Student Manual

STL academy

Subject

Java Fundamentals

Vol.01

Empowering Youth!

ISO 21001 | Guinness World Records Record Holder | Asia Book of Records

Skill & Assessment Partner NASSCOM®

stl.tech | stlacad.tech

Java Programmer

Submitted to :- **Bihar Skill Development Mission, Labour Resources Department, GoB**	Submitted By :- **Sterlite Technologies Ltd**
	Session : 2022-23

Course name:

- Course Id-
- Candidate Eligibility : Diploma/ Graduate
- Course Duration: (In hours) 580

CONTACT DETAILS OF THE BODY SUBMITTING THE QUALIFICATION FILE

Name and address of submitting body:

Sterlite Technologies Ltd

Name and contact details of individual dealing with the submission

Name : Mrs./Mr. Srikant Pattnaik

Position in the organization : Manager

Tel number (s) (Mobile no.) : 9702048264

Website : www.stlacad.tech

E-mail address : srikant.pattnaik@stl.tech

STL academy

JAVA PROGRAMMER

STUDENT GUIDE

About the Student Guide

The student guide contains modules which will help you to acquire relevant knowledge and skills (generic and domain-specific skills) related to the 'Java Programmer' job role. Knowledge in each module is easily understood and grasped by you before you move on to the next module. Comprehensible diagrams & images from world of work have been included to bring about visual appeal and to make the text lively and interactive for you. You can also try to create your own illustrations using your imagination or taking the help of your trainer.

Let us now see what the sections in the modules have for you.

Section 1: Learning Outcome

This section introduces you to the learning objectives and knowledge criteria covered in the module. It also tells you what you will learn through the various topics covered in the module.

Section 2: Relevant Knowledge

This section provides you with the knowledge to achieve relevant skill and proficiency to perform tasks of the Java Programmer. The knowledge developed through the module will enable you to perform certain activities related to the job market. You should read through the textual information to develop an understanding on the various aspects of the module before you complete the exercise(s).

Section 3: Exercises

Each module has exercises, which you should practice on completion of the learning sessions of the module. You will perform the activities in the classroom, at home or at the workplace. The activities included in this section will help you to develop necessary knowledge, skills and attitude that you need for becoming competent in performing the tasks at workplace. The activities should be done under the supervision of your trainer who will guide you in completing the tasks and also provide feedback to you for improving your performance.

Section 4: Assessment Questionnaire

The review questions included in this section will help you to check your progress. You must be able to answer all the questions before you proceed to the next module.

STL academy

CONTENTS

MODULE 1
JAVA BASICS

Section 1: Learning Outcomes

After completing this module, you will be able to:

- Explain Object-oriented concepts
- Tell History of Java, Java Buzzwords
- Describe various Data types
- Explain about Variables
- Explain JVM Architecture
- Describe Scope and Life time of Variables
- Explain Arrays, Operators, Control, Statements, Type Conversion, Casting, Simple java program, Constructors
- Distinguish between Static Method String and String Buffer Classes
- Execute a Simple Java program

Section 2: Relevant Knowledge

1.1 Review of Object-oriented Concepts

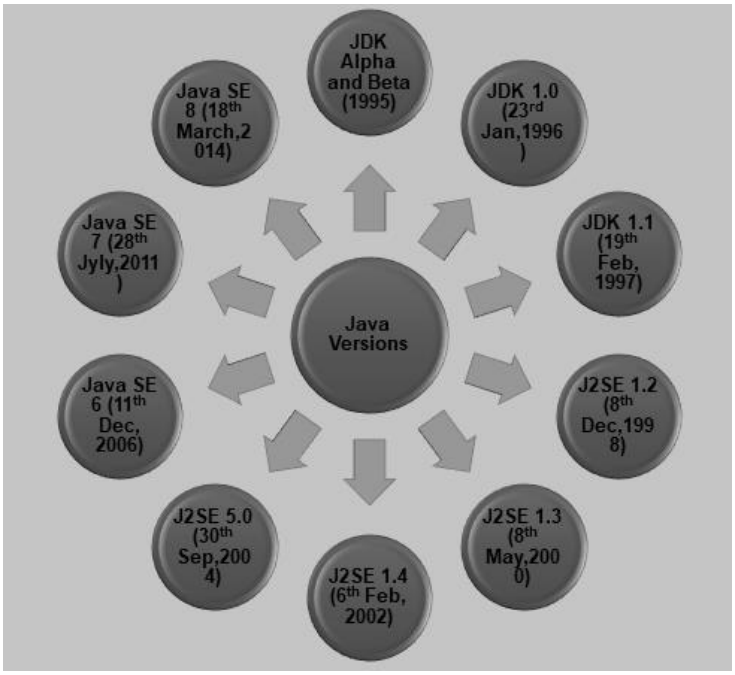
Basically, java like any other object-oriented programming language involves real world entities like:

- Object - This is something that has a physical state and behaviour like a book, person, car etc.
- Class - A collection of objects performing a certain task.
- Inheritance - When an object gets properties from a parent class.
- Polymorphism - A task being performed in different ways
- Abstraction - Showing the important things to the user and hiding the functionalities that are not necessary for users to see.
- Encapsulation - To stick code and data together into a single unit.
- Coupling - Refers a class taking information or depending on another class.
- Association - Refers to the relationship between objects.

1.2 History of Java, Java Buzzwords

- It was initiated in June 1991 by James Gosling, Mike Sheridan and Patrick Naughton.
- James Gosling first called it “Greentalk” and the file extension was ‘.gt’.
- Then it was called Oak and was developed as a part of the green project.
- It was named Oak because Oak is a symbol of strength and chosen as a national tree of many countries like France, Germany, U.S.A, Romania etc.
- Since it was already a trademark by Oak Technologies, it was named Java in 1995.
- Several words like silk, DNA, jolt, dynamic, java, revolutionary but java was preferred because of its uniqueness and most people in the team preferred java.

Java Version History



Why Use Java?

- Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
- It is one of the most popular programming languages in the world
- It is easy to learn and simple to use
- It is open-source and free
- It is secure, fast and powerful
- It has a huge community support (tens of millions of developers)
- Java is an object-oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
- As Java is close to C++ and C#, it makes it easy for programmers to switch to Java or vice versa.

Java Install

- Some PCs might have Java already installed.
- To check if you have Java installed on a Windows PC, search in the start bar for Java or type the following in Command Prompt (cmd.exe):

```
C:\Users\Your Name>java -version
```

If Java is installed, you will see something like this (depending on version):
Note:- If you do not have Java installed on your computer, you can download it for free at oracle.com.

```
java version "11.0.1" 2018-10-16 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.1+13-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.1+13-LTS, mixed mode)
```

Java Quick Start

- In Java, every application begins with a class name, and that class must match the filename.
- Let's create our first Java file, called Main.java, which can be done in any text editor (like Notepad).
- The file should contain a "Hello World" message, which is written with the following code:

Main.java

```
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

Try it Yourself »

- Don't worry if you don't understand the code above - we will discuss it in detail in later chapters.
- For now, focus on how to run the code above.
- Save the code in Notepad as "Main.java". Open Command Prompt (cmd.exe), navigate to the directory where you saved your file, and type "javac Main.java":

```
C:\Users\Your Name>javac Main.java
```

- This will compile your code. If there are no errors in the code, the command prompt will take you to the next line. Now, type "java Main" to run the file:

```
C:\Users\Your Name>java Main
```

The output should read:

```
Hello World
```

Try it Yourself »

Java Syntax

- In the previous chapter, we created a Java file called Main.java, and we used the following code to print "Hello World" to the screen:

Example Explained

- Every line of code that runs in Java must be inside a class. In our example, we named the class Main. A class should always start with an uppercase first letter.
- Note: Java is case-sensitive: "MyClass" and "myclass" has different meaning.
- The name of the java file must match the class name. When saving the file, save it using the class name and add ".java" to the end of the filename.

The Main Method

The main() method is required and you will see it in every Java program:

```
public static void main(String[] args)
```

System.out.println()

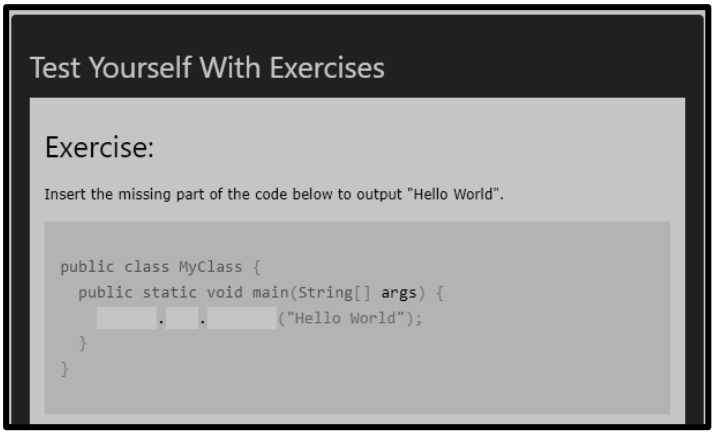
Inside the main() method, we can use the println() method to print a line of text to the screen:

```
public static void main(String[] args) {
  System.out.println("Hello World");
}
```

Note: The curly braces {} marks the beginning and the end of a block of code

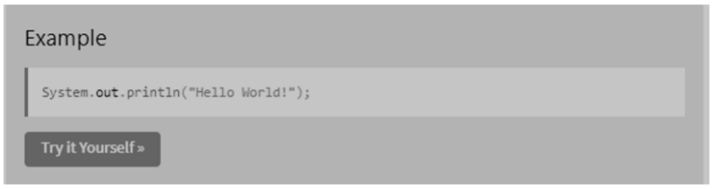
Simple Java Programme

```
class simple{
  public Static void main (string args[]) {
  System.out.println(“Hello World! ”);
   }
}
```



Java Output

You learned from the previous chapter that you can use the println() method to output values or print text in Java:



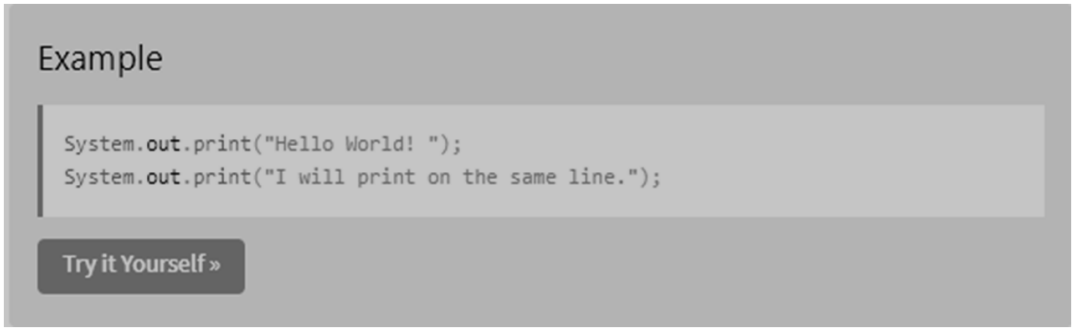
You can add as many println() methods as you want. Note that it will add a new line for each method:

You can also output numbers, and perform mathematical calculations:

Note that we don't use double quotes ("") inside `println()` to output numbers.

The Print() Method

There is also a print() method, which is similar to println().
The only difference is that it does not insert a new line at the end of the output:



Note that we add an extra space (after "Hello World!" in the example above), for better readability.

Java Comments

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

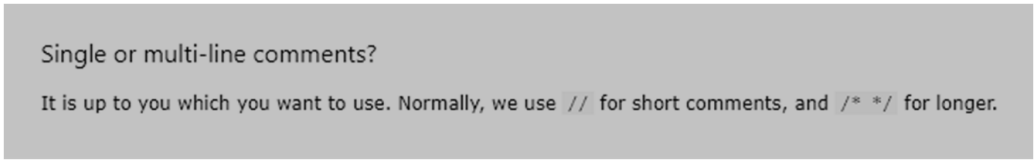
Single-line Comments

Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by Java

Java Multi-line Comments

- Multi-line comments start with /* and ends with */.
- Any text between /* and */ will be ignored by Java.
- This example uses a multi-line comment (a comment block) to explain the code:

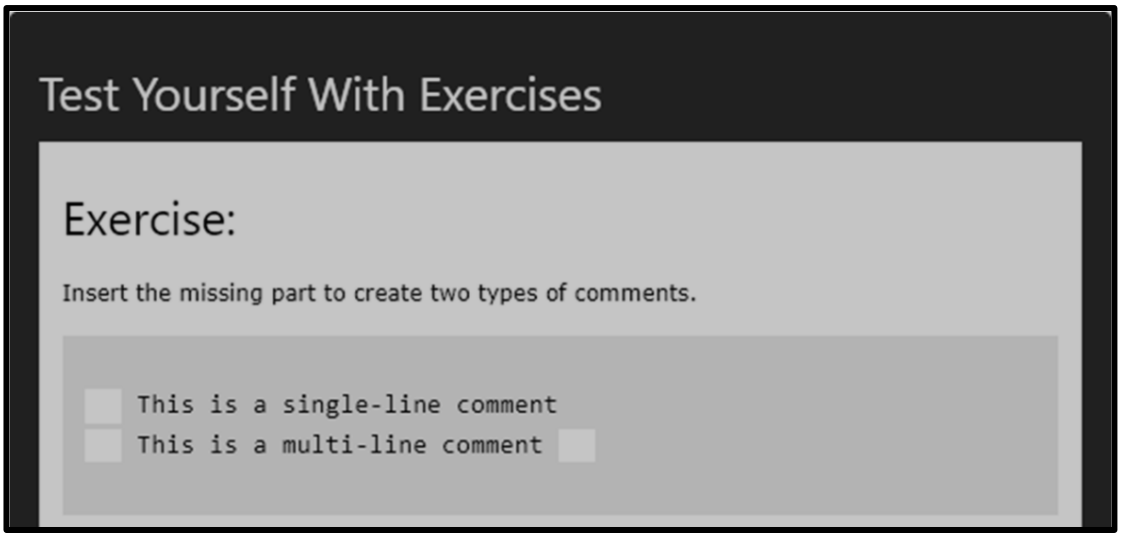


1.3 Variables

Variables are storage locations.

They are identified by unique names which are called identifiers which can be short or more descriptive eg. X, Area, George. When you are naming variables, you must follow certain rules which include:

- They can contain letters, digits, underscores, and dollar signs.
- They must begin with a letter
- Cannot contain white space and must start in lower case
- They are case sensitive
- Reserved words cannot be used as names



Java Variables

Variables are containers for storing data values.

In Java, there are different types of variables, for example:

- String - stores text, such as "Hello". String values are surrounded by double quotes
- int - stores integers (whole numbers), without decimals, such as 123 or -123.
- float - stores floating point numbers, with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes.
- boolean - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

Where type is one of Java's types (such as int or String), and variableName is the name of the variable (such as x or name). The equal sign is used to assign values to the variable.

Syntax

```
type variableName = value;
```

To create a variable that should store text, look at the following example:

Example

Create a variable called **name** of type `String` and assign it the value "**John**":

```
String name = "John";
System.out.println(name);
```

You can also declare a variable without assigning the value, and assign the value later:

```
int myNum;
myNum = 15;
System.out.println(myNum);
```

That if you assign a new value to an existing variable, it will overwrite the previous value:

Change the value of `myNum` from `15` to `20`:

```
int myNum = 15;
myNum = 20;  // myNum is now 20
System.out.println(myNum);
```

A demonstration of how to declare variables of other types:

Example

```
int myNum = 5;
float myFloatNum = 5.99f;
char myLetter = 'D';
boolean myBool = true;
String myText = "Hello";
```

Final Variables

If you don't want others (or yourself) to overwrite existing values, use the final keyword (this will declare the variable as "final" or "constant", which means unchangeable and read-only):

Example

```
final int myNum = 15;
myNum = 20;  // will generate an error: cannot assign a value to a final variable
```

Try it Yourself »

Java Print Variables

Display Variables

The println() method is often used to display variables.

To combine both text and a variable, use the + character:

Example

```
String name = "John";
System.out.println("Hello " + name);
```

Test Yourself With Exercises

Exercise:

Create a variable named `carName` and assign the value `Volvo` to it.

```
      =      ;
```

Submit Answer »

Start the Exercise

You can also use the + character to add a variable to another variable:

For numeric values, the + character works as a mathematical operator (notice that we use int (integer) variables here):

Example

```
String firstName = "John ";
String lastName = "Doe";
String fullName = firstName + lastName;
System.out.println(fullName);
```

Example

```
int x = 5;
int y = 6;
System.out.println(x + y); // Print the value of x + y
```

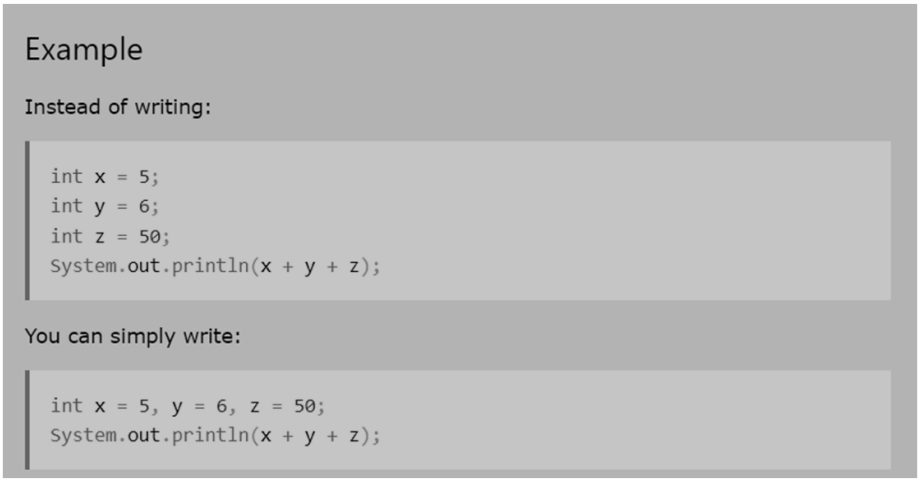
Try it Yourself »

From the example above, you can expect:

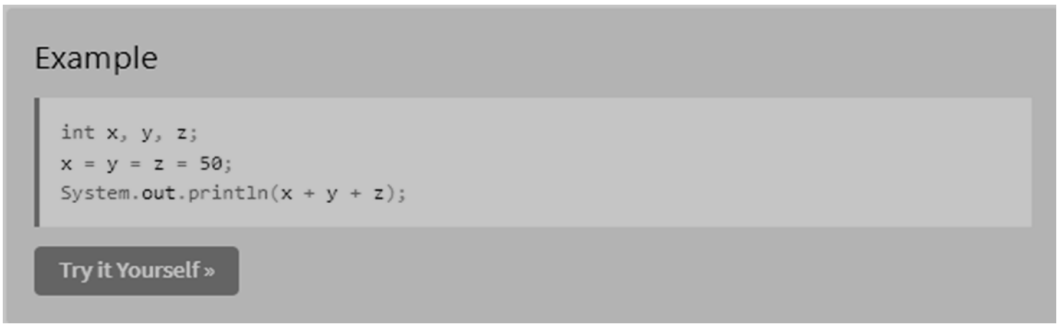
- x stores the value 5
- y stores the value 6
- Then we use the println() method to display the value of x + y, which is 11

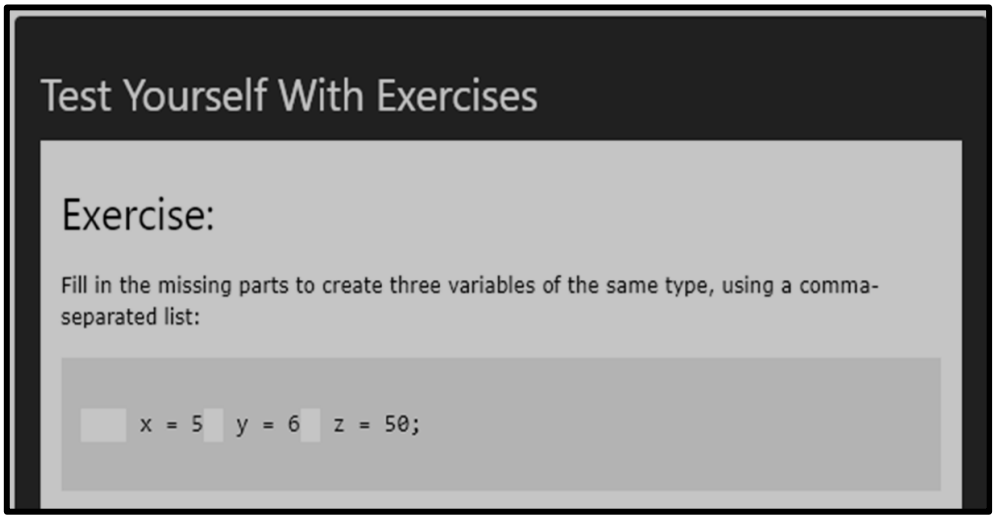
Java Declare Multiple Variables

- Declare Many Variables
- To declare more than one variable of the same type, you can use a comma-separated list:
- One Value to Multiple Variables



- You can also assign the same value to multiple variables in one line:





Java Identifiers

- All Java variables must be identified with unique names.
- These unique names are called identifiers.
- Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: It is recommended to use descriptive names in order to create understandable and maintainable code:

The general rules for naming variables are:

- Names can contain letters, digits, underscores, and dollar signs
- Names must begin with a letter
- Names should start with a lowercase letter and it cannot contain whitespace
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive ("myVar" and "myvar" are different variables)
- Reserved words (like Java keywords, such as int or boolean) cannot be used as names.

Example

```
// Good
int minutesPerHour = 60;

// OK, but not so easy to understand what m actually is
int m = 60;
```

1.4 Data Types

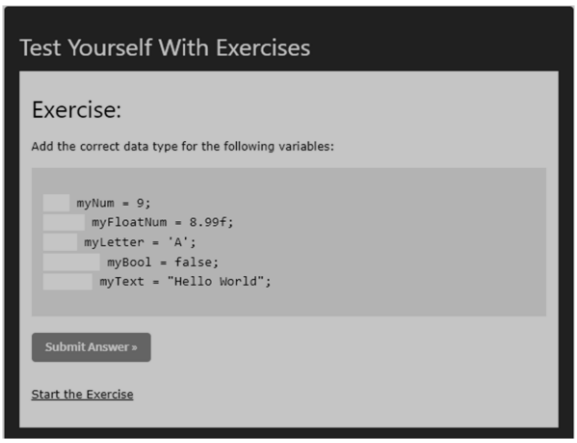
- Data types are the classification of data that which specifies the type of value a variable hold.
- Data types can be categorized into two groups:
- Primitive data types - includes byte, short, int, long, float, double, boolean and char
- Non-primitive data types - such as String, Arrays and Classes.
- PRIMITIVE DATA TYPES - A primitive data type specifies the size and type of variable values, and it has no additional methods.
- There are eight primitive data types in Java:

Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

Data Type	Size	Description
`byte`	1 byte	Stores whole numbers from -128 to 127
`short`	2 bytes	Stores whole numbers from -32,768 to 32,767
`int`	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
`long`	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`float`	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
`double`	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
`boolean`	1 bit	Stores true or false values
`char`	2 bytes	Stores a single character/letter or ASCII values



Java Numbers

- Primitive number types are divided into two groups:
- Integer types stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are byte, short, int and long. Which type you should use, depends on the numeric value.
- Floating point types represents numbers with a fractional part, containing one or more decimals. There are two types: float and double.

Even though there are many numeric types in Java, the most used for numbers are `int` (for whole numbers) and `double` (for floating point numbers). However, we will describe them all as you continue to read.

Integer Types

Byte

The byte data type can store whole numbers from -128 to 127. This can be used instead of int or other integer types to save memory when you are certain that the value will be within -128 and 127:

Example

```
byte myNum = 100;
System.out.println(myNum);
```

SHORT

The short data type can store whole numbers from -32768 to 32767:

Example

```
short myNum = 5000;
System.out.println(myNum);
```

Try it Yourself »

INT

The int data type can store whole numbers from -2147483648 to 2147483647. In general, and in our tutorial, the int data type is the preferred data type when we create variables with a numeric value.

Example

```
int myNum = 100000;
System.out.println(myNum);
```

LONG

The long data type can store whole numbers from -9223372036854775808 to 9223372036854775807. This is used when int is not large enough to store the value. Note that you should end the value with an "L":

Floating Point Types

- You should use a floating point type whenever you need a number with a decimal, such as 9.99 or 3.14515.
- The float and double data types can store fractional numbers. Note that you should end the value with an "f" for floats and "d" for doubles:

Boolean Types

A boolean data type is declared with the boolean keyword and can only take the values true or false:

Example

```
boolean isJavaFun = true;
boolean isFishTasty = false;
System.out.println(isJavaFun);     // Outputs true
System.out.println(isFishTasty);   // Outputs false
```

Java Characters

- Characters
- The char data type is used to store a single character. The character must be surrounded by single quotes, like 'A' or 'c':

Example

```
char myGrade = 'B';
System.out.println(myGrade);
```

Try it Yourself »

Strings

- The String data type is used to store a sequence of characters (text). String values must be surrounded by double quote.

Example

```
String greeting = "Hello World";
System.out.println(greeting);
```

Java Non-Primitive Data Types

Non-primitive data types are called reference types because they refer to objects.

The main difference between primitive and non-primitive data types are:

- Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type has always a value, while non-primitive types can be null.
- A primitive type starts with a lowercase letter, while non-primitive types start with an uppercase letter.
- The size of a primitive type depends on the data type, while non-primitive types have all the same size.

STL academy

Java Type Casting

- Type casting is when you assign a value of one primitive data type to another type.
- In Java, there are two types of casting:
  - Widening Casting (automatically) - converting a smaller type to a larger type size byte -> short -> char -> int -> long -> float -> double
  - Narrowing Casting (manually) - converting a larger type to a smaller size type double -> float -> long -> int -> char -> short -> byte

Widening Casting

Widening casting is done automatically when passing a smaller size type to a larger size type:

Example

```
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
}
```

Try it Yourself »

Narrowing Casting

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

Example

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

Try it Yourself »

STL academy

Java Operators

These are symbols used for doing operations they include:

- Unary
- Bitwise operator
- Logical operator
- Ternary
- Relational operator
- Arithmetic
- Shift operator
- Assignment

- Operators are used to perform operations on variables and values.
- In the example below, we use the + operator to add together two values:

Arithmetic Operators

Example

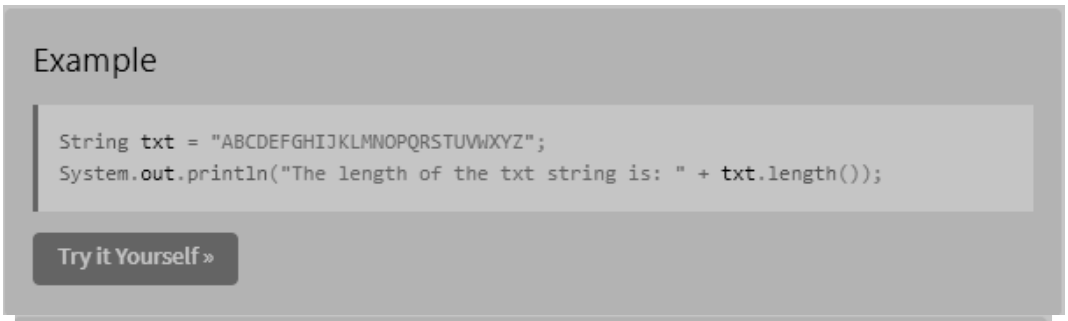
```
int x = 100 + 50;
```

Operator	Name	Description	Example	Try it
+	Addition	Adds together two values	x + y	Try it »
-	Subtraction	Subtracts one value from another	x - y	Try it »
*	Multiplication	Multiplies two values	x * y	Try it »
/	Division	Divides one value by another	x / y	Try it »
%	Modulus	Returns the division remainder	x % y	Try it »
++	Increment	Increases the value of a variable by 1	++x	Try it »
--	Decrement	Decreases the value of a variable by 1	--x	Try it »

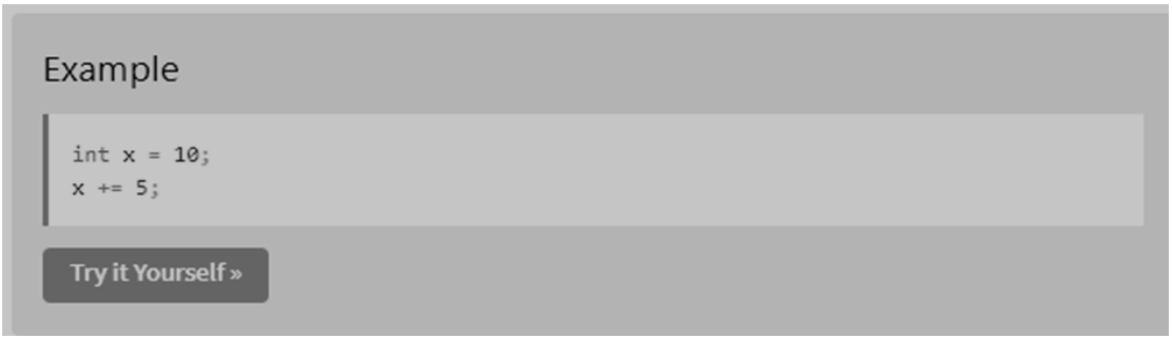
STL academy

Java Assignment Operators

- Assignment operators are used to assign values to variables.
- In the example below, we use the assignment operator (=) to assign the value 10 to a variable called x:

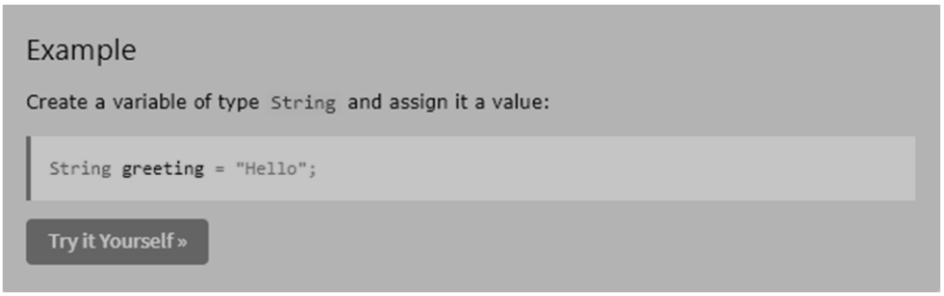


- The addition assignment operator (+=) adds a value to a variable:



Java Strings

- Strings are used for storing text.
- A String variable contains a collection of characters surrounded by double quotes:

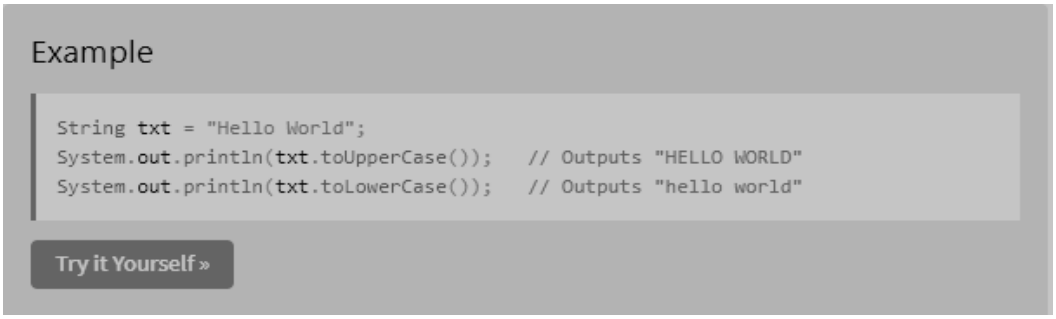


String Length

A String in Java is actually an object, which contain methods that can perform certain operations on strings. For example, the length of a string can be found with the length() method:

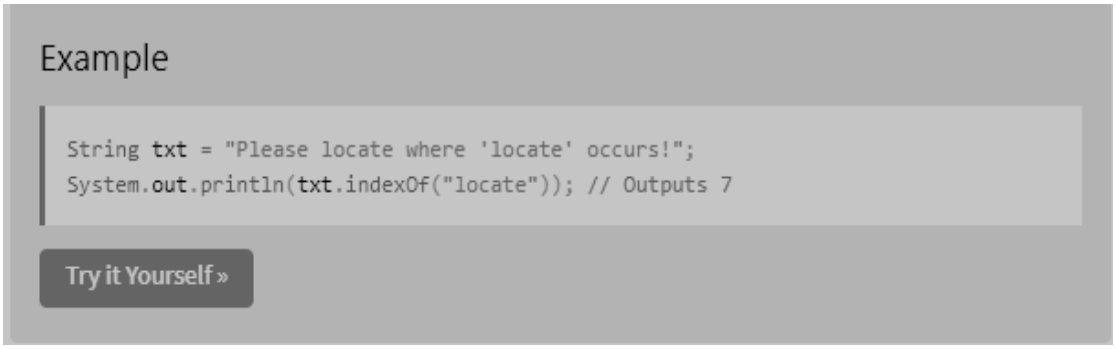
More String Methods

There are many string methods available, for example toUpperCase() and toLowerCase():



Finding a Character in a String

The indexOf() method returns the index (the position) of the first occurrence of a specified text in a string (including whitespace):



Java counts positions from zero.
0 is the first position in a string, 1 is the second, 2 is the third ...

Java String Concatenation

The + operator can be used between strings to combine them. This is called concatenation:

Example

```
String firstName = "John";
String lastName = "Doe";
System.out.println(firstName + " " + lastName);
```

Try it Yourself »

Note that we have added an empty text (" ") to create a space between firstName and lastName on print.

You can also use the concat() method to concatenate two strings:

Example

```
String firstName = "John ";
String lastName = "Doe";
System.out.println(firstName.concat(lastName));
```

Try it Yourself »

Adding Numbers and Strings

WARNING!

Java uses the + operator for both addition and concatenation.

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

Example

```
int x = 10;
int y = 20;
int z = x + y;  // z will be 30 (an integer/number)
```

Try it Yourself »

Java Special Characters

Because strings must be written within quotes, Java will misunderstand this string, and generate an error:

```
String txt = "We are the so-called "Vikings" from the north.";
```

The solution to avoid this problem, is to use the backslash escape character. The backslash (\) escape character turns special characters into string characters:

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

The sequence \" inserts a double quote in a string:

Example

```
String txt = "We are the so-called \"Vikings\" from the north.";
```

Try it Yourself »

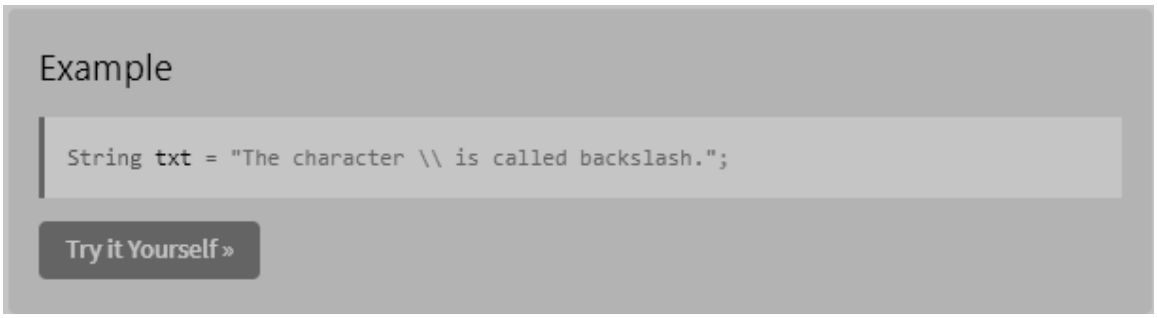
The sequence \' inserts a single quote in a string:

Example

```
String txt = "It\'s alright.";
```

Try it Yourself »

The sequence \\ inserts a single backslash in a string:



Other common escape sequences that are valid in Java are

Code	Result	Try it
\n	New Line	Try it »
\r	Carriage Return	Try it »
\t	Tab	Try it »
\b	Backspace	Try it »
\f	Form Feed	

Java Math

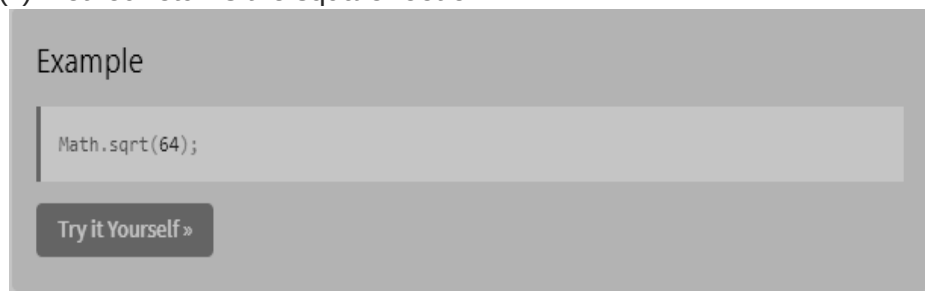
The Java Math class has many methods that allows you to perform mathematical tasks on numbers.
Math.max(x,y)

The Math.max(x,y) method can be used to find the highest value of x and y:

Math.min(x,y)
The Math.min(x,y) method can be used to find the lowest value of x and y:

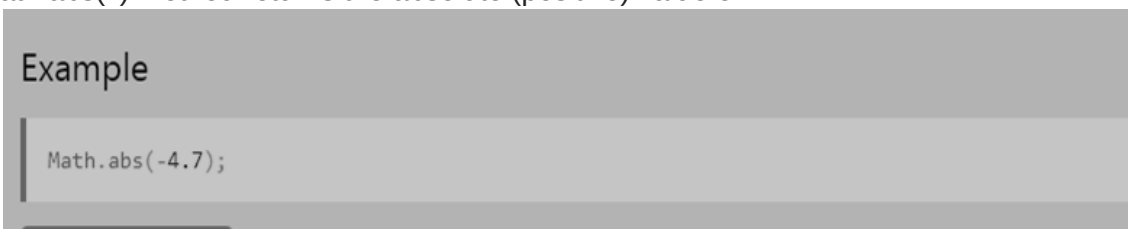
Math.sqrt(*x*)

The Math.sqrt(*x*) method returns the square root of *x*:



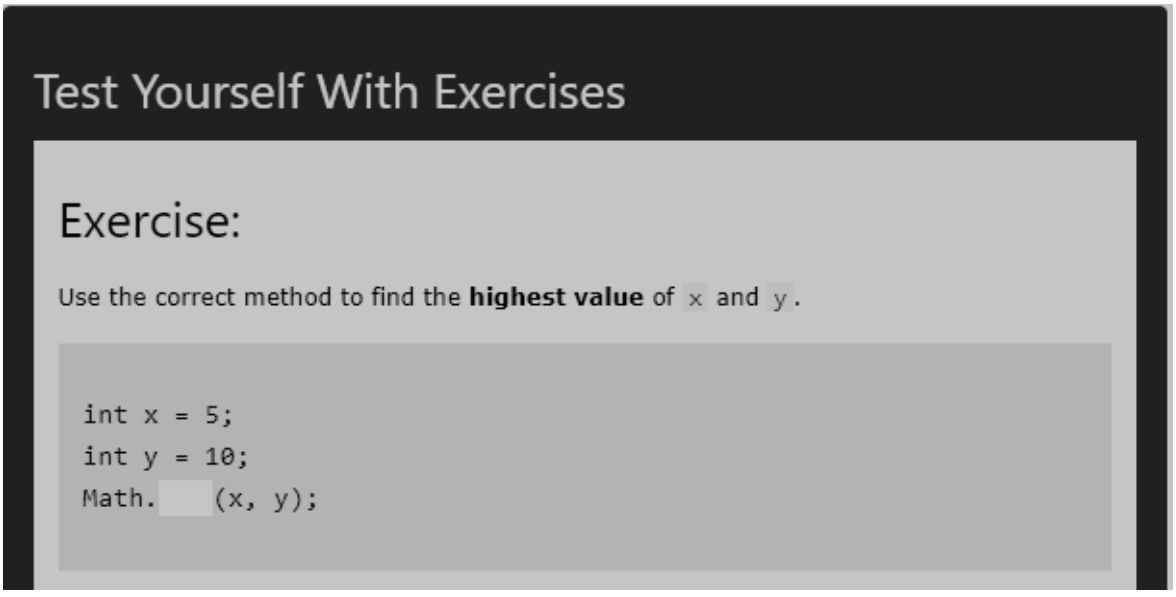
Math.abs(x)

The Math.abs(*x*) method returns the absolute (positive) value of *x*:



Random Numbers

Math.random() returns a random number between 0.0 (inclusive), and 1.0 (exclusive):



STL academy

Java Booleans

Very often, in programming, you will need a data type that can only have one of two values, like:

- YES / NO
- ON / OFF
- TRUE / FALSE

For this, Java has a boolean data type, which can take the values true or false.

Boolean Values

A boolean type is declared with the boolean keyword and can only take the values true or false:

Example

```
boolean isJavaFun = true;
boolean isFishTasty = false;
System.out.println(isJavaFun);     // Outputs true
System.out.println(isFishTasty);   // Outputs false
```

Try it Yourself »

However, it is more common to return boolean values from boolean expressions, for conditional testing (see below).

Boolean Expression

- A **Boolean expression** is a Java expression that returns a Boolean value: true or false.
- You can use a comparison operator, such as the **greater than** (>) operator to find out if an expression (or a variable) is true:

Example

```
int x = 10;
int y = 9;
System.out.println(x > y); // returns true, because 10 is higher than 9
```

Try it Yourself »

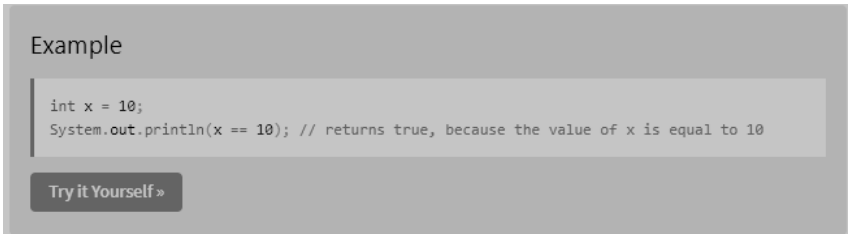
Or even easier:

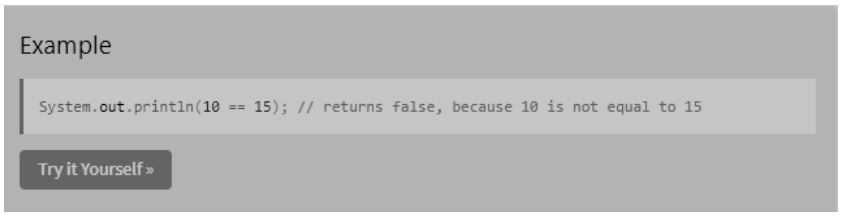
Example

```
System.out.println(10 > 9); // returns true, because 10 is higher than 9
```

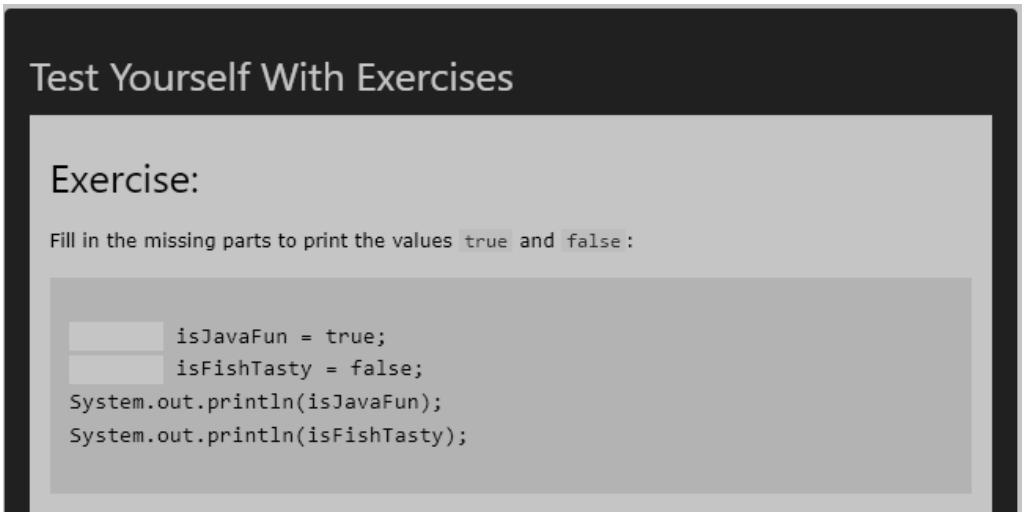
Try it Yourself »

In the examples below, we use the equal to (==) operator to evaluate an expression:





The Boolean value of an expression is the basis for all Java comparisons and conditions. You will learn more about conditions in the next chapter.



Java If ... Else

Java Conditions and If Statements

- Java supports the usual logical conditions from mathematics:
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b
- Equal to a == b
- Not Equal to: a != b

You can use these conditions to perform different actions for different decisions.

- Java has the following conditional statements:
- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

STL academy

The if Statement

- Use the if statement to specify a block of Java code to be executed if a condition is true.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

Note that `if` is in lowercase letters. Uppercase letters (If or IF) will generate an error.

In the example below, we test two values to find out if 20 is greater than 18. If the condition is true, print some text:

Example

```
if (20 > 18) {
  System.out.println("20 is greater than 18");
}
```

Try it Yourself »

We can also test variables:

Example

```
int x = 20;
int y = 18;
if (x > y) {
  System.out.println("x is greater than y");
}
```

Try it Yourself »

Example Explained

In the example above we use two variables, **x** and **y**, to test whether x is greater than y (using the > operator). As x is 20, and y is 18, and we know that 20 is greater than 18, we print to the screen that "x is greater than y".

The else Statement

Use the else statement to specify a block of code to be executed if the condition is false.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

Example

```
int time = 20;
if (time < 18) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
// Outputs "Good evening."
```

Try it Yourself »

Example Explained

In the example above, time (20) is greater than 18, so the condition is false. Because of this, we move on to the else condition and print to the screen "Good evening". If the time was less than 18, the program would print "Good day".

The else if Statement

Use the else if statement to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

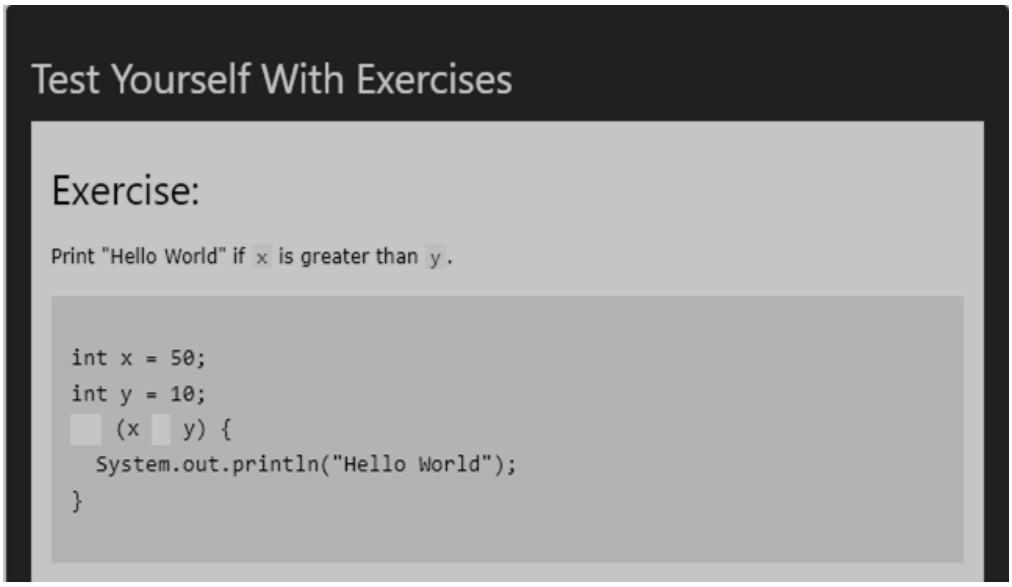
Example

```
int time = 22;
if (time < 10) {
  System.out.println("Good morning.");
} else if (time < 20) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
// Outputs "Good evening."
```

Try it Yourself »

Example explained

In the example above, time (22) is greater than 10, so the **first condition** is false. The next condition, in the else if statement, is also false, so we move on to the else condition since **condition1** and **condition2** is both false - and print to the screen "Good evening". However, if the time was 14, our program would print "Good day."



Java Short Hand If...Else (Ternary Operator)

There is also a short-hand if else, which is known as the ternary operator because it consists of three operands.

It can be used to replace multiple lines of code with a single line, and is most often used to replace simple if else statements:

```
variable = (condition) ? expressionTrue :  expressionFalse;
```

Syntax
Instead of writing:
Example
You can simply write:

Example:

```
int time = 20;
String result = (time < 18) ? "Good day." : "Good evening.";
System.out.println(result);
```

Test Yourself With Exercises

Exercise:

Insert the missing parts to complete the following "short hand if...else" statement:

```
int time = 20;
String result =  time < 18    "Good day."   "Good evening.";
System.out.println(result);
```

Java Switch

- Java Switch Statements
- Use the switch statement to select one of many code blocks to be executed.

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

STL academy

Java Switch Working:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- The break and default keywords are optional, and will be described later in this chapter

Below are the few examples to understand the working:

Example

```
int day = 4;
switch (day) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednesday");
    break;
  case 4:
    System.out.println("Thursday");
    break;
  case 5:
    System.out.println("Friday");
    break;
  case 6:
    System.out.println("Saturday");
    break;
  case 7:
    System.out.println("Sunday");
    break;
}
// Outputs "Thursday" (day 4)
```

Try it Yourself »

The Break Keyword

- When Java reaches a break keyword, it breaks out of the switch block.
- This will stop the execution of more code and case testing inside the block.
- When a match is found, and the job is done, it's time for a break. There is no need for more testing.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

Test Yourself With Exercises

Exercise:

Insert the missing parts to complete the following `switch` statement.

```
int day = 2;
switch (    ) {
    1:
    System.out.println("Saturday");
    break;
    2:
    System.out.println("Sunday");
    ;
}
```

1.5 JVM Architecture

It is very necessary to understand and learn about JVM because it enables us to write code –

- More efficiently.
- A virtual machine is the implementation of a physical machine.
- Java Runtime Environment (JRE) executes byte code and JRE is an implementation of JVM.
- WORA (Write Once Run Anywhere) is the concept which java language was developed with and it runs on a virtual machine.
- A compiler compiles a java file into a java .class file, the that .class file is input into the JVM.
- This loads and executes the class file.

1.6 Scope and Time, Life Time of Variables

- Scope of a variable is the area or parts of the program where the variable can be accessed
- Life time of a variable refers to the amount of time a variable stays alive in the memory

Method Scope

- Variables declared directly inside a method are available anywhere in the method following the line of code in which they were declared:

Example

```
public class Main {
  public static void main(String[] args) {

    // Code here CANNOT use x

    int x = 100;

    // Code here can use x
    System.out.println(x);
  }
}
```

Block Scope

- A block of code refers to all of the code between curly braces {}.
- Variables declared inside blocks of code are only accessible by the code between the curly braces, which follows the line in which the variable was declared:

Example

```
public class Main {
  public static void main(String[] args) {

    // Code here CANNOT use x

    { // This is a block

      // Code here CANNOT use x

      int x = 100;

      // Code here CAN use x
      System.out.println(x);

    } // The block ends here

    // Code here CANNOT use x

  }
}
```

1.7 Arrays, operators, control, statements, type coversion, casting, simple java program, constructors

Arrays

Arrays are a collection of multiple values of the same type stored in the same variable. They can be categorised into a single-dimension or multi-dimension array.

Example:

To declare an array, define the variable type with **square brackets**:

```
String[] cars;
```

- We have now declared a variable that holds an array of strings. To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

To create an array of integers, you could write:

```
int[] myNum = {10, 20, 30, 40};
```

Control statements

They are used to set order in a program instead of it running from top to bottom. They are three categories of control statements:

- Selection statements
- Jump statements
- Iteration statements

Type Conversion

This is the conversion of one data type either from primitive to non-primitive and vice-versa also known as casting.

Constructors

It is a special method that is called when an object of a class is created also used to initialize objects.

1.8 Static Method string and String Buffer classes

Static method

- It is most commonly used to access static variables.
- Declared using the keyword “static”
- A static method belongs to a class not an instance of a class

String

Strings are used to store text characters. They are immutable and have a fixed length. String methods include:

To Upper Case ()
To Lower Case ()
Length ()

String Buffer classes

- String buffer classes are like strings but have much functionality of strings
- Unlike strings they are not immutable and they are growable and writable
- String Buffer methods include:

CharAt ()
Length ()
Index of ()

Section 3: Exercises

Exercise 1: Write a Java program to print 'Hello' on screen and then print your name on a separate line.
Expected Output:
Hello
Alexandra Abramov

Exercise 2: Write a Java program to print the sum of two numbers.

Test Data:

24 + 26

Expected Output:

50

Exercise 3: Write a Java program to divide two numbers and print on the screen.

Test Data:

50/3

Expected Output:

16

Exercise 4: Write a Java program to print the result of the following operations.

Test Data:

a. -5 + 8 * 6

b. (55+9) % 9

c. 20 + -3*5 / 8

d. 5 + 15 / 3 * 2 - 8 % 3

Expected Output:

43

1

19

13

Exercise 5: Write a Java program that takes two numbers as input and display the product of two numbers.

Test Data:

Input first number: 25

Input second number: 5

Expected Output:

25 x 5 = 125

Exercise 6: Write a Java program to print the sum (addition), multiply, subtract, divide and remainder of two numbers.

Test Data:

Input first number: 125

Input second number: 24

Expected Output:

125 + 24 = 149

125 - 24 = 101

125 x 24 = 3000

125 / 24 = 5

125 mod 24 = 5

Exercise 7: Write a Java program that takes a number as input and prints its multiplication table upto 10.

Test Data:

Input a number: 8

Expected Output:

8 x 1 = 8

8 x 2 = 16

8 x 3 = 24

...8 x 10 = 80

Exercise 8: Write a Java program to display the following pattern.

Sample Pattern:

```
  J   a  v    v a
  J   a a  v   v a a
J J  aaaaa   V V  aaaaa
 JJ a    a   V a    a
```

Exercise 9: Write a Java program to compute the specified expressions and print the output.

Test Data:

((25.5 * 3.5 - 3.5 * 3.5) / (40.5 - 4.5))

Expected Output

2.138888888888889

Exercise 10: Write a Java program to compute a specified formula.

Specified Formula:

4.0 * (1 - (1.0/3) + (1.0/5) - (1.0/7) + (1.0/9) - (1.0/11))

Expected Output

2.9760461760461765

STL academy

Section 4: Assessment Questionnaire

1. Number of primitive data types in Java is?
   a) 6
   b) 7
   c) 8
   d) 9

2. What is the size of float and double in java?
   a) 32 and 64
   b) 32 and 32
   c) 64 and 64
   d) 64 and 32

3. Automatic type conversion is possible in which of the possible cases?
   a) Byte to int
   b) Int to long
   c) Long to int
   d) Short to int

4. Find the output of the following code.

```
int Integer = 24;
   char String  = 'I';
   System.out.print(Integer);
   System.out.print(String);
```

   a) Compile error
   b) Throws exception
   c) I
   d) 24 I

5. Find the output of the following program.

```
public class Solution{
    public static void main(String[] args){
       short x = 10;
       x =  x * 5;
       System.out.print(x);
        }
   }
```

   a) 50
   b) 10
   c) Compile error
   d) Exception

6. Find the output of the following program.

```
public class Solution{
    public static void main(String[] args){
        byte x = 127;
        x++;
        x++;
        System.out.print(x);
        }
    }
```

a) -127
b) 127
c) 129
d) 245

7. Select the valid statement.
a) char[] ch = new char(5)
b) char[] ch = new char[5]
c) char[] ch = new char()
d) char[] ch = new char[]

8. Find the output of the following program.

```
public class Solution{
    public static void main(String[] args){
        int[]  x = {120, 200, 016};
        for(int i = 0; i < x.length; i++){
            System.out.print(x[i] + " ");
        }
    }
}
```

a) 120 200 016
b) 120 200 14
c) 120 200 16
d) None

9. When an array is passed to a method, what does the method receive?
a) The reference of the array
b) A copy of the array
c) Length of the array
d) Copy of first element

10. Select the valid statement to declare and initialize an array.
a) int[] A = {}
b) int[] A = {1, 2, 3}
c) int[] A = (1, 2, 3)
d) int[][] A = {1,2,3}

11. What is JAVA?

12. How does Java enable high performance?

13. Name the Java IDE's?

14. What do you mean by Constructor?

15. What is Inheritance?

16. What is Encapsulation?

17. What is SerialVersionUID?

18. What is the purpose of a Volatile Variable?

19. What is the disadvantage of Synchronization?

20. What is Synchronization?

21. When to use the Runnable interface Vs Thread class in Java?

22. Difference between start() and run() method of thread class.

23. Difference between notify() method and notifyAll() method in Java.

24. Explain about join () method.

25. What is the final keyword in Java?

26. What are the Exception handling keywords in Java?

27. What are the advantages of Exception handling?

28. What are the types of Exceptions?

29. Explain the Priority Queue.

30. What is the meaning of Collections in Java?

----------End of the Module----------

MODULE 2
INHERITANCE AND POLYMORPHISM

Section 1: Learning Outcomes

After completing this module, you will be able to:

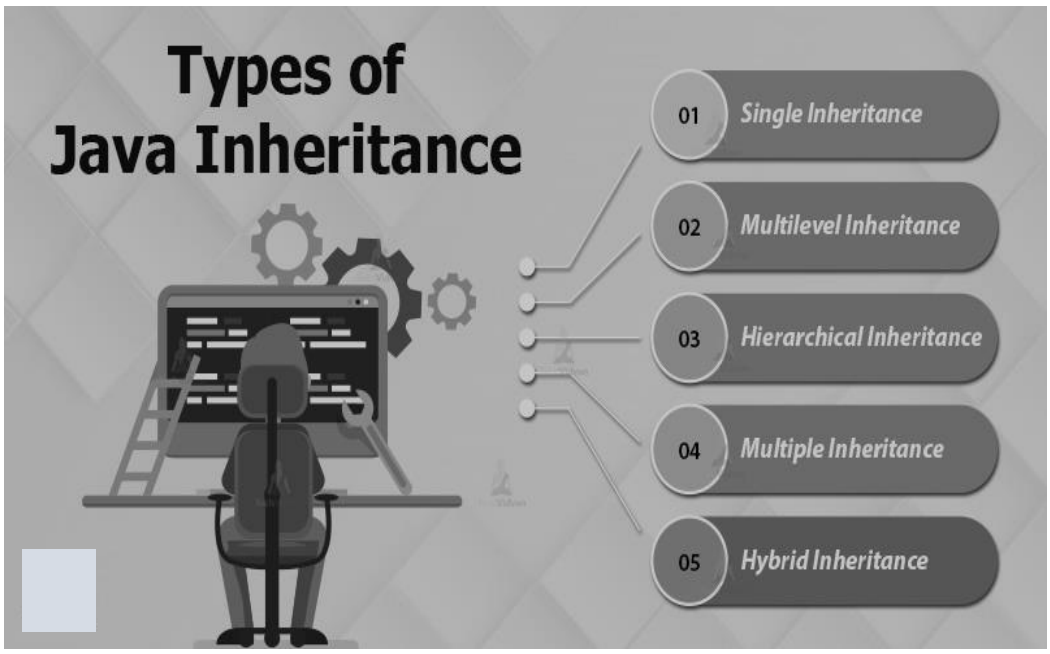
- Describe basic concepts of Inheritance and Polymorphism
- Explain Types of inheritance and Member access rules
- Explain Method Overloading as well as Method overriding
- Describe Abstract Classes
- State Usage of this and Super key word
- Explain Packages and Interfaces: Defining package, Access protection, importing packages
- Explain I/O Streams: Concepts of streams, Stream classes-Byte and Character stream
- Classify Reading console Input and Writing Console output
- Describe File Handling

Section 2: Relevant Knowledge

2.1 Basic Concepts

Inheritance: As discussed earlier inheritance is when a child or sub-class gets some features from a parent class. Inheritance is one of the many principles in object-oriented programming.

Polymorphism: This is when a task can be done in several different ways.



2.2 Types of Inheritance and Member Access Rules

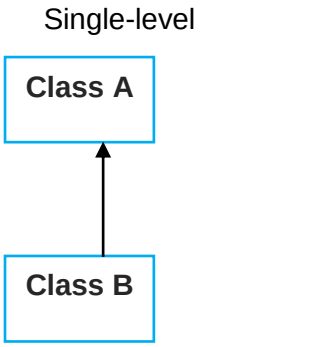
They are three types of inheritance in java:

1. Single-level
2. Multi-level
3. Hierarchical

Single Level Inheritance

When a class extends only one class, then it is called single level inheritance.

```
class A
{
   //code
}
class B extends A
{
   //code
}
```



One class inherits feature from another class

Example: Sample program for single level inheritance

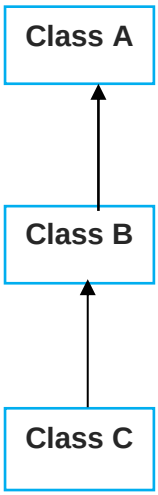
```
class Parent
{
    public void m1()
    {
       System.out.println("Class Parent method");
    }
}
public class Child extends Parent
{
   public void m2()
   {
      System.out.println("Class Child method");
   }
   public static void main(String args[])
   {
      Child obj = new Child();
      obj.m1();
      obj.m2();
   }
}
```

Output:

- Class Parent method
- Class Child method

Multi-level

Here there is a chain of inheritance one class inherits from another and the chain goes on.



Syntax:

```
class A
{
   //code
}
class B extends A
{
   //code
}
class C extends B
{
   //code
}
```

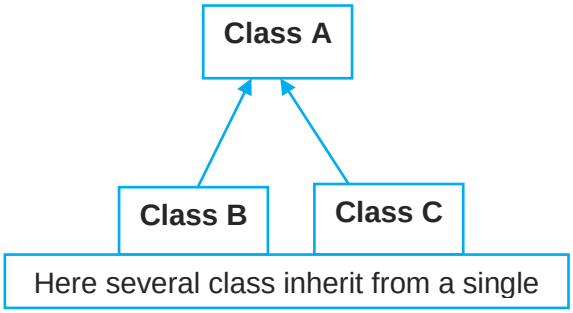
Example: Sample program for multilevel inheritance

```
class Grand
{
   public void m1()
   {
    System.out.println("Class Grand method");
   }
}
class Parent extends Grand
{
   public void m2()
   {
      System.out.println("Class Parent method");
    }
}
class Child extends Parent
{
   public void m3()
   {
    System.out.println("Class Child method");
   }
   public static void main(String args[])
   {
    Child obj = new Child();
    obj.m1();
    obj.m2();
    obj.m3();
   }
}
```

Output:

- Class Grand method
- Class Parent method
- Class Child method

Hierarchical



Syntax:

```
class A
{
   // code
}
class B extends A
{
   // code
}
class C extends A
{
   //code
}
```

Example: Sample program for hierarchical inheritance

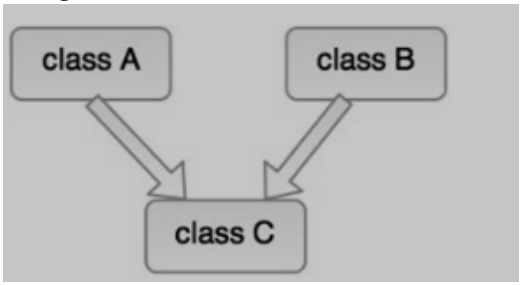
```
class A
{
  public void m1()
  {
    System.out.println("method of Class A");
  }
}
class B extends A
{
  public void m2()
 {
    System.out.println("method of Class B");
  }
}
class C extends A
{
  public void m3()
  {
    System.out.println("method of Class C");
  }
}
class D extends A
{
  public void m4()
  {
    System.out.println("method of Class D");
  }
}
public class MainClass
{
  public void m5()
  {
    System.out.println("method of Class MainClass");
  }
  public static void main(String args[])
  {
    A obj1 = new A();
    B obj2 = new B();
    C obj3 = new C();
    D obj4 = new D();
    obj1.m1();
    obj2.m1();
    obj3.m1();
    obj4.m1();
  }
}
```

Output:

method of Class A
method of Class A
method of Class A
method of Class A

Multiple Inheritances

- In multiple inheritance, one derive class can extend more than one base class.
- Multiple inheritances are basically not supported by Java, because it increases the complexity of the code.
- But they can be achieved by using interfaces.



Example: Sample program for multiple inheritance

```
class X
{
  void display()
  {
     System.out.println("class X dispaly method ");
  }
}
class Y
{
  void display()
  {
     System.out.println("class Y display method ");
  }
}
public class Z extends X,Y
{
   public static void main(String args[])
   {
      Z obj=new Z();
      obj.display();
  }
}
```

Output:

Compile time error

STL academy

Access Modifiers

Access modifiers are simply a keyword in Java that provides accessibility of a class and its member. They set the access level to methods, variable, classes and constructors.

Types of access modifier

There are 4 types of access modifiers available in Java.

- public
- default
- protected
- private

Public

The member with public modifiers can be accessed by any classes. The public methods, variables or class have the widest scope.

Example: Sample program for public access modifier

```
public static void main(String args[])
{
   // code
}
```

Default

When we do not mention any access modifier, it is treated as default. It is accessible only within same package.

Example: Sample program for default access modifier

```
int a = 25;
String str = "Java";
boolean m1()
{
   return true;
}
```

Protected

- The protected modifier is used within same package. It lies between public and default access modifier.
- It can be accessed outside the package but through inheritance only.
- A class cannot be protected.

Example: Sample program for protected access modifier

```
class Employee
{
   protected int id = 101;
   protected String name = "Jack";
}
public class ProtectedDemo extends Employee
{
   private String dept = "Networking";
   public void display()
   {
      System.out.println("Employee Id : "+id);
      System.out.println("Employee name : "+name);
      System.out.println("Employee Department : "+dept);
   }
   public static void main(String args[])
   {
      ProtectedDemo pd = new ProtectedDemo();
      pd.display();
   }
}
```

Output:

Employee Id : 101
Employee name : Jack
Employee Department : Networking

Private

- The private methods, variables and constructor are not accessible to any other class.
- It is the most restrictive access modifier.
- A class except a nested class cannot be private.

Example: Sample program for private access modifier

```
public class PrivateDemo
{
   private int a = 101;
   private String s = "TutorialRide";
   public void show()
   {
      System.out.println("Private int a = "+a+"\nString s = "+s);
   }
   public static void main(String args[])
   {
      PrivateDemo pd = new PrivateDemo();
      pd.show();
      System.out.println(pd.a+" "+pd.s);
   }
}
```

Output:

Private int a = 101
String s = TutorialRide
101 TutorialRide

Java Inheritance (Subclass and Superclass)

- In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:
- subclass (child) - the class that inherits from another class
- superclass (parent) - the class being inherited from

To inherit from a class, use the extends keyword.
In the example below, the Car class (subclass) inherits the attributes and methods from the Vehicle class (superclass):

Example

Example

```
class Vehicle {
  protected String brand = "Ford";        // Vehicle attribute
  public void honk() {                    // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}

class Car extends Vehicle {
  private String modelName = "Mustang";    // Car attribute
  public static void main(String[] args) {

    // Create a myCar object
    Car myCar = new Car();

    // Call the honk() method (from the Vehicle class) on the myCar object
    myCar.honk();

    // Display the value of the brand attribute (from the Vehicle class) and the value of the modelName from the Car class
    System.out.println(myCar.brand + " " + myCar.modelName);
  }
}
```

Did you notice the `protected` modifier in Vehicle?

We set the **brand** attribute in **Vehicle** to a `protected` access modifier. If it was set to `private`, the Car class would not be able to access it.

Why And When To Use "Inheritance"?

- It is useful for code reusability: reuse attributes and methods of an existing class when you create a new class.

Tip: Also take a look at the next chapter, Polymorphism, which uses inherited methods to perform different tasks.

The Final Keyword

If you don't want other classes to inherit from a class, use the final keyword:

If you try to access a `final` class, Java will generate an error:

```
final class Vehicle {
  ...
}

class Car extends Vehicle {
  ...
}
```

The output will be something like this:

```
Main.java:9: error: cannot inherit from final Vehicle
class Main extends Vehicle {
                   ^
1 error)
```

Try it Yourself »

STL academy

Java Polymorphism

- Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.
- Like we specified in the previous chapter; Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.
- For example, think of a superclass called Animal that has a method called animalSound(). Subclasses of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

Example

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}
```

Now we can create Pig and Dog objects and call the animalSound() method on both of them:

Example

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}

class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();  // Create a Animal object
    Animal myPig = new Pig();  // Create a Pig object
    Animal myDog = new Dog();  // Create a Dog object
    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }
}
```

Try it Yourself »

Java Inner Classes

- In Java, it is also possible to nest classes (a class within a class).
- The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.
- To access the inner class, create an object of the outer class, and then create an object of the inner class:

Example

```
class OuterClass {
  int x = 10;

  class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}

// Outputs 15 (5 + 10)
```

Try it Yourself »

Private Inner Class

Unlike a "regular" class, an inner class can be private or protected. If you don't want outside objects to access the inner class, declare the class as private:

Example

```
class OuterClass {
  int x = 10;

  private class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}
```

If you try to access a private inner class from an outside class, an error occurs:

```
Main.java:13: error: OuterClass.InnerClass has private access in OuterClass
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
              ^
```

Try it Yourself »

Static Inner Class

An inner class can also be static, which means that you can access it without creating an object of the outer class:

Example

```
class OuterClass {
  int x = 10;

  static class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass.InnerClass myInner = new OuterClass.InnerClass();
    System.out.println(myInner.y);
  }
}

// Outputs 5
```

Try it Yourself »

Access Outer Class From Inner Class

One advantage of inner classes, is that they can access attributes and methods of the outer class:

Example

```
class OuterClass {
  int x = 10;

  class InnerClass {
    public int myInnerMethod() {
      return x;
    }
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.myInnerMethod());
  }
}

// Outputs 10
```

Try it Yourself »

Java Abstraction

Abstract Classes and Methods

- Data abstraction is the process of hiding certain details and showing only essential information to the user.
- Abstraction can be achieved with either abstract classes or interfaces (which you will learn more about in the next chapter).
- The abstract keyword is a non-access modifier, used for classes and methods:
  - **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
  - **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).
- An abstract class can have both abstract and regular methods:

```
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

From the example above, it is not possible to create an object of the Animal class:

```
Animal myObj = new Animal(); // will generate an error
```

To access the abstract class, it must be inherited from another class. Let's convert the Animal class we used in the Polymorphism chapter to an abstract class:

Example

```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

STL academy

Interfaces

Another way to achieve abstraction in Java, is with interfaces.
An interface is a completely "abstract class" that is used to group related methods with empty bodies:

Example

```
// interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void run(); // interface method (does not have a body)
}
```

To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class:

Example

```
// Interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Try it Yourself »

STL academy

Notes on Interfaces:

- Like **abstract classes**, interfaces **cannot** be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default `abstract` and `public`
- Interface attributes are by default `public`, `static` and `final`
- An interface cannot contain a constructor (as it cannot be used to create objects)

Why And When To Use Interfaces?

1) To achieve security - hide certain details and only show the important details of an object (interface).

2) Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can **implement** multiple interfaces. **Note:** To implement multiple interfaces, separate them with a comma (see example below).

Multiple Interfaces

To implement multiple interfaces, separate them with a comma.

Example

```
// Interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Try it Yourself »

STL academy

Java Enums

- An enum is a special "class" that represents a group of **constants** (unchangeable variables, like final variables).
- To create an enum, use the enum keyword (instead of class or interface), and separate the constants with a comma. Note that they should be in uppercase letters:

Example

```
enum Level {
  LOW,
  MEDIUM,
  HIGH
}
```

You can access enum constants with the **dot** syntax:
Level myVar = Level.MEDIUM;

Enum is short for "enumerations", which means "specifically listed".

Enum inside a Class

You can also have an enum inside a class:

Example

```
public class Main {
  enum Level {
    LOW,
    MEDIUM,
    HIGH
  }

  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;
    System.out.println(myVar);
  }
}
```

The output will be:

```
MEDIUM
```

Try it Yourself »

Enum in a Switch Statement

Enums are often used in switch statements to check for corresponding values.

Example

```
enum Level {
  LOW,
  MEDIUM,
  HIGH
}

public class Main {
  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;

    switch(myVar) {
      case LOW:
        System.out.println("Low level");
        break;
      case MEDIUM:
         System.out.println("Medium level");
        break;
      case HIGH:
        System.out.println("High level");
        break;
    }
  }
}
```

The output will be:

```
Medium level
```

Try it Yourself »

Loop Through an Enum

The enum type has a values() method, which returns an array of all enum constants. This method is useful when you want to loop through the constants of an enum:

Example

```
for (Level myVar : Level.values()) {
  System.out.println(myVar);
}
```

The output will be:

```
LOW
MEDIUM
HIGH
```

Try it Yourself »

Difference between Enums and Classes

An `enum` can, just like a `class`, have attributes and methods. The only difference is that enum constants are `public`, `static` and `final` (unchangeable - cannot be overridden).

An `enum` cannot be used to create objects, and it cannot extend other classes (but it can implement interfaces).

Why And When To Use Enums?

Use enums when you have values that you know aren't going to change, like month days, days, colors, deck of cards, etc.

Java User Input

- The Scanner class is used to get user input, and it is found in the java.util package.
- To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation.
- In our example, we will use the nextLine() method, which is used to read Strings:

Example

```
import java.util.Scanner;  // Import the Scanner class

class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username");

    String userName = myObj.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user input
  }
}
```

Run Example »

If you don't know what a package is, read our Java Packages Tutorial.

Input Types

In the example above, we used the `nextLine()` method, which is used to read Strings. To read other types, look at the table below:

Method	Description
`nextBoolean()`	Reads a `boolean` value from the user
`nextByte()`	Reads a `byte` value from the user
`nextDouble()`	Reads a `double` value from the user
`nextFloat()`	Reads a `float` value from the user
`nextInt()`	Reads a `int` value from the user
`nextLine()`	Reads a `String` value from the user
`nextLong()`	Reads a `long` value from the user
`nextShort()`	Reads a `short` value from the user

In the example below, we use different methods to read data of various types:

Example

```
import java.util.Scanner;

class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);

    System.out.println("Enter name, age and salary:");

    // String input
    String name = myObj.nextLine();

    // Numerical input
    int age = myObj.nextInt();
    double salary = myObj.nextDouble();

    // Output input by user
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Salary: " + salary);
  }
}
```

Run Example »

Note: If you enter wrong input (e.g. text in a numerical input), you will get an exception/error message (like "InputMismatchException").

You can read more about exceptions and how to handle errors in the Exceptions chapter.

Java Date and Time

Java does not have a built-in Date class, but we can import the java. Time package to work with the date and time API. The package includes many date and time classes. For example:

Class	Description
LocalDate	Represents a date (year, month, day (yyyy-MM-dd))
LocalTime	Represents a time (hour, minute, second and nanoseconds (HH-mm-ss-ns))
LocalDateTime	Represents both a date and a time (yyyy-MM-dd-HH-mm-ss-ns)
DateTimeFormatter	Formatter for displaying and parsing date-time objects

If you don't know what a package is, read our Java Packages Tutorial.

Display Current Date

To display the current date, import the java.time.LocalDate class, and use its now() method:

Example

```
import java.time.LocalDate; // import the LocalDate class

public class Main {
  public static void main(String[] args) {
    LocalDate myObj = LocalDate.now(); // Create a date object
    System.out.println(myObj); // Display the current date
  }
}
```

The output will be:

```
2022-07-15
```

Try it Yourself »

Display Current Time

To display the current time (hour, minute, second, and nanoseconds), import the java.time.LocalTime class, and use its now() method:

Example

```
import java.time.LocalTime; // import the LocalTime class

public class Main {
  public static void main(String[] args) {
    LocalTime myObj = LocalTime.now();
    System.out.println(myObj);
  }
}
```

The output will be:

```
14:32:33.275933
```

Try it Yourself »

Display Current Date and Time

To display the current date and time, import the java.time.LocalDateTime class, and use its now() method:

Example

```
import java.time.LocalDateTime; // import the LocalDateTime class

public class Main {
  public static void main(String[] args) {
    LocalDateTime myObj = LocalDateTime.now();
    System.out.println(myObj);
  }
}
```

The output will be:

```
2022-07-15T14:32:33.275657
```

Try it Yourself »

Formatting Date and Time

The "T" in the example above is used to separate the date from the time. You can use the DateTimeFormatter class with the ofPattern() method in the same package to format or parse date-time objects. The following example will remove both the "T" and nanoseconds from the date-time:

Example

```
import java.time.LocalDateTime; // Import the LocalDateTime class
import java.time.format.DateTimeFormatter; // Import the DateTimeFormatter class

public class Main {
  public static void main(String[] args) {
    LocalDateTime myDateObj = LocalDateTime.now();
    System.out.println("Before formatting: " + myDateObj);
    DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");

    String formattedDate = myDateObj.format(myFormatObj);
    System.out.println("After formatting: " + formattedDate);
  }
}
```

The output will be:

```
Before Formatting: 2022-07-15T14:32:33.275826
After Formatting: 15-07-2022 14:32:33
```

Try it Yourself »

The ofPattern() method accepts all sorts of values, if you want to display the date and time in a different format. For example:

Value	Example	Tryit
yyyy-MM-dd	"1988-09-29"	Try it »
dd/MM/yyyy	"29/09/1988"	Try it »
dd-MMM-yyyy	"29-Sep-1988"	Try it »
E, MMM dd yyyy	"Thu, Sep 29 1988"	Try it »

- The ArrayList class is a resizable array, which can be found in the java.util package.
- The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one).
- While elements can be added and removed from an ArrayList whenever you want. The syntax is also slightly different:

Example

Create an `ArrayList` object called **cars** that will store strings:

```
import java.util.ArrayList; // import the ArrayList class

ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object
```

If you don't know what a package is, read our Java Packages Tutorial.

Add Items

The ArrayList class has many useful methods. For example, to add elements to the ArrayList,use the add() method:

Example

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
  }
}
```

Try it Yourself »

Access an Item

To access an element in the ArrayList, use the get() method and refer to the index number:

Example

```
cars.get(0);
```

Try it Yourself »

Remember: Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

Change an Item

To modify an element, use the set() method and refer to the index number.

Example

```
cars.set(0, "Opel");
```

Try it Yourself »

Remove an Item

To remove an element, use the remove() method and refer to the index number:

Example

```
cars.remove(0);
```

Try it Yourself »

To remove all the elements in the ArrayList, use the clear() method:

Example

```
cars.clear();
```

Try it Yourself »

ArrayList Size

To find out how many elements an ArrayList have, use the size method

Example

```
cars.size();
```

Try it Yourself »

Loop Through an ArrayList

Loop through the elements of an ArrayList with a for loop, and use the size() method to specify how many times the loop should run:

Example

```
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    for (int i = 0; i < cars.size(); i++) {
      System.out.println(cars.get(i));
    }
  }
}
```

Try it Yourself »

You can also loop through an ArrayList with the **for-each** loop:

Example

```
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

Try it Yourself »

Other Types

- Elements in an ArrayList are actually objects. In the examples above, we created elements (objects) of type "String".
- Remember that a String in Java is an object (not a primitive type).
- To use other types, such as int, you must specify an equivalent wrapper class: Integer.
- For other primitive types, use: Boolean for boolean, Character for char, Double for double, etc:

Example

Create an `ArrayList` to store numbers (add elements of type `Integer`):

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(10);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(25);
    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```

Try it Yourself »

STL academy

Sort an ArrayList

Another useful class in the java.util package is the Collections class, which include the sort() method for sorting lists alphabetically or numerically:

Example

Sort an ArrayList of Strings:

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    Collections.sort(cars);  // Sort cars
    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

Try it Yourself »

Example

Sort an ArrayList of Integers:

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(33);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(34);
    myNumbers.add(8);
    myNumbers.add(12);

    Collections.sort(myNumbers);  // Sort myNumbers

    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```

Try it Yourself »

The LinkedList class is almost identical to the ArrayList:

Example

```
// Import the LinkedList class
import java.util.LinkedList;

public class Main {
  public static void main(String[] args) {
    LinkedList<String> cars = new LinkedList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
  }
}
```

Try it Yourself »

ArrayList vs. LinkedList

- The LinkedList class is a collection which can contain many objects of the same type, just like the ArrayList.
- The LinkedList class has all of the same methods as the ArrayList class because they both implement the List interface.
- This means that you can add items, change items, remove items and clear the list in the same way.
- However, while the ArrayList class and the LinkedList class can be used in the same way, they are built very differently.

How the ArrayList works

- The ArrayList class has a regular array inside it. When an element is added, it is placed into the array. If the array is not big enough, a new, larger array is created to replace the old one and the old one is removed.

When To Use

- Use an ArrayList for storing and accessing data, and LinkedList to manipulate data.
- For many cases, the ArrayList is more efficient as it is common to need access to random items in the list, but the LinkedList provides several methods to do certain operations more efficiently:

Method	Description	Try it
addFirst()	Adds an item to the beginning of the list.	Try it »
addLast()	Add an item to the end of the list	Try it »
removeFirst()	Remove an item from the beginning of the list.	Try it »
removeLast()	Remove an item from the end of the list	Try it »
getFirst()	Get the item at the beginning of the list	Try it »
getLast()	Get the item at the end of the list	Try it »

- In the ArrayList chapter, you learned that Arrays store items as an ordered collection, and you have to access them with an index number (int type).
- A HashMap however, store items in "key/value" pairs, and you can access them by an index of another type (e.g. a String).
- One object is used as a key (index) to another object (value). It can store different types: String keys and Integer values, or the same type, like: String keys and String values:

Example

Create a `HashMap` object called **capitalCities** that will store `String` **keys** and `String` **values**:

```
import java.util.HashMap; // import the HashMap class

HashMap<String, String> capitalCities = new HashMap<String, String>();
```

Add Items

- The HashMap class has many useful methods. For example, to add items to it, use the put() method:

Example

```
// Import the HashMap class
import java.util.HashMap;

public class Main {
  public static void main(String[] args) {
    // Create a HashMap object called capitalCities
    HashMap<String, String> capitalCities = new HashMap<String, String>();

    // Add keys and values (Country, City)
    capitalCities.put("England", "London");
    capitalCities.put("Germany", "Berlin");
    capitalCities.put("Norway", "Oslo");
    capitalCities.put("USA", "Washington DC");
    System.out.println(capitalCities);
  }
}
```

Try it Yourself »

Access an Item

To access a value in the HashMap, use the get() method and refer to its key:

Remove an Item

To remove an item, use the remove() method and refer to the key:

Example

```
capitalCities.remove("England");
```

Try it Yourself »

To remove all items, use the `clear()` method:

Example

```
capitalCities.clear();
```

Try it Yourself »

HashMap Size

To find out how many items there are, use the size() method:

Example

```
capitalCities.size();
```

Try it Yourself »

Loop Through a HashMap

Loop through the items of a HashMap with a **for-each** loop.

Note: Use the keySet() method if you only want the keys, and use the values() method if you only want the values:

Example

```
// Print keys
for (String i : capitalCities.keySet()) {
  System.out.println(i);
}
```

Try it Yourself »

Example

```
// Print values
for (String i : capitalCities.values()) {
  System.out.println(i);
}
```

Try it Yourself »

Example

```
// Print keys and values
for (String i : capitalCities.keySet()) {
  System.out.println("key: " + i + " value: " + capitalCities.get(i));
}
```

Try it Yourself »

- Superclass members can be inherited to a subclass provided there is eligibility of access modifiers.
- Here are the behaviors of access specifiers in java:

1. Private members of a superclass cannot be inherited to a subclass because they are not available directly.
2. Public members can be inherited to all subclass
3. Protected members can be inherited to a subclass but their usage is limited in a package.
4. Default members can be inherited in the same package.

2.3 Method Overloading as well as Method Overriding

Method overloading

Two or more methods have the same name but different type of parameters, different parameters name, different number of parameters and this is called method overloading.

Example:

```
void add () {…..}
void add (int a) {…..}
float add (int a, float b) {…..}
```

Method Overriding

If a child class has the same method that is declared in the parent class, we refer this as method overriding.

2.4 Usage of 'This' and 'Super' Key Word

'This' keyword

- It is commonly used to eliminate confusion between class attributes and parameters with the same name.
- It is used to refer a current object in a method constructor.
- It can also be used to:
  1. Invoke current class method
  2. Pass argument in the constructor call
  3. Pass argument in the method call
  4. Invoke current class constructor
  5. Return the current class object

Java Packages

- A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:
  - Packages (create your own packages)
  - Built-in Packages (packages from the Java API)
  - User-defined

Built-In Packages

- The library is divided into packages and classes.
- You can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package.
- To use a class or a package from the library, you need to use the import keyword:

Import A Class

- If you find a class you want to use, for example, the Scanner class, which is used to get user input, write the following code:

Example

```
import java.util.Scanner;
```

- In the example above, java.util is a package, while Scanner is a class of the java.util package.
- To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read a complete line:

2.5 Packages and Interfaces: Defining package, Access protection, importing packages

Packages

- It's a group of classes and interfaces that are together.
- They can be imported.
- Can be created using the keyword package.
- Done using the import keyword

Example:

```
package package_name;
public class class_name
{
(body of class)
}
```

Interfaces

- It is implemented using "implement" keyword.
- Created using "interface" keyword.
- Can be extended by another interface
- It can also be implemented by a class

Example:

```
Interface interface_name {
variable declaration;
method declaration;
}
```

2.6 I / O Streams: Concepts of streams, Stream classes- Byte and Character stream

I/O Streams

A stream is a sequence of data generated input source and consumed by the output destination.

Byte-Stream

- Java byte are used to perform input/output of 8-bit bytes.
- Commonly used classes are File Input Stream and File Output Stream

Example:

```
Import java.io.*;
Public class copyfile {
Public static void main(string args[] throws IOException) {
FileInputStream in  =  null;
FileOutputStream out  =  null;
try {
```

CHARACTER STREAMS

```
In = new FileInputStream(“input.txt”);
out = new FileOutputStream(“output.txt”);
Int c;
While (( c = in.read()) !=-1) {
   out.write(c);
}
} finally {
if  ( in != null) {
In.close();
}
if (out ! =null) {
Out.close();
}
}
}
}
```

Character Byte Stream

- Character stream are used to perform input and output for 16-bit Unicode
- Despite of the many classes used inn character streams the commonly used one are FileReader and FileWriter.
- The main difference between FileReader and FileWriter is that FileReader reads two bytes at a time while FileWriter writes two bytes at a time.

Example:

```
Import java.io.*;
Public class CopyFile {
Public static void main(string args[]) throws IOException {
FileReader in = null;
FileWriter out = null;
Try {
in = new FileReader("input.txt");
out = new FileWriter("output.txt");
int c;
while (( c = in.read()) ! = -1) {
Out.write(c);
}
} finally {
if (n ! = null) {
in.close();
}
If ( n ! = null) {
Out.close();
}
}
}
```

2.7 Reading Console Input and Writing Console Output

Reading Console Input

- We use console class to read from console by default. The console class provides method to access character based, console associated with the current java process.
- The console has three ways to read input:
  - String readline()
  - Char [] readpassword()
  - Reader reader()

Writing console output

- We can easily write the output data to console using System.out.println() statement.
- We can also use printf() method to write formatted text to console

2.8 File Handling

- We have a file class that is found inside the java.io package. With the help of this class, we can easily work with files in java.
- The file class can be used by creating an object of the class and then specifying the name of the file.

Reasons for file handling

- It enables reading and writing data to a file.
- File handling is an integral part of any programming language because it allows storing the output of any particular program in a file and allows us to perform certain operation on it.

Importing File Class: Import java.io.File
Class GFG {
Public static void main(string[] args)
{
//File name specified
File obj = new File (“myfile.txt”);
System.out.println(“File Created!”);
}
}
Output
File Created!

Section 3: Exercises

Exercise 1: Tinku has written the code like below. But it is showing compile time error. Can you identify what mistake he has done?

```
1  class X
2  {
3      //Class X Members
4  }
5
6  class Y
7  {
8      //Class Y Members
9  }
10
11 class Z extends X, Y
12 {
13     //Class Z Members
14 }
```

Exercise 2: What will be the output of this program?

```
class A
{
    int i = 10;
}

class B extends A
{
    int i = 20;
}

public class MainClass
{
    public static void main(String[] args)
    {
        A a = new B();

        System.out.println(a.i);
    }
}
```

Exercise 3: What will be the output of the below program?

```
class A
{
    {
        System.out.println(1);
    }
}

class B extends A
{
    {
        System.out.println(2);
    }
}

class C extends B
{
    {
        System.out.println(3);
    }
}

public class MainClass
{
    public static void main(String[] args)
    {
        C c = new C();
    }
}
```

Exercise 4: What will be the output of the following program?

```
class A
{
    String s = "Class A";
}

class B extends A
{
    String s = "Class B";

    {
        System.out.println(super.s);
    }
}

class C extends B
{
    String s = "Class C";

    {
        System.out.println(super.s);
    }
}

public class MainClass
{
    public static void main(String[] args)
    {
        C c = new C();

        System.out.println(c.s);
    }
}
```

Exercise 5: What will be the output of this program?

```
class A
{
    static
    {
        System.out.println("THIRD");
    }
}

class B extends A
{
    static
    {
        System.out.println("SECOND");
    }
}

class C extends B
{
    static
    {
        System.out.println("FIRST");
    }
}

public class MainClass
{
    public static void main(String[] args)
    {
        C c = new C();
    }
}
```

STL academy

Section 4: Assessment Questionnaire

1. Polymorphism is supported by the c++ by using following ways:
   a) Function Overloading
   b) Operator Overloading
   c) Virtual Functions
   d) All of the Above

2. Compile time polymorphism is supported by:
   a) Function Overloading
   b) Virtual Function
   c) Operator Overloading
   d) Both A&C

3. Run time polymorphism is supported by:
   a) Function Overloading
   b) Operator Overloading
   c) Virtual Function
   d) Both A And B

4. Selecting the appropriate overloaded function by the compiler is known as:
   a) Late Binding
   b) Early Binding
   c) Both A And B
   d) None of the Above

5. Object to function binding is done at compile time then is it known as:
   a) Early Binding
   b) Compile Time Binding
   c) None of the Above
   d) Both A and B

6. Run time polymorphism is done by using:
   a) Function Overloading
   b) Operator Overloading
   c) Virtual Function
   d) None of the Above

7. Operator overloading is:
   a) Run Time Polymorphism
   b) Compile Time Polymorphism
   c) None of the Above
   d) Both A And B

8. Which of the following operator cannot be overloaded?
   a) Scope Resolution Operator(::)
   b) Size of Operator (Sizeof[])
   c) Conditional Operator(?:)
   d) All of the Above
9. Which of the operator cannot be overloaded?
   a) >=
   b) &
   c) <=
   d) ::
10. While performing operator overloading which function/keyword we have to use?
   a) Function
   b) Operator
   c) Op
   d) None of the Above
11. Which of the statement is not true about operator overloading?
   a) We can overload only Existing Operator
   b) Basic Meaning cannot be Changed
   c) Binary Operator should have Return Type
   d) All of the Above
12. Pick up the correct statement related with operator overloading.
   a) We can Overload a Class Access Operator
   b) We can Change the Meaning of Basic Operator
   c) Binary Operator Should Have a Return Type
   d) Both A And B
13. We are overloading a unary operator without friend function how many arguments we have to pass?
   a) 1
   b) 2
   c) 0
   d) None of the above
14. Suppose we are overloading a binary operator with friend function, how many parameter of argument we have to pass?
   a) 1
   b) 2
   c) 3
   d) None of the above
15. We are overloading a binary operator without friend function how many arguments we have to pass?
   a) 1
   b) 2
   c) 0
   d) None of the above

----------End of the Module----------

MODULE 3
EXCEPTION HANDLING

Section 1: Learning Outcomes

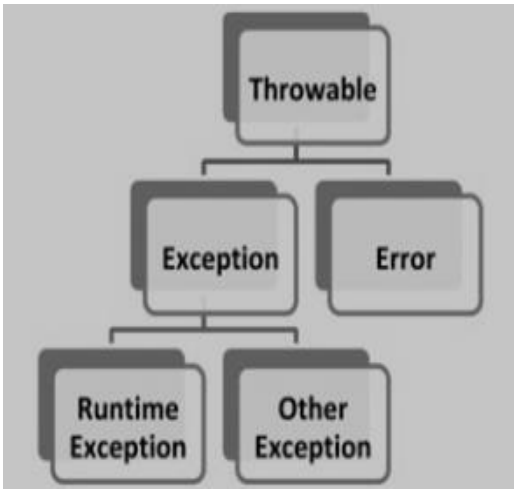
After completing this module, you will be able to:

- Explain Exception Type
- Explain Usage of Try and Catch, Throw, Throws and Finally keywords
- Describe Multi-Threading: Concepts of Thread, Thread life cycle
- Create threads using Thread class and Runnable interface
- Explain Synchronization
- Explain Inter Thread communication
- Classify Thread Priorities

Section 2: Relevant Knowledge

3.1 Exception Types

- Java is an object-oriented programming language. It provides support for various mechanisms such as exception handling.
- This feature of Java enables developers to manage the runtime errors caused by the exceptions.
- Exceptions are the unwanted errors or bugs or events that restrict the normal execution of a program.
- Each time an exception occurs, program execution gets disrupted. An error message is displayed on the screen.
- There are several reasons behind the occurrence of exceptions. These are some conditions where an exception occurs:
  - Whenever a user provides invalid data.
  - The file requested to be accessed does not exist in the system.
  - When the Java Virtual Machine (JVM) runs out of memory.
  - Network drops in the middle of communication.
- Now let us explore different types of exceptions in Java.
- The parent class of all the exception classes is the java.lang.Exception class. Figure illustrates the different types of Java exceptions.



- **There are some important methods available in the Throwable class which are as follows:**
  1. public String getMessage() – Provides information about the exception that has occurred through a message, which is initialized in the *Throwable constructor*.
  2. public Throwable getCause() – Provides root cause of the exception as represented by a *Throwable object*.
  3. public void printStackTrace() – Used to display the output of *toString()* along with the stack trace to *System.err* (error output stream).
  4. public StackTraceElement [] getStackTrace() – Returns an array with each element present on the stack trace. The index 0 element will symbolize the top of the call stack, and the last element of array will identify the bottom of the call stack
- There are mainly two types of exceptions in Java as follows:
  - Checked exception
  - Unchecked exception

Checked Exception

- Checked exceptions are also known as compile-time exceptions as these exceptions are checked by the compiler during the compilation process to confirm whether the exception is handled by the programmer or not. If not, then the system displays a compilation error. For example, SQLException, IOException, InvocationTargetException, and ClassNotFoundException.
- To illustrate the concept of checked exception, let us consider the following code snippet:

```
import java.io.*;
class demo1 {
    public static void main(String args[]) {
        FileInputStream input1 = null;

        /* FileInputStream(File filename) is a constructor that will throw
         *     FileNotFoundException (a checked exception)
         */

        input1 = new FileInputStream("D:/file.txt");
        int m;

        // The read() of FileInputStream will also throw a checked exception
        while(( m = input1.read() ) != -1) {
            System.out.print((char)m);
        }

        // The close() will close the file input stream, and it will also throw a exceptio
n
        input1.close();
    }
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
Unhandled exception type FileNotFoundException
Unhandled exception type IOException
Unhandled exception type IOException
```

Throw Keyword

It is clearly displayed in the output that the program throws exceptions during the compilation process. There are two methods of resolving such issues. You can declare the exception with the help of the throw keyword.

```
import java.io.*;
class demo1 {
    public static void main(String args[]) throws IOException {
        FileInputStream input1 = null;
        input1 = new FileInputStream("D:/file.txt");

        int m;
        while ((m = input1.read()) != -1) {
            System.out.print((char)m);
        }

        input1.close();
    }
}
```

Output: The file will be displayed on the screen.

Try-catch Block

Apart from the above-mentioned method, there is another way to resolve exceptions. You can manage them with the help of **try-catch blocks**.

```
import java.io.*;
class demo1 {
    public static void main(String args[]) {
        FileInputStream input1 = null;
        try {
            input1 = new FileInputStream("D:/file.txt");
        } catch(FileNotFoundException input2) {
            system.out.println("The file does not " + "exist at the location");
        }

        int m;
        try {
            while((m = input1.read()) != -1) {
                System.out.print((char)m);
            }

            input1.close();
        } catch(IOException input3) {
            system.out.println("I/O error occurred: "+ input3);
        }
    }
}
```

Output: The code will run smoothly and the file will be displayed.

Now, let us learn about other checked exceptions. Some of them are:

SQLException

This type of exception occurs while executing queries on a database related to the SQL syntax. For example, consider the following code snippet:

```
public void setClientInfo(String sname, String svalue) throws SQLClientInfoException {
    try {
        checkClosed();
        ((java.sql.Connection) this.mc).setClientInfo(sname, svalue);
    } catch (SQLException sqlEx) {
        try {
            checkAndFireConnectionError(sqlEx);
        } catch (SQLException sqlEx2) {
            SQLClientInfoException client_Ex = new SQLClientInfoException();
            client_Ex.initCause(sqlEx2);
            throw client_Ex;
        }
    }
}
```

Output: This code will generate a SQLException.

IOException

This type of exception occurs while using file I/O stream operations. For example, consider the following code snippet:

```
import java.io.*;
public class sample_IOException {
    private static String filepath = "D:\User\guest\Desktop\File2.txt";

    public static void main(String[] args) {
        BufferedReader br1 = null;
        String curline;

        try {
            br1 = new BufferedReader(new FileReader(filepath));

            while ((curline = br1.readLine()) != null) {
                System.out.println(curline);
            }
        } catch (IOException e) {
            System.err.println("IOException found :" + e.getMessage());
        } finally {
            try {
                if (br1 != null)
                    br1.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Output: This code will generate an IOException.

ClassNotFoundException

This type of exception is thrown when the JVM is not able to find the required class. It may be due to a command-line error, a classpath issue, or a missing .class file. For example, consider the following code snippet:

```
public class sample_ClassNotFoundException {
    private static final String CLASS_TO_LOAD = "main.java.Utils";

    public static void main(String[] args) {
        try {
            Class loadedClass = Class.forName(CLASS_TO_LOAD);
            System.out.println("Class " + loadedClass + " found!");
        } catch (ClassNotFoundException ex) {
            System.err.println("ClassNotFoundException was found: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

Output: This code will generate a ClassNotFoundException.

InvocationTargetException

This type of exception wraps an exception thrown by an invoked method or a constructor. The thrown exception can be accessed with the help of the getTargetException method. For example, consider the following code snippet:

```
package main.samplejava;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
public class Example {
    @SuppressWarnings("unused")
    private int test_sample(String s1) {
        if (s1.length() == 0)
            throw new IllegalArgumentException("The string should have at least one charac
ter!");
        System.out.println("Inside test_sample: argument's value equals to: "" + s1 +
""");
        return 0;
    }

    public static void main(String... args) {
        try {
            Class<?> c1 = Class.forName("main.samplejava. Example");
            Object t1 = c1.newInstance();
            Method[] declared_Methods = c1.getDeclaredMethods();
            for (Method method : declared_Methods) {
                String methodName = method.getName();
                if (methodName.contains("main"))
                    continue;
```

```
                System.out.format("Invoking %s()%n", methodName);

                try {
                    method.setAccessible(true);
                    Object returnValue = method.invoke(t1, "");
                    System.out.format("%s() returned: %d%n", methodName, returnValue);
                } catch (InvocationTargetException ex) {
                    System.err.println("An InvocationTargetException was caught!");
                    Throwable cause = ex.getCause();
                    System.out.format("Invocation of %s failed because of: %s%n",
                        methodName, cause.getMessage());
                }
            }
        } catch (ClassNotFoundException | InstantiationException | IllegalAccessException
ex) {
            System.err.println("The following exception was thrown:");
            ex.printStackTrace();
        }
    }
}
```

```
Invoking testMethod()
An InvocationTargetException was caught!
Invocation of testMethod failed because of: The string must contain at least one characte
r!
```

Output: This code will generate an Instantiation Exception.

Unchecked Exception

- The unchecked exceptions are those exceptions that occur during the execution of the program. Hence, they are also referred to as Runtime exceptions.
- These exceptions are generally ignored during the compilation process.
- They are not checked while compiling the program. For example, programming bugs like logical errors, and using incorrect APIs.

To illustrate the concept of an unchecked exception, let us consider the following code snippet:

```
import java.util.Scanner;
public class Sample_RunTimeException {
    public static void main(String[] args) {
        // Reading user input
        Scanner input_dev = new Scanner(System.in);
        System.out.print("Enter your age in Numbers: ");
        int age1 = input_dev.nextInt();
        if (age1>20) {
            System.out.println("You can view the page");
        } else {
            System.out.println("You cannot view the page");
        }
    }
}
```

Output 1:

```
Enter your age in Numbers: 21
You can view the page
```

Output 2:

```
Enter your age in Numbers: Twelve
Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor (Unknown Source)
at java.util.Scanner.next (Unknown Source)
at java.util.Scanner.nextInt (Unknown Source)
at java.util.Scanner.nextInt (Unknown Source)
at exceptiondemo.sample_runtimedemo.main(Sample_RunTimeExceptionDemo.java:11)
```

Now, let us learn about other unchecked exceptions.
Some of them are:

NullPointerException

This type of exception occurs when you try to access an object with the help of a reference variable whose current value is null or empty. For example, consider the following code snippet:

```
// Program to demonstrate the NullPointerException
class SampleNullPointer {
    public static void main(String args[]) {
        try {
            String a1 = null; // null value
            System.out.println(a1.charAt(0));
        } catch(NullPointerException e) {
            System.out.println("NullPointerException is found in the program.");
        }
    }
}
```

Output: NullPointerException is found in the program.

ArrayIndexOutofBound

This type of exception occurs when you try to access an array with an invalid index value. The value you are providing is either negative or beyond the length of the array.

For example, consider the following code snippet:

```
// Program to demonstrate the ArrayIndexOutOfBoundException
class sample_ArrayIndexOutOfBound {
    public static void main(String args[]) {
        try {
            int b[] = new int[6];
            b[8] = 2; // we are trying to access 9th element in an array of size 7
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println ("The array index is out of bound");
        }
    }
}
```

Output: The array index is out of bound

IllegalArgumentException

This type of exception occurs whenever an inappropriate or incorrect argument is passed to a method. For example, if a method is defined with non-empty string as parameters. But you are providing null input strings. Then, the IllegalArgumentException is thrown to indicate the user that you cannot pass a null input string to the method.

Consider the following code snippet to demonstrate this type of exception:

```
import java.io.File;
public class Sample_IllegalArgumentException {
    public static String createRelativePath(String par, String f_name) {
        if (par == null)
            throw new IllegalArgumentException("You cannot provide null parent path!");

        if (f_name == null)
            throw new IllegalArgumentException("Please enter the complete filename!");

        return par + File.separator + f_name;
    }

    public static void main(String[] args) {
        // This command will be successfully executed.
        system.out.println(IllegalArgumentExceptionExample.createRelativePath("dir1", "fil
e1"));
        system.out.println();

        // This command will throw an IllegalArgumentException.
        System.out.println(IllegalArgumentExceptionExample.createRelativePath(null, "file
1"));
    }
}

Output: This code will generate an IllegalArgumentException.
```

IllegalStateException

This type of exception occurs when the state of the environment does not match the operation being executed. For example, consider the following code snippet, which demonstrates this type of exception:

```
/**
 * This code will publish the current book.
 * If the book is already published, it will throw an IllegalStateException.
 **/
public void pub() throws IllegalStateException {
    Date pub_at = getPub_at();

    if (pub_at == null) {
        setPub_at(new Date());
        Logging.log(String.format("Published '%s' by %s.", getTitle(), getAuthor()));
    } else {
        throw new IllegalStateException(
        String.format("Cannot publish '%s' by %s (already published on %s).",
            getTitle(), getAuthor(), pub_at));
    }
}
```

Output: This code will generate IllegalStateException.

If a publication date already exists in the system, then it will produce an IllegalStateException that indicates that the book cannot be published again.

NumberFormatException

This type of exception occurs when you pass a string to a method that cannot be converted to a number. For example, consider the following code snippet:

```
// Program to demonstrate the NumberFormatException
class Sample_NumberFormat {
    public static void main(String args[]) {
        try {
            // "Test" is not a number
            int n = Integer.parseInt ("Test") ;
            System.out.println(n);
        } catch(NumberFormatException e) {
            System.out.println("Number format exception");
        }
    }
}
```

Output: This code will generate NumberFormatException.

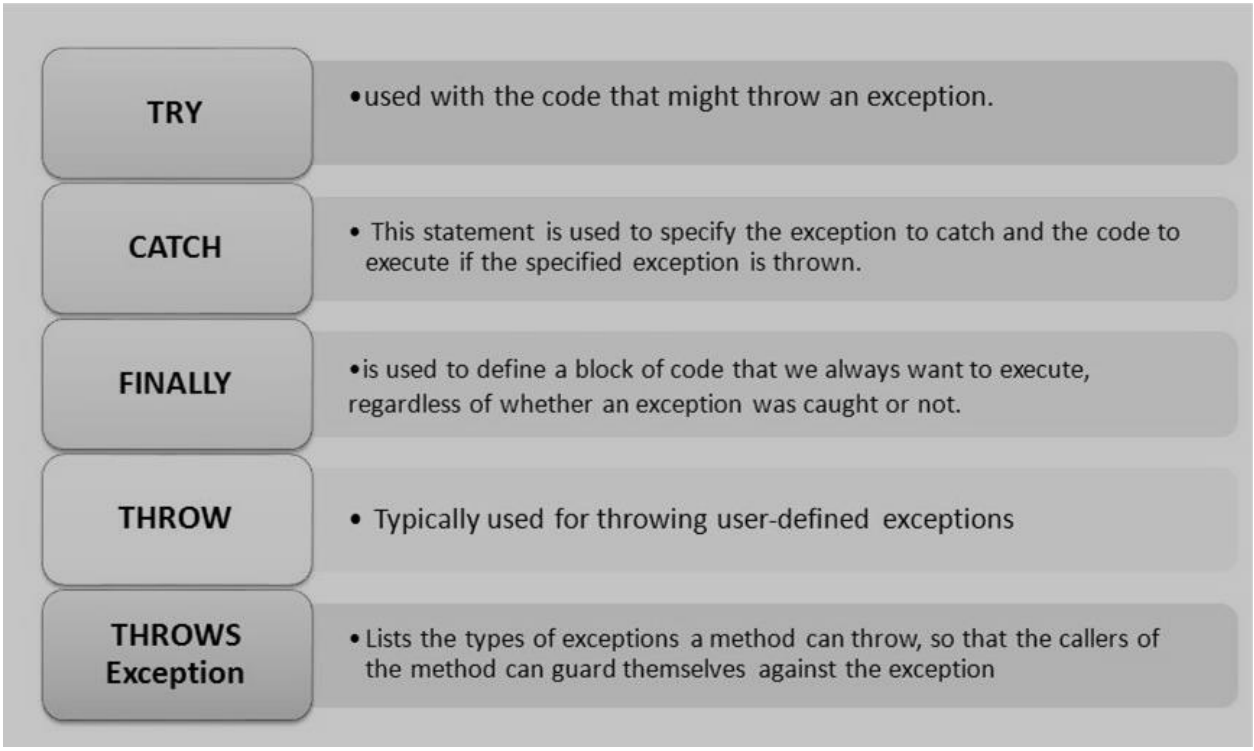
ArithmeticExceaption

This type of exception occurs when you perform an incorrect arithmetic operation. For example, if you divide any number by zero, it will display such an exception. Let us consider the following code snippet:

```
// Program to demonstrate the ArithmeticException
class Sample_ArithmeticException {
    public static void main(String args[]) {
        try {
            int p = 30, q = 0;
            int r = p/q;  // It cannot be divided by zero
            System.out.println ("Result = " + r);
        } catch(ArithmeticException e) {
            System.out.println ("Number cannot be divided by 0");
        }
    }
}
```

Output: This code will generate an ArithmeticException.

3.2 Usage Of Try, Catch, Throw, Throws and Finally Keywords



Try

Species the block of code with the keyword “try” that may give rise to exception.

```
 try
{
//set of statements that can raise exceptions
}
```

Catch

When an exception is raised using the “try” keyword it needs to be caught and we use the “catch” keyword.

```
Catch (Exception e)
{
//code to handle exception e
}
```

Try-Catch

```
Try
{
 //code causing exception
}
Catch (exception (exception-type) e (object)) {
//code to handle exception e
}
```

Throw

Throw keyword is used to throw an exception explicitly.

Throws

Unlike throw keyword it does not throw an exception it shows an exception might occur in a program also it is used to declare an exception.

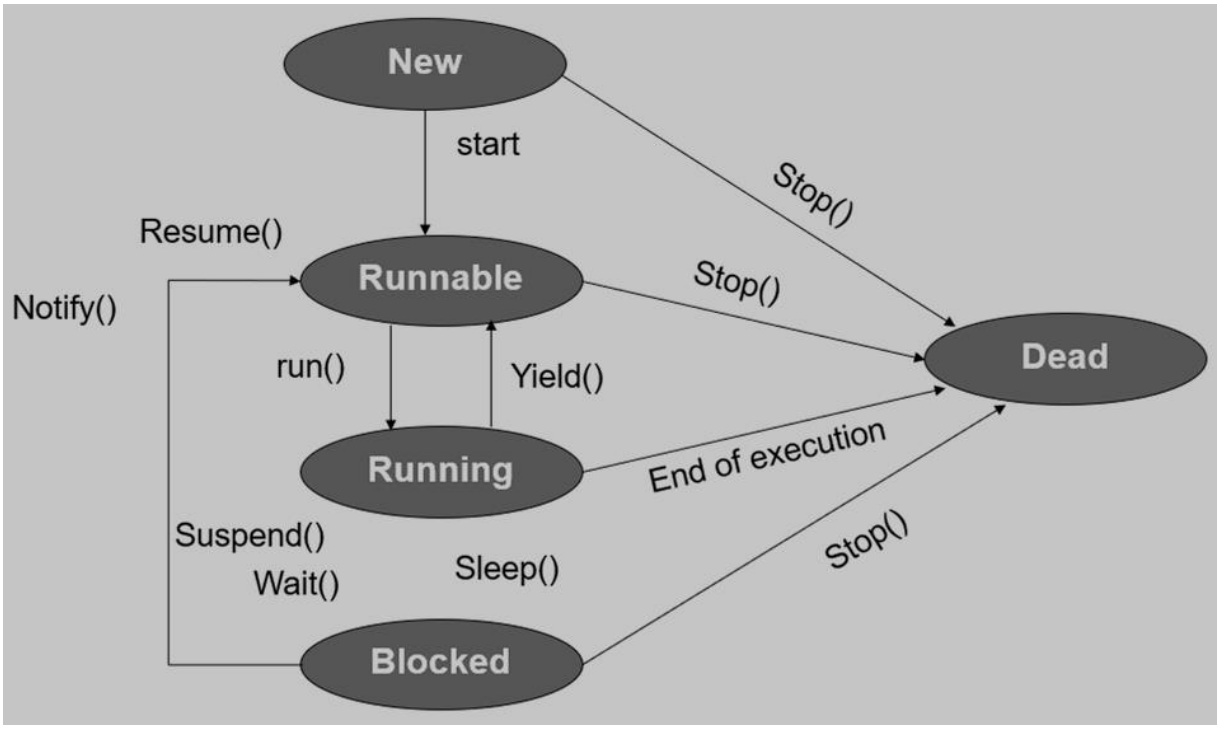
Finally

- We may have an important code in our program that needs to be executed regardless of whether an exception is thrown.
- The code is placed is a special block using the “finally” keyword.
- This block follows the try-catch block.

3.3 Multi-Threading: Concepts of Thread, Thread life cycle

- Multi-threading – is a feature in java that allows execution concurrently of two or more parts of a program.
- Threads are the parts found in such a program that runs concurrently.
- Threads are created using two methods:
  1. Extending Thread Class
  2. Implementing Runnable Interface

Thread life cycle



3.4 Creating Threads using Thread Class and Runnable Interface

Creation using extending thread class

- First we create a class that extends the java.lang.Thread class
- What this class does is it overrides the run() method available in the thread class
- Usually a thread begins its life inside a run() method.
- Then we create an object of our new class then call start() method to start execution of a thread

//program

```
Class ExtendingThread extends Thread
{
String s[] =
{“welcome”, “to”, “java”, “programming”, “Language”};
Public static void main(String args[])
{
Extending Thread t = new
Extending Thread(“Extending Thread Class”);
}
Public ExtendingThread(String n)
{
 super(n);
 start();
}
Public void run()
{
String name = getName();
For(int i=0; i<s.length; i++)
{
 Try
{
Sleep(500);
}
Catch(Exception e)
{
}
System.out.println(name + “:”+s[i];
}
}
}
```

Creation by implementing Runnable Interface

- First a new class is created that implements java.lang.Runnable interface and override run() method.
- Then a thread object is instatiated and a start() method is called on this object.

//program

```
Public class ExampleClass implements Runnable {
@override
Public void run() {
 system.out.println(“Thread has ended”);
}
Public static void main(String[] args) {
ExamplesClass ex = new ExampleClass();
Thread t1 = new Thread(ex);
T1.start();
system.out.println(“Hi”);
}
}
```

Output:

Hi
Thread has ended

Java Thread

- Threads allows a program to operate more efficiently by doing multiple things at the same time.
- Threads can be used to perform complicated tasks in the background without interrupting the main program.

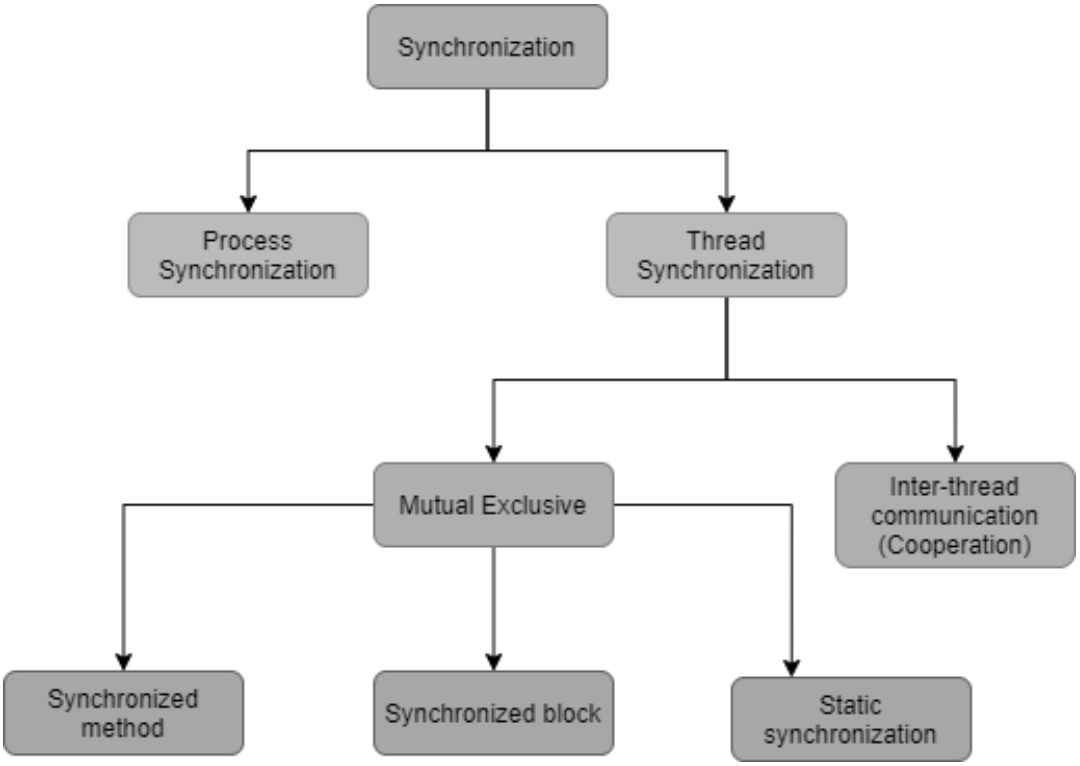
3.5 Synchronization

- Synchronization refers to the ability to control the access of multiple threads to any shared resource.
- Synchronization is of importance when we want to allow only one thread to access shared resource.

Reasons for synchronization

1. Preventing consistency problem.
2. Preventing thread interference.

Types of Synchronization



Thread Synchronization

- There are two types of thread synchronization mutual exclusive and inter-thread communication.
- Mutual Exclusive
  - Synchronized method.
  - Synchronized block.
  - Static synchronization.
- Cooperation (Inter-thread communication in java)

Mutual Exclusive

- Mutual Exclusive helps keep threads from interfering with one another while sharing data. It can be achieved by using the following three ways:
- By Using Synchronized Method
- By Using Synchronized Block
- By Using Static Synchronization

Concept of Lock in Java

- Synchronization is built around an internal entity known as the lock or monitor. Every object has a lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.
- From Java 5 the package java.util.concurrent.locks contains several lock implementations.

3.6 Inter-Thread Communication

- Inter-Thread Communication is the process in which two threads communicate with each other by using wait (), notify (), and notifyAll () methods.
- The Thread which is required updation has to call the wait() method on the required object then immediately the Thread will be entered into a waiting state. So, The Thread which is performing updation of an object is responsible to give notification by calling notify () method.
- After getting notification the waiting thread will get those updation.

wait(), notify() and notifyAll()

- Before going further let's discuss some points about the wait (), notify (), and notifyAll () method.
  1. wait (), notify () and notifyAll () methods are available in Object class but not in Thread class because Thread can call these methods on any common object.
  2. To call wait (), notify () and notifyAll () methods compulsory the current thread should be the owner of that object i.e., the Current thread should have a lock of that object and the current thread should be in the synchronized area.
- Hence, we can call wait(), notify(), and notifyAll() methods only from synchronized area otherwise we will get runtime exception saying IllegalMonitorStateException.
- Once a thread calls the wait() method on the given object 1st it releases the lock of that object immediately and entered into the waiting state.
- Once a Thread calls notify () (or) notifyAll () methods it releases the lock of that object but may not immediately.
- Except for these (wait (), notify (), and notifyAll ()) methods, there is no other place or method where the lock releases will happen.
- One important thing to note is that when a thread calls wait(), notify(), notifyAll() methods on any object then it releases the lock of that particular object but not all the lock it has.
- Also note that on which object we are calling wait(), notify() and notifyAll() methods that corresponding object lock we have to get but not other object locks. Below is the example for this:

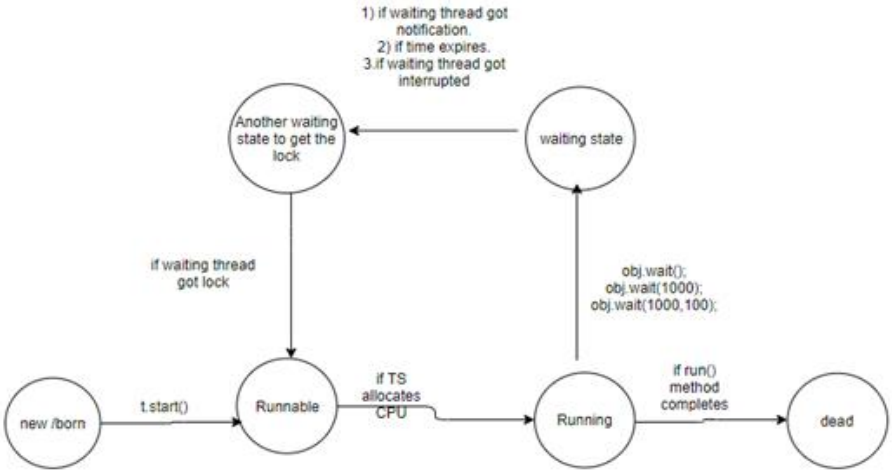
```
Stack s1 = new Stack();
Stack s2 = new Stack();
synchronized(s1)
{
s2.wait(); // invalid one and gives IllegalMonitorException
}
```

```
Stack s1 = new Stack();
Stack s2 = new Stack();
synchronized(s1)
{
s1.wait(); // valid
}
```

Let's understand this in the table:

Method	IS Thread Release Lock
yield()	No
join()	No
sleep()	No
wait()	Yes
notify()	Yes
notifyAll()	Yes

Thread States



Here we see that there is an important role of wait( ) in **inter-thread** communication. So let's clear the difference between the wait( ) and sleep( ) methods so that we can understand the rest of the scenario clearly.

Difference Between wait( ) and sleep( ):

Wait( )	Sleep( )
This method is always called from the synchronized block.	For this method, there is no such requirement for calling.
Whenever this method is called monitor is released. Here monitor means releases the lock.	Whenever this method is called monitor is not released.
This method awakes or becomes active again when notify( ) and notifyAll( ) method is called.	This method does not awake or does not become active when notify( ) or notifyAll( ) method is called.
wait( ) method is generally used based on some condition.	sleep( ) method is generally used when you want to stop the execution of code for a particular time and then want to resume again.
wait( ) is not a static method.	sleep( ) is static method.
wait( ) is used for multi-thread synchronization. So, we can say that in interthread communication wait( ) plays an important role.	sleep( ) is used for controlling the execution time of one thread.

3.7 Thread Priorities

- Every thread has a priority and priorities are represented by a number between 1 and 10.
- A thread scheduler schedules the threads according to their priority this is known as preemptive scheduling.
- The Thread.setPriority() method is used to change the priority of the thread.
- First, the checkAccess() method is called implicitly with no arguments, to check whether this thread has permission to modify the resources or not. If this thread does not have modification access, it results in a SecurityException exception.

Note: checkAccess() is called implicitly by the setPriority() method, not explicitly.

Priorities

There are three properties in the Thread class related to priority:

- **MAX_PRIORITY:** The maximum value is 10, known as the maximum priority of a thread.
- **NORM_PRIORITY:** The normal value is 5, known as the normal priority of a thread.
- **MIN_PRIORITY:** The minimum value is 1, known as the minimum priority of a thread.

  Note: The user-defined priority will be between 1 and 10.

Syntax

```
final void setPriority(int newPriority)
```

Parameters

It takes a single integer value as an argument.

- newPriority: Priority to set this thread.

Return value

The void method does not return any value.

Exceptions

- It throws the following exceptions:
- **IllegalArgumentException:** If the priority is out of range MIN_PRIORITY to MAX_PRIORITY.
- **SecurityException**: If this thread has no access to modify its resources.

Code

```
1  public class setPriorityExample extends Thread  {
2          // run() method to start it's execution
3      public void run()  {
4          // Printing about this thread
5          System.out.println("Priority of "+ Thread.currentThread().getName()
6                  + " is: " + Thread.currentThread().getPriority());
7      }
8      public static void main(String args[]) {
9          // Creating 4 threads
10         setPriorityExample thread1= new setPriorityExample();
11         setPriorityExample thread2= new setPriorityExample();
12         setPriorityExample thread3= new setPriorityExample();
13         setPriorityExample thread4= new setPriorityExample();
14         // Set Priority & name of thread1 Thread
15         thread1.setPriority(Thread.MAX_PRIORITY);
16         thread1.setName("Thread-1");
17         // Set Priority & name of thread2 Thread
18         thread2.setPriority(Thread.MIN_PRIORITY);
19         thread2.setName("Thread-2");
20         // Set Priority & name of thread3 Thread
21         thread3.setPriority(Thread.NORM_PRIORITY);
22         thread3.setName("Thread-3");
23         // Set Priority & name of userdefined Thread thread4
24         thread4.setPriority(6);
25         thread4.setName("Thread-4");
26         // Calling the run() method to start execution
27         thread1.start();
28         thread2.start();
29         thread3.start();
30         thread4.start();
31     }
32 }
```

Explanation

- **Line 5:** We print the current thread name and its priority value in the run() method.
- **Lines 10 to 13:** We create four threads as the setPriorityExample class is inherited from the Thread class.
- **Line 15:** We set thread1 priority to Max_priority, which is 10.
- **Line 18:** We set thread2 priority to MIN_priority, which is 1.
- **Line 21:** We label thread3 as a normal thread, which means it has no priority. The priority value is 5.
- **Line 24:** We label thread4 as a user-defined thread, which means the priority of the thread will be set by the user. In this case, the priority value is 6.
- **Lines 27 to 30:** We call the start() method to invoke JVM for thread execution.

IllegalArgumentException

```
1  public class setPriorityExample extends Thread  {
2      // run() method to start it's execution
3      public void run()  {
4          // Printing about this thread
5          System.out.println("Priority of "+ Thread.currentThread().getName()
6                  + " is: " + Thread.currentThread().getPriority());
7      }
8      public static void main(String args[]) {
9          // Creating 4 threads
10         setPriorityExample thread= new setPriorityExample();
11
12         // Set Priority & name of thread Thread
13         thread.setPriority(15);
14         thread.setName("Thread-1");
15
16         thread.start();
17     }
18 }
```

Explanation

The above code generates an error IllegalArgumentException because we try to set priority value = 15 in line 13, which is greater than the MAX_PRIORITY value.

Section 3: Exercises

Exercise 1: Participate in group discussion on following topics.

a) What is the Main Objective of Exception Handling?
b) What is the Importance of Exception Handling in Java?
c) Can we Handle Exception without Catch Block?
d) What happens if an Exception is not provided by User?

STL academy

Section 4: Assessment Questionnaire

1. When does Exceptions in Java arises in code sequence?
   a) Run Time
   b) Compilation Time
   c) Can Occur Any Time
   d) None of the mentioned
2. Which of these keywords is not a part of exception handling?
   a) try
   b) finally
   c) thrown
   d) catch

3. Which of these keywords must be used to monitor for exceptions?
   a) try
   b) finally
   c) throw
   d) catch

4. What will be the output of the following Java program?

```
1.  class exception_handling
2.  {
3.  public static void main(String args[])
4.  {
5.  try
6.  {
a.  int a, b;
b.  b = 0;
c.  a = 5 / b;
d.  System.out.print("A");
7.  }
8.  catch(ArithmeticException e)
9.  {
a.  System.out.print("B");
10. }
11. }
12. }
```

   a) A
   b) B
   c) Compilation Error
   d) Runtime Error

5. Which of these keywords must be used to handle the exception thrown by try block in some rational manner?
   a) Try
   b) Finally
   c) Throw
   d) catch

6. What will be the output of the following Java program?

```
1.  class exception_handling
2.  {
3.      public static void main(String args[])
4.      {
5.      try
6.      {
7.              System.out.print("Hello" + " " + 1 / 0);
8.      }
9.      catch(ArithmeticException e)
10.     {
11.             System.out.print("World");
12.     }
13.     }
14. }
```

   a) Hello
   b) World
   c) HelloWorld
   d) Hello World

7. Which of the following should be true of the object thrown by a thrown statement?
   a) Should be assignable to String type
   b) Should be assignable to Exception type
   c) Should be assignable to Throwable type
   d) Should be assignable to Error type

8. Which part of code gets executed whether exception is caught or not?
   a) finally
   b) try
   c) catch
   d) throw

9. At runtime, error is recoverable.
   a) True
   b) False

10. Which of the following is a super class of all exception type classes?
    a) Catchable
    b) RuntimeExceptions
    c) String
    d) Throwable

11. Will it be possible to only include a ‘try’ block without the ‘catch’ and ‘finally’ blocks?

12. Will it be possible to keep the statements after the ‘finally’ block if the control is returning from the finally block itself?

13. What do you mean by an exception?

14. Explain how exceptions can be handled in Java. What is the exception handling mechanism behind the process?
15. Is it possible to keep other statements in between ‘try’, ‘catch’, and ‘finally’ blocks?

16. Differentiate error and exception in Java.

17. What are the types of exceptions? Explain them.

18. What are runtime exceptions in Java? Give a few examples.

19. Define OutOfMemoryError in Java.

20. Differentiate between NoClassDefFoundError and ClassNotFoundException in Java.

21. Does the ‘finally’ block get executed if either of ‘try’ or ‘catch’ blocks return the control?

22. It is often recommended to keep clean-up operations like closing the DB resources inside the ‘finally’ block. Why is it necessary?

23. How would you differentiate between final, finally and finalize in Java?

24. What are customized exceptions in java?

25. How would you explain a ClassCastException in Java?

26. Differentiate between throw, throws and throwable in Java.

27. Define chained exceptions in Java.
28. Which class is defined as a super class for all types of errors and exceptions in Java?
29. What can classify as a correct combination of try, catch and finally blocks?
30. Why do you use printStackTrace() method?

----------End of the Module----------

MODULE 4
AWT (ABSTRACT WINDOW TOOLKIT) CONTROLS

Section 1: Learning Outcomes

After completing this module, you will be able to:

- Explain AWT Class hierarchy
- Explain User interface components- Labels, Button, Text Components, Check Box, Check Box Group, Choice, List Box
- Describe Event Handling: Events, Event sources, Event Listeners, Event Delegation Model (EDM)
- State Adapter classes
- Explain Inner classes
- Explain Working with Frame class, Color, Fonts and layout managers

Section 2: Relevant Knowledge

4.1 AWT Class Hierarchy

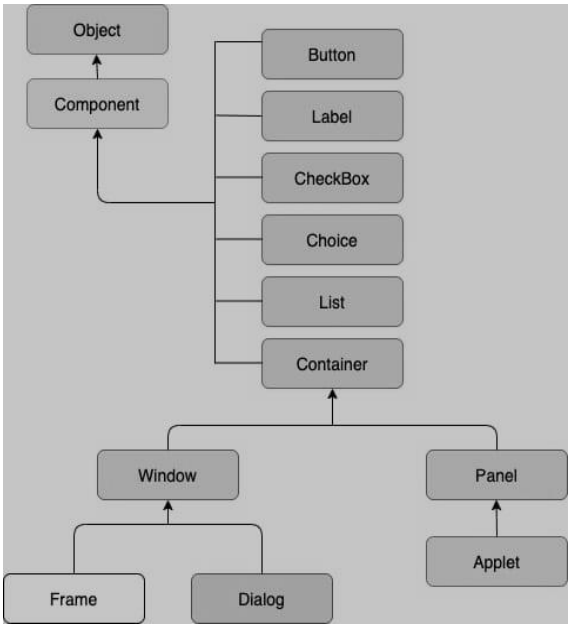
- Abstract Window Toolkit is an API used to develop Graphical User Interface or window-based applications.
- AWT components are displayed according to the view of the operating system this makes them platform-independent.
- AWT is a heavy weight because its components are using the resources of the underlying OS.
- Java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice etc.

Why AWT is platform independent?

- Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.
- For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix.
- The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.
- In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

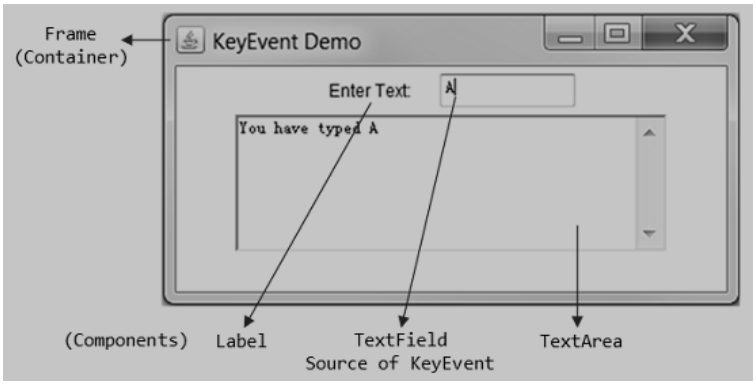
Java AWT Hierarchy

- The hierarchy of Java AWT classes are given below.



4.2 User Interface Components

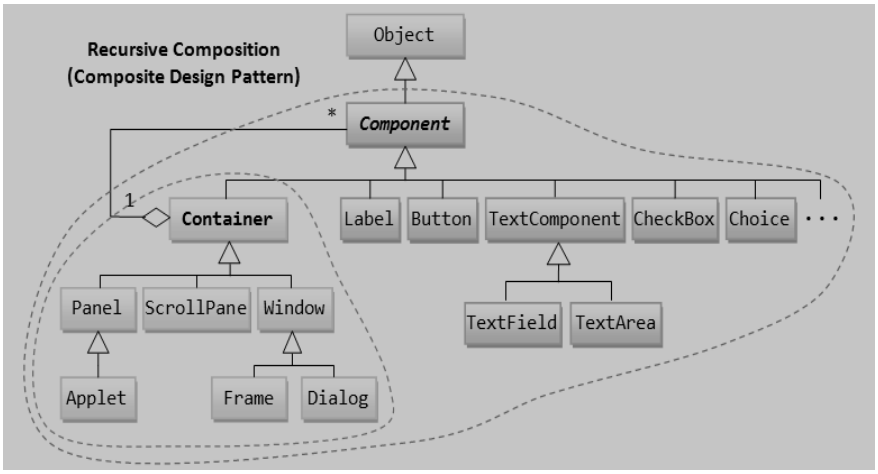
- Labels, Button, Text Components, Check Box, Check Box Group, Choice, List Box



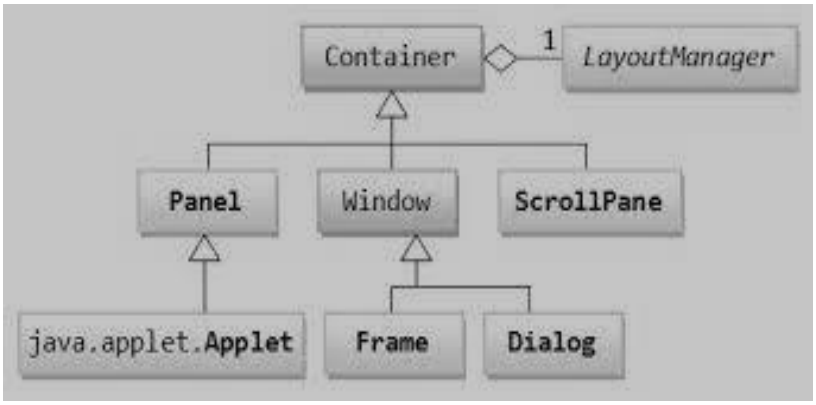
There are classes for every component in java AWT.

Container

- This contains all other components in java AWT.
- Since a container is a component, you can have a container inside a container.



Types of Containers



1. Window
2. Panel
3. Frame
4. Dialog

Window

- This is a container that has no borders or menu bars.
- In order to create a window you must use frame, dialog or another window.

Panel

- A container that doesn't contain tittle bar, border or menu bar.
- May contain other components like button, text field etc.

Frame

- This is the container that has tittle bar and border and can have menu bars.
- It may also have other components like button, text field, scroll bar etc.
- The most widely used.

STL academy

Useful Methods of Component Class

Method	Description
public void add(Component c)	Inserts a component on this component.
public void setSize(int width,int height)	Sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	Defines the layout manager for the component.
public void setVisible(boolean status)	Changes the visibility of the component, by default false.

Java AWT Example

- To create simple AWT example, you need a frame. There are two ways to create a GUI using Frame in AWT.

1. By extending Frame class (inheritance)
2. By creating the object of Frame class (association)

AWT Example by Inheritance

- Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

AWTExample1.java

```
// importing Java AWT class
import java.awt.*;

// extending Frame class to our class AWTExample1
public class AWTExample1 extends Frame {

  // initializing using constructor
  AWTExample1() {

    // creating a button
    Button b = new Button("Click Me!!");

    // setting button position on screen
    b.setBounds(30,100,80,30);

    // adding button into frame
    add(b);

    // frame size 300 width and 300 height
    setSize(300,300);
```

```
    // setting the title of Frame
    setTitle("This is our basic AWT example");

    // no layout manager
    setLayout(null);

    // now frame will be visible, by default it is not visible
    setVisible(true);
}

// main method
public static void main(String args[]) {

// creating instance of Frame class
AWTExample1 f = new AWTExample1();

}

}
```

STL academy

Event and Listener (Java Event Handling)

- Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

Register the component with the Listener

Registration Methods

- For registering the component with the Listener, many classes provide the registration methods.

For example:

- Button
  - public void addActionListener(ActionListener a){}
- MenuItem
  - public void addActionListener(ActionListener a){}
- TextField
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- TextArea
  - public void addTextListener(TextListener a){}
- Checkbox
  - public void addItemListener(ItemListener a){}
- Choice
  - public void addItemListener(ItemListener a){}
- List
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

Java Event Handling Code

- We can put the event handling code into one of the following places:
- Within class
- Other class
- Anonymous class

1. Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){

//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);

//register listener
b.addActionListener(this);//passing current instance
```

2. Java event handling by outer class

```
import java.awt.*;
import java.awt.event.*;
class AEvent2 extends Frame{
TextField tf;
AEvent2(){
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
//register listener
Outer o=new Outer(this);
b.addActionListener(o);//passing outer class instance
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent2();
}
}
```

```
import java.awt.event.*;
class Outer implements ActionListener{
AEvent2 obj;
Outer(AEvent2 obj){
this.obj=obj;
}
public void actionPerformed(ActionEvent e){
obj.tf.setText("welcome");
}
}
```

3. Java event handling by anonymous class

```
import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame{
TextField tf;
AEvent3(){
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(50,120,80,30);

b.addActionListener(new ActionListener(){
public void actionPerformed(){
tf.setText("hello");
}
});
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent3();
}
}
```

Java AWT Button

- A button is basically a control component with a label that generates an event when pushed. The Button class is used to create a labeled button that has platform independent implementation.
- The application result in some action when the button is pushed.
- When we press a button and release it, AWT sends an instance of ActionEvent to that button by calling processEvent on the button.
- The processEvent method of the button receives the all the events, then it passes an action event by calling its own method processActionEvent.
- This method passes the action event on to action listeners that are interested in the action events generated by the button.
- To perform an action on a button being pressed and released, the ActionListener interface needs to be implemented.
- The registered new listener can receive events from the button by calling addActionListener method of the button.
- The Java application can use the button's action command as a messaging protocol.

AWT Button Class Declaration

```
public class Button extends Component implements Accessible
```

Button Class Constructors

Following table shows the types of Button class constructors:

Sr. no.	Constructor	Description
1.	Button( )	It constructs a new button with an empty string i.e. it has no label.
2.	Button (String text)	It constructs a new button with given string as its label.

Button Class Methods

Sr. no.	Method	Description
1.	void setText (String text)	It sets the string message on the button
2.	String getText()	It fetches the String message on the button.
3.	void setLabel (String label)	It sets the label of button with the specified string.

4.	String getLabel()	It fetches the label of the button.
5.	void addNotify()	It creates the peer of the button.
6.	AccessibleContext getAccessibleContext()	It fetched the accessible context associated with the button.
7.	void addActionListener(ActionListener l)	It adds the specified action listener to get the action events from the button.
8.	String getActionCommand()	It returns the command name of the action event fired by the button.
9.	ActionListener[ ] getActionListeners()	It returns an array of all the action listeners registered on the button.
10.	T[ ] getListeners(Class listenerType)	It returns an array of all the objects currently registered as FooListeners upon this Button.

11.	protected String paramString()	It returns the string which represents the state of button.
12.	protected void processActionEvent (ActionEvent e)	It process the action events on the button by dispatching them to a registered ActionListener object.
13.	protected void processEvent (AWTEvent e)	It process the events on the button
14.	void removeActionListener (ActionListener l)	It removes the specified action listener so that it no longer receives action events from the button.
15.	void setActionCommand(String command)	It sets the command name for the action event given by the button.

STL academy

Java AWT Button Example

Example 1:

ButtonExample.java

```
import java.awt.*;
public class ButtonExample {
public static void main (String[] args) {

    // create instance of frame with the label
    Frame f = new Frame("Button Example");

    // create instance of button with label
    Button b = new Button("Click Here");

    // set the position for the button in frame
    b.setBounds(50,100,80,30);

    // add button to the frame
    f.add(b);
    // set size, layout and visibility of frame
```

```
    // set the position for the button in frame
    b.setBounds(50,100,80,30);

    // add button to the frame
    f.add(b);
    // set size, layout and visibility of frame
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

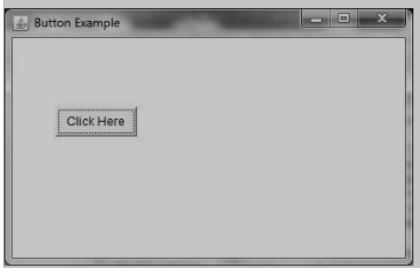
To compile the program using command prompt type the following commands

```
C:\Users\Anurati\Desktop\abcDemo>javac ButtonExample.java
```

If there's no error, we can execute the code using:

```
C:\Users\Anurati\Desktop\abcDemo>java ButtonExample
```

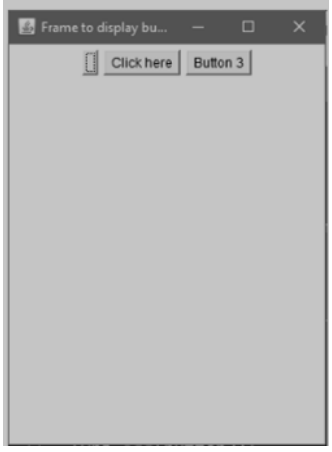
Output:



Example 2:

```
// importing necessary libraries
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ButtonExample2
{
// creating instances of Frame class and Button class
Frame fObj;
Button button1, button2, button3;
// instantiating using the constructor
ButtonExample2() {
fObj = new Frame ("Frame to display buttons");
button1 = new Button();
button2 = new Button ("Click here");
button3 = new Button();
button3.setLabel("Button 3");
fObj.add(button1);
fObj.add(button2);
fObj.add(button3);
fObj.setLayout(new FlowLayout());
fObj.setSize(300,400);
fObj.setVisible(true);
}
// main method
public static void main (String args[])
{
new ButtonExample2();
}
}
```

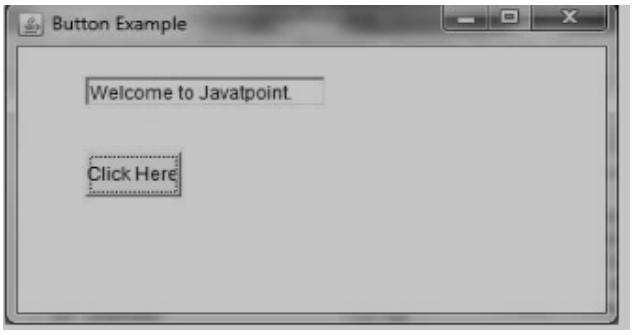
Output:



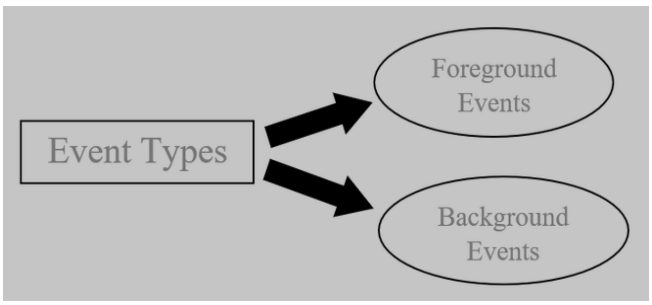
Java AWT Button Example with ActionListener

- In the following example, we are handling the button click events by implementing ActionListener Interface.
- ButtonExample3.java

```
// importing necessary libraries
import java.awt.*;
import java.awt.event.*;
public class ButtonExample3 {
public static void main(String[] args) {
    // create instance of frame with the label
    Frame f = new Frame("Button Example");
    final TextField tf=new TextField();
    tf.setBounds(50,50, 150,20);
    // create instance of button with label
    Button b=new Button("Click Here");
    // set the position for the button in frame
    b.setBounds(50,100,60,30);
    b.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {
          tf.setText("Welcome to Javatpoint.");
       }
    });
// adding button the frame
    f.add(b);
// adding textfield the frame
    f.add(tf);
// setting size, layout and visibility
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```



Types of Events



Foreground Events

- This are the events that require user interaction like mouse click on a button etc.

Background Events

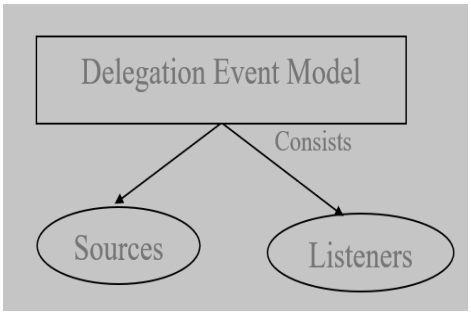
- This are events that do not require user interaction like OS failure and interrupts

Event Handlingdecides

- This is the mechanism that controls the event and decides the things that will happen after an event occurrence.

Delegation Event Mode

- This model has sources and listeners.



Source

- Events are normally generated from sources like checkbox, buttons, list, menu-item, scrollbar etc

Listeners:

- Their work is to handle the events generated from the source
- In order to perform event handling we need to register the source with the listener.

We use the following syntax:
addTypeListener()

4.3 Event Handling: Events, Event Sources, Event Listeners, Event Delegation Model (EDM) Event

- This can be defined as changing the state of an object or behavior by performing actions.
- An action can be a button click, movement of cursor, keypress on keyboard, scrolling of page etc.
- Java.awt.event is package is used to provide various event classes

4.4 Adapter Classes

- The adapter classes provide the default implementation of listener interfaces.
- If you inherit adapter class you won't be forced to provide the implementation of all the methods of listener interfaces.
- Adapter class saves code.
- Adapter class are found in java.awt.event, java.awt.dnd and javax.swing.event packages

4.5 Inner Classes

- An inner class is a class declared inside a class or interface.
- Inner classes have the following advantages:
  - Optimizes the code module
  - Makes code clean and readable
  - Private methods of the outer class can be accessed, so bringing a new dimension and making it close to real world.

STL academy

Types of Inner Classes

- We have four types of inner classes in java i.e.:
  1. Nested Inner Class
  2. Anonymous Inner Classes
  3. Static Nested Classes
  4. Method Local Inner Classes

Nested Inner Class

- It can access any private instance variable of the outer class.
- It has access modifier, protected, public, private and default modifier like any other instance variable.
- An interface class can be nested and have access specifiers.

```
//program
Class OuterClass {
 //…
Class NestedClass {
 //…
}
}
```

Method Local Inner Classes

- Can be declared within the method of an outer class.

```
//program
Class CPU {
Double price;
//nested class
Class processor{
//member of nested class
Double cores;
String manufacturer;
Double getCache() {
 return 4.3;
}
}
//nested protected class
Protected class RAM {
//member of protected nested class
Double memory;
String manufacturer;
Double getClockSpeed() {
 return 5.5;
}
}
}
Public class main {
Public static void main(String[] args) {
//create object of outer class CPU
CPU cpu = new CPU();
//create object of outer class CPU
CPU cpu = new CPU();
// create an object of inner class processor //using outer class
CPU .Processor processor = cpu.new processor();
//create an object of inner class RAM using //outer class CPU
CPU.RAM ram = cpu.new RAM();
System.out.println(“processor cache = “ + processor.getCache());
System.out.println(“Ram Clock Speed = “+ ram.getclockspeed());
}
}
```

Output:

Processor Cache = 4.3

Ram clock speed = 5.5

Static Nested Classes

- This are like a static member of outer class.
- They are not technically inner classes.

```
//program
Class TestOuter1{
Static int data = 30;
Static class inner{
Void msg(){system.out.println(“data is” +data);}
}
Public static void main(main(string args[]){
TestOuter1.inner obj = new TestOuter1.inner();
Obj.msg();
}
}
```

Output: data is 30

Anonymous Inner Classes

- This are declared without any name at all.
- Created in two ways:

1. As a subclass of the specified type
2. As an implement of the specified interface

```
//program
Abstract class person{
Abstract void eat();
}
Class TestAnonymousinner{
Public static void main(string args[]){
Person p = new person(){
Void eat(){system.out.println(“nice people”);}
};
p.eat();
}
}
```

Output:
nice people

4.6 Working with Frame Class, Color, Fonts and Layout Managers

Working with Color

- Color can be controlled by accessing color class.
- The AWT color system allows you to specify any color you want

You can indicate colors in java using:

1. Color classes static color variables: color.cyan
2. Mix your color using the RGB model.
3. Mix your color using HSB model.

Working with Font

- The AWT supports multiple type fonts.
- Fonts are found in the font class.
- Creating and selecting a font:
    - You construct a font object that describes that font
    - They are five font style attributes supported by JVM: Monospace, DialogInput, Dialog, Serif, San-serif.

STL academy

Section 3: Exercises

Exercise 1: Participate in group discussion on following topics.

a) What is the purpose of AWT in Java?
b) Discuss any eight AWT Classes?
c) What are AWT controls, discuss three main aspects of it?
d) What are the limitations of AWT in Java?

Section 4: Assessment Questionnaire

1. Which method used to place some text in the text field?
   A. getText(String str)
   B. setText(String str)
   C. putText(String str)
   D. None of the above

2. What is the listener used to handle the events of a text field?
   A. java.awt.ActionListener interface
   B. java.awt.event.ActionListener
   C. awt.event.ActionListener interface
   D. java.awt.event.ActionListener interface

3. Which method used to change the foreground (text) color of components like text field?
   A. setBackground(Color clr)
   B. setForeground(Color clr)
   C. setColor(Color clr)
   D. setEditable(boolean state)

4. getLabel() method used to retrieve the label of a button.
   A. Yes
   B. No
   C. Can be yes or no
   D. Cannot say

5. How many layout managers defined in java.awt package?
   A. 2
   B. 3
   C. 4
   D. 5

6. A _____ dictates the style of arranging the components in a container.
   A. border layout
   B. grid layout
   C. panel
   D. layout manager

7. AWT stands for:
   A. All Window Toolkit
   B. Abstract Work Toolkit
   C. Abstract Window Toolkit
   D. Abstract Window Text

8. The subclass of a java.awt.Component class is known as a ?
   A. system
   B. component
   C. container
   D. component manager

9. What is the super class of all components of Java?
   A. java.all.Component
   B. all.awt.Component
   C. java.awt.Component
   D. awt.Component

10. The subclass of a java.awt.Container class is known as a container.
    A. TRUE
    B. FALSE
    C. Can be true or false
    D. Cannot say
11. What are AWT Peers?
12. What is the difference between a Swing and AWT Components?
13. What are the different types of Controls in AWT?
14. What are the benefits of Swing over AWT?
15. What are the Component and Container Class?
16. What is the use of the Window Class?
17. What is Clipping?
18. What is the difference between the Paint() and Repaint() Method?
19. What is a Container in a GUI?
20. What is the default layout for Applet?
21. What is the difference between a Menuitem and a Checkboxmenuitem?
22. How are the elements of different layouts organized?
23. What is the difference between Exclusive and Non-Exclusive?
24. What are the subclasses of the Container Class?
25. Which method is method to set the layout of a Container?
26. What are the default layouts for an Applet, a Frame and a Panel?
27. What is the difference between a Scrollbar and a Scrollpane?
28. How can we get all public methods of an Object dynamically?
29. Define Canvas?
30. Explain the use of Update Method?

----------End of the Module----------

MODULE 5
SWING

Section 1: Learning Outcomes

After completing this module, you will be able to:

- Explain Introduction to Swing
- Explain Hierarchy of swing components
- Describe Top level containers - JFrame, JDialog, JPanel, JButton, JToggleButton
- Explain Life cycle of an Applet
- Develope applets
- Understand Simple applet
- Distinguish between Applets and Applications
- Program Dutch National Flag Problem in Java | Java Program to Short an Array of 0's, 1's and 2's

Section 2: Relevant Knowledge

5.1 Introduction to Swing

- Java Swing tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications.
- It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

There are many differences between java AWT and swing that are given below.

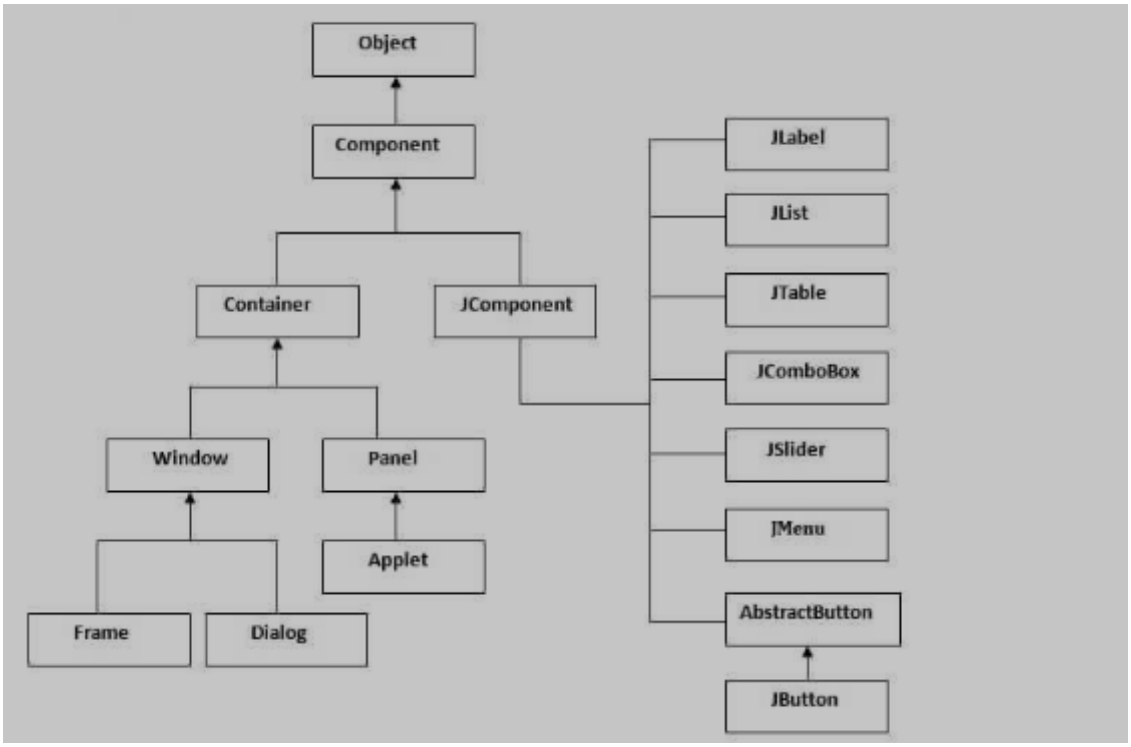
No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent.**	Java swing components are **platform-independent.**
2)	AWT components are **heavyweight.**	Swing components are **lightweight.**
3)	AWT **doesn't support pluggable look and feel.**	Swing **supports pluggable look and feel.**
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC.**

What is JFC

- The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

5.2 Hierarchy of Java Swing classes

- The hierarchy of java swing API is given below.



Commonly used Methods of Component Class

- The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Java Swing Examples

- There are two ways to create a frame:
  - By creating the object of Frame class (association)
  - By extending Frame class (inheritance)

- We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

```
import javax.swing.*;
public class FirstSwingExample {
public static void main(String[] args) {
JFrame f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
}
```

5.3 Top Level Containers

A top-level container has one sole purpose of holding other swing components.

Example of top-level containers:

- JFrame
- JWindow
- JAppalet
- JDialog

Example of Swing by Association inside Constructor

- We can also write all the codes of creating JFrame, JButton and method call inside the java Constructor.

```
import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
JFrame f;
Simple2(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);

add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
new Simple2();
}}
```

Simple example of Swing by Inheritance

- We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

```
import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
JFrame f;
Simple2(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);

add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
new Simple2();
}}
```

Java JFrame

- The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class.
- JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI.
- Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

STL academy

Nested Class

Modifier and Type	Class	Description
protected class	JFrame.AccessibleJFrame	This class implements accessibility support for the JFrame class.

Fields

Modifier and Type	Field	Description
protected AccessibleContext	accessibleContext	The accessible context property.
static int	EXIT_ON_CLOSE	The exit application default window close operation.
protected JRootPane	rootPane	The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane.
protected boolean	rootPaneCheckingEnabled	If true then calls to add and setLayout will be forwarded to the contentPane.

Constructors

Constructor	Description
JFrame()	It constructs a new frame that is initially invisible.
JFrame(GraphicsConfiguration gc)	It creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.
JFrame(String title)	It creates a new, initially invisible Frame with the specified title.
JFrame(String title, GraphicsConfiguration gc)	It creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

Useful Methods

Modifier and Type	Method	Description
protected void	addImpl(Component comp, Object constraints, int index)	Adds the specified child Component.
protected JRootPane	createRootPane()	Called by the constructor methods to create the default rootPane.
protected void	frameInit()	Called by the constructors to init the JFrame properly.
void	setContentPane(Containe contentPane)	It sets the contentPane property
static void	setDefaultLookAndFeelDecorated(boolean defaultLookAndFeelDecorated)	Provides a hint as to whether or not newly created JFrames should have their Window decorations (such as borders, widgets to close the window, title...) provided by the current look and feel.
void	setIconImage(Image image)	It sets the image to be displayed as the icon for this window.
void	setJMenuBar(JMenuBar menubar)	It sets the menubar for this frame.
void	setLayeredPane(JLayeredPane layeredPane)	It sets the layeredPane property.
JRootPane	getRootPane()	It returns the rootPane object for this frame.
TransferHandler	getTransferHandler()	It gets the transferHandler property.

JFrame Example

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class JFrameExample {
   public static void main(String s[]) {
      JFrame frame = new JFrame("JFrame Example");
      JPanel panel = new JPanel();
      panel.setLayout(new FlowLayout());
      JLabel label = new JLabel("JFrame By Example");
      JButton button = new JButton();
      button.setText("Button");
      panel.add(label);
      panel.add(button);
      frame.add(panel);
      frame.setSize(200, 300);
      frame.setLocationRelativeTo(null);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}
```

Output

Java JDialog

- The JDialog control represents a top-level window with a border and a title used to take some form of input from the user. It inherits the Dialog class.
- Unlike JFrame, it doesn't have maximize and minimize buttons.

JDialog Class Declaration

Let's see the declaration for javax.swing.JDialog class.

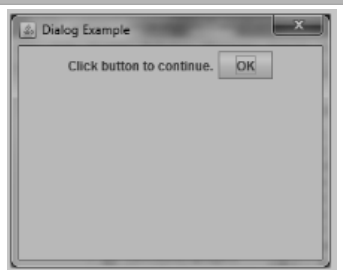
```
public class JDialog extends Dialog implements WindowConstants, Accessible, RootPaneContainer
```

Commonly used Constructors:

Constructor	Description
JDialog()	It is used to create a modeless dialog without a title and without a specified Frame owner.
JDialog(Frame owner)	It is used to create a modeless dialog with specified Frame as its owner and an empty title.
JDialog(Frame owner, String title, boolean modal)	It is used to create a dialog with the specified title, owner Frame and modality.

Java JDialog Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DialogExample {
   private static JDialog d;
   DialogExample() {
      JFrame f= new JFrame();
      d = new JDialog(f , "Dialog Example", true);
      d.setLayout( new FlowLayout() );
      JButton b = new JButton ("OK");
      b.addActionListener ( new ActionListener()
      {
         public void actionPerformed( ActionEvent e )
         {
            DialogExample.d.setVisible(false);
         }
      });
      d.add( new JLabel ("Click button to continue."));
      d.add(b);
      d.setSize(300,300);
      d.setVisible(true);
   }
   public static void main(String args[])
   {
      new DialogExample();
   }
}
```



Java JPanel

- The JPanel is a simplest container class.
- It provides space in which an application can attach any other component.
- It inherits the JComponents class.
- It doesn't have title bar.

JPanel Class Declaration

```
public class JPanel extends JComponent implements Accessible
```

Commonly used Constructors:

Constructor	Description
JPanel()	It is used to create a new JPanel with a double buffer and a flow layout.
JPanel(boolean isDoubleBuffered)	It is used to create a new JPanel with FlowLayout and the specified buffering strategy.
JPanel(LayoutManager layout)	It is used to create a new JPanel with the specified layout manager.

Java JPanel Example

```
import java.awt.*;
import javax.swing.*;
public class PanelExample {
    PanelExample()
      {
      JFrame f= new JFrame("Panel Example");
      JPanel panel=new JPanel();
      panel.setBounds(40,80,200,200);
      panel.setBackground(Color.gray);
      JButton b1=new JButton("Button 1");
      b1.setBounds(50,100,80,30);
      b1.setBackground(Color.yellow);
      JButton b2=new JButton("Button 2");
      b2.setBounds(100,100,80,30);
      b2.setBackground(Color.green);
      panel.add(b1); panel.add(b2);
      f.add(panel);
            f.setSize(400,400);
            f.setLayout(null);
            f.setVisible(true);
      }
      public static void main(String args[])
      {
      new PanelExample();
      }
  }
```

Output:

Java JButton

- The JButton class is used to create a labeled button that has platform independent implementation.
- The application result in some action when the button is pushed. It inherits AbstractButton class.

JButton class declaration

Let's see the declaration for javax.swing.JButton class.

```
public class JButton extends AbstractButton implements Accessible
```

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

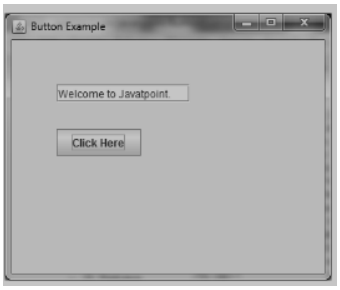
Java JButton Example

```
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    f.add(b);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:

Java JButton Example with ActionListener

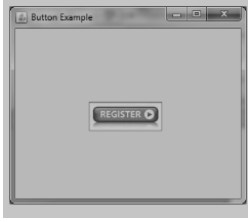
```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    final JTextField tf=new JTextField();
    tf.setBounds(50,50, 150,20);
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    b.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
            tf.setText("Welcome to Javatpoint.");
        }
    });
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```



Example of displaying image on the button:

```
import javax.swing.*;
public class ButtonExample{
ButtonExample(){
JFrame f=new JFrame("Button Example");
JButton b=new JButton(new ImageIcon("D:\\icon.png"));
b.setBounds(100,100,100, 40);
f.add(b);
f.setSize(300,400);
f.setLayout(null);
f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
public static void main(String[] args) {
    new ButtonExample();
}
}
```

Output:



Java JToggleButton

- JToggleButton is used to create toggle button, it is two-states button to switch on or off.

Nested Classes

Modifier and Type	Class	Description
protected class	JToggleButton.AccessibleJToggleButton	This class implements accessibility support for the JToggleButton class.
static class	JToggleButton.ToggleButtonModel	The ToggleButton model

Constructors

Constructor	Description
JToggleButton()	It creates an initially unselected toggle button without setting the text or image.
JToggleButton(Action a)	It creates a toggle button where properties are taken from the Action supplied.
JToggleButton(Icon icon)	It creates an initially unselected toggle button with the specified image but no text.
JToggleButton(Icon icon, boolean selected)	It creates a toggle button with the specified image and selection state, but no text.
JToggleButton(String text)	It creates an unselected toggle button with the specified text.
JToggleButton(String text, boolean selected)	It creates a toggle button with the specified text and selection state.
JToggleButton(String text, Icon icon)	It creates a toggle button that has the specified text and image, and that is initially unselected.
JToggleButton(String text, Icon icon, boolean selected)	It creates a toggle button with the specified text, image, and selection state.

Methods

Modifier and Type	Method	Description
AccessibleContext	getAccessibleContext()	It gets the AccessibleContext associated with this JToggleButton.
String	getUIClassID()	It returns a string that specifies the name of the l&f class that renders this component.
protected String	paramString()	It returns a string representation of this JToggleButton.
void	updateUI()	It resets the UI property to a value from the current look and feel.

STL academy

JToggleButton Example

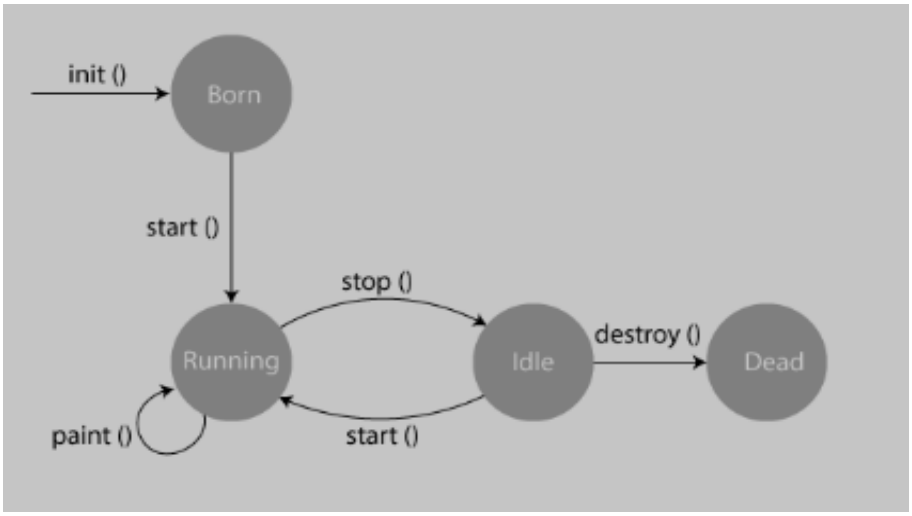
```
import java.awt.FlowLayout;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import javax.swing.JFrame;
import javax.swing.JToggleButton;

public class JToggleButtonExample extends JFrame implements ItemListener {
   public static void main(String[] args) {
      new JToggleButtonExample();
   }
   private JToggleButton button;
   JToggleButtonExample() {
      setTitle("JToggleButton with ItemListener Example");
      setLayout(new FlowLayout());
      setJToggleButton();
      setAction();
      setSize(200, 200);
      setVisible(true);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
   private void setJToggleButton() {
      button = new JToggleButton("ON");
      add(button);
   }
   private void setAction() {
      button.addItemListener(this);
   }
   public void itemStateChanged(ItemEvent eve) {
      if (button.isSelected())
         button.setText("OFF");
      else
         button.setText("ON");
   }
}
```

Output

5.4 Life Cycle of an Applet

- In Java, an applet is a special type of program embedded in the web page to generate dynamic content. Applet is a class in Java.
- The applet life cycle can be defined as the process of how the object is created, started, stopped, and destroyed during the entire execution of its application.
- It basically has five core methods namely init(), start(), stop(), paint() and destroy().
- These methods are invoked by the browser to execute.
- Along with the browser, the applet also works on the client side, thus having less processing time.

Methods of Applet Life Cycle



- There are five methods of an applet life cycle, and they are:

➢ **init():** The init() method is the first method to run that initializes the applet. It can be invoked only once at the time of initialization. The web browser creates the initialized objects, i.e., the web browser (after checking the security settings) runs the init() method within the applet.

➢ **start():** The start() method contains the actual code of the applet and starts the applet. It is invoked immediately after the init() method is invoked. Every time the browser is loaded or refreshed, the start() method is invoked. It is also invoked whenever the applet is maximized, restored, or moving from one tab to another in the browser. It is in an inactive state until the init() method is invoked.

➢ **stop():** The stop() method stops the execution of the applet. The stop () method is invoked whenever the applet is stopped, minimized, or moving from one tab to another in the browser, the stop() method is invoked. When we go back to that page, the start() method is invoked again.

➢ **destroy():** The destroy() method destroys the applet after its work is done. It is invoked when the applet window is closed or when the tab containing the webpage is closed. It removes the applet object from memory and is executed only once. We cannot start the applet once it is destroyed.

➢ **paint():** The paint() method belongs to the Graphics class in Java. It is used to draw shapes like circle, square, trapezium, etc., in the applet. It is executed after the start() method and when the browser or applet windows are resized.

Sequence of method execution when an applet is executed:

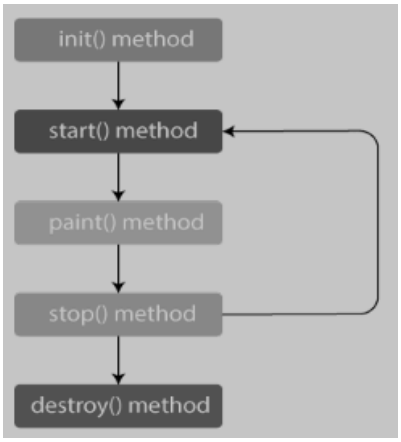
- init()
- start()
- paint()

Sequence of method execution when an applet is executed:

- stop()
- destroy()
- Applet Life Cycle Working
- The Java plug-in software is responsible for managing the life cycle of an applet.
- An applet is a Java application executed in any web browser and works on the client-side. It doesn't have the main() method because it runs in the browser. It is thus created to be placed on an HTML page.
- The init(), start(), stop() and destroy() methods belongs to the applet.Applet class.
- The paint() method belongs to the awt.Component class.
- In Java, if we want to make a class an Applet class, we need to extend the Applet
- Whenever we create an applet, we are creating the instance of the existing Applet class. And thus, we can use all the methods of that class.

Flow of Applet Life Cycle:

- These methods are invoked by the browser automatically. There is no need to call them explicitly.



Syntax of entire Applet Life Cycle in Java

```
class TestAppletLifeCycle extends Applet {
public void init() {
// initialized objects
}
public void start() {
// code to start the applet
}
public void paint(Graphics graphics) {
// draw the shapes
}
public void stop() {
// code to stop the applet
}
public void destroy() {
// code to destroy the applet
}
}
```

5.5 Developing Applets

- A component designed using component-based architecture can be developed into java applet.

Steps to make an applet:

1. Create a class MyTopJPanel that is a subclass of javax.swing.JPanel. Then lay your components in the constructor of the MyTopJPanel class.
2. Create a class called MyApplet that is a subclass of javax.swing.Japplet
3. In the init method of MyApplet, instantiate MyTopJPanel and set it as the applet's content pane

```
//program
Import java.applet.Applet;
Import java.awt.Graphics;
Public class first extends Applet{
Public void paint(Graphics g){
g.drawString("welcome", 150,150);
}
}
```

5.6 Differences Between Applets and Applications

Applications are programs that stand-alone and are run independently without the need of a web browser while applets run on the client-side and are designed to be included in a HTML web document.

- Java application and Java applet both are Java programs
- But there is a slight difference between them.
- Like in any program, the execution of the Java application always begins with the main( ) method, while in the case of an applet, initialization takes through the init( ).
- There is no need to invoke the main( ) method.
- Java applications are types of stand-alone applications that run directly on the underlying operating system with the help of virtual machine.
- These perform various general operations for their users and do not require any APIs or browsers enabled by Java.
- On the other hand, the applets are small programs that can be embedded into a web page.
- The applets are used to make the website more dynamic and entertaining.
- We can use the OBJECT or APPLET tag to embed an applet on an HTML page and manage it on any web server.
- A web browser that is compatible with Java can easily run applets.
- Hardware and operating system of the devices does not affect Java applets.
- One thing is clear and that is the overall appearance of Java applications are always the same on different OS. If the affected system browser has a well-installed JVM, then it can easily work with the help of these JVMs.

STL academy

Java Application

Java application is basically a Java program (collection of instructions) that runs stand alone in a client or server, and works on an underlying OS (operating system) that receives virtual machine support. Graphical User Interface (GUI) is not required for execution of a Java application.

Java Application Features

- It resembles Java programs strongly.
- We do not require any web browser for the execution of these programs. Execution can be easily done via the local system.
- The implementation of these applications requires main( ) function.
- The local file systems and networks are fully accessed by these applications.

Java Applet

- The Java applet works on the client side, and runs on the web browser. It is a Java application that the user can easily embed on a web page.
- Java Applet Features
- Java applet is a small and easy-to-write Java program.
- One can easily install Java applet along with various HTML documents.
- One needs a web browser (Java based) to use applets.
- Applet do not have access to the network or local disk and can only access browser-specific services.
- It cannot perform system operations on local machines.
- Java applet cannot establish access to local system.

Java Application Vs. Java Applet

Parameters	Java Application	Java Applet
Meaning	A Java Application also known as application program is a type of program that independently executes on the computer.	The Java applet works on the client side, and runs on the browser and makes use of another application program so that we can execute it.
Requirement of main( ) method	Its execution starts with the main( ) method only. The use of the main( ) is mandatory.	It does not require the use of any main() method. Java applet initializes through init( ) method.
Execution	It cannot run independently, but requires JRE to run.	It cannot start independently but requires APIs for use (Example. APIs like Web API).
Installation	We need to install the Java application first and obviously on the local computer.	Java applet does not need to be pre-installed.
Connectivity with server	It is possible to establish connections with other servers.	It cannot establish connection to other servers.
Operation	It performs read and write tasks on a variety of files located on a local computer.	It cannot run the applications on any local computer.
File access	It can easily access a file or data available on a computer system or device.	It cannot access the file or data found on any local system or computer.
Security	Java applications are pretty trusted, and thus, come with no security concerns.	Java applets are less reliable. So, they need to be safe.

STL academy

5.7 Dutch National Flag Problem in Java |

Java Program to Short an Array of 0's, 1's and 2's

- Dutch National Flag (DNF) problem is one of the most popular programming problems proposed by the famous Dutch computer scientist Edsger Dijkstra.
- As its name suggest, it is based on the flag of Netherlands that consists tri colors i.e. red, white, and blue.
- The task is to randomly arrange the bolls of red, white, and blue in such a way that balls of the same color are placed together.
- We will solve the above problem using an array.
- Instead of using the colors, here we will use 0's, 1's, and 2's that represents red, white, and blue colors, respectively. In order to match the color of the bolls we will sort the array.
- For example, consider the following array.
  - **Input: {0, 1, 2, 2, 1, 0, 0, 2, 0, 1, 1, 0}**
  - **Output: {0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2}**
- So, Dutch National flag problem is the same as short an array of 0's, 1's, and 2's.
- In this section, we will create a Java program for the same.

Solution to the Problem

- There are many solutions are available that may vary in performance and characteristics.

Naive Approach

The simple solution is to perform the counting sort over the array. In this sorting, we count the frequency of the elements (0, 1, and 2), after that put them in the correct order. The disadvantage of the approach is that it traverses two times over the array, one for sorting, second for putting elements in the correct order.

In order to overcome the above shortcoming, we reorganize the array in such a way that solves the problem in single traversal. It uses an alternative linear-time partition routine (the same as 3-way partition) that separates the values into the following three groups.

- The values (red) less than the pivot.
- The values (white) equal to the pivot, and
- The values (blue) greater than the pivot.

Algorithm

```
procedure three-way-partition(A : array of values, mid : value):
  i ← 0
  j ← 0
  k ← size of A - 1
  while j <= k:
    if A[j] < mid:
      swap A[i] and A[j]
      i ← i + 1
      j ← j + 1
    else if A[j] > mid:
      swap A[j] and A[k]
      k ← k - 1
    else:
      j ← j + 1
```

Let's understand it through an example.

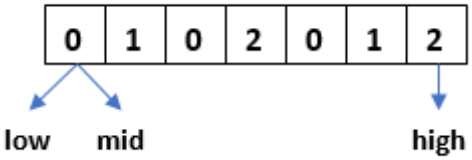
Example

- In the following example, we have used three variables low, mid, and high. The low and mid pointer point at start and the high pointer point at the end of the array. There are three cases:
  - If **array[mid]=0**, swap **array[mid] with array[low]** and increment both the pointers by one.
  - If **array[mid]=1, no swapping** performed but increment the mid pointer.
  - If **array[mid]=2**, swap **array[mid] with array[high]** and decrement the high pointer by one.

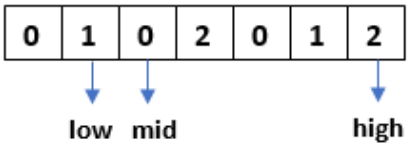
Consider the following array.

Initially low and mid points the start of the array i.e. 0 and the high points end of the array i.e. 2.

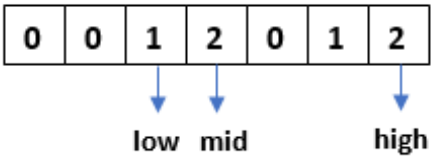
Step 1: Check if **array[mid]=0**, swap array[mid] with array[low] and increment both the pointers by 1.



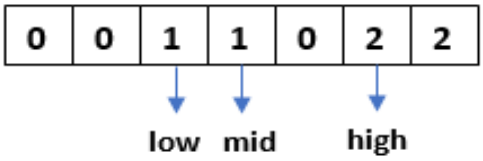
Step 2: Check if **array[mid]=1**, no swapping is required. Increment the mid pointer by 1.



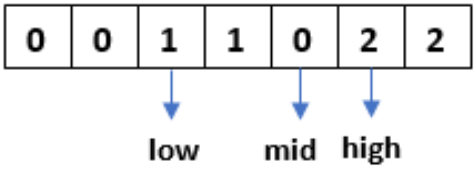
Step 3: Check if **array[mid]=0,** swap array[mid] with array[low] and increment both the pointers by 1.



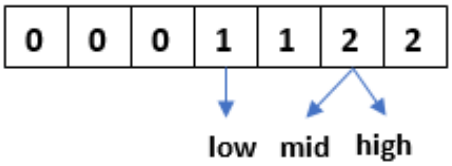
Step 4: Check if **array[mid]=2**, swap array[mid] with array[high] and decrement the high pointer by 1.



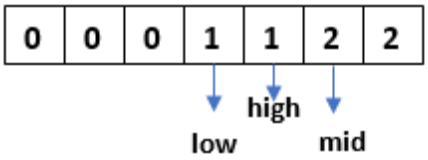
Step 5: Check if **array[mid]=1**, no swapping required increments the pointers by 1.



Step 6: Check if **array[mid]=0,** swap array[mid] with array[low] and increment both the pointers by 1.



Step 7: Check if **array[mid]=2**, swap array[mid] with array[high] and decrement the high pointer by 1.



We overserve that the array is sorted.

Let's implement the above approach in Java programs with different logics.

DutchNationalFlag.java

```
import java.util.Arrays;
public class DutchNationalFlag
{
   // Linear time partition routine to sort an array containing 0, 1, and 2.
   // It is similar to 3-way partitioning for the Dutch national flag problem.
   public static void threeWayPartition(int[] A)
   {
      int start = 0, mid = 0;
      int pivot = 1;
      int end = A.length - 1;
      while (mid <= end)
      {
         if (A[mid] < pivot)        // if current element is 0
         {
            swap(A, start, mid);
            ++start;
            ++mid;
         }
         else if (A[mid] > pivot)   // if current element is 2
         {
            swap(A, mid, end);
            --end;
         }
         else {                     // if current element is 1
            ++mid;
         }
      }
   }
   // Utility function to swap elements `A[i]` and `A[j]` in the array
   private static void swap(int[] A, int i, int j)
   {
      int temp = A[i];
      A[i] = A[j];
      A[j] = temp;
   }
//driver code
   public static void main (String args[])
   {
   //array to be short
      int[] A = { 0, 1, 2, 2, 1, 0, 0, 2, 0, 1, 1, 0 };

      threeWayPartition(A);
   //prints sorted array
      System.out.println(Arrays.toString(A));
   }
}
```

Output:

```
[0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2]
```

Let's see another logic for the same.

DutchFlagProblem.java

```
import java.io.*;
public class DutchFlagProblem
{
   static void DNFS(int arr[], int arr_size)
   {
      int low = 0;
      int high = arr_size - 1;
      int mid = 0, temp=0; // temp variable is used for swapping
      while (mid <= high)
      {
         switch (arr[mid])
         {
      case 0: // if mid pointer points at 0
      {
      //swapping logic
        temp = arr[low];
        arr[low] = arr[mid];
        arr[mid] = temp;
        low++;
      mid++;
      break;
    }
    case 1: // if mid pointer points at 1
    //no swapping required only increment the mid pointer
      mid++;
      break;
    case 2: // if mid pointer points at 2
    {
    //swapping logic
      temp = arr[mid];
      arr[mid] = arr[high];
      arr[high] = temp;
      high--;
      break;
    }
        }
    }
 }
```

```
//function to print array sorted and unsorted array
static void printArray(int arr[], int arr_size)
{
    int i;
    //loop iterate over the array
    for (i = 0; i < arr_size; i++)
    //prints array elements
        System.out.print(arr[i]+" ");
        //prints space
    System.out.println("");
}
/*Driver function to check for above functions*/
public static void main (String args[])
{
    int arr[] = {0, 0, 1, 2, 0, 1, 2};
    //finds length of the array
    int arr_size = arr.length;
System.out.println("Array before Sorting: ");
    printArray(arr, arr_size);
     //function calling before sorting
 DNFS(arr, arr_size);
 //printing sorted array
 System.out.println("Array after sorting: ");
 //function calling after sorting
     printArray(arr, arr_size);
 }
}
```

Output:

```
Array before Sorting:
0 0 1 2 0 1 2
Array after sorting:
0 0 0 1 1 2 2
```

The above program shorts the array in linear time without using extra space. The array is traversed only once. Hence, the time complexity of the is O(n), where n is the size of the array.

Java Calculate Average of List

- In Java, List is a linear data structure that store the ordered collection of data.
- It also accepts the duplicate values but preserve the insertion order.
- Sometimes, it is required to find the minimum and maximum element of the list, sum, and average of the list, etc.
- In this section, we will calculate the average of List using Java for loop, Java 8 Stream, and IntSummaryStatistics class.

Java Programs to Calculate the Average of List

Using Java for Loop

- First, we will initialize a variable sum to 0 that holds the sum of the list elements.
- Declare another element say average (avg), it holds the average of the list.
- We have created an instance of the ArrayList class and invoked the add() method to add the elements to the List.
- A for loop iterate over the list and get an element at each iteration and add that element to the variable sum.
- After that, we have calculated the average of the list, dividing the sum by the size of the list (number of elements).

Let's implement the above approach in a Java program.

AverageOfList.java

```
import java.util.*;
public class AverageOfList
{
public static void main(String args[])
{
int sum = 0, avg;
ArrayList<Integer> list = new ArrayList<Integer>();
//adding elements to the List
list.add(12);
list.add(34);
list.add(10);
list.add(48);
list.add(65);
//loop iterates over the List
for(int i = 0; i < list.size(); i++)
//getting elements from the list and adding to the variable sum
sum = sum + list.get(i);
//finds the average of the list
avg = sum / list.size();
//prints the result
System.out.println("The average of the List: " + avg);
}
}
```

Output:

```
The average of the List: 33
```

Let's see another logic for the same.

FindAverageOfList.java

```
import java.util.*;
public class FindAverageOfList
{
public static void main(String args[])
{
//creates an array
//the asList() method return a fixed-size list backed by the specified array
List<Integer> list = Arrays.asList(10, 20, 30, 40, 50, 60, 70, 80, 90, 100);
//variable to store sum
int sum = 0;
//for-each loop iterates over the list
for (int i : list)
{
//adds elements to the variable sum
sum+=i;
}
//checks if the list is empty
if(list.isEmpty())
{
System.out.println("List is empty!");
}
else
{
//calculate average and returns the same
System.out.println("The average of the List is: " + sum/(float)list.size());
}
}
}
```

Output:

```
The average of the List is: 55.0
```

Using IntSummaryStatistics Class

IntSummaryStatistics is a class that belongs to **java.util** package. It implements the IntStream class. The class is dedicated to work with streams. A state object for collecting statistics such as **count, min, max, sum,** and **average**. In the following Java programs, we have used the following methods:

- **stream():** The method returns a **sequential Stream**with this collection as its source.
- **mapToInt():** It is a lazy **intermediate operation**. The operations are invoked on a Stream instance and once the operation finish their processing, they give a Stream instance as output.
- **summaryStatistics():** The method returns an **IntSummaryStatistics** describing various summary data about the elements of this stream like count of number of elements in the IntStream, average of all elements present in IntStream, minimum and maximum element in the IntStream and so on. It is a terminal operation. It means, it may traverse the stream to produce a result.
- **getAverage():** The method returns the **arithmetic mean** of values recorded, or zero if no values have been recorded.

CalculateAverageOfList.java

```
import java.util.*;
public class CalculateAverageOfList
{
public static void main(String args[])
{
//creates an array
//the asList() method return a fixed-size list backed by the specified array
List<Integer> list = Arrays.asList(23, 56, 78, 12, 46, 89, 11, 134, 234, 1, 5, 9);
IntSummaryStatistics iss = list.stream().mapToInt((a) -> a). summaryStatistics();
System.out.println("The average of the List is: "+iss.getAverage());
}
}
```

Output:

```
The average of the List is: 58.166666666666664
```

compareToIgnoreCase Java

- In Java, the method compareToIgnoreCase() belongs to the String class that belong to java.lang package.
- It is used for comparing any two strings by ignoring the lower- and upper-case differences.
- The method does the comparison of strings using the Unicode value of each character present in both of the strings.
- The way we pass both the strings to the method compareToIgnoreCase() is similar to the method compareTo(), and the following results is returned by the method.
- A positive number is returned if string 1 is greater than string 2.
- A negative number is returned if string 1 is less than string 2.
- Zero is returned if string 1 is equal to the string 2.

Syntax:

The syntax of the method is:

```
int compareToIgnoreCase(String s)
```

Parameter: String s is the parameter of the method that is to be compare.

Return Type: It returns an integer value.

CompareToIgnoreCase Java Program

Let's see how one can use the method in a Java program.

FileName: CompareToIgnoreCase.java

```
public class CompareToIgnoreCase
{

// main method
public static void main(String argvs[])
{
String string1 = "Book";
String string2 = "book";
String string3 = "look";
String string4 = "abc";
String string5 = "BEEN";

System.out.println(string1.compareToIgnoreCase(string2));
System.out.println(string1.compareToIgnoreCase(string3));
System.out.println(string1.compareToIgnoreCase(string4));
System.out.println(string1.compareToIgnoreCase(string5));

}
}
```

Output:

```
0
-10
1
10
```

Explanation:

- Since we are using the compareToIgnoreCase() for string comparison; therefore, string1 and string2 are treated as the same strings. Therefore, the first output is 0.
- For the second output, the string string1 ("Book") is compared with the string3 ("look"), and the gap between 'B' and 'l' is 10 as per the alphabetical sequence (remember the case sensitiveness is ignored. Thus, 'l' can be treated as 'L'). Since 'B' comes after 'l'; therefore, -10 is the answer.
- For the third output, the character 'B' is compared with the character 'a', and we know that the gap between 'B' and 'a' is 1. So, the output is 1.
- For the fourth output, "Book" is compared with "BEEN". Here, the first character of "Book" is compared with the first character of "Been", and we get 0. So, the second character of both the strings ('o' of "Book", and 'E' of BEEN) is taken into consideration, and we see that the gap between 'o' and 'E' is 10. Since 'o' comes after 'E'; therefore, the output is plus 10.

Custom compareToIgnoreCase() Method

- We can also define own compareToIgnoreCase() method in Java. The method takes two parameters: one is string string1, and the other is string string2 and will return an integer. The following program illustrates the same.

FileName: CompareToIgnoreCase1.java

```
public class CompareToIgnoreCase1
{
public int compareToIgnoreCase(String string1, String string2)
{
int i = 0;
int j = 0;
// converting each character of both the strings to the
// lowercase so that the case insensitiveness is maintained
string1 = string1.toLowerCase();
string2 = string2.toLowerCase();
int size1 = string1.length();
int size2 = string2.length();
while(i < size1 && j < size2)
{
   char c1 = string1.charAt(i);
   char c2 = string2.charAt(j);

   // if both the characters are equal, continue for the next iteration
   if(c1 == c2)
   {
      i = i + 1;
      j = j + 1;
   }

   else
   {
      // the codePointAt() method returns the Unicode value of characters
      // here, we are finding the difference of the Unicode value of the character
      // sitting at the index i of the string string1 and the character
      // sitting at the index j of the string string2
      return (string1.codePointAt(i) - string2.codePointAt(j));
   }
   }
// if all the characters of both the strings are equal as well as
// both the strings are of the same length; we return 0
if(i == size1 && j == size2)
{
```

```
    return 0;
}
// if string1 has some extra characters present in the end
// that are not present in the string2, then count the extra characters
// and return it
if(i < size1)
{
    return size1 - i;
}
// if string2 has some extra characters present in the end
// that are not present in the string1, then count the extra characters put a minus sign
// and return it
return (j - size2);
}
// main method
public static void main(String argvs[])
{
// creating an object of the class CompareToIgnoreCase1
CompareToIgnoreCase1 obj = new CompareToIgnoreCase1();
String string1 = "Book";
String string2 = "book";
String string3 = "look";
String string4 = "abc";
String string5 = "BEEN";
// invoking our own compareToIgnoreCase() method and displaying the result
System.out.println(obj.compareToIgnoreCase(string1, string2));
System.out.println(obj.compareToIgnoreCase(string1, string3));
System.out.println(obj.compareToIgnoreCase(string1, string4));
System.out.println(obj.compareToIgnoreCase(string1, string5));
}
}
```

Output:

```
0
-10
1
10
```

Points to Remember

- The following are some important points to remember while working with the method *compareToIgnoreCase()*.

1. For maintaining the case insensitiveness, all the alphabets of both the strings are converted into lowercase letters (not in uppercase). The following example will make things clearer.

FileName: CompareToIgnoreCase2.java

```
public class CompareToIgnoreCase2
{

// main method
public static void main(String argvs[])
{
String string1 = "6";
String string2 = "Book";

System.out.println(string1.compareToIgnoreCase(string2));

}
}
```

Output:

```
-44
```

Explanation:

- The Unicode value of '6' is 54 and of 'B' is 66. So, the answer should be 54 - 66 = -12. However, we are getting -44. It is because the string "Book" is getting converted into "book", and the Unicode value of 'b' is 98. Thus, 54 - 98 is -44, which is shown in the output. It shows that the alphabets of the strings are converted into lowercase letters, not into uppercase letters. It is because of this reason our custom compareToIgnoreCase() method uses the toLoweCase() method instead of toUpperCase() method.
- In the method compareToIgnoreCase(), the comparison stops immediately when the first mismatch is found. It is evident when we look at the first program of this section, where "Book" is compared with "abc". Here, the first characters of each string are compared, and we get a mismatch. Thus, the rest of the characters of both the strings are ignored, and the comparison loop stops there to return the appropriate results.
- When there are some extra characters present in any one of the strings, and the remaining characters match with the other string, then only the count of the extra characters is returned. Observe the following example.

FileName: CompareToIgnoreCase3.java

```
public class CompareToIgnoreCase3
{

// main method
public static void main(String argvs[])
{
String string1 = "Booksss";
String string2 = "Book";

System.out.println(string1.compareToIgnoreCase(string2));

}
}
```

Output:

```
3
```

Explanation:

- Here, string *string1* has some extra letters in the end. The comparison happens till the character 'k'; after that, there is no letter in the string *string2.*
- For the substring "sss", it is not possible to make a comparison. Therefore, only the numbers of extra letters are counted and returned.
- Hence, the output is 3. Note that instead of string string1, if there is some extra letters in the string string2, then the output will in minus. For example: *"book".CompareToIgnoreCase ("Booksss");* results in -3.

Trimorphic Numbers in Java

- In previous section, we have discussed many numbers programs.
- that are usually asked in interviews. In this section, we are going to discuss what is trimorphic number and how to check if the number is trimorphic or not.

Trimorphic Number

- A number T is said to be **trimorphic** if the cube of T (T*T*T or T^3) terminates with T. Let's understand it with the help of examples.

Example 1:

Input:

T = 501

Output:

501 is a trimorphic number.

Explanation:
If we multiply 501 by itself 3 times (501 x 501 x 501), we get 125751501, and 125751501 ends with the number 501. Hence, **501** is the trimorphic number.

Example 2:

Input:

T = 7

Output:

7 is not a trimorphic number.

Explanation:
If we multiply 7 by itself 3 times (7 x 7 x 7), we get 343, and 343 does not end with the number 7. Hence, 7 is not the trimorphic number.

Algorithm

Step1: Take a number T and find its cube. Let's assume that $T^3 = Y$.

Step 2: Find the number of digits present in the number T. Let's assume there is n number of digits present in the number T.

Step 3: Find the value of 10^n. Assume that the value of 10^n is p.

Step 4: Compute the value of Y - T. Let's say Y - T = r

Step 5: Check whether the value of r % p is equal to 0 or not. If the value is 0, then T is a trimorphic number; otherwise, not.

Trimorphic Number Java Program

Let's implement the above algorithm in a Java program.

FileName: TrimorphicNumber.java

```
public class TrimorphicNumber
{
ring your opinions?
 // a method that finds the total number of
 // digits present in the number N
 public int findDigCount(int N)
 {
    int countDig = 0;

    while(N != 0)
    {
       N = N / 10;
       countDig = countDig + 1;
    }


    return countDig;
 }
```

```
// a method that finds the value of b ^ e
public int findPow(int b, int e)
{
   int ans = 1;

   while(true)
   {
      if((e % 2) == 1)
      {
         ans = ans * b;
         e = e - 1;
      }

      if(e == 0)
      {
         return ans;
      }

      b = b * b;
      e = e / 2;
   }
```

```
}

// a method that checks whether the number T is trimorphic or not
public boolean checkTrimorphic(int T)
{
    // step 1
    int Y = T * T * T;

    // step 2
    int n = findDigCount(T);

    // step 3
    int p = findPow(10, n);

    // step 4
    int r = Y - T;

    // step 5
    return (r % p) == 0;
}

// main method
public static void main(String argvs[])
{
// creating an object of the class TrimorphicNumber
TrimorphicNumber obj = new TrimorphicNumber();

for(int i = 1; i <= 10; i++)
{
boolean isTrimorphic = obj.checkTrimorphic(i);
if(isTrimorphic)
{
System.out.println(i + " is a trimorphic number.");
}
else
{
System.out.println(i + " is not a trimorphic number.");
}
}
}
}
```

Output:

```
1 is a trimorphic number.
2 is not a trimorphic number.
3 is not a trimorphic number.
4 is a trimorphic number.
5 is a trimorphic number.
6 is a trimorphic number.
7 is not a trimorphic number.
8 is not a trimorphic number.
9 is a trimorphic number.
10 is not a trimorphic number.
```

- In the above program, we have to do a few things separately.
- For example, we have to compute the power.
- Also, the total number of digits present in the number T.
- If we use string, we can avoid these mentioned operations. Let's find it out in the following program.

FileName: TrimorphicNumber1.java

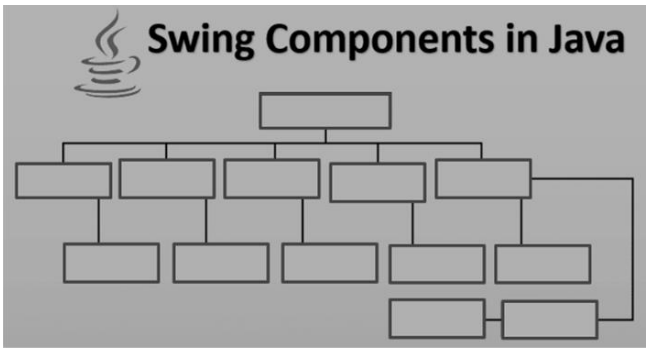
```
public class TrimorphicNumber1
{
// a method that checks whether the number T is trimorphic or not
public boolean checkTrimorphic(int T)
{
   int Y = T * T * T;
   String T1 = String.valueOf(T);
   String Y1 = String.valueOf(Y);
   int size1 = T1.length();
   int size2 = Y1.length();
   for(int i = size1 - 1, j = size2 - 1; i >= 0 && j >= 0;)
   {
      char a = T1.charAt(i);
      char b = Y1.charAt(j);
      if(a != b)
      {
         return false;
      }
      i = i - 1;
      j = j - 1;
   }
   return true;
}
// main method
public static void main(String argvs[])
{
// creating an object of the class TrimorphicNumber1
TrimorphicNumber1 obj = new TrimorphicNumber1();
for(int i = 1; i <= 10; i++)
{
boolean isTrimorphic = obj.checkTrimorphic(i);
if(isTrimorphic)
{
System.out.println(i + " is a trimorphic number.");
}
else
{
System.out.println(i + " is not a trimorphic number.");
}
}
}
}
```

Output:

```
1 is a trimorphic number.
2 is not a trimorphic number.
3 is not a trimorphic number.
4 is a trimorphic number.
5 is a trimorphic number.
6 is a trimorphic number.
7 is not a trimorphic number.
8 is not a trimorphic number.
9 is a trimorphic number.
10 is not a trimorphic number.
```

Section 3: Exercises

Exercise 1: Identify all Swing Components in Java.



Exercise 2: Participate in a group discussion on following topics:

a) Hierarchy of swing components
b) Top level containers - JFrame, JDialog, JPanel, JButton, JToggleButton
c) Life cycle of an Applet
d) Difference between Applets and Applications

Section 4: Assessment Questionnaire

1. Give the abbreviation of AWT?
   a) Applet Windowing Toolkit
   b) Abstract Windowing Toolkit
   c) Absolute Windowing Toolkit
   d) None of the above

2. Where are the following four methods commonly used?
   1. public void add(Component c)
   2. public void setSize(int width,int height)
   3. public void setLayout(LayoutManager m)
   4. public void setVisible(boolean)

   a) Graphics class
   b) Component class
   c) Both A & B
   d) None of the above

3. Which is the container that doesn't contain title bar and MenuBars but it can have other components like button, textfield etc?
   a) Window
   b) Frame
   c) Panel
   d) Container

4. These two ways are used to create a Frame
    1. By creating the object of Frame class (association)
    2. By extending Frame class (inheritance)
    a) True
    b) False

5. Package of drawstring() method is in? Options are:
    a) java.applet
    b) java.io
    c) java.awt
    d) javax.swing

6. The Java Foundation Classes (JFC) is a set of GUI components which simplify the development of desktop applications.
    a) True
    b) False

7. In Graphics class which method is used to draws a rectangle with the specified width and height?
    a) public void drawRect(int x, int y, int width, int height)
    b) public abstract void fillRect(int x, int y, int width, int height)
    c) public abstract void drawLine(int x1, int y1, int x2, int y2)
    d) public abstract void drawOval(int x, int y, int width, int height)
8. Which object can be constructed to show any number of choices in the visible window?
    a) Labels
    b) Choice
    c) List
    d) Checkbox

9. Which method is used to set the graphics current color to the specified color in the graphics class?
    a) public abstract void setFont(Font font)
    b) public abstract void setColor(Color c)
    c) public abstract void drawString(String str, int x, int y)
    d) None of the above

10. Which is used to store data and partial results, as well as to perform dynamic linking, return values for methods, and dispatch exceptions?
    a) Window
    b) Panel
    c) Frame
    d) Container

11. Which of these methods can be used to know which key is pressed? Options are:
    a) getActionEvent()
    b) getActionKey()
    c) getModifier()
    d) getKey()

12. Which of the following statements about GUI components is wrong? Options are:
    a) Swing exists since version 1.2 of the jdk
    b) You cannot place AWT components on Swing containers
    c) AWT stands for Abstract Window Toolkit
    d) The AWT classes are deprecated

13. Which of these packages contains all the classes and methods required for even handling in Java? Options are:
    a) java.applet
    b) java.awt
    c) java.awt.event
    d) java.event

14. Implement the Listener interface and overrides its methods is required to perform in event handling.
    a) True
    b) False

15. The following specifies the advantages of:
    1. It is lightweight.
    2. It supports pluggable look and feel.
    3. It follows MVC (Model View Controller) architecture.
    a) Swing
    b) AWT
    c) Both A & B
    d) None of the above

16. Which class provides many methods for graphics programming?
    a) java.awt
    b) java.Graphics
    c) java.awt.Graphics
    d) None of the above

17. To use the ActionListener interface it must be implemented by a class there are several ways to do that find in the following?
    a) Creating a new class
    b) using the class the graphical component
    c) an anonymous inner class
    d) All mentioned above

18. The ActionListener interface is used for handling action events, For example,it's used by a:
    a) JButton
    b) JCheckbox
    c) JMenuItem
    d) All of these

19. The following steps are required to perform:
    1. Implement the Listener interface and overrides its methods
    2. Register the component with the Listener
    a) Exception Handling
    b) String Handling
    c) Event Handling
    d) None of the above

20. Which is a component in AWT that can contain another component like buttons, textfields, labels etc.?
    a) Window
    b) Container
    c) Panel
    d) Frame
21. What is JFC?
22. What is AWT?
23. What are the differences between Swing and AWT?
24. What are heavy weight components?
25. What is lightweight component?
26. What is double buffering?
27. What is an event in Swing?
28. What is an event handler in swing?
29. What is a layout manager?
30. What is the difference between applications and applets?

----------End of the Module----------

MODULE 6
IMPLEMENTING PAAS (PLATFORM AS A SERVICE)

Section 1: Learning Outcomes

After completing this module, you will be able to:

- Explain the concept of Platform as a Service (PaaS)
- State Pros and Cons of PaaS
- Define PaaS Architecture
- Describe PaaS and Its Services
- Explain the PaaS Monitoring
- Tell the Benefits of Cloud Monitoring

Section 2: Relevant Knowledge

6.1 Exploring the Technical Foundation for PaaS

PaaS: Platform as a Service

It Stands for **"Platform as a Service"**

- Programming Language + OS + Server + Database
- Provides Encapsulation
- Build, Compile & Run Programs
- Users Manage Data & Application Resources

Who Use it ?:- Developers

Pros & Cons of PaaS

Pros	*Cons*
• Scalable & Cost Effective • Faster Market for Developers • Easy Development for Web Application • Private & Public Deployment	• Provider Language Only • Developers Limited • Migration Issues • Vendor Lock-in

PaaS Market Size, Share, and Leading Vendors

- The PaaS market's reported size and how it compares to other cloud services depend on the source.
- For example, according to Gartner, PaaS will be dwarfed by IaaS in 2021, with $27.5 billion vs. $61.9 billion in revenue, respectively.

Worldwide Public Cloud Service Revenue Forecast (Billions of U.S. Dollars)

	2018	2019	2020	2021	2022
Cloud Business Process Services (BPaaS)	45.8	49.3	53.1	57.0	61.1
Cloud Application Infrastructure Services (PaaS)	15.6	19.0	23.0	27.5	31.8
Cloud Application Services (SaaS)	80.0	94.8	110.5	126.7	143.7
Cloud Management and Security Services	10.5	12.2	14.1	16.0	17.9
Cloud System Infrastructure Services (IaaS)	30.5	38.9	49.1	61.9	76.6
Total Market	**182.4**	**214.3**	**249.8**	**289.1**	**331.2**

BPaaS = business process as a service; IaaS = infrastructure as a service; PaaS = platform as a service; SaaS = software as a service
Note: Totals may not add up due to rounding.

PaaS- Delivery Ways

PaaS can be delivered in three ways:

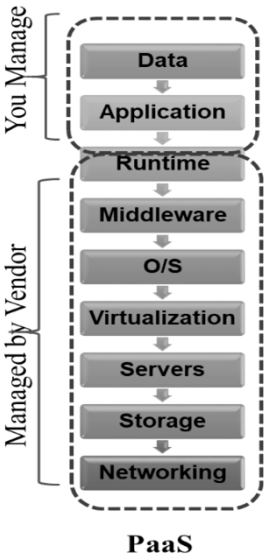
- As a public cloud service from a provider, where the consumer controls software deployment with minimal configuration options, and the provider provides the networks, servers, storage, operating system (OS), middleware (e.g. Java runtime, .NET runtime, integration, etc.), database and other services to host the consumer's application.
- As a private service (software or appliance) behind a firewall.
- As software deployed on public infrastructure as a service.

How does PaaS work?

- PaaS does not replace a company's entire IT infrastructure for software development. It is provided through a cloud service provider's hosted infrastructure.
- Users most frequently access the offerings through a web browser.
- PaaS can be delivered through public, private and hybrid clouds to deliver services such as application hosting and Java development.
- Other PaaS services include the following:
  - Development team collaboration
  - Application design and development
  - Application testing and deployment
  - Web service integration
  - Information security
  - Database integration
- Users will normally have to pay for PaaS on a per-use basis. However, some providers charge a flat monthly fee for access to the platform and its applications.

PaaS Architecture

- PaaS enables developers to develop, test, and deploy in the same environment. A typical PaaS architecture consists of the following categories:
  - Integration and Middleware: It refers to the software that offers runtime services.
  - API: It implies Application Platform Interface, which acts as a communication between client and server that offers abstraction (running the details in the background) and core connectivity.
  - Hardware: It comprises of all hard requirements to handle the resources.
- This facilitates and allows the users to build and run applications without the complexity of constructing and maintaining the infrastructure as the PaaS architecture covers the requirements.



Understanding PaaS with Types of Services

PaaS leads to faster development as there is no need for the user to worry about setting and maintaining the infrastructure. PaaS services are available in 3 types:

- Public
- Private
- Hybrid

What Services Does PaaS Include?

Although the most common use case of PaaS is web app deployment, many other cloud services also fall under it.

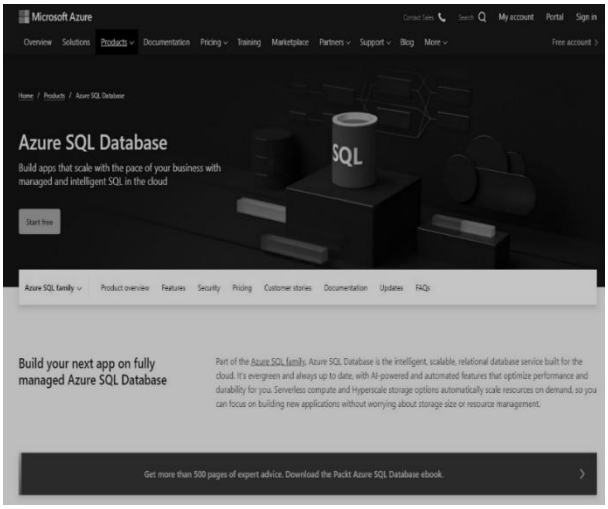
- Database as a Service (DBaaS)
- Internet of Things (IoT) Platforms
- Mobile Services (APIs)
- Push Notification APIs
- Machine Learning
- Hadoop, Spark, & Other Data Processing Frameworks

Database as a Service (DBaaS)

- A cloud-hosted database that you manually install on a virtual machine is only an implementation of IaaS.

- To be considered a PaaS offering, it needs to be an integrated solution that offers storage, computing power, and relational database capabilities.

- An example of this is the Azure SQL Database service, which offers a fully managed database with automated updates, scalability, smart threat protection, and AI-powered search.



Internet of Things (IoT) Platforms

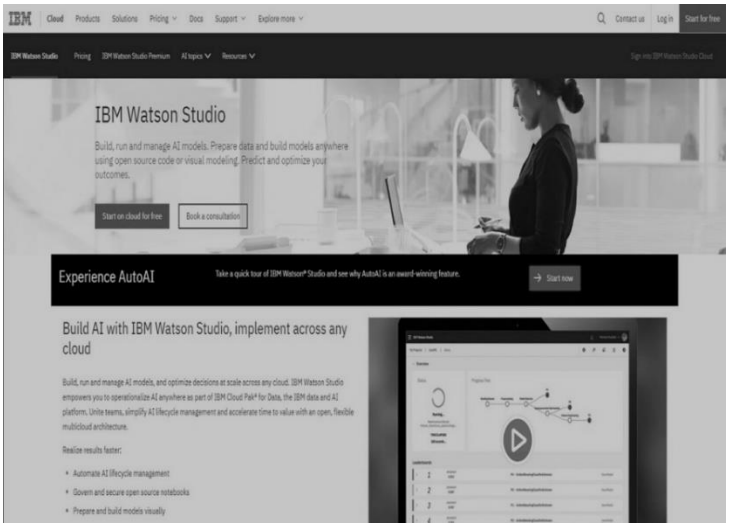
- More items are powered by computers and connected to the internet than ever before.
- The new HTTP/3 standard will only accelerate that further.
- Connected devices now include lights, thermostats, ovens, washing machines, locks, and even truck engines.
- The bare bones of connectivity to the internet could be considered IaaS, but complex APIs for controlling and sharing data across devices and apps fall under PaaS.

Mobile Services (APIs)

- Companies are no longer settling for email when sending notifications and marketing campaigns to their customers.
- They also use automated SMS messages at scale.
- With SMS APIs, companies can build automated messages into their applications.
- For example, they can text customers to:
  - Remind them of scheduled calls or meetings.
  - Promote a new related product or service.
  - Ask for feedback on a recent customer service encounter.
  - Recruit customers to join a case study or survey.
  - These services are sometimes categorized separately as Communications Platform as a Service (CPaaS), a PaaS subcategory.

Machine Learning

- If you genuinely want to take advantage of your data, it's not enough to just store it in the cloud. The data is still just sitting around, only in a new location.
- You need to set up algorithms to sift through your data and find meaningful insights and actionable steps.
- With cloud-based machine learning platforms, you can easily create models (from templates), apply them to your databases, and scale your computing power as needed.



Public PaaS

- Public Platform as a Service runs on the public cloud the user have to focus on building application.
- It helps developers to be more agile, which helps them to develop and deliver faster. And the vendor manages and maintains the infrastructure.

Private PaaS

- A private Platform as a Service is a good choice for companies that wish to maintain some of their own hardware.
- It's also a good alternative for companies who wish to maintain part of their information, in some cases sensitive, in their own data centers.

Hybrid PaaS

- Hybrid Platform as a Service offers flexibility to choose what percent of the user's infrastructure in his control.
- Private PaaS provides scalability for hybrid PaaS. Well, a hybrid is a combination of a bit of private and public.
- These platforms reduce the time taken to develop and deploy, increase flexibility, help users achieve performance and better results, and maintain control over the cost.

Serverless vs PaaS

- Both serverless and PaaS provide the same facilities, as they both are backend architectures that hide the backend from the developers.
- They only differ in scalability, timing, startup time and tools, and deployment process.
- Differences are:
  - The pricing of serverless is exact as it charges developers for the time the application utilizes. On the other hand, PaaS pricing is not as precise as serverless, as PaaS vendors charge a monthly fee for the services offered.
  - PaaS provides more control over the deployment environment, while on the other hand, serverless provides less control over the environment.
  - Serverless applications are active most of the time. The built-in PaaS applications can be up and run quickly, but they are not as lightweight as serverless. Serverless provides agility to its built-in applications makes it more suitable for web applications.
- It is not that serverless services are more affordable.
- It depends on the type of application we are developing and the facilities and services we require.
- We have to choose between PaaS and serverless according to the project requirements.

Common PaaS scenarios

Organisations typically use PaaS for these scenarios:

Development framework

- PaaS provides a framework that developers can build upon to develop or customise cloud-based applications.
- Similar to the way you create an Excel macro, PaaS lets developers create applications using built-in software components.
- Cloud features such as scalability, high-availability and multi-tenant capability are included, reducing the amount of coding that developers must do.

Analytics or business intelligence

Tools provided as a service with PaaS allow organisations to analyse and mine their data, finding insights and patterns and predicting outcomes to improve forecasting, product design decisions, investment returns and other business decisions.

Additional services

PaaS providers may offer other services that enhance applications, such as workflow, directory, security and scheduling.

STL academy

PaaS Feature

- **Programming Models, Languages, and Frameworks**

Programming mod-els made available by IaaS providers define how users can express their applications using higher levels of abstraction and efficiently run them on the cloud platform.

- **Persistence Options**

A persistence layer is essential to allow applications to record their state and recover it in case of crashes, as well as to store user data.

Security, Privacy and Trust

- Security and privacy affect the entire cloud computing stack, since there is a massive use of third-party services and infrastructures that are used to host important data or to perform critical operations.
- In this scenario, the trust toward providers is fundamental to ensure the desired level of privacy for applications hosted in the cloud.
- When data are moved into the Cloud, providers may choose to locate them anywhere on the planet.
- The physical location of data centers determines the set of laws that can be applied to the management of data.

Data Lock-In and Standardization

- The Cloud Computing Interoperability Forum (CCIF) was formed by organizations such as Intel, Sun, and Cisco in order to —enable a global cloud computing ecosystem whereby organizations are able to seamlessly work together for the purposes for wider industry adoption of cloud computing technology.
- The development of the Unified Cloud Interface (UCI) by CCIF aims at creating a standard programmatic point of access to an entire cloud infrastructure.
- In the hardware virtualization sphere, the Open Virtual Format (OVF) aims at facilitating packing and distribution of software to be run on VMs so that virtual appliances can be made portable.

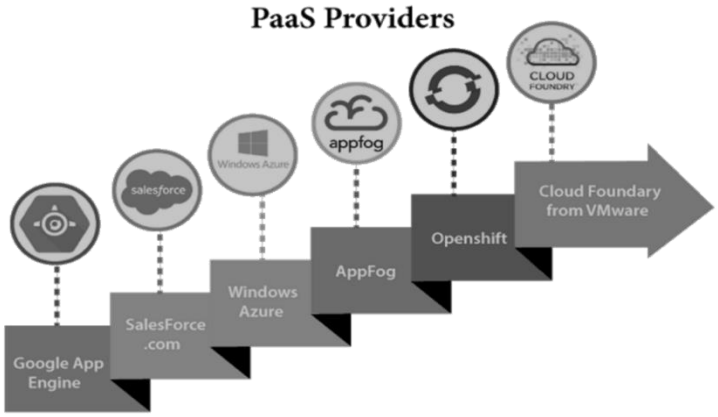
Availability, Fault-Tolerance and Disaster Recovery

- It is expected that users will have certain expectations about the service level to be provided once their applications are moved to the cloud.
- These expectations include availability of the service, its overall performance, and what measures are to be taken when something goes wrong in the system or its components.
- In summary, users seek for a warranty before they can comfortably move their business to the cloud.
- SLAs, which include QoS requirements, must be ideally set up between customers and cloud computing providers to act as warranty.
- An SLA specifies the details of the service to be provided, including availability and performance guarantees.
- Additionally, metrics must be agreed upon by all parties, and penalties for violating the expectations must also be approved.

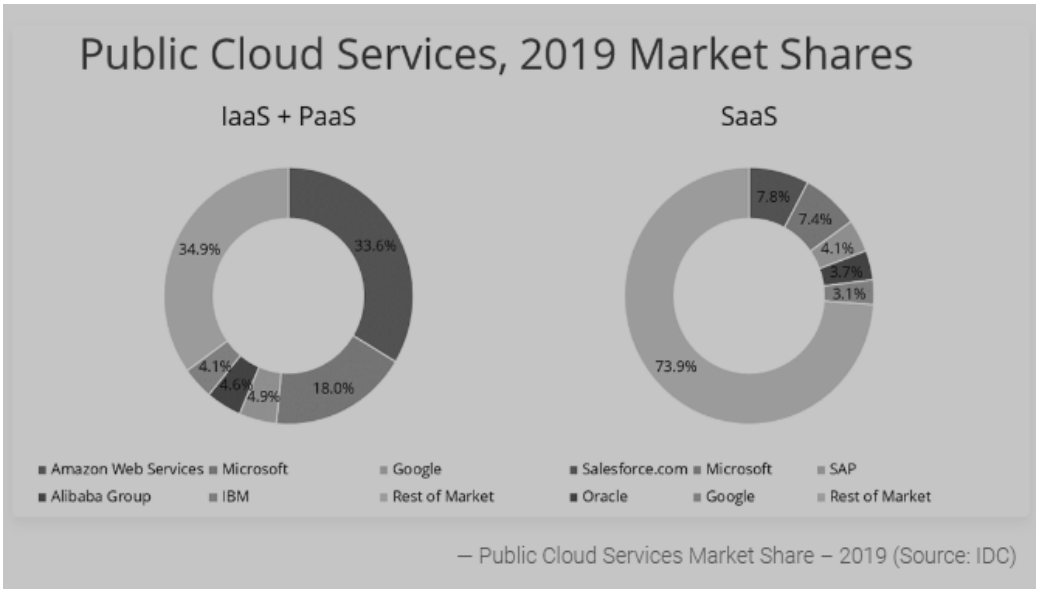
Resource Management and Energy-Efficiency

- The multi-dimensional nature of virtual machines complicates the activity of finding a good mapping of VMs onto available physical hosts while maximizing user utility.
- Dimensions to be considered include:
  - Number of CPUs
  - Amount of memory
  - Size of virtual disks
  - Network bandwidth
- Dynamic VM mapping policies may leverage the ability to suspend, migrate, and resume VMs as an easy way of preempting low-priority allocations in favor of higher-priority ones.
- Migration of VMs also brings additional challenges such as detecting when to initiate a migration, which VM to migrate, and where to migrate.
- In addition, policies may take advantage of live migration of virtual machines to relocate data center load without significantly disrupting running services.

Popular PaaS Providers



Leading Vendors and Their Market Share



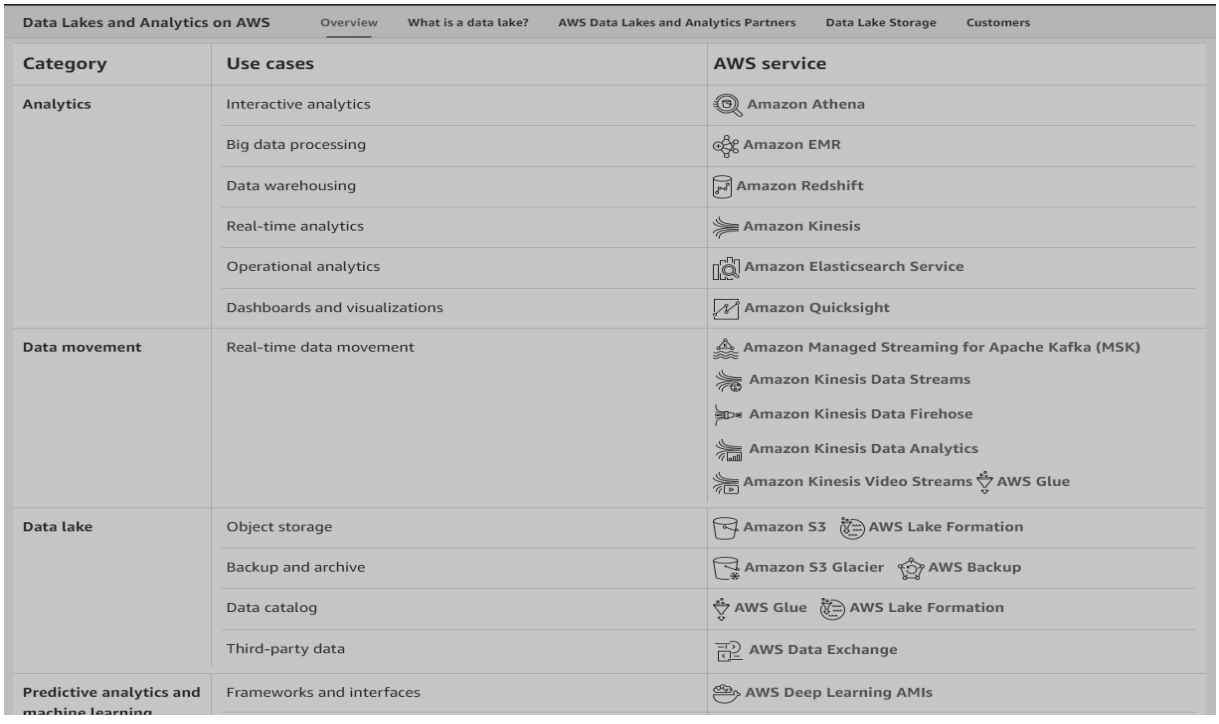
Popular PaaS Providers

Providers	Services
Google App Engine (GAE)	App Identity, URL Fetch, Cloud storage client library, Logservice
Salesforce.com	Faster implementation, Rapid scalability, CRM Services, Sales cloud, Mobile connectivity, Chatter.
Windows Azure	Compute, security, IoT, Data Storage.
AppFog	Justcloud.com, SkyDrive, GoogleDocs
Openshift	RedHat, Microsoft Azure.
Cloud Foundry from VMware	Data, Messaging, and other services.

The 4 Leading PaaS Providers:

AWS

- AWS is the original cloud computing provider, having launched the revolution with its primary EC2 product in 2006.
- The head start cemented them as the clear market leader, and it's still the largest cloud services company in the world.But for PaaS specifically, what does it bring to the table?
- A quick look at Amazon's services overview will tell you everything you need to know.

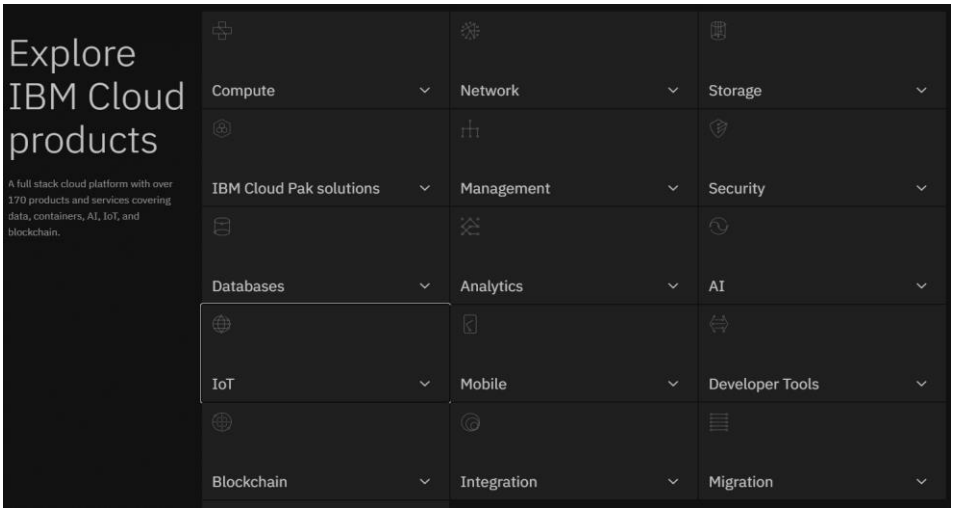


Data Lakes and Analytics on AWS — Overview | What is a data lake? | AWS Data Lakes and Analytics Partners | Data Lake Storage | Customers

Category	Use cases	AWS service
Analytics	Interactive analytics	Amazon Athena
	Big data processing	Amazon EMR
	Data warehousing	Amazon Redshift
	Real-time analytics	Amazon Kinesis
	Operational analytics	Amazon Elasticsearch Service
	Dashboards and visualizations	Amazon Quicksight
Data movement	Real-time data movement	Amazon Managed Streaming for Apache Kafka (MSK) Amazon Kinesis Data Streams Amazon Kinesis Data Firehose Amazon Kinesis Data Analytics Amazon Kinesis Video Streams AWS Glue
Data lake	Object storage	Amazon S3 AWS Lake Formation
	Backup and archive	Amazon S3 Glacier AWS Backup
	Data catalog	AWS Glue AWS Lake Formation
	Third-party data	AWS Data Exchange
Predictive analytics and machine learning	Frameworks and interfaces	AWS Deep Learning AMIs

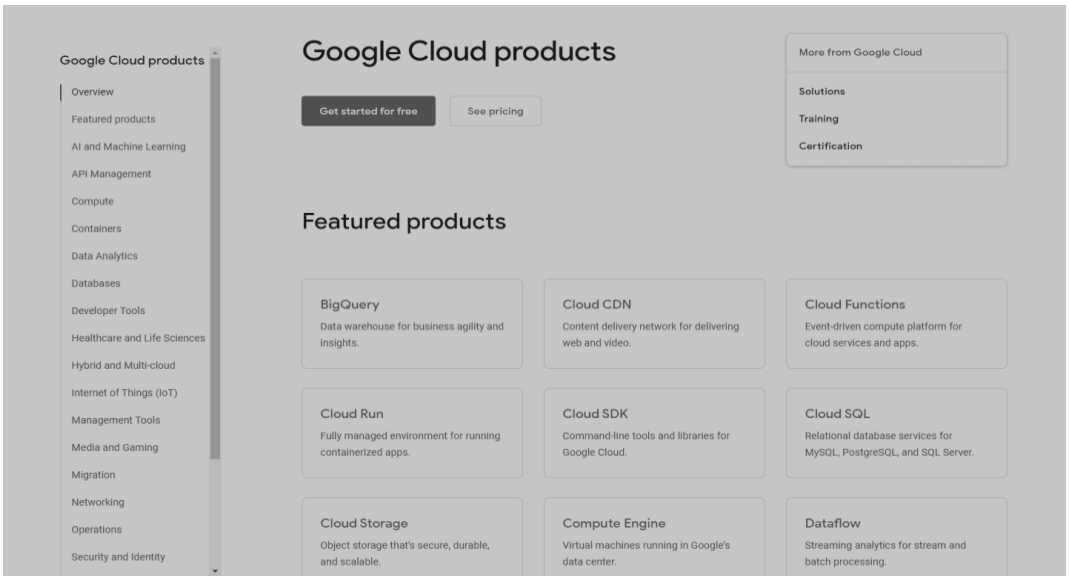
IBM Cloud

- An early innovator in computing, IBM has put a lot of money and effort into developing its cloud services suite.
- IBM first launched its PaaS services as IBM Bluemix in 2014.
- In 2017, IBM dropped the Bluemix brand and grouped its PaaS, IaaS, and private cloud offerings under the IBM Cloud umbrella.
- With a wide range of enterprise clients, IBM Cloud has quickly grown to become one of the leading PaaS providers since its launch in 2011and that shows in its range of services:



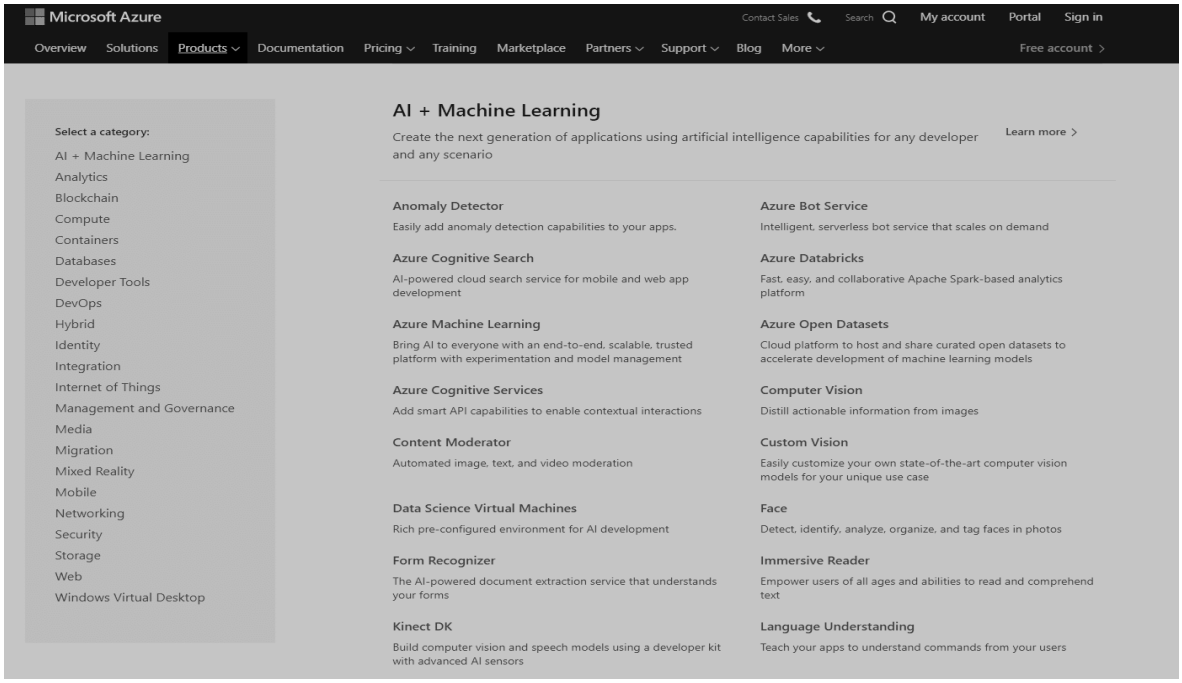
Google Cloud

- Google isn't just a search engine. It's also one of the leading SaaS companies, with Google Docs, Drive, Gmail, and the entire Google Workspace.
- Google also lets you rent the infrastructure and platforms that make it possible to handle billions of visitors every month.
- Launched in 2008, Google Cloud was the second major player to enter the market. Its extensive list of products shows why it's still one of the market leaders.



Microsoft Azure

- Microsoft isn't just responsible for the operating systems on most desktop and laptop computers around the world.
- It also has one of the largest public cloud services collections, including Office 365, Microsoft Teams (SaaS), and Azure (IaaS & PaaS).
- The Azure cloud platform includes a range of services from AI and machine learning to analytics, development tools, data processing, and more.



6.2 Selecting an Appropriate Implementation

Development framework

PaaS gives a framework that developers can customize cloud-based applications

Analytics or business intelligence

- Tools provided as a service with PaaS allow organizations to analyze and mine their data, product design decisions and other business decisions.

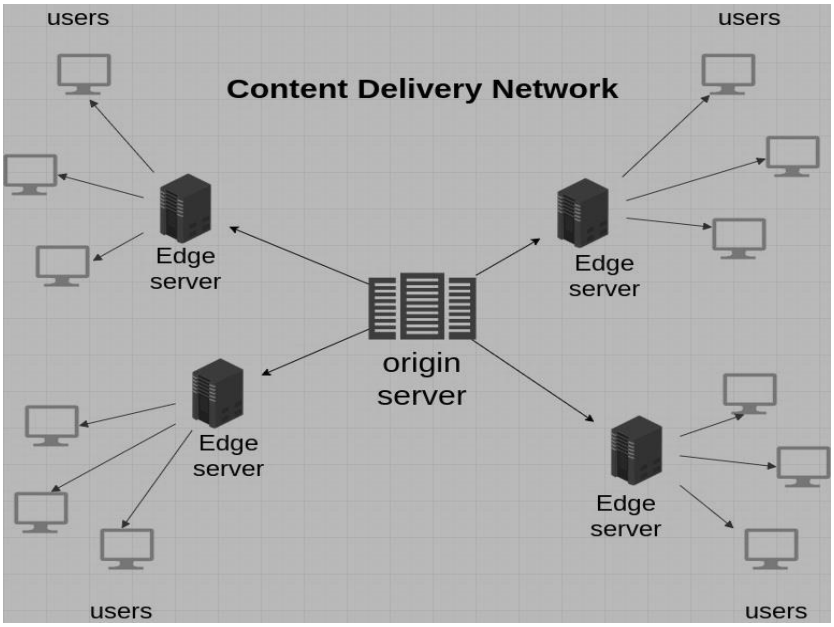
Additional services

- PaaS providers may offer other services that enhance applications, such as directory, workflow, security and scheduling.

6.3 Managing Cloud Storage

Building Content Delivery Networks Using Clouds

- Numerous “storage cloud” providers (or “Storage as a Service”) have recently emerged that can provide Internet-enabled content storage and delivery capabilities in several continents, offering service-level agreement (SLA)- backed performance and uptime promises for their services.
- Customers are charged only for their utilization of storage and transfer of content (i.e., a utility computing model), which is typically on the order of cents per gigabyte.



Following are the Popular Content Delivery Network Provider

- Microsoft Azure
- Amazon S3 and CloudFront
- Nirvanix SDN
- Rackspace Cloud Files

- This represents a large paradigm shift away from typical hosting arrangements that were prevalent in the past, where average customers were locked into hosting contracts (with set monthly/yearly fees and excess data charges) on shared hosting services like DreamHost.

- Larger enterprise customers typically utilized pervasive and high-performing Content Delivery Networks (CDNs), who operate extensive networks of “edge” servers that deliver content across the globe.

6.4 Controlling Unstructured Data in the Cloud

- Unstructured data is harder to analyse and process than structured data, which is why it often goes unused.
- But cloud computing and AI tools equipped with machine learning are introducing new ways to manage this data, which contains a myriad of valuable customer insights.

What Is Unstructured Data Management?

- Unstructured data management is the process of collecting, storing, organizing, and analysing data that doesn't have any predefined structure.
- While structured data management can easily be performed using everyday tools like Excel, Google Sheets, and relational databases, unstructured data management, requires more advanced tools, complex rules, and techniques to transform it into quantifiable data.
- Companies handle large amounts of unstructured data from different sources every day: market data, customer feedback, in-app reviews, social media, and so on.
- All this data contains valuable information that helps companies to make informed decisions and shape data-driven strategies, improve processes and products, reduce costs, and gain a competitive advantage.

Challenges of Managing Unstructured Data

Businesses often face a series of challenges that might be holding them back from managing unstructured data:

- **Data quality**: Unstructured data often needs to be cleaned before it can be organized. Duplicate, outdated, unreliable, or inaccurate data that contains outliers, can lead to poor quality data that will skew results when performing unstructured data analysis. It can be challenging for businesses to clean and prepare huge amounts of data, but it's a critical step if you want to get the most out of your data.
- **Siloed data**: Each team collects their own data, stored in different systems and formats. However, data should be stored in one place that's accessible to everyone making data retrieval quick and painless. Businesses will need to spend time routing their data before putting new processes in place, but they might not have the resources to do so.
- **Data Growth & Costs**: As your unstructured data grows, you'll need somewhere to store it, adding to the already increasing costs involved in data management. However, you can ensure that data is compressed (and remove any duplicates) to minimize the amount of space you use, helping you keep costs down and manage data in the most efficient way possible.

Why Should You Manage Unstructured Data?

Having a solid data management strategy to collect, organize, and analyze unstructured data can help overcome the above challenges, and eventually lead to:

- **Increased productivity:** Employees know where to find data when they need it because it's all in one place and it's easy to search. If you're using machine learning tools to manage your data, you can even speed up internal processes and reduce response times.
- **Accurate & fast decisions:** High-quality data is reliable and drives better decision-making. Using tools to analyze unstructured data in real-time allows you to detect urgent issues and act quickly. Also, uncovering trends in large datasets helps you anticipate market shifts.
- **Better compliance:** Ensuring your data is organized and always up-to-date makes it easier to keep up with current regulations and standards and avoid any kind of legal trouble.
- **Improved data security:** Data breaches and cyber-attacks are a common threat to organizations. Effective data management helps you keep your data safe, create backups, and monitor in real-time to identify potential risks.

In short, knowing how to manage your data effectively can help you extract more value from unstructured data and translate this value into opportunity.

How to Manage Unstructured Data?

- Unstructured data(or qualitative data) is a highly valuable business asset. Product reviews, brand mentions, open-ended survey responses, and general feedback about your support team, all provide valuable customer insights.
- But customers' opinions are not always easy to manage and understand because they're unstructured.
- Manually analyzing unstructured text is time-consuming (hence, expensive), and prone to human error and bias. Plus, it doesn't scale. As your business grows, along with your data, you'll need tools that help you cut through the noise and find what's relevant.
  So, how can you get started?

There are **four steps** you'll need to follow to manage unstructured data:

1. Make Content Accessible, Organized, and Searchable

- First, you'll need space to store unstructured data.
- Public cloud-based storage is the obvious option because it's easy for everyone to access, and enables remote collaboration. Plus, it's scalable and cost-effective: if you need more space, you can always upgrade to a higher tier. Amazon Simple Storage Service (Amazon S3), Google Cloud, and Azure Data Lake Storage are the main cloud storage providers for big data.
- The alternative to storing data in the cloud is to invest in on-premise storage hardware, like servers or external drives. Some businesses prefer to store their critical data in-house due to security concerns or data protection regulations.
- While on-premise storage offers full control over your data, it also involves higher costs like IT support, maintenance, and security infrastructure.
- You can also opt for hybrid data storage, which combines on-premise and cloud-based storage.
- To decide which storage option is best for your business, you'll need to consider the type of data that you handle (highly-regulated industries like finance or healthcare may require more security), your budget, and other requirements (for example, a start-up that's planning to grow quickly may prefer a flexible cloud-based solution).
- When storing your data, make sure that it's easy to search and filter, so you can explore datasets using keywords and quickly identify what you need. Adding metadata to files and documents is essential for summarizing content and making it searchable.
- If you change your data storage, you'll need to migrate data, which involves careful planning. Plus, you may also need to convert data from one format to another.

2. Clean your Unstructured Data

- Unstructured datasets are very noisy. They often contain spelling mistakes, HTML tags, punctuation marks, hashtags, special characters, and so on.
- To improve the quality of your datasets you need to preprocess data, also known as 'data cleaning'. This step must be completed before performing any kind of text analysis.

3. Analyze Unstructured Data with AI Tools

- Once you've stored, organized, and cleaned unstructured data, the next step is to analyze it. Using AI-powered text analysis tools, like MonkeyLearn Studio, is the most effective way to transform text data into valuable insights.
- Text analysis tools combine machine learning and Natural Language Processing (NLP) to understand and process text data at scale.
- These tools work by automatically classifying text by topic, sentiment, intent, and more and extracting specific information like keywords or named entities.
- For example, if you need to manage your Twitter mentions, you could use a sentiment analyzer to analyze sentiments in real time and quickly identify urgent issues.

4. Visualize your Data

- Compelling data visualizations help summarize unstructured data. Through charts, reports, or interactive dashboards, you can transform boring spreadsheets into clear and actionable information to share with your teammates.

6.5 Improving Data Availability

- Understanding the importance of data availability is easy enough. Achieving high data availability, however, is harder.
- If you're wondering how, you can improve data availability, keep reading. This post discusses some data availability best practices.

➢ **Embrace redundancy**

- Perhaps the most basic step you can take to improve data availability is to ensure that your data is redundant – or, in other words, that you have multiple sources of the data available. That way, a failure in one of the disks, servers or databases that hosts your data will not lead to a disruption in availability.
- The challenge with redundancy is striking the right balance between redundancy and cost-efficiency. In a world where budgets are a thing, the number of copies of a database or data server that you can have running at the same time is limited.
- However, by studying data such as how often a given server or database fails and assessing how important different data workloads are, you can make an informed decision about how much redundancy to implement for each data source.

➢ **Automate failover**

- Data redundancy is great, but data redundancy combined with automated failover is even greater.
- That is because automated failover (as the term implies) means that when a component of your infrastructure fails, a backup component automatically replaces it.
- By eliminating the need to wait for a human engineer to detect a failure and switch to a backup system, Automated failover minimizes or completely avoids disruption to data availability.
- Many monitoring and management tools for virtual servers and databases make it possible to configure automated failover. And if yours doesn't, some simple scripting should suffice for ensuring that backup systems come online automatically when a primary system fails.

➢ **Avoid single points of failure**

- Another simple step that you can take to improve data availability is to avoid single points of failure – meaning infrastructure components or applications that would cause your data to become unavailable if they stop working correctly.
- The concept here is similar to the point about redundancy made above, but there are differences between redundancy and eliminating single points of failure.
- You can have a redundant storage infrastructure composed of multiple servers and disks but still be at risk of having them become unavailable if, for example, the network router on which they depend crashes. In that case, your router would be your single point of failure. A high availability architecture would avoid that risk by ensuring that not all data passes through a single router.

6.6 Deploying Relational Databases in the Cloud

- A relational database is a collection of information that organizes data in predefined relationships where data is stored in one or more tables (or "relations") of columns and rows, making it easy to see and understand how different data structures relate to each other.
- Relationships are a logical connection between different tables, established on the basis of interaction among these tables.
- Relational databases are among the most popular types of databases because they allow users to easily understand data points that are related to each other in one or more tables and provide relational operators such as structured query language (SQL) to query and manipulate the data. As a result, they are often used for a well-structured data model or for transactional queries in which the data structure does not change often.
- Learn how Google Cloud's relational databases Cloud SQL, Cloud Spanner and AlloyDB for PostgreSQL can help you reduce operational costs and help you build transformative applications.

Relational Database defined

A relational database (RDB) is a way of structuring information in tables, rows, and columns. An RDB has the ability to establish links—or relationships–between information by joining tables, which makes it easy to understand and gain insights about the relationship between various data points.

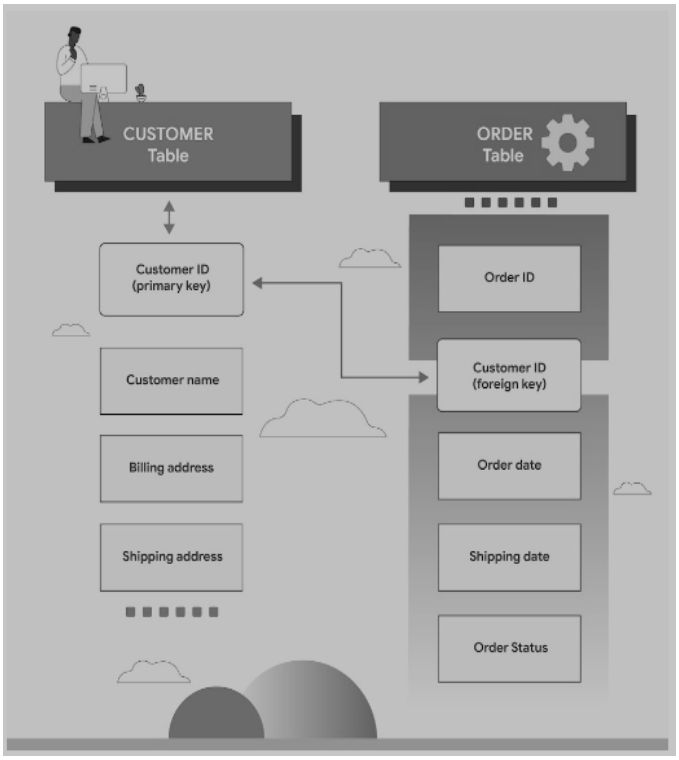
STL academy

The Relational Database Model

- Developed by E.F. Codd from IBM in the 1970s, the relational database model allows any table to be related to another table using a common attribute. Instead of using hierarchical structures to organize data, Codd proposed a shift to using a data model where data is stored, accessed, and related in tables without reorganizing the tables that contain them.
- Think of the relational database as a collection of spreadsheet files that help businesses organize, manage, and relate data. In the relational database model, each “spreadsheet” is a table that stores information, represented as columns (attributes) and rows (records or *tuples*).
- Attributes (columns) specify a data type, and each record (or row) contains the value of that specific data type. All tables in a relational database have an attribute known as the **primary key**, which is a unique identifier of a row, and each row can be used to create a relationship between different tables using a foreign key—a reference to a primary key of another existing table.

Let’s take a look at how the relational database model works in practice:

Say you have a **customer** table and an **Order** table.



- The Customer table contains data about the customer:
  - Customer ID (primary key)
  - Customer name
  - Billing address
  - Shipping address
- In the **Customer** table, the customer ID is a primary key that uniquely identifies who the customer is in the relational database. No other customer would have the same Customer ID.
- The **Order** table contains transactional information about an order:
  - Order ID (primary key)
  - Customer ID (foreign key)
  - Order date
  - Shipping date
  - Order status
- Here, the primary key to identify a specific order is the Order ID. You can connect a customer with an order by using a foreign key to link the customer ID from the **Customer** table.
- The two tables are now related based on the shared customer ID, which means you can query both tables to create formal reports or use the data for other applications. For instance, a retail branch manager could generate a report about all customers who made a purchase on a specific date or figure out which customers had orders that had a delayed delivery date in the last month.
- The above explanation is meant to be simple. But relational databases also excel at showing very complex relationships between data, allowing you to reference data in more tables as long as the data conforms to the predefined relational schema of your database.
- As the data is organized as pre-defined relationships, you can query the data declaratively. A declarative query is a way to define what you want to extract from the system without expressing how the system should compute the result. This is at the heart of a relational system as opposed to other systems.

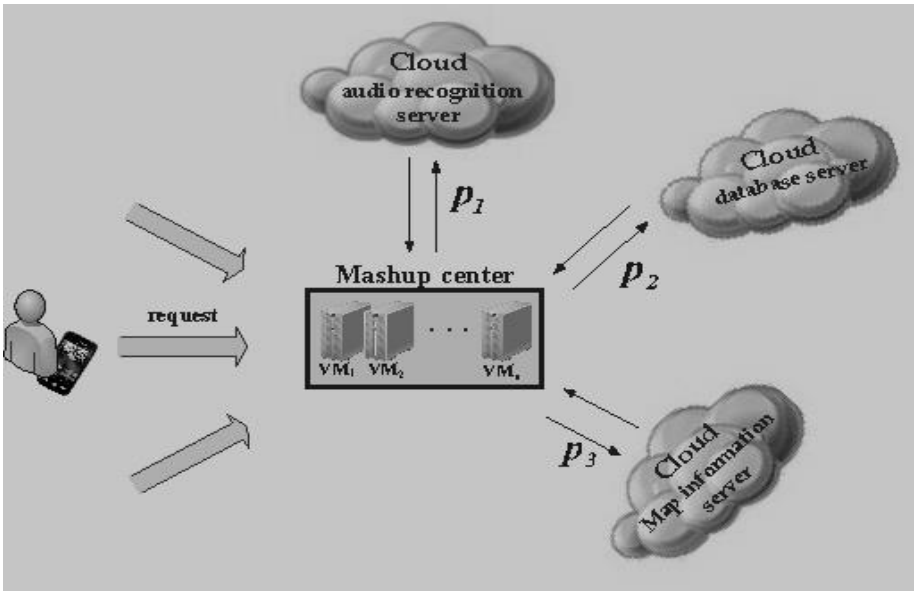
Examples of relational databases

- Now that you understand how relational databases work, you can begin to learn about the many relational database management systems that use the relational database model. A relational database management system (RDBMS) is a program used to create, update, and manage relational databases.
- Some of the most well-known RDBMSs include MySQL, PostgreSQL, MariaDB, Microsoft SQL Server, and Oracle Database.
- Cloud-based relational databases like Cloud SQL, Cloud Spanner and AlloyDB have become increasingly popular as they offer managed services for database maintenance, patching, capacity management, provisioning and infrastructure support.

6.7 Employing Support Services

Resource Cloud Mashups

- Outsourcing computation and/or storage away from the local infrastructure is not a new concept itself: Already the grid and Web service domain presented (and uses) concepts that allow integration of remote resource for seemingly local usage.



Interoperability and Vendor Lock-In

- Since most cloud offerings are proprietary, customers adopting the according services or adapting their respective applications to these environments are implicitly bound to the respective provider.

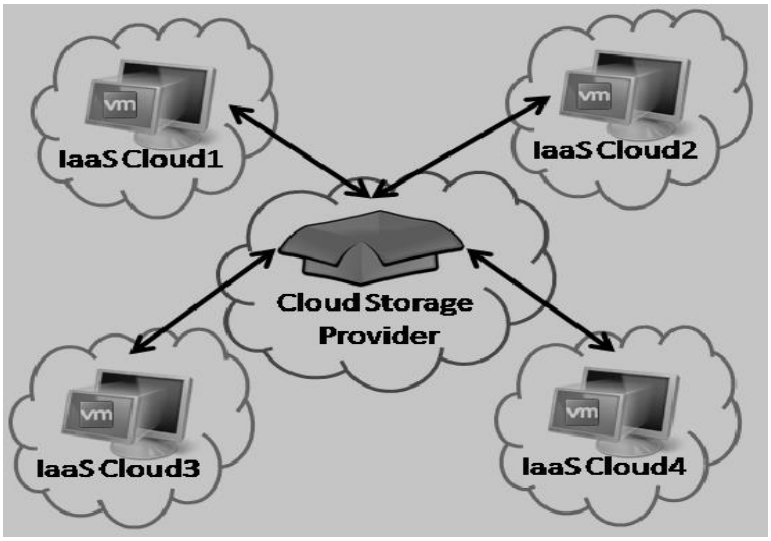
- Movement between providers is restricted by the effort the user wants to vest into porting the capabilities to another environment, implying in most cases reprogramming of the according applications.

- This makes the user dependent not only on the provider's decisions, but also on his/her failures: As the example of the Google crash on the May 14, 2009 showed, relying too much on a specific provider can lead to serious problems with service consumption.

The Problem of Interoperability

The Web service domain has already shown that interoperability cannot be readily achieved through the definition of common interfaces or specifications:

- The standardization process is too slow to capture the development in academy and industry.
- Specifications (as predecessors to standards) tend to diverge quickly with the standardization process being too slow.
- "Competing" standardization bodies with different opinions prefer different specifications.



A Need for Cloud Mashups

- By integrating multiple cloud infrastructures into a single platform, reliability and scalability is extended by the degree of the added system(s).
- Platform as a Service (PaaS) providers often offer specialized capabilities to their users via a dedicated API, such as Google App Engine providing additional features for handling (Google) documents, and MS Azure is focusing particularly on deployment and provisioning of Web services, and so on.
- Through aggregation of these special features, additional, extended capabilities can be achieved (given a certain degree of interoperability), ranging from extended storage and computation facilities (IaaS) to combined functions, such as analytics and functionalities.
- The Cloud Computing Expert Working Group refers to such integrated cloud systems with aggregated capabilities across the individual infrastructures as Meta-Clouds and Meta-Services, respectively.

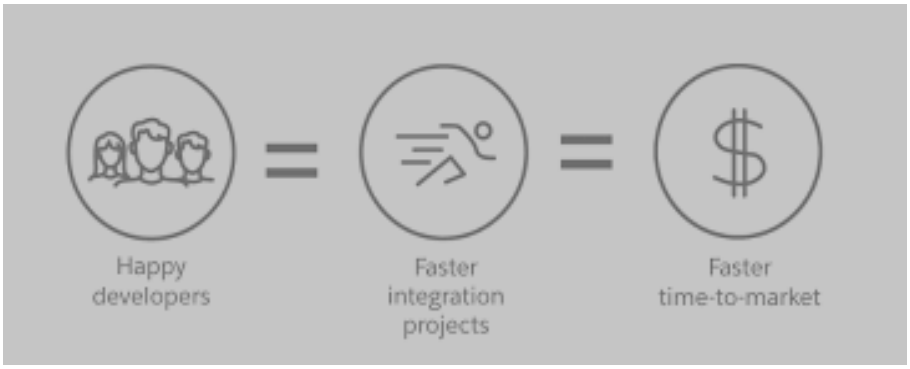
With the main focus of cloud-based services being “underneath” thetypical Web service levelthat is, more related to resources and platformskey interoperability issues relate to compatible data structures, related programming models, interoperable operating images, and so on. Thus, to realize a mashup requires at least:

- A compatible API/programming model, respectively an engine that can parse the APIs of the cloud platforms to be combined (PaaS).
- A compatible virtual machine, respectively an image format that all according cloud infrastructures can host (IaaS).
- Interoperable or transferrable data structures that can be interpreted by all engines and read by all virtual machines involved. This comes as a side effect to the compatibility aspects mentioned above.

Why do developers use PaaS?

Faster time to market

- PaaS is used to build applications more quickly than would be possible if developers had to worry about building, configuring, and provisioning their own platforms and backend infrastructure.
- With PaaS, all they need to do is write the code and test the application, and the vendor handles the rest



Why do developers use PaaS?

One environment from start to finish

- PaaS permits developers to build, test, debug, deploy, host, and update their applications all in the same environment.
- This enables developers to be sure a web application will function properly as hosted before they release, and it simplifies the application development lifecycle.

Price

- PaaS is more cost-effective than leveraging IaaS in many cases. Overhead is reduced because PaaS customers don't need to manage and provision virtual machines.
- In addition, some providers have a pay-as-you-go pricing structure, in which the vendor only charges for the computing resources used by the application, usually saving customers money. However, each vendor has a slightly different pricing structure, and some platform providers charge a flat fee per month.

Ease of licensing

- PaaS providers handle all licensing for operating systems, development tools, and everything else included in their platform.

Future of the PaaS market and business model

- PaaS has emerged as a cost-effective and capable cloud platform for developing, running and managing applications -- and the PaaS market is expected to gain popularity and grow through 2027. As an example, IDC predicted that the cloud and PaaS market should see a compound annual growth rate of 28.8 percent in 2021 through 2025.
- Such expectations are based on the need for businesses to accelerate application time to market, reduce complexity, shed local infrastructure, build collaboration -- especially for remote and geographically distributed teams -- and streamline application management tasks.
- PaaS expansion and growth are also being driven by cloud migration and cloud-first or cloud-native application development efforts in concert with other emerging cloud technologies, such as IoT.
- The role of iPaaS is also expected to make considerable gains by 2027 as businesses of all sizes seek to modernize, connect and share data between disparate software applications and deliver unified tools across the business and their customer base.

6.8 Monitoring Cloud-Based Services

What is Cloud Monitoring?

- Cloud monitoring is a method of reviewing, observing, and managing the operational workflow in a cloud-based IT infrastructure.
- Manual or automated management techniques confirm the availability and performance of websites, servers, applications, and other cloud infrastructure.
- This continuous evaluation of resource levels, server response times, and speed predicts possible vulnerability to future issues before they arise.

Types of Cloud Monitoring

The main types of cloud monitoring are:

- Database Monitoring
- Website Monitoring
- Virtual Network Monitoring
- Cloud Storage Monitoring
- Virtual Machine Monitoring

Database Monitoring

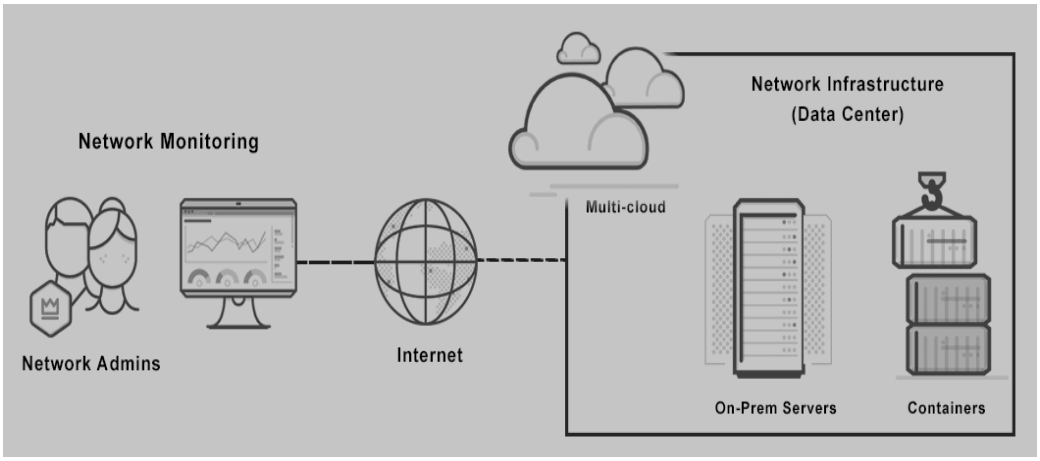
- Because most cloud applications rely on databases, this technique reviews processes, queries, availability, and consumption of cloud database resources.
- This technique can also track queries and data integrity, monitoring connections to show real-time usage data.
- For security purposes, access requests can be tracked as well. For example, an uptime detector can alert if there's database instability and can help improve resolution response time from the precise moment that a database goes down.

Website Monitoring

A website is a set of files that is stored locally, which, in turn, sends those files to other computers over a network. This monitoring technique tracks processes, traffic, availability, and resource utilization of cloud-hosted sites.

Virtual Network Monitoring

- This monitoring type creates software versions of network technology such as firewalls, routers, and load balancers. Because they're designed with software, these integrated tools can give you a wealth of data about their operation.
- If one virtual router is endlessly overcome with traffic, for example, the network adjusts to compensate. Therefore, instead of swapping hardware, virtualization infrastructure quickly adjusts to optimize the flow of data.



Cloud Storage Monitoring

- This technique tracks multiple analytics simultaneously, monitoring storage resources and processes that are provisioned to virtual machines, services, databases, and applications.
- This technique is often used to host infrastructure-as-a-service (IaaS) and software-as-a-service (SaaS) solutions.
- For these applications, you can configure monitoring to track performance metrics, processes, users, databases, and available storage.
- It provides data to help you focus on useful features or to fix bugs that disrupt functionality.

Virtual Machine Monitoring

- This technique is a simulation of a computer within a computer; that is, virtualization infrastructure and virtual machines.
- It's usually scaled out in IaaS as a virtual server that hosts several virtual desktops.
- A monitoring application can track the users, traffic, and status of each machine. You get the benefits of traditional IT infrastructure monitoring with the added benefit of cloud monitoring solutions.

Benefits of Cloud Monitoring

Monitoring is a skill, not a full-time job

- In today's world of cloud-based architectures that are implemented through DevOps projects, developers, site reliability engineers (SREs), and operations staff must collectively define an effective cloud monitoring strategy.
- Such a strategy should focus on identifying when service-level objectives (SLOs) are not being met, likely negatively affecting the user experience. So, then what are the benefits of leveraging cloud monitoring tools? With cloud monitoring:
  - Scaling for increased activity is seamless and works in organizations of any size
  - Dedicated tools (and hardware) are maintained by the host
  - Tools are used across several types of devices, including desktop computers, tablets, and phones, so your organization can monitor apps from any location
  - Installation is simple because infrastructure and configurations are already in place
  - Your system doesn't suffer interruptions when local problems emerge, because resources aren't part of your organization's servers and workstations
  - Subscription-based solutions can keep your costs low

Monitoring in Public, Private and Hybrid Clouds

- A private cloud gives you extensive control and visibility. Because systems and the software stack are fully accessible, cloud monitoring is relaxed when it's operated in a private cloud.
- Monitoring in public or hybrid clouds, however, can be tough.
- Let's review the focal points:
  - Because the data exists between private and public clouds, a hybrid cloud environment presents curious challenges. Limited security and compliance create problems for data access. Your administrator can solve these issues by deciding which data to store in various clouds and which data to asynchronously update.
  - A private cloud gives you more control, but to promote optimal performance, it's still wise to monitor workloads. Without a clear picture of workload and network performance, it's nearly impossible to justify configuration or architectural changes or to quantify quality-of-service implementations.

Cloud Monitoring Best Practices

- Observe your cloud service usage and fees. Increased costs can be triggered when scaling kicks in to meet demand. Strong monitoring solutions should track how much activity is on the cloud and its associated cost.
- Identify metrics and events that affect your bottom line. Not everything that can be measured needs to be reported.
- Use a single platform to report all data. You need solutions that can report data from different sources to a single platform. This consolidated information enables you to calculate uniform metrics and results in a complete performance view.
- Trigger rules with data. If activity surpasses or drops below certain levels, the right solution should be to add or subtract servers to maintain efficiency and performance.
- Separate your centralized data. Your organization must store your monitoring data separately from your proprietary apps, but the information should still be centralized for easy access.
- Monitor the user experience. To get the full picture of performance, review metrics such as response times and frequency of use.
- Try failure. Test tools to see what happens when an outage or a data breach occurs. This evaluation can create new standards for the alert system.

6.9 Testing in the Cloud

- Cloud testing is the process of using the cloud computing resources of a third-party service provider to test software applications.
- This can refer to the testing of cloud resources, such as architecture or cloud-native software as a service (SaaS) offering, or using cloud tools as a part of quality assurance (QA) strategy.
- Cloud testing can be valuable to organizations in a number of ways.
  - For organizations testing cloud resources, this can ensure optimal performance, availability and security of data, and minimize downtime of the associated infrastructure or platform. Organizations test cloud-based SaaS products to ensure applications are functioning properly.
  - For companies testing other types of applications, use of cloud computing tools, as opposed to on-premises QA tools, can help organizations cut down on testing costs and improve collaboration efforts between QA teams.

Types of Cloud Testing

While *cloud testing* in broad terms refers to testing applications through cloud computing resources, there are three main types of cloud testing that vary by purpose:

Testing of cloud resources. The cloud's architecture and other resources are assessed for performance and proper functioning. This involves testing a provider's platform as a service (PaaS) or infrastructure as a service (IaaS). Common tests may assess scalability, disaster recovery (DR), and data privacy and security.

Testing of cloud-native software. QA testing of SaaS products that reside in the cloud.

Testing of software with cloud-based tools. Using cloud-based tools and resources for QA testing.

Why is Cloud Testing needed?

- Automated testing is almost always more complicated to set up and execute than manual testing.
- With cloud automated testing, the process is simplified for the following reasons:
  - Cloud testing platforms are set up to facilitate tests for multiple users and teams on multiple devices simultaneously. That means QA teams won't have to share test environments with other teams/projects.
  - Even if some tests have to be queued, a cloud-based testing environment worth the cost is schematized to expedite tests without compromising accuracy.
  - Efficient cloud testing platforms also possess features to accelerate and enhance collaboration between teams or members of the same teams. This helps monitor all team members' progress and keeps everyone on the same page about project direction and achievements.
- This is also available for cloud-based manual testing platforms, such as BrowserStack's Live for Teams, but this is much rarer than the former.

Benefits of Cloud Testing

Here are some of the primary benefits associated with cloud testing:

- **Cost-effectiveness**
  Cloud testing is more cost-efficient than traditional testing, as customers only pay for what they use.
- **Availability and collaboration**
  Resources can be accessed from any device with a network connection. QA testing efforts are not limited by physical location. This, along with built-in collaboration tools, can make it easier for testing teams to collaborate in real time.
- **Scalability**
  Compute resources can be scaled up or down, according to testing demands.
- **Faster testing**
  Cloud testing is faster than traditional testing, as it circumvents the need for many IT management tasks. This can lead to faster time to market.

- **Customization**
  A variety of testing environments can often be simulated.
- **Simplified disaster recovery**
  DR efforts for data backup and recovery are less intensive than traditional methods.

Best Practices of Cloud Testing

- Look for a cloud testing platform that offers the devices and browsers the target audience is likely to use while using the software in question. For instance, BrowserStack offers 2000+ real browsers and devices. Chances are that users of this cloud will have access to the devices their potential customers would prefer.
- Before choosing a platform, put in the research. The ideal cloud should offer high-security levels, reasonably consistent tech support, and ensure that wait times for queued tests are not too long. The point of moving tests to the cloud is to speed them up without compromising on quality or security.
- A cloud testing platform worth the cost should cater not just to individual testers but to WA managers as well. Especially in these times of remote testing, the cloud should provide team-wide testing on a single plan, as well as features designed to help QA managers keep track of project progress and individual activity of each team member.

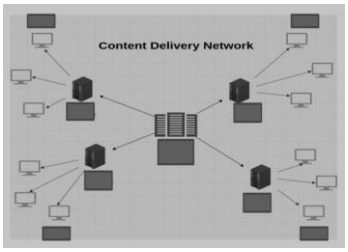
STL academy

Section 3: Exercises

Exercise 1: Write down the services provided by PaaS service providers in below Table.

Providers	Services
Google App Engine (GAE)	
Salesforce.com	
Windows Azure	
AppFog	
Openshift	
Cloud Foundry from VMware	

Exercise 2: Write Down all the Parts of Content Delivery Network in below Diagram.



Exercise 3: Participate in a group discussion on following topics:

a) Concept of Platform as a Service (PaaS)
b) Pros and Cons of PaaS
c) PaaS Architecture
d) PaaS and Its Services
e) PaaS Monitoring
f) Benefits of Cloud Monitoring

Section 4: Assessment Questionnaire

1. What is PaaS?
2. Who is the End Customer of PaaS?
3. What are ways to Deliver PaaS?
4. What Services Does PaaS Includes?
5. What is Public PaaS?
6. What is Hybrid PaaS?
7. What are the Common PaaS Scenarios?
8. What are the Features of PaaS?
9. What are Popular PaaS Providers?
10. List few of the Content Delivery Network Using Cloud.

----------End of the Module----------

MODULE 7
UTILIZING INFRASTRUCTURE AS A SERVICE (IaaS)

Section 1: Learning Outcomes

After completing this module, you will be able to:

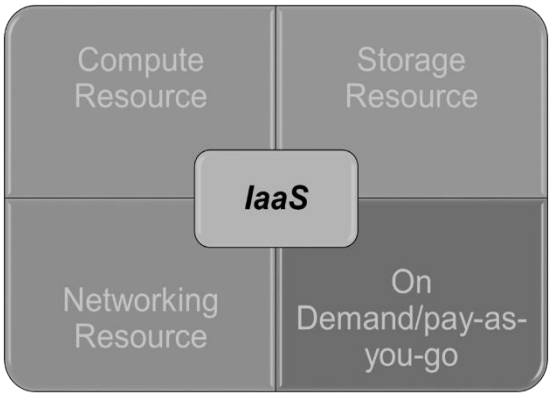
- Explain the Utilization of Infrastructure as a Service (IaaS)
- Differentiate between various enabling technologies
- Define Scalable Server Clusters
- Describe the Elastic Storage Devices information
- Show how to access IaaS
- Handle Dynamic & Static IP Addresses
- Enlist Tools & Support for Management & Monitoring

Section 2: Relevant Knowledge

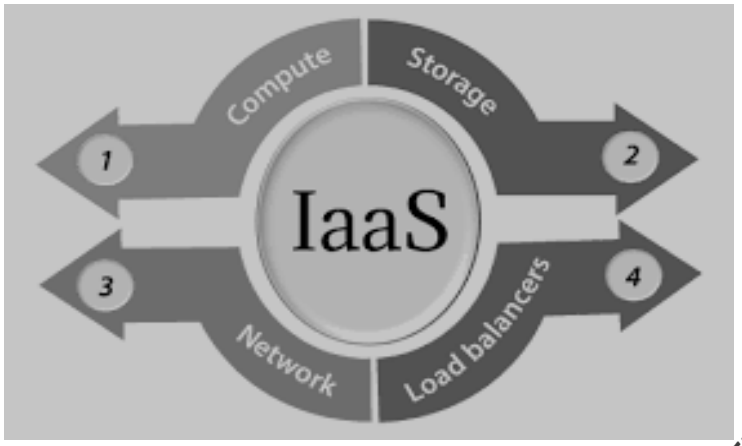
7.1 Utilizing Infrastructure as a Service (IaaS)

Infrastructure as a Service

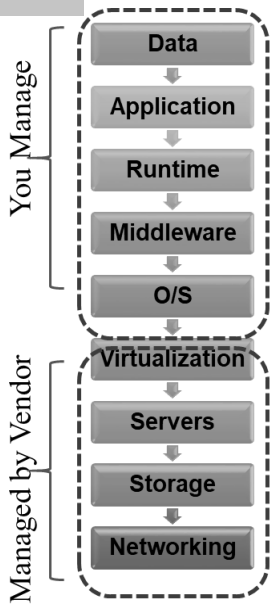
- Infrastructure as a service (IaaS) is a type of cloud computing service that offers essential compute, storage and networking resources on demand, on a pay-as-you-go basis.
- IaaS is one of the four types of cloud services, along with software as a service (SaaS), platform as a service (PaaS) and serverless.
- Migrating your organisation's infrastructure to an IaaS solution helps you reduce maintenance of on-premises data centres, save money on hardware costs and gain real-time business insights.
- IaaS solutions give you the flexibility to scale your IT resources up and down with demand. They also help you quickly provision new applications and increase the reliability of your underlying infrastructure.



- IaaS lets you bypass the cost and complexity of buying and managing physical servers and datacentre infrastructure.
- Each resource is offered as a separate service component and you only pay for a particular resource for as long as you need it.
- A cloud computing service provider like Azure manages the infrastructure, while you purchase, install, configure and manage your own software including operating systems, middleware and applications.
- Offering virtualized resources (computation, storage, and communication) on demand is known as Infrastructure as a Service (IaaS).



A cloud infrastructure enables on-demand provisioning of servers running several choices of operating systems and a customized software stack. Infrastructure services are considered to be the bottom layer of cloud computing systems.

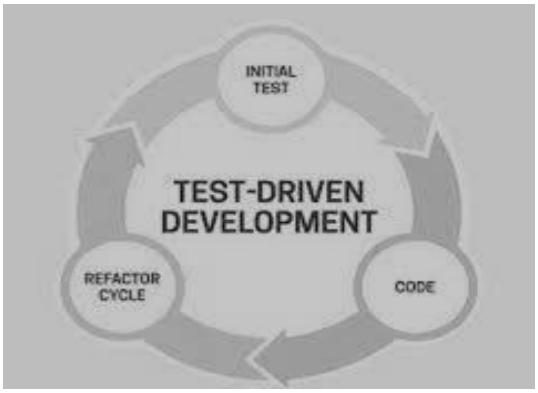


IaaS

Common IaaS use cases include:

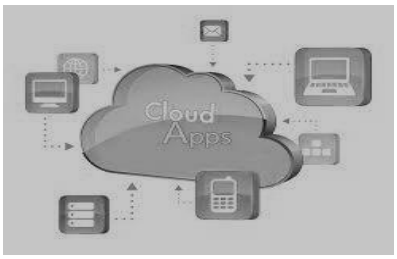
Test and Development

Teams can rapidly create development and test environments to bring new applications to market faster. IaaS enables teams to create test and development environments automatically, as part of their development pipeline.



Web Apps

- IaaS provides all the infrastructure needed to run large scale web applications, including storage, web servers, and networking.
- Organizations can quickly deploy web applications using IaaS services, and easily scale their infrastructure when application requirements increase or decrease.



Storage, Backup and Recovery

- Organizations can avoid the high upfront cost of storage and the complexity of storage management.
- Leveraging cloud storage services eliminates the need for trained personnel to manage data and comply with legal and regulatory requirements, and helps organizations respond to storage requirements on-demand.
- It also simplifies the planning and management of backup and recovery systems.

High-Performance Computing

- High-performance computing (HPC) can help solve complex and complex problems with millions of variables and calculations, by running them on supercomputers or large clusters of computers.
- The major IaaS providers offer services that place HPC within the reach of ordinary businesses, allowing them to use HPC on demand instead of making a huge investment in HPC infrastructure.

Big Data Analytics

- Big data processing and analysis is critical in today's economy, and requires complex infrastructure including large-scale storage systems, distributed processing engines, and high-speed databases.
- IaaS providers provide all this infrastructure as a managed service, and most of them also offer PaaS services that can perform the actual analytics, including machine learning and AI.

7.2 Enabling Technologies

AWS IaaS Services

Amazon S3

- Amazon Simple Storage Service (S3) is the first and most popular Amazon service, which provides object storage at unlimited scale.
- S3 is easy to access via the Internet and programmatically via API, and is integrated into a wide range of applications.
- It provides 11 9's of durability (99.999999999%), and offers several storage tiers, allowing users to move data that is used less frequently into a low-cost archive tier within S3.

AWS EC2

- Amazon Elastic Compute Cloud (Amazon EC2) offers scalable computing resources.
- It lets you run as many virtual servers as you want, configure your network and security, and manage storage.
- You can increase or decrease resources on-demand according to changing business requirements, and set up auto scaling to scale resources up and down according to actual workloads.

AWS EBS

- Amazon Elastic Block Store (Amazon EBS) is a block-level storage service for use with Amazon EC2 instances.
- When mounted on an Amazon EC2 instance, you can use Amazon EBS volumes like any other raw block storage device.
- It can be formatted and mirrored for specific file systems, host operating systems, and applications.

AWS Lambda

- AWS Lambda is a serverless, on-demand IT service that provides developers with a fully managed, event-driven cloud system that executes code.
- AWS Lambda uses Lambda functions anonymous functions that are not associated with identifiers enabling users to package any code into a function and run it, independently of other infrastructure

Azure IaaS Services

Linux Virtual Machines in Azure

- Traditionally Azure focused on Windows virtual machines, but now has a robust offering for Linux users as well.
- Azure virtual machines (VMs) are scalable on-demand compute resources provided by Azure. Microsoft Azure supports popular Linux distributions deployed and managed by multiple partners.
- Linux machine images are available in the Azure Marketplace for the following Linux distributions (more distributions are added on an ongoing basis):
  - FreeBSD
  - Red Hat Enterprise
  - CentOS
  - SUSE Linux Enterprise
  - Debian
  - Ubuntu
  - CoreOS
  - RancherOS

Azure Managed Disk

- Azure managed disks are block-level storage volumes managed by Azure and used by Azure virtual machines.
- A managed disk is similar to a physical disk on a local server, but it is virtualized.
- For managed disks, you only need to specify the disk size and disk type, and provision—Azure does the rest. The available hard drive types are:
  - Standard hard disks (HDD)
  - Standard SSD
  - Premium SSDs
  - Ultra-disks—optimized for sub-millisecond latency

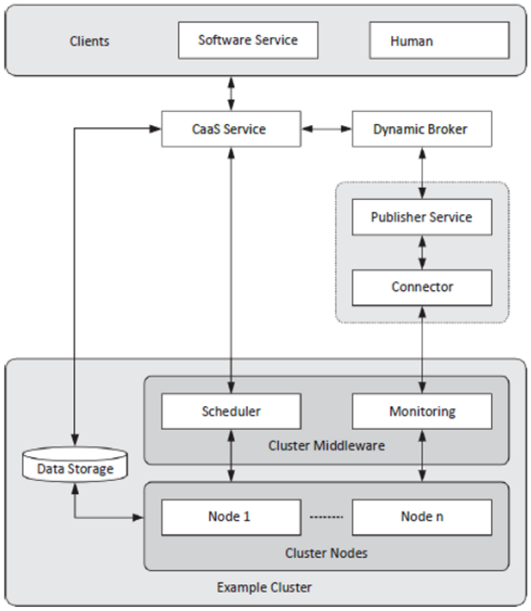
7.3 Scalable Server Clusters

Cluster As A Service: The Logical Design

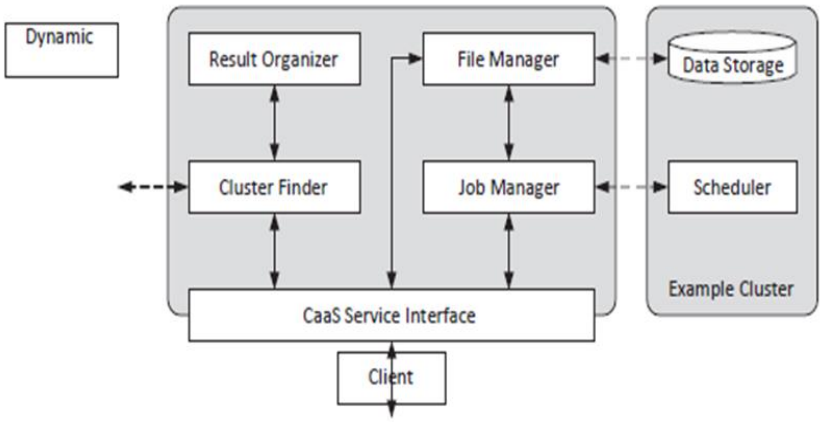
- Simplification of the use of clusters could only be achieved through higher layer abstraction that is proposed here to be implemented using the service-based Cluster as a Service (CaaS) Technology.
- The purpose of the CaaS Technology is to ease the publication, discovery, selection, and use of existing computational clusters.

CaaS Overview

- The exposure of a cluster via a Web service is intricate and comprises several services running on top of a physical cluster. Figure shows the complete CaaS technology.
- A typical cluster is comprised of three elements:
  - Nodes
  - Data storage
  - Middleware
- The middleware virtualizes the cluster into a single system image; thus, resources such as the CPU can be used without knowing the organization of the cluster.
- As time progresses, the amount of free memory, disk space, and CPU usage of each cluster node changes. Information about how quickly the scheduler can take a job and start it on the cluster also is vital in choosing a cluster.



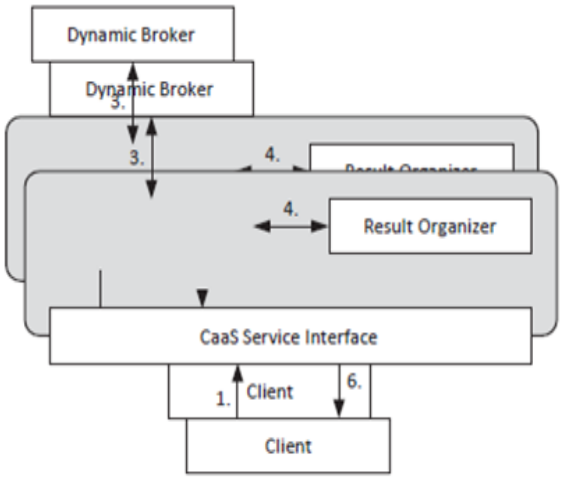
Complete CaaS system



CaaS Service design

Cluster Discovery

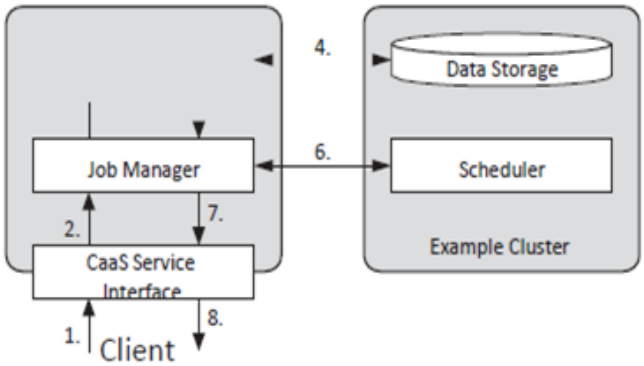
- Before a client uses a cluster, a cluster must be discovered and selected first. Figure 3.5 shows the workflow on finding a required cluster.
- To start, clients submit cluster requirements in the form of attribute values to the CaaS Service Interface:
  (1) The requirements range from the number of nodes in the cluster to the installed software (both operating systems and software APIs). The CaaS Service Interface invokes the Cluster Finder module.
  (2) that communicates with the Dynamic Broker
  (3) returns service matches (if any). To address the detailed results from the Broker, the Cluster Finder module invokes the Results Organizer module
  (4) that takes the Broker results and returns an organized version that is returned to the client



Cluster discovery

Job Submission

- After selecting a required cluster, all executables and data files have to be transferred to the cluster and the job submitted to the scheduler for execution.
- As clusters vary significantly in the software middleware used to create them, it can be difficult to place jobs on the cluster.
- To do so requires knowing how jobs are stored and how they are queued for execution on the cluster.



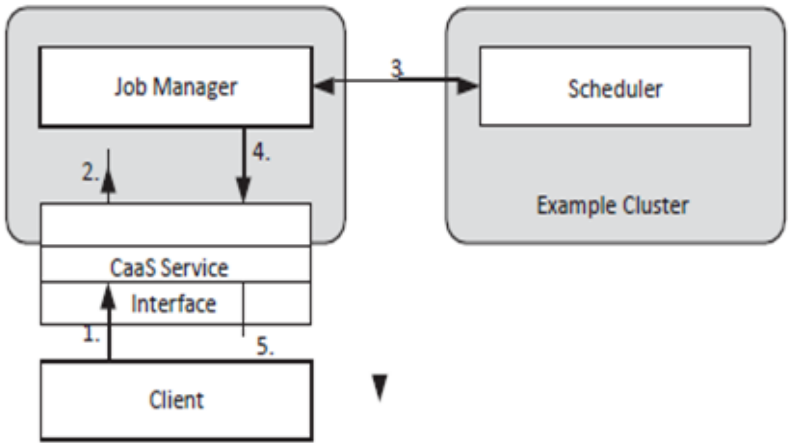
Job submission

Job Monitoring

- During execution, clients should be able to view the execution progress of their jobs. Even though the cluster is not the owned by the client, the job is.
- It is the right of the client to see how the job is progressing and (if the client decides) terminate the job and remove it from the cluster.

Result Collection

- The final role of the CaaS Service is addressing jobs that have terminated or completed their execution successfully.
- In Both cases, error or data files need to be transferred to the client. Figure presents the workflow and CaaS Service modules used to retrieve error or result files from the cluster.



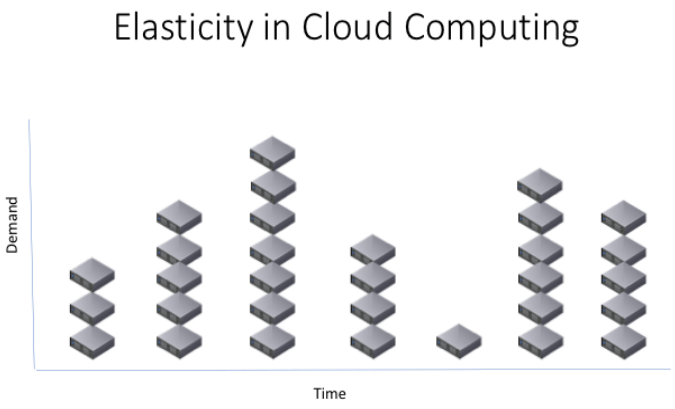
Job result collection

7.4 Elastic Storage Devices

What is Elasticity?

Elasticity

- One of the main advantages of cloud computing is the capability to provide, or release, resources on-demand.
- These elasticity capabilities should be enacted automatically by cloud computing providers to meet demand variations, just as electrical companies are able (under normal operational circumstances) to automatically deal with variances in electricity consumption levels.
- Clearly the behavior and limits of automatic growth and shrinking should be driven by contracts and rules agreed on between cloud computing providers and consumers.



Why is Cloud Elasticity Important?

- Without Cloud Elasticity, organizations would have to pay for capacity that remained unused for most of the time, as well as manage and maintain that capacity with OS upgrades, patches, and component failures.
- It is Cloud Elasticity that in many ways defines cloud computing and differentiates it from other computing models such as client-server, grid computing, or legacy infrastructure.
- Cloud Elasticity helps businesses avoid either over-provisioning (deploying and allocating more IT resources than needed to serve current demands) or under-provisioning (not allocating enough IT resources to meet existing or imminent demand).
- Organizations that over-provision spend more than is necessary to meet their needs, wasting valuable capital which could be applied elsewhere. Even if an organization is already utilizing public cloud, without elasticity, thousands of dollars could be wasted on unused VMs every year.
- Under-provisioning can lead to the inability to serve existing demand, which could lead to unacceptable latency, user dissatisfaction, and ultimately loss of business as customers abandon long online and take their business to more responsive organizations. In this way the lack of Cloud Elasticity can lead to lost business and severe bottom-line impacts.

How does Cloud Elasticity Work?

- Cloud Elasticity enables organizations to rapidly scale capacity up or down, either automatically or manually.
- Cloud Elasticity can refer to ‘cloudbursting’ from on-premises infrastructure into the public cloud for example to meet a sudden or seasonal demand.
- Cloud Elasticity can also refer to the ability to grow or shrink the resources used by a cloud-based application.
- Cloud Elasticity can be triggered and executed automatically based on workload trends, or can be manually instantiated, often in minutes. Before organizations had the ability to leverage Cloud Elasticity, they would have to either have additional stand-by capacity already on hand or would need to order, configure, and install additional capacity, a process which could take weeks or months.
- If and when demand eases, capacity can be removed in minutes.
- In this manner organizations pay only for the amount of resources in use at any given time, without the need to acquire or retire on-premises infrastructure to meet elastic demand

Use Cases of Cloud Elasticity

Typical use cases for Cloud Elasticity include:

- Retail or e-tail holiday seasonal demand, in which demand increases dramatically from Black Friday shopping specials until the end of the holiday season in early January
- School district registration which spikes in demand during the spring and wanes after the school term begins
- Businesses that see a sudden spike in demand due to a popular product introduction or social media boost, such as a streaming service like Netflix adding VMs and storage to meet demand for a new release or positive review.
- Disaster Recovery and Business Continuity (DR/BC). Organizations can leverage public cloud capabilities to provide off-site snapshots or backups of critical data and applications, and spin up VMs in the cloud if on-premises infrastructure suffers an outage or loss.
- Scale virtual desktop infrastructure in the cloud for temporary workers or contractors or for applications such as remote learning
- Scale infrastructure into the cloud for test and development activities and tear it down once test/dev work is complete.
- Unplanned projects with short timelines
- Temporary projects like data analytics, batch processing, media rendering, etc.

What are the Benefits of Cloud Elasticity?

The benefits of cloud elasticity include:

Agility

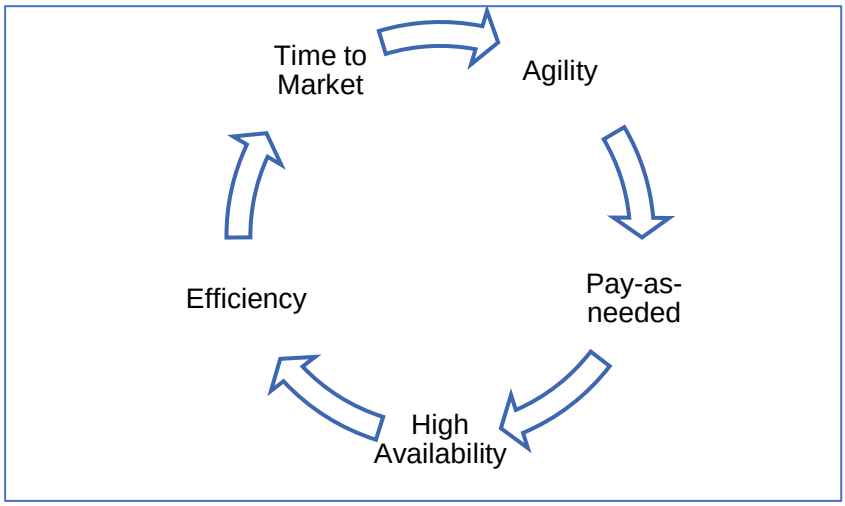
By eliminating the need to purchase, configure, and install new infrastructure when demand changes, Cloud Elasticity prevents the need to plan for such unexpected demand spikes, and enables organizations to meet any unexpected demand, whether due to seasonal spike, mention on Reddit, or selection by Oprah's book club.

Pay-as-needed pricing

- Rather than paying for infrastructure whether or not is being used, Cloud Elasticity enables organizations to pay only for the resources that are in use at any given point tin time, closely tracking IT expenditures to the actual demand in real-time.
- In this way, although spending may fluctuate, organizations can 'right-size' their infrastructure as elasticity automatically allocates or deallocates resources on the basis of real-time demand.
- Amazon has stated that organizations that adopt its instance scheduler with their EC2 cloud service can achieve savings of over 60 percent versus organizations that do not.

High Availability

- Cloud elasticity facilitates both high availability and fault tolerance, since VMs or containers can be replicated if they appear to be failing, helping to ensure that business services are uninterrupted and that users do not experience downtime.
- This helps ensure that users perceive a consistent and predictable experience, even as resources are provisioned or deprovisioned automatically and without impact on operations.



Efficiency

As with most automations, the ability to autonomously adjust cloud resources as needed enables IT staff to shift their focus away from provisioning and onto projects that are more beneficial to the organization.

Speed/Time-to-market

Organizations have access to capacity in minutes instead of the weeks or months it may take through a traditional procurement process.

Secure Distributed Data Storage in Cloud Computing

Cloud Storage: From LANs to WANs

- Cloud computing will be a revolutionary change in computing services.
- Users will be allowed to purchase CPU cycles, memory utilities, and information storage services conveniently just like how we pay our monthly water and electricity bills.

Existing Commercial Cloud Services

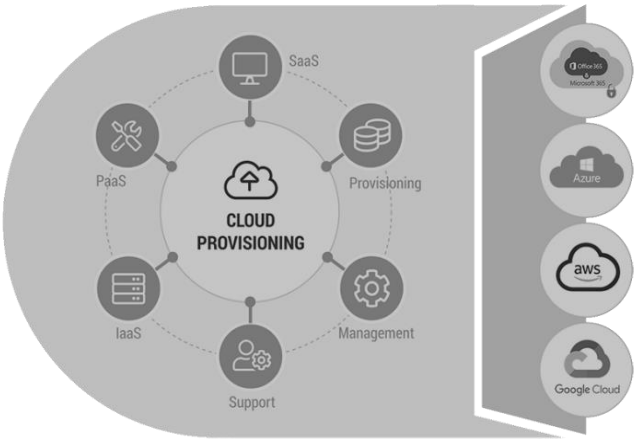
- In normal network-based applications, user authentication, data confidentiality, and data integrity can be solved through IPSec proxy using encryption and digital signature.
- The key exchanging issues can be solved by SSL proxy. These methods have been applied to today's cloud computing to secure the data on the cloud and also secure the communication of data to and from the cloud.
- The service providers claim that their services are secure.

7.5 Accessing IaaS

- Cloud provisioning means allocating a cloud service provider's resource to a customer.
- It is a key feature of cloud computing. It refers to how a client gets cloud services and resources from a provider.
- The cloud services that customers can subscribe to include infrastructure-as-a-service (IaaS), software-as-a-service (SaaS), and platform-as-a-service (PaaS) in public or private environments.

Types of Cloud Provisioning

- There are various cloud provisioning delivery models.
- Each model depends on the types of resources or services an organization purchases, how and when the cloud service provider delivers them, and how customers pay for them.
- The three models
  - Advanced
  - Dynamic
  - User self-provisioning



Advanced Cloud Provisioning

- Also known as "post-sales cloud provisioning," customers get the resources upon contract or service signup. They sign formal contracts with the cloud service provider. The provider then prepares and delivers the agreed-upon resources or services.
- The customers are charged a flat fee or billed every month.

Dynamic Cloud Provisioning

- Also referred to as "on-demand cloud provisioning," customers are provided with resources on runtime.
- In this delivery model, cloud resources are deployed to match customers' fluctuating demands. Deployments can scale up to accommodate spikes in usage and down when demands decrease.
- Customers are billed on a pay-per-use basis. When this model is used to create a hybrid cloud environment, it is sometimes called "cloud bursting."

User Cloud Provisioning

- In this delivery model, customers add a cloud device themselves. Also known as “cloud self-service,” clients buy resources from the cloud service provider through a web interface or portal.
- The model usually involves creating a user account and paying for resources with a credit card. The resources are quickly spun up and made available for use within hours, if not minutes.
- An example of this includes an employee purchasing cloud-based productivity applications via Microsoft 365 or G Suite.

Cloud Provisioning Benefits

Cloud provisioning has several benefits that are not available with traditional provisioningapproaches, such as:

Scalability

- The traditional information technology (IT) provisioning model requires organizations to make large investments in their on-premises infrastructure. That needs extensive preparation and forecasting of infrastructure needs since on-premises infrastructures are often set up to last for many years.
- The cloud provisioning model, meanwhile, lets companies simply scale up and down their cloud resources depending on their short-term usage requirements.

Speed

Organizations’ developers can quickly spin up several workloads on-demand, so the companies no longer require IT administrators to provide and manage computing resources.

Cost Savings

While traditional on-premises technology requires large upfront investments, many cloud service providers let their customers pay for only what they consume. But the attractive economics of cloud services presents challenges, too, which may require organizations to develop a cloud management strategy.

Cloud Provisioning Challenges

Like any other technology, cloud provisioning also presents several challenges, including:

Complex management and monitoring

- Organizations may need several provisioning tools to customize their cloud resources.
- Many also deploy workloads on more than one cloud platform, making viewing everything on a central console more challenging.

Resource and service dependencies

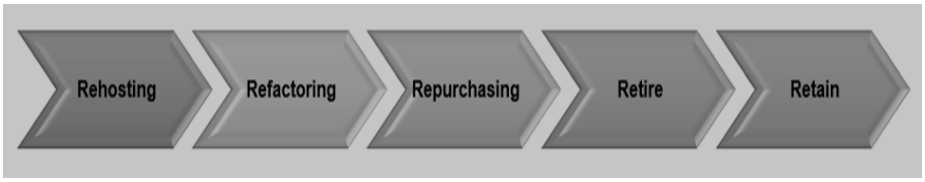
- Cloud applications and workloads often tap into basic infrastructure resources, such as computing, networking, and storage. But public cloud service providers offer higher-level ancillary services like serverless functions and machine learning (ML) and big data capabilities.
- Such services may carry dependencies that can lead to unexpected overuse and surprise costs.

Policy enforcement

- User cloud provisioning helps streamline requests and manage resources but requires strict rules to make sure unnecessary resources are not provided. That is time-consuming since different users require varying levels of access and frequency.
- Setting up rules to know who can provide which resources, for how long, and with what budgetary controls can be difficult.

Adopting IaaS: Cloud Migration Strategies

- Following are the most common approaches to cloud migration, taken from the influential “5 R’s” model proposed by Gartner.
  - Rehosting
  - Replatforming, Refactoring, or Re-architecture
  - Repurchasing
  - Retire
  - Retain



Rehosting

- Re-hosting (also known as "lift and shift") is the fastest way to move your application to the cloud.
- This is usually the first approach taken in a cloud migration project because it allows moving the application to the cloud without any changes.
- Both physical and virtual servers are migrated to infrastructure as a service (IaaS).
- Lift and shift is commonly used to improve performance and reliability for legacy applications.

Replatforming, Refactoring, or Re-architecture

- This migration strategy involves detailed planning and a high investment, but it is the only strategy that can help you get the most out of the cloud.
- Applications that undergo replatforming or re-architecture are completely rebuilt on cloud-native infrastructure.
- They scale up and down on-demand, are portable between cloud resources and even between different cloud providers.

Repurchasing

- In most cases, repurchasing is as easy as moving from an on-premise application to a SaaS platform.
- Typical examples are switching from internal CRM to Salesforce.com, or switching from internal email server to Google’s G Suite.
- It is a simple license change, which can reduce labor, maintenance, and storage costs for the organization.

Retire

- When planning a move to the cloud, it often turns out that part of the company's IT product portfolio is no longer useful and can be decommissioned.
- Removing old applications allows you to focus time and budget on high priority applications and improve overall productivity.

Retain

- Moving to the cloud doesn't make sense for all applications. You need a strong business model to justify migration costs and downtime.
- Additionally, some industries require strict compliance with laws that prevent data migration to the cloud. Some on-premises solutions should be kept on-premises, and can be supported in a hybrid cloud migration model.

What is Cloud Management?

- Cloud management refers to the exercise of control over public, private or hybrid cloud infrastructure resources and services.
- A well-designed cloud management strategy can help IT pros control those dynamic and scalable computing environments.
- Cloud management can also help organizations achieve three goals:
  - *Self-service* refers to the flexibility achieved when IT pros access cloud resources, create new ones, monitor usage and cost, and adjust resource allocations.
  - *Workflow automation* lets operations teams manage cloud instances without human intervention.
  - *Cloud analysis* helps track cloud workloads and user experiences.
- Without a competent IT staff in place, it's difficult for any cloud management strategy to succeed.
- These individuals must possess knowledge of the proper tools and best practices while they keep in mind the cloud management goals of the business.
- Companies are more likely to improve cloud computing performance, reliability, cost containment and environmental sustainability when they adhere to tried-and-true cloud optimization practices.
- There are many ways to approach cloud management, and they are ideally implemented in concert.
- Cost-monitoring tools can help IT shops navigate complex vendor pricing models. Applications run more efficiently when they use performance optimization tools and with architectures designed with proven methodologies.
- Many of these tools and strategies dovetail with environmentally sustainable architectural strategies to lower energy consumption.
- Cloud management decisions must ultimately hinge on individual corporate priorities and objectives, as there is no single approach.

What is Cloud Monitoring?

- Cloud monitoring measures the conditions of a workload and the various quantifiable parameters that relate to overall cloud operations.
- Results are monitored in specific, granular data, but that data often lacks context.
- Cloud observability is a process similar to cloud monitoring in that it helps assess cloud health. Observability is less about metrics than what can be gleaned from a workload based on its externally visible properties.
- There are two aspects of cloud observability: methodology and operating state. Methodology focuses on specifics, such as metrics, tracing and log analysis.
- Operating state relies on tracking and addresses state identification and event relationships, the latter of which is a part of DevOps.

STL academy

Why Cloud Monitoring?

Cloud monitoring can perform the following capabilities:

- Monitoring cloud data across distributed locations
- Eliminating potential breaches by providing visibility into files, applications, and users
- Continually monitoring the cloud to ensure real-time file scans
- Regular auditing and reporting to ensure security standards
- Merging monitoring tools with different cloud providers

AWS First-Party Monitoring Tools

- There are multiple services and utilities available from AWS that you can use to monitor your systems and access.
- Some of these tools are included in existing services, while others are available for additional costs.
  - AWS CloudTrail
  - AWS CloudWatch
  - AWS Certificate Manager
  - Amazon EC2 Dashboard

AWS CloudTrail

- CloudTrail is a service that you can use to track events across your account.
- The service automatically records event logs and activity logs for your services and stores the data in S3.
- Collected data includes user identities, traffic origin IPs, and timestamps.
- You can view all management events for free for the most recent 90 days. Data events and insights based on your data are also available for an additional fee.

AWS CloudWatch

- CloudWatch is a service you can use to aggregate, visualize, and respond to service metrics.
- CloudWatch has two main components: alarms, which create alerts according to thresholds for single metrics, and events, which can automate responses to metric values or system changes.

AWS Certificate Manager

- Certificate Manager is a tool you can use to provision, manage, and apply transport layer security (TLS) and secure sockets layer (SSL) certificates.

- These certificates are used to prove your services or devices' authenticity and enable you to secure network connections.

Amazon EC2 Dashboard

- EC2 Dashboard is a monitoring tool for the Amazon EC2 virtual machine service.
- You can use this dashboard to monitor and maintain your EC2 instances and infrastructure.
- The dashboard lets you view instance states and service health, manage alarms and status reports, view scheduled events, and assess volume and instance metrics

Google Cloud Platform Monitoring Tools

Azure Monitoring Tools

Benefits of Cloud Monitoring

There are innumerable benefits cloud monitoring provides. Even businesses that solely rely on a private cloud architecture can enjoy key cloud monitoring deliverables, including:

- Improving the security of cloud applications and networks
- Simplifying the implementation of continuity plans, enabling proactive (rather than reactive) risk remediation
- Achieving and maintaining ideal application performance
- Optimizing service availability thanks to rapid issue reporting and rapid resolutions
- Reduction of surprise cloud cost leaks thanks to complete architecture visibility
- Simple scaling in the event cloud activity increases
- Usability on multiple devices, ensuring cloud awareness at all times

Cloud Monitoring Best Practices

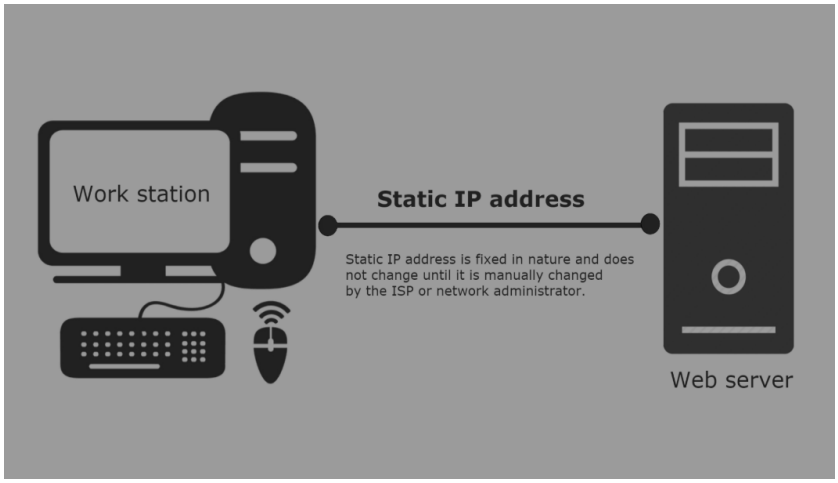
As you implement a cloud monitoring service, keep the following best practices in mind to ensure you experience the full benefits.

- Decide which activity(ies) need to be monitored. Choose the metrics that matter the most to your bottom line.
- Consolidate report data onto a single platform to eliminate confusion and complexity that arises from juggling multiple cloud services and infrastructures. Your solution should report data from various sources and present them in one platform, enabling you to calculate metrics comprehensively.
- Keep track of subscription and service fees. The more you use your cloud monitoring service, the pricier it will be to use. Choosing a more advanced service can track how much activity is occurring on the cloud and determine costs from there.
- Be aware of which users are using which cloud applications to track accountability. You'll also need to know what these users see when they're using certain applications, and you'll want to monitor response time, frequency of use, and other metrics overall.
- Automate rules with the appropriate data to account for activities that go over or below your thresholds, ensuring you're able to add or remove servers to maintain consistent performance.
- Separate your monitoring data from your applications and services, and centralize this information to ensure your stakeholders have easy access.
- Always test your cloud monitoring tools at a regular cadence. While a service may seem operational, an outage or breach will truly put it to the test, so test your tools to ensure there are no surprises.

7.6 Handling Dynamic & Static IP Addresses

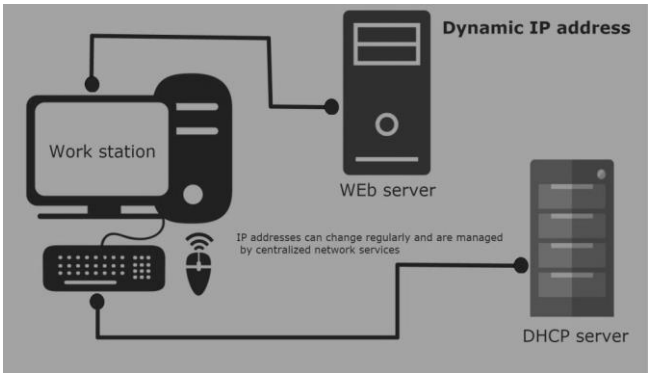
Static IP Addresses

- A static IP address is an IP address that doesn't change. Static IP addresses usually stay the same unless our network architecture changes or our devices are out of commission.
- Static IP addresses are typically used for servers or other important networking equipment. They're popular within business settings because they ensure that the devices connected to them keep a consistent address.
- They also work well for remote access solutions.
- A static IP address is assigned to a device by an ISP. Typically, static IP addresses add to the cost of your internet service.



Dynamic IP addresses

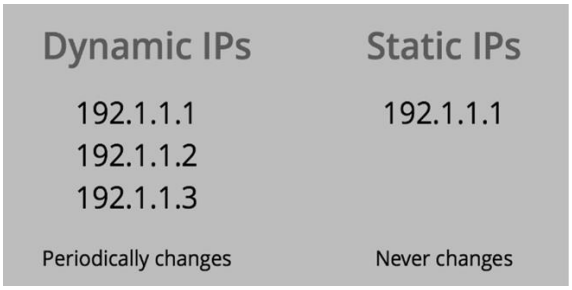
- A dynamic IP address is an IP address that can regularly change.
- An ISP will buy a large number of dynamic IP addresses and assign them to their customer's devices.
- Dynamic IP addresses are often reassigned. Reassigning IP addresses helps internet providers save money and ensure a higher level of security.
- It also means that they don't need to take the time to reestablish any network connections if we go on a vacation or move to a new location.
- Dynamic IP addresses are more common for consumer equipment and personal use. A dynamic IP address is assigned to a device by our ISP's Dynamic Host Configuration Protocol (DHCP) servers.
- The DHCP server typically uses network routers to assign addresses to devices.



Static IP vs Dynamic IP

- There's not always a clear answer when it comes to deciding between a static or a dynamic IP address.
- When choosing between the two, it's important to consider your connection environment.
- In other words, static IP addresses tend to be better for businesses, while dynamic IP addresses tend to be better for personal or home networks.

Let's take a look at some of the pros and cons of static and dynamic IP addresses:



Static IPA: Pros

- **Remote access:** Static IP addresses make it easy for us to work remotely using a Virtual Private Network (VPN).
- **Server hosting:** Static IP addresses make it easy for people to find us using DNS.
- **DNS support:** With static IP, it's easier to manage DNS servers.
- **Geolocation services:** With static IP addresses, our geolocation services are more accurate. This is because our services will match the IP address to its physical location.
- **Reliable connection:** A static IP address is fixed, which typically results in a more reliable connection.
- **Easy to find:** A static IP address can make it easier to find specific devices on a network.

Static IP Addresses: Cons

- **Security concerns:** With a static IP address, anyone with the proper tools can find where our devices are located. VPNs can help with this.
- **Cost:** Static IP addresses are not as cost-effective as dynamic IP addresses. Typically, ISPs charge more for them.

Dynamic IP Addresses: Pros

- **Easy configuration:** DHCP servers automatically assign IP addresses to our devices, so we don't need to worry about setting it up ourselves.
- **Cost:** Dynamic IP addresses are usually cheaper than static IP addresses.
- **Unlimited IP addresses:** Dynamic IP addresses can be reused. Whenever our devices need a new dynamic IP address, our network or router can automatically configure them for us.
- **Security:** Dynamic IP addresses make it more difficult for potential attackers to locate our networked devices.
- This is because dynamic IP addresses can change frequently, so it's harder to track a device.
- This helps with physical and online security. We can also increase our security measures by using a VPN.

Dynamic IP Addresses: Cons

- **DNS Compatibility:** If we wanted to host an email server, for example, it may be difficult to use a dynamic IP address because DNS doesn't work well with dynamic IP addresses. We could use a dynamic DNS service, but those tend to be expensive.
- **Remote Connectivity:** If we don't have the proper remote access software, it'll be difficult to connect using a dynamic IP address. A VPN can help with this.
- **Increased Downtime**: Sometimes, our ISP can't assign us a dynamic IP address. This can slow down our internet connection.
- **Inaccurate geolocation:** Dynamic IP addresses may affect our geolocation services because our IP address may not reflect our physical location.

7.7 Tools & Support for Management & Monitoring

- Logging into each server in turn and manually checking performance is very time-consuming.
- This is why server management and monitoring tools have become so important.
- Greater complications arise when you expand your infrastructure to include a number of servers. You don't want to get trapped into buying from one vendor and it may be advantageous to use a blend of operating systems in your network.
- Remote server administration tools will enable you to manage servers on several sites and improve your efficiency.

Here's list of the best server management software and monitoring tools:

- Logging into every server and manually checking its performance can be time-consuming this brings about the necessity of server management and monitoring tools.
- Keeping track of utilization, maintenance issue has become a burden compared to setting of servers.
- **Server Density** This tool monitors onsite and cloud-based servers; it covers Linux and Windows onsite servers and AWS, or Azure-based cloud servers.
- **Checkmk** A monitor for networks, servers, and applications whether they are on site or in the Cloud. Available in free and paid versions. Installs on Linux.
- **Nagios XI** A network monitoring system that also covers applications and servers. You can opt for a free version, called Nagios Core.
- **Instrumental** This is an application monitor that also covers server statuses, specifically: CPU usage, disk status, load information, memory activity, network interaction, and page swapping events.
- **Zabbix** A free system monitor that tracks CPU operations, memory utilization, I/O error rates, disk space, fan status, temperature, and power supply performance on your servers.
- **OP5** A server monitor that enables capacity planning and trend analysis.
- **Monitis** A cloud-based service that monitors applications and server statuses.
- **Anturis** A cloud-based server monitor that tracks CPU, memory, disk usage, page swaps, power supply status, fan status, and temperature.
- **Motadata** A network, server, and application monitoring tool that tracks disk volumes and spare capacity, CPU usage, and memory utilization.

Examples of Management and Monitoring Tools

1. ScolarWinds Server and Application Monitor (SAM)
2. Atera
3. ManageEngine Opmanager
4. Site24x7 Server Monitoring
5. N-able RMM
6. Ninjaone
7. Syxsense-Manage
8. ManageEngine Applications Manager

Section 3: Exercises

Exercise 1: Draw flow chart of complete CaaS systems and CaaS service design.

Exercise 2: Participate in group discussion on following topics:

a) Infrastructure as a Service in Cloud Computing
b) Various enabling technologies
c) Scalable Server Clusters
d) Elastic Storage Devices
e) Accessing IaaS
f) Dynamic and Static IP Address
g) Tools and Support for Management and Monitoring

Section 4: Assessment Questionnaire

1. What is IaaS?
2. What is Elasticity?
3. What are the Use Cases of Cloud Elasticity?
4. What is Cloud Provisioning??
5. What are the types of Cloud Provisioning?
6. What is 5R Approach of Cloud Migration?
7. List few of the AWS Monitoring Tools.
8. What is Static and Dynamic IP Address?

----------End of the Module----------

MODULE 8
MAKING A BUSINESS CASE

Section 1: Learning Outcomes

After completing this module, you will be able to:

- Explain the Business Case planning for Cloud Adoption
- Calculate the Financial Implications
- Compare In-house Facilities to the Cloud
- Estimate Economic Factors Downstream
- Safeguard access to Assets in the cloud
- Adopt Security, Availability and Disaster Recovery Strategies
- Select appropriate Service-Level Agreements

Section 2: Relevant Knowledge

8.1 Business Case Planning for Cloud Adoption

What is a business case?

- Your organization depends on information technology (IT) for its operations, and probably for creating and supplying its products as well. It's a significant expense. For these reasons, a move to the cloud must be carefully considered and planned.
- A business case provides a view of the technical and financial timeline of your environment and can represent the opportunities for reinvestment into further modernization.
- Developing a business case includes building a financial plan that takes technical considerations into account and aligns with business outcomes.

- It helps you foster support from your Finance team and other areas of the business, helps accelerate cloud migration, and enables business agility.

Key Components of a Business Case

When you're planning your business case to migrate to the cloud, there are several key components to consider.

Environment scope, technical and financial

- As you build out the on-premises view of your environment, think about how your environment scope, from both a technical and financial perspective, is aligned.
- You want to be sure the technical environment you're using for your plan matches up to the financial data.

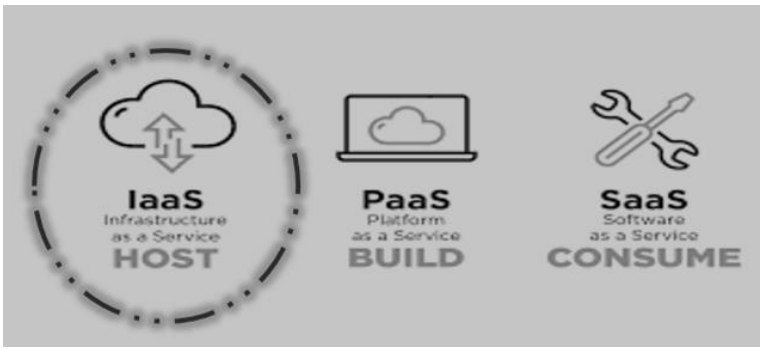
Baseline financial data: Cost to run today

- When you build out your business case, it's important to pull your baseline financial data. Common questions you can ask to gather the financial data needed are:
  - How much does it cost to run my environment today?
  - What am I spending on servers in an average year?
  - What am I spending in my data center operations categories, for example, power or lease costs?
  - When is the next hardware refresh?

Three Types of Cloud Compared

Infrastructure as a Service (IaaS)

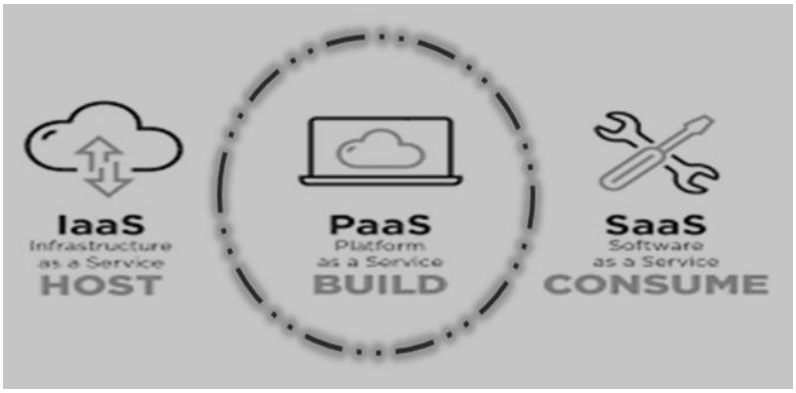
- The CPU, data storage, bandwidth and an operating system delivered as a flexible service (ordered and provisioned incrementally).
- Each customer can then load their application software stack on top, taking advantage of this easily-resized Cloud Infrastructure (CI) on demand.
- This allows for scaling up or down as needed, providing a huge advantage to businesses in terms of flexibility and preservation of capital.



Platform as a Service (PaaS)

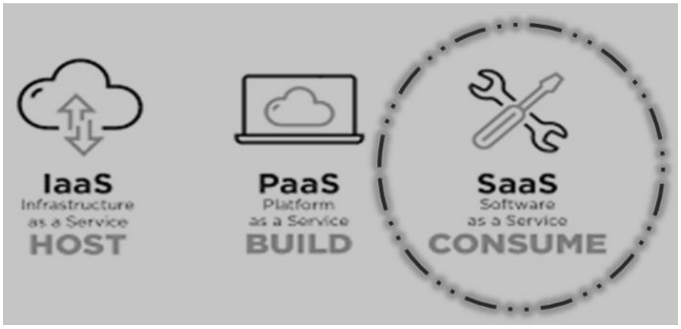
- Simply, this adds to CI a full software stack; for example, Linux.
- Each customer is then able to write or load applications into this environment, with the provider responsible for expanding or contracting all elements to adapt to the changing requirements of the users.
- Where a high degree of availability or service is required, this provides an impressive advantage for businesses looking to react and scale rapidly.

customization or control.



Software as a Service (SaaS)

- This takes CI and Cloud Platform (CP) and adds a fully managed application, with examples of this being salesforce.com, dropbox.com or facebook.com.
- The user will consume these apps, incrementally and usually on a per-user basis, with very little long-term commitment.
- This does provide advantages to businesses needing an application for a short-term or test basis, or in a business where pure applications and dedicated staff are scarce or expensive. But, this format also provides very little customization or control.



Approach to Building a Business Case

- In building a business case, ultimately, we are driving an expression of ROI. And, eventually, this leads into comparison between the deployment of capital (CapEx) vs. the preservation of capital - aka the deployment of operational expense (OpEx).
- This is purely a traditional “buy vs build” analysis and mostly straight forward and we'll present examples of this type of analysis and calculation within this series. However when it comes to the cloud, there are different approaches to building the business case to ensure all unique costs are truly identified:
  - Infrastructure business case
  - Applications business case
  - Talent business case

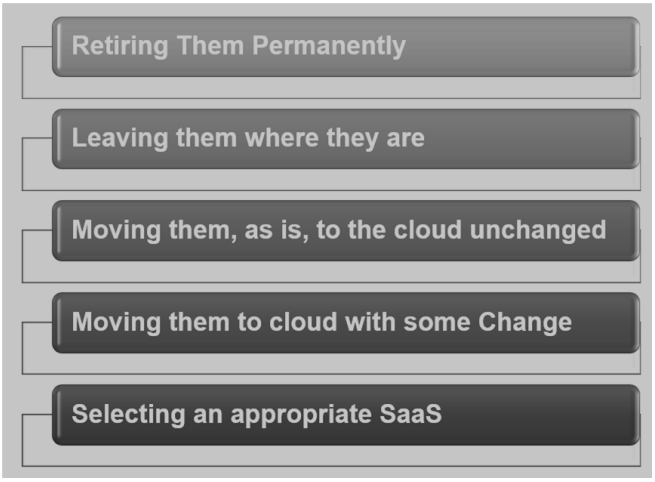
Infrastructure Case

- Usually, there is some compelling event to initiate a move to the cloud, such as a compute upgrade due to increased demands from users or applications, end-of-life of data centre facility assets or a facility move where everything needs to be built again.
- The initial and most significant savings are usually found when abandoning infrastructure in favour of the cloud, with infrastructure savings being the most significant part of the business case in terms of cost savings.
- The reason why is that in-house IT is typically under-utilised, because when infrastructure purchases are being considered, not all applications that will be deployed on it are known and so a margin is added for this misunderstood capacity requirement.
- Additionally, over-deployment is a result of companies configuring infrastructure for peak loads.
- We add to these the other fixed and variable costs we identified previously: cost of specialised data centre assets to house, power and cool servers, the cost of real estate which includes carrying finance charges, lease costs and other terms, the skilled staffing costs of maintaining the data centre and the systems within it.
- Other costs include back-ups, redundancy at a second facility, certifications, security and decommissioning costs when moving to the cloud.

Applications case

- With the three main types of cloud there are a number of options of what to do with applications, but this requires a look at what the main drivers are.
- This includes a major fork-lift to update an in-house custom application, a shift to a new application requiring higher availability and performance, and scarcity of maintenance resources such as talent and quality control. Therefore, there's many options for then dealing with the applications.
- Each of these reasons will have different cost implications, which will need to be outlined for the business case. For example, leaving one application where it is, but moving others, places a greater share of the remaining infrastructure costs on that application.

Reasons to Move Applications to the Cloud



8.2 Calculating the Financial Implication

How to Calculate Cloud Computing Cost

- Determining the potential costs of cloud computing can be as complex as the technology itself.
- Not only are you faced with the different pricing structures of cloud providers, you must find a reasonable way to estimate the resources that you will need in the future.
- Despite your best efforts, there is no guarantee that your calculations will be correct. That’s because of the variable cost architecture of the cloud.

From CAPEX to OPEX

- The traditional computing model was dependent on significant capital expenditures (CAPEX).
- The one-time purchase of hardware, software and licenses meant that companies had to squeeze as much work out of these resources as possible throughout the life cycle of these platforms.
- There was always a strong focus on the maintenance and configuration of proprietary machines. This meant that vendor support was essential to keeping systems current and healthy.

Cost Centers in the Cloud

The three cost centres of the cloud

- Compute
- Storage
- Network

These provide the outline for the calculation of cloud costs. But these broad areas don't account for everything. The options and variables related to cloud usage can make predicting costs something of a guessing game. But the main components give us something to work with.

Compute

- The costs for compute depend entirely on what you're going to do with it. How much processing power is required for your computing projects? A common way to deal with computing capability has been to purchase more than you need. Better to have too much than too little, as they say. But, the scalability of cloud computing means that you can take a much different approach.
- If you have a temporary project that is entirely contingent upon the response of website visitors, you can use cloud computing to scale up automatically and scale back down just as quickly. The computing systems that you set up with cloud providers can purchase for on-demand usage or for a fixed period of time.
- The parameters involved in selecting a CPU include the operating system and the expected usage (in percent). Cloud providers will then calculate the cost of CPU based on their cost per gigabyte (GB) of virtual RAM.

Cost Centres in the Cloud

Storage

- Advancements in storage technology have brought down prices considerably.
- It is no longer necessary to have dedicated hardware for each client or project.
- Virtual disks have replaced their physical counterparts.
- The same scalability in the compute sphere applies to storage as well. Storage is calculated in units of GB of virtual disk.

Network

- This area is generally measured in GB of data transfer. But bandwidth is also calculated in terabytes (TB) or petabytes (PB).
- While these are the main cost centers, not everything that a cloud provider offers will fit neatly in these three categories.
- Each provider packages their offerings in different ways. It might even seem like comparing apples to oranges when putting one service up against another.

Multicloud Infrastructure

- A further complication in calculating cloud computing costs is the fact that most companies are turning to multiple providers to handle their cloud computing needs.
- The research firm IDC found that 84 percent of IT executives expect to use multiple clouds from different providers. It's not enough to determine what your favorite cloud company charges if you are going to spread out your cloud presence to more than one company.
- The three biggest cloud providers, according to Gartner, are
  - Amazon Web Services (AWS)
  - Microsoft
  - Google
- Companies like Cisco and Nutanix offer support for the management of a multicloud strategy. They can even use automated processes to redirect traffic to more efficient or economical resources – even if it means dynamically moving to another cloud provider.

8.3 Comparing In-house Facilities to the Cloud

Adoption and Consumption Strategies

The selection of strategies for enterprise cloud computing is critical for IT capability as well as for the earnings and costs the organization experiences, motivating efforts toward convergence of business strategies and IT. Some critical questions toward this convergence in the enterprise cloud paradigm are as follows:

- Will an enterprise cloud strategy increase overall business value?
- Are the effort and risks associated with transitioning to an enterprise cloud strategy worth it?
- Which areas of business and IT capability should be considered for the enterprise cloud?
- Which cloud offerings are relevant for the purposes of an organization?
- How can the process of transitioning to an enterprise cloud strategy be piloted and systematically executed?

These questions are addressed from two strategic perspectives:

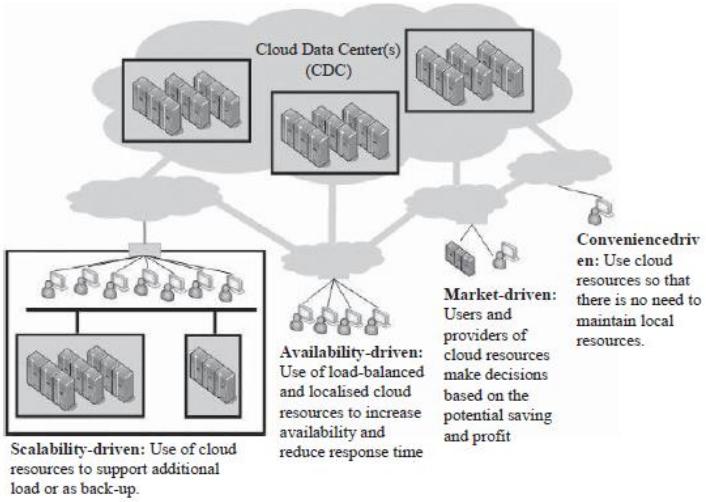
(1) Adoption
(2) Consumption

where an organization makes a decision to adopt a cloud computing model based on fundamental drivers for cloud computing— scalability, availability, cost and convenience.

Enterprise cloud adoption strategies

1. **Scalability-Driven Strategy:** The objective is to support increasing workloads of the organization without investment and expenses exceeding returns.
2. **Availability-Driven Strategy:** Availability has close relations to scalability but is more concerned with the assurance that IT capabilities and functions are accessible, usable and acceptable by the standards of users.
3. **Market-Driven Strategy:** This strategy is more attractive and viable for small, agile organizations that do not have (or wish to have) massive investments in their IT infrastructure on their profiles and requests service requirements.
4. **Convenience-Driven Strategy:** The objective is to reduce the load and need for dedicated system administrators and to make access to IT capabilities by users easier, regardless of their location and connectivity (e.g. over the Internet)

Enterprise cloud adoption strategies



Enterprise cloud adoption strategies using fundamental cloud drivers

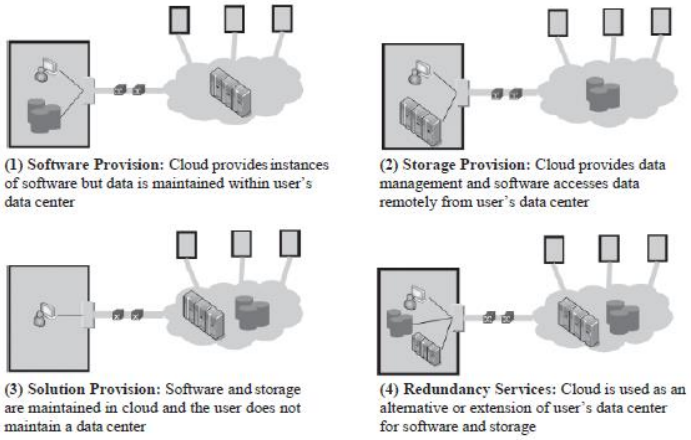
There are four consumptions strategies identified, where the differences in objectives, conditions and actions reflect the decision of an organization to trade-off hosting costs, controllability and resource elasticity of IT resources for software and data. These are discussed in the following:

1. Software Provision: This strategy is relevant when the elasticity requirement is high for software and low for data, the controllability concerns are low for software and high for data, and the cost reduction concerns for software are high, while cost reduction is not a priority for data, given the high controllability concerns for data, that is, data are highly sensitive.

2. Storage Provision: This strategy is relevant when the elasticity requirements is high for data and low for software, while the controllability of software is more critical than for data. This can be the case for data intensive applications, where the results from processing in the application are more critical and sensitive than the data itself.

3. Solution Provision: This strategy is relevant when the elasticity and cost reduction requirements are high for software and data, but the controllability requirements can be entrusted to the CDC.

4. Redundancy Services: This strategy can be considered as a hybrid enterprise cloud strategy, where the organization switches between traditional, software, storage or solution management based on changes in its operational conditions and business demands.



Enterprise cloud consumption strategies

Business Benefits of Cloud Computing

There are some clear business benefits to building applications in the cloud. A few of these are listed here:

Almost Zero Upfront Infrastructure Investment

- If you have to build a large- scale system, it may cost a fortune to invest in real estate, physical security, hardware (racks, servers, routers, backup power supplies), hardware management (power management, cooling), and operations personnel.
- Because of the high upfront costs, the project would typically require several rounds of management approvals before the project could even get started. Now, with utility-style cloud computing, there is no fixed cost or start-up cost

Just-in-Time Infrastructure

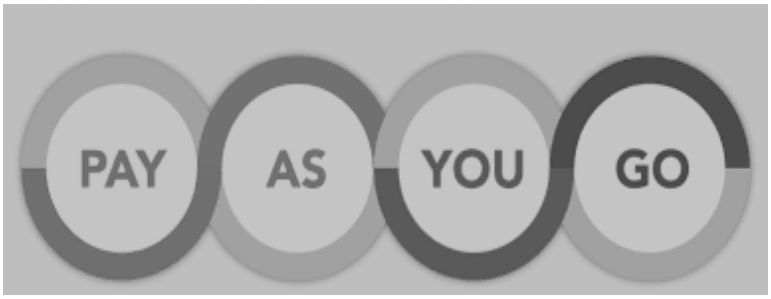
- In the past, if your application became popular and your systems or your infrastructure did not scale, you became a victim of your own success.
- By deploying applications in-the-cloud with just-in-time self-provisioning, you do not have to worry about pre-procuring capacity for large-scale systems.
- This increases agility, lowers risk, and lowers operational cost because you scale only as you grow and only pay for what you use.

More Efficient Resource Utilization

- System administrators usually worry about procuring hardware (when they run out of capacity) and higher infrastructure utilization (when they have excess and idle capacity).
- With the cloud, they can manage resources more effectively and efficiently by having the applications request and relinquish resources on-demand.

Usage-Based Costing

- With utility-style pricing, you are billed only for the infrastructure that has been used. You are not paying for allocated infrastructure but instead for unused infrastructure. This adds a new dimension to cost savings.



- You can see immediate cost savings (some- times as early as your next month's bill) when you deploy an optimization patch to update your cloud application.
- For example, if a caching layer can reduce your data requests by 70%, the savings begin to accrue immediately and you see the reward right in the next bill. Moreover, if you are building platforms on the top of the cloud, you can pass on the same flexible, variable usage-based cost structure to your own customers.

Reduced Time to Market

- Parallelization is one of the great ways to speed up processing.
- If one compute-intensive or data-intensive job that can be run in parallel takes 500 hours to process on one machine, with cloud architectures, it would be possible to spawn and launch 500 instances and process the same job in 1 hour.
- Having available an elastic infrastructure provides the application with the ability to exploit parallelization in a cost-effective manner reducing time to market.

Technical Benefits of Cloud Computing

Some of the technical benefits of cloud computing includes:

Automation— "Scriptable Infrastructure"

- You can create repeatable build and deployment systems by leveraging programmable (API-driven) infrastructure.
- **Auto-scaling***:* You can scale your applications up and down to match your unexpected demand without any human intervention. Auto-scaling encourages automation and drives more efficiency.

Proactive Scaling

Scale your application up and down to meet your anticipated demand with proper planning understanding of your traffic patterns so that you keep your costs low while scaling.

More Efficient Development Life Cycle

Production systems may be easily cloned for use as development and test environments. Staging environments may be easily promoted to production.

Improved Testability

Never run out of hardware for testing. Inject and automate testing at every stage during the development process. You can spawn up an "instant test lab" with preconfigured environments only for the duration of testing phase.

Disaster Recovery and Business Continuity

The cloud provides a lower cost option for maintaining a fleet of DR servers and data storage. With the cloud, you can take advantage of geo-distribution and replicate the environment in other location within minutes.

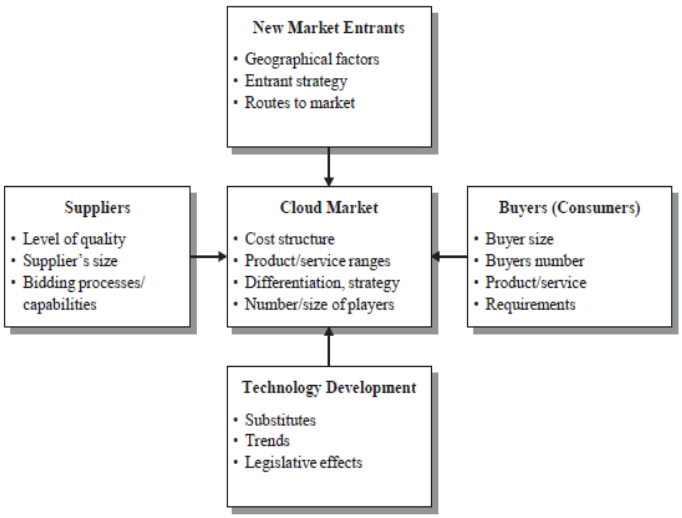
"Overflow" the Traffic to the Cloud

With a few clicks and effective load balancing tactics, you can create a complete overflow-proof application by routing excess traffic to the cloud.

8.4 Estimating Economic Factors Downstream

Business Drivers Toward a Marketplace

- In order to create an overview of offerings and consuming players on the market, it is important to understand the forces on the market and motivations of each player.
- The Porter model consists of five influencing factors/views (forces) on the market. The intensity of rivalry on the market is traditionally influenced by industry-specific characteristics:



Porter's five forces market model (adjusted for the cloud market)

8.5 Safeguarding Access to Assets in the Cloud

Security Best Practices

- In a multi-tenant environment, cloud architects often express concerns about security.
- Physical security is typically handled by your service provider (Security Whitepaper, which is an additional benefit of using the cloud. Network and application-level security is your responsibility, and you should implement the best practices as applicable to your business.
- It is recommended to take advantage of these tools and features mentioned to implement basic security and then implement additional security best practices using standard methods as appropriate or as they see fit.
- Security should be implemented in every layer of the cloud application architecture.

Protect Your Data in Transit

- If you need to exchange sensitive or confidential information between a browser and a Web server, configure SSL on your server instance. You'll need a certificate from an external certification authority like VeriSign or Entrust.
- The public key included in the certificate authenticates your server to the browser and serves as the basis for creating the shared session key used to encrypt the data in both directions.
- Create a virtual private cloud by making a few commands line calls (using Amazon VPC). This will enable you to use your own logically isolated resources within the AWS cloud, and then connect those resources directly to your own data center using industry-standard encrypted IPSec VPN connections.
- You can also set up an OpenVPN server on an Amazon EC2 instance and install the OpenVPN client on all user PCs.

Protect your Data at Rest

- If you are concerned about storing sensitive and confidential data in the cloud, you should encrypt the data (individual files) before uploading it to the cloud.
- For example, encrypt the data using any open source or commercial PGP-based tools before storing it as Amazon S3 objects and decrypt it after download.
- This is often a good practice when building HIPPA-compliant applications that need to store protected health information (PHI).
- On Amazon EC2, file encryption depends on the operating system. Amazon EC2 instances running Windows can use the built-in Encrypting File System (EFS) feature available in Windows. This feature will handle the encryption and decryption of files and folders automatically and make the process transparent to the users.

Secure Your Application

- Every Amazon EC2 instance is protected by one or more security groups that is, named sets of rules that specify which ingress (i.e., incoming) network traffic should be delivered to your instance.
- You can specify TCP and UDP ports, ICMP types and codes, and source addresses. Security groups give you basic firewall-like protection for running instances.

8.6 Security, Availability and Disaster Recovery Strategies

What is disaster recovery in cloud computing?

- Disaster recovery (DR) is the process that goes into preparing for and recovering from a disaster.
- This disaster could take one of a number of forms, but they all end up in the same result: the prevention of a system from functioning as it normally does, preventing a business from completing its daily objectives.

What kind of disasters should you prepare for?

There are three main categories of disaster that can affect businesses:

Natural disasters: Natural disasters such as floods or earthquakes are rarer but not infrequent. If a disaster strikes an area that contains a server that hosts the cloud service you're using, this could disrupt services and require disaster recovery operations.

Technical disasters: Perhaps the most obvious of the three, technical disasters encompass anything that could go wrong with the cloud technology. This could include power failures or a loss of network connectivity.

Human disasters:

- Human failures are a common occurrence and are usually accidents that happen whilst using the cloud services. These could include inadvertent misconfiguration or even malicious third-party access to the cloud service.

- The cloud providers are responsible for everything they have direct control over. This includes the resiliency of the general infrastructure such as the hardware, software, network and facilities. You, the customer, are usually responsible for areas such as the cloud configuration, secure data backups, the workload architecture and the availability.

Why is disaster recovery important?

- Creating protocols and contingencies for disaster recovery is vital for the smooth operation of business. In the event of a disaster, a company with disaster recovery protocols and options can minimize the disruption to their services and reduce the overall impact on business performance.
- Minimal service interruption means a reduced loss of revenue which, in turn, means user dissatisfaction is also minimised.
- Having plans for disaster in place also means your company can define its Recovery Time Objective (RTO) and its Recovery Point Objective (RPO). The RTO is the maximum acceptable delay between the interruption and continuation of the service and the RPO is the maximum amount of time between data recovery points.
- Quantifying these areas can help your company identify its optimal protection level for disaster recovery and choose the right protocols to implement such as backups and multiple servers.

What are some examples of cloud computing disasters?

Although uncommon, disasters in cloud computing have occured in the past and even to some of the largest cloud providers such as AWS.

OVHCloud

A data centre run by OVHCloud was destroyed in early 2021 by a fire. All four data centres had been too close, and it took over six hours for firefighters at the scene to put out the blaze. This severely affected the cloud services run by OVHCloud and spelt disaster for companies whose entire assets were hosted on those servers.

AWS

- In June 2016, storms in Sydney battered the electrical infrastructure and caused an extensive power outage. This led to the failure of a number of Elastic Compute Cloud instances and Elastic Block Store volumes which hosted critical workloads for a number of large companies.
- This meant that some heavily trafficked websites and the online presence of some of the biggest brands was decimated for over ten hours on a weekend, severely affecting business.=

Amazon

In February 2017 an Amazon employee was attempting to debug an issue with the billing system when they accidentally took more servers offline than they needed to. This started a domino effect that removed two other server subsystems which then snowballed to other subsystems. This meant that thousands of people were unable to access Amazon servers for a few hours.

What are the benefits of cloud disaster recovery in the cloud?

- Using the cloud for cloud disaster recovery means that data backups don’t have to be maintained by the customer on disks or physical hard drives.
- The distributed nature of the cloud means that services can be spread out to different servers in different geographical locations, essentially providing complete protection against local natural disasters.
- Some of the responsibility can be offloaded onto the cloud provider.
- The cloud provider is responsible for the core resilience of the infrastructure of the cloud, removing this worry from the customer.
- Cloud disaster recovery using the cloud also proves to be cost-effective. Because cloud providers only charge for the services that they use, your business can pick and choose which services it wants from the provider.
- This leads to a huge cost reduction by increasing the personalization of the package that your business pays for.

How should you prepare your recovery plans, step by step?

Here are 5 steps that can help you prepare a recovery plan:

1. Your disaster recovery plan should be part of your business continuity plan

This should involve definitions of RTO and RPO to help you decide which cloud services you'll need and improve cost efficiency.

2. If you haven't done so already, define the RTO and RPO for your disaster recovery

This forms the basis of your disaster recovery plan and, in turn, the kinds of disaster recovery services you'll need.

3. Design your plan with your recovery goals in mind

This involves looking at your RTO and RPO points to decide which disaster recovery pattern you'll need to meet those criteria. Your recovery goals should outline the maximum and minimum affects to your services

4. Design for end-to-end recovery

Your plan should include recovery for every aspect of your business that needs to be operational.

5. Create specific tasks to ensure a smooth-running process

The more specific your tasks are, the easier the recovery process will be and the fewer chances there will be of deviating from the plan.

8.7 Selecting Appropriate Service-Level Agreements

SLA Management in Cloud Computing

- In the early days of web-application deployment, performance of the application at peak load was a single important criterion for provisioning server resources.
- The capacity build up was to cater to the estimated peak load experienced by the application.
- The activity of determining the number of servers and their capacity that could satisfactorily serve the application end-user requests at peak loads is called capacity planning.
- Enterprises developed the web applications and deployed on the infrastructure of the third-party service providers.
- These providers get the required hardware and make it available for application hosting. Typically, the QoS parameters are related to the availability of the system CPU, data storage, and network for efficient execution of the application at peak loads. This legal agreement is known as the service-level agreement (SLA).

STL academy

Types of SLA

- Service-level agreement provides a framework within which both seller and buyer of a service can pursue a profitable service business relationship. It outlines the broad understanding between the service provider and the service consumer for conducting business and forms the basis for maintaining a mutually beneficial relationship.
- There are two types of SLAs from the perspective of application hosting. These are described in detail here.

Infrastructure SLA: The infrastructure provider manages and offers guarantees on availability of the infrastructure, namely, server machine, power, network connectivity, and so on.

Application SLA: In the application co-location hosting model, the server capacity is available to the applications based solely on their resource demands. Therefore, the service

Key Components of a Service-Level Agreement

Service-Level Parameter	Describes an observable property of a service whose value is measurable.
Metrics	These are definitions of values of service properties that are measured from a service-providing system or computed from other metrics and constants. Metrics are the key instrument to describe exactly what SLA parameters mean by specifying how to measure or compute the parameter values.
Function	A function specifies how to compute a metric's value from the values of other metrics and constants. Functions are central to describing exactly how SLA parameters are computed from resource metrics
Measurement directives	These specify how to measure a metric.

Hardware availability	99% uptime in a calendar month
Power availability	99.99% of the time in a calendar month
Data centre network availability	99.99% of the time in a calendar month
Backbone network availability	99.999% of the time in a calendar month
Service credit for unavailability	Refund of service credit prorated on downtime period
Outage notification guarantee	Notification of customer within 1 hr of complete downtime
Internet latency guarantee	When latency is measured at 5-min intervals to an upstream provider, the average doesn't exceed 60 msec
Packet loss guarantee	Shall not exceed 1% in a calendar month

Key contractual components of an application SLA

Service-level parameter metric	• Web site response time (e.g., max of 3.5 sec per user request)
	• Latency of web server (WS) (e.g., max of 0.2 sec per request)
	• Latency of DB (e.g., max of 0.5 sec per query)
Function	• Average latency of WS = (latency of web server 1 +latency of web server 2) /2
	• Websiteresponsetime= Averagelatencyofwebserver+ latency ofdatabase
Measurement directive	• DB latency available via http://mgmtserver/em/latency
	• WS latency available viahttp://mgmtserver/ws/instanceno/ latency
Service-level objective	• Service assurance
	• website latency , 1 sec when concurrent connection , 1000
Penalty	• 1000 USD for every minute while the SLO was breached

Challenges for Provisioning the Infrastructure on Demand

From the SLA perspective there are multiple challenges for provisioning the infrastructure on demand. These challenges are as follows:

- The application is a black box to the MSP and the MSP has virtually no knowledge about the application runtime characteristics.
- The MSP needs to understand the performance bottlenecks and the scalability of the application.
- The MSP analyses the application before it goes on-live. However, subsequent operations/enhancements by the customer's to their applications or auto updates beside others can impact the performance of the applications, thereby making the application SLA at risk.
- The risk of capacity planning is with the service provider instead of the customer.

Life Cycle of SLA

- Each SLA goes through a sequence of steps starting from identification of terms and conditions, activation and monitoring of the stated terms and conditions, and eventual termination of contract once the hosting relationship ceases to exist.
- Such a sequence of steps is called SLA life cycle and consists of the following five phases:
  1. Contract definition
  2. Publishing and discovery
  3. Negotiation
  4. Operationalization
  5. De-commissioning

Here, we explain in detail each of these phases of SLA life cycle.

Contract Definition

Generally, service providers define a set of service offerings and corresponding SLAs using standard templates.

Publication and Discovery

Service provider advertises these base service offerings through standard publication media, and the customers should be able to locate the service provider by searching the catalogue.

Negotiation

Once the customer has discovered a service provider who can meet their application hosting need, the SLA terms and conditions needs to be mutually agreed upon before signing the agreement for hosting the application.

Operationalization

SLA operation consists of SLA monitoring, SLA ac- counting, and SLA enforcement. SLA monitoring involves measuring parameter values and calculating the metrics defined as a part of SLA and determining the deviations.

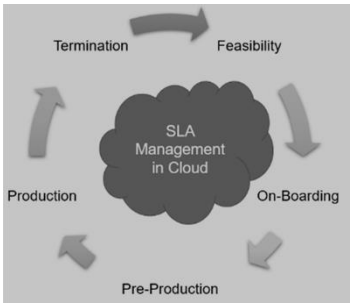
De-commissioning

SLA decommissioning involves termination of all activities performed under a particular SLA when the hosting relationship between the service provider and the service consumer has ended.

SLA Management in Cloud

SLA management of applications hosted on cloud platforms involves five phases.

1. Feasibility
2. On-boarding
3. Pre-production
4. Production
5. Termination



Feasibility Analysis

- MSP conducts the feasibility study of hosting an application on their cloud platforms. This study involves three kinds of feasibility:
  (1) Technical Feasibility
  (2) Infrastructure Feasibility
  (3) Financial Feasibility
- The technical feasibility of an application implies determining the following:
  1. Ability of an application to scale out.
  2. Compatibility of the application with the cloud platform being used within the MSP's data center.
  3. The need and availability of a specific hardware and software required for hosting and running of the application.
  4. Preliminary information about the application performance and whether they can be met by the MSP.
- Performing the infrastructure feasibility involves determining the availability of infrastructural resources in sufficient quantity so that the projected demands of the application can be met.

On-Boarding of Application

- Once the customer and the MSP agree in principle to host the application based on the findings of the feasibility study, the application is moved from the customer servers to the hosting platform.
- The application is accessible to its end users only after the on- boarding activity is completed.

On-boarding activity consists of the following steps:

a. Packing of the application for deploying on physical or virtual environments. Application packaging is the process of creating deployable components on the hosting platform (could be physical or virtual). Open Virtualization Format (OVF) standard is used for packaging the application for cloud platform.

b. The packaged application is executed directly on the physical servers to capture and analyse the application performance characteristics.

c. The application is executed on a virtualized platform and the application performance characteristics are noted again.

d. Based on the measured performance characteristics, different possible SLAs are identified. The resources required and the costs involved for each SLA are also computed.

e. Once the customer agrees to the set of SLOs and the cost, the MSP starts creating different policies required by the data center for automated management of the application. These policies are of three types:
   (1) Business
   (2) Operational
   (3) Provisioning

- Business policies help prioritize access to the resources in case of contentions.

Preproduction

- Once the determination of policies is completed as discussed in previous phase, the application is hosted in a simulated production environment.
- Once both parties agree on the cost and the terms and conditions of the SLA, the customer sign-off is obtained. On successful completion of this phase the MSP allows the application to go on-live.

Production

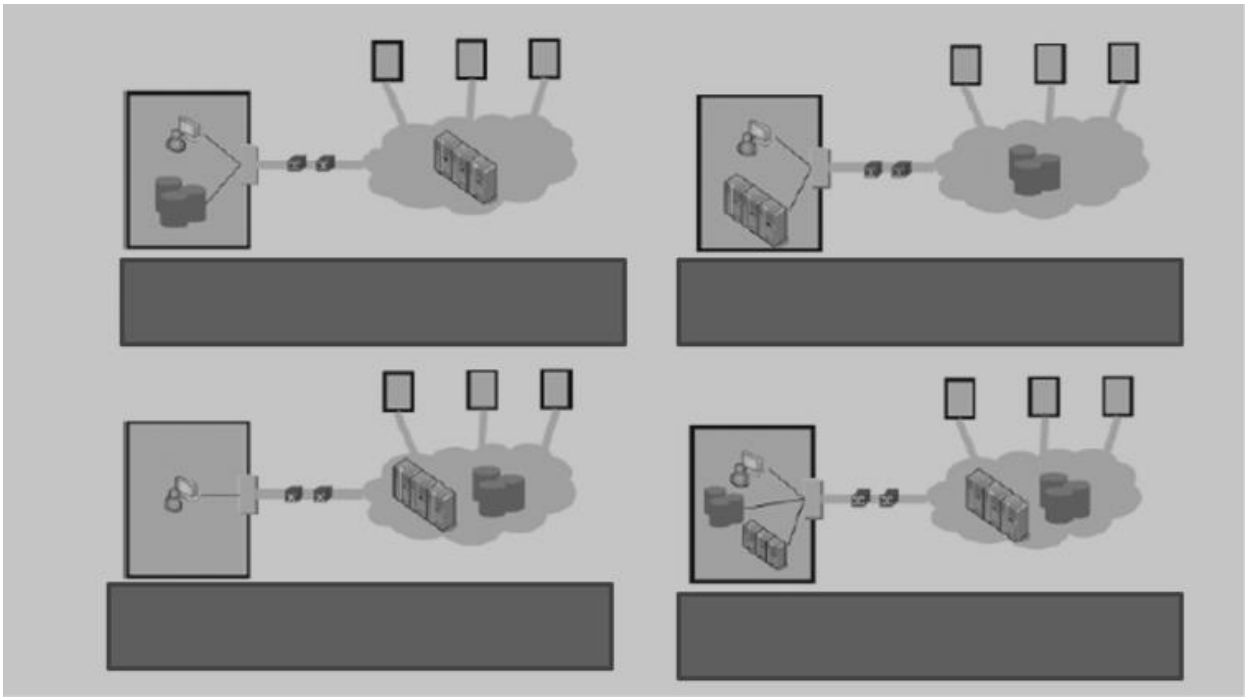
- In this phase, the application is made accessible to its end users under the agreed SLA.
- In the case of the former, on-boarding activity is repeated to analyze the application and its policies with respect to SLA fulfilment. In case of the latter, a new set of policies are formulated to meet the fresh terms and conditions of the SLA.

Termination

When the customer wishes to withdraw the hosted application and does not wish to continue to avail the services of the MSP for managing the hosting of its application, the termination activity is initiated

Section 3: Exercise

Exercise 1: Write down the name of all enterprise cloud consumption strategies in below diagram**.**



Exercise 2: Participate in the group discussion on following topics:

a) Calculation of the Financial Implications
b) Comparing In-house facilities to the Cloud
c) Estimation of economic factors downstream
d) Safeguarding access to Assets in the Cloud
e) Adopting the Security, Availability and Disaster Recovery Strategies
f) Selecting appropriate Service-Level Agreements

Section 4: Assessment Questionnaire

1. What are the business benefits of cloud computing?
2. Explain SLA in Cloud computing?
3. What are the types of SLA?
4. List the Key Components of SLA?
5. Explain Phases Life cycle of SLA?
6. List out the Security practices?
7. What kind of Disaster happen? How to recover disaster in Cloud Computing?

----------End of the Module----------

MODULE 9
MIGRATING TO THE CLOUD

Section 1: Learning Outcomes

After completing this module, you will be able to:

- Explain about the Technical Consideration for Cloud Migration
- Define the term 'Cloud Migration'
- Re-architect applications for the cloud
- Integrate the cloud with existing applications
- Avoid vendor lock-in
- Plan the migration and selecting a vendor

Section 2: Relevant Knowledge

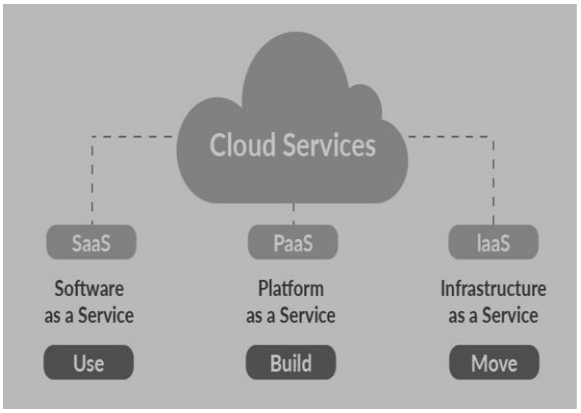
9.1 Re-architecting Applications for the Cloud

What Is Cloud Application Migration?

- The term "application migration" refers to the process of shifting software applications between computing environments.
- The process may apply to moving applications between a public cloud to a private cloud or moving applications from a local server to a cloud environment.
- Cloud migration helps organizations leverage the advantages of the cloud for their applications, including cost reduction, a higher level of scalability, and quick application updates.

What Are Your Cloud Migration Options?

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)



Software Migration Challenges to Overcome

Here are some of the main challenges involved in software migration.

Unexpected Costs

- When you migrate an application, you could face unexpected costs resulting from the complexity of the migration process.
- For example, you may have to train staff in using the new system or toolset, requiring extra hours and expenses. For your migration to be successful, you need to assess the expected costs realistically, considering potential complications.

Disruptions and Downtime

- Migration can impact processes that are critical to your business functions. If you experience an unanticipated outage, you may lose customers and revenue.
- To reduce unexpected downtime, you should consider the potential issues that may affect performance so you can address them in advance.

Maintaining Privacy

- It is essential to protect the privacy of your business operations and data when migrating to a third-party system, such as a cloud server. Whenever you work with a third-party vendor, you need to carefully oversee the migration process and ensure the proper SLAs are in place.

Maintaining Compliance

- You need to ensure that the new environment is compliant with regulations such as HIPAA.
- It is important to have a compliance strategy in place before you begin the migration process to find suitable vendors and solutions.

Stakeholder Commitment Issues

- Migration projects often take a long time to complete, testing the commitment of key stakeholders.
- You need to have a clearly defined long-term with measurable targets to help keep team leaders and department heads on board.

Using Different Systems Simultaneously

- Organizations typically migrate applications gradually to maintain business continuity, resulting in a period of overlap between the data and functions of the old and new environments.
- This overlap can create confusion as to which system should be used for each task. You need to have a clear plan outlining the data storage requirements of each migration phase.

Application Migration Plan Stages

Your application migration plan is key to making the process manageable. While the specifics of a migration plan differ for each organization, any application migration plan should address the following basic elements.

Identify and Assess Your Applications

- First, you need to discover and audit all applications used in your enterprise environment. You should assess the importance and complexity of your applications, categorizing them as business-critical or non-critical.
- An application assessment should include any requirements for modifications or re-coding, helping you decide whether to migrate or replace the application.

Determine Which Legacy Applications to Migrate

- Most organizations continue to use legacy applications long after the introduction of new technologies.
- You might want to keep your legacy applications to avoid the expense or disruption of acquiring a replacement—as long as they perform adequately.
- When migrating to a new environment, especially in the cloud, legacy applications can be difficult to migrate or maintain. You can migrate some applications unchanged or with minor alterations but replacing other applications with cloud-compatible alternatives could be cheaper.

Calculate Your TCO

- Software migration carries a significant risk of unanticipated costs. Review your application migration plan to evaluate the total cost of ownership (TCO).
- You can compare various scenarios to see which options strike an acceptable balance between cost savings and performance. Consider factors such as the maintenance costs, the cost of replacing or acquiring new applications, and training.

Assess the Project Duration and Identify Potential Risks

Do your best to forecast the likely duration of your migration project and consider the risks of unexpected hurdles. Your forecast will not be perfect, but it should help reduce the risk of overblown costs and disruptions.

Managed Application Migration

Cloud providers offer managed services that can make it easier to migrate your applications to the cloud. Here are a few types of application migration services you can use to plan, execute, and automate an application migration.

Migration Blueprint

- In a complete blueprint service offer, your vendor helps you define your migration objectives and strategy by recognizing your users' needs and your organizational requirements.
- They also collect details about your environment and applications, developing a complete action plan for the migration process.

Migration Deployment

- If you select a managed deployment, your vendor helps you strategize and plan your migration.
- They also help you manage the migration and any related troubleshooting and testing. This method is typically a turn-key option that features full-scale and end-to-end support.

Cloud Managed Services

- A managed cloud service option provides observation and maintenance of your cloud-based IT environment.
- Your managed cloud service provider takes responsibility for functions, including acquiring as-a-service providing on your behalf.
- They also manage cloud security. Application migration may also be part of the packaged service.

Application Modernization

- Application modernization services provide custom development services.
- They can help you prepare legacy applications for utilization in the cloud, by adapting them to run in virtualized environments or containers.

9.2 Migrating to the Cloud

What is Cloud Migration?

- Cloud migration is the process of moving data, applications or other business elements to a cloud computing environment.
- There are various types of cloud migrations an enterprise can perform. One common model is to transfer data and applications from a local on-premises data center to the public cloud.
- A cloud migration could also entail moving data and applications from one cloud platform or provider to another; this model is known as cloud-to-cloud migration.
- A third type of migration is a reverse cloud migration, cloud repatriation or cloud exit, where data or applications are moved off of the cloud and back to a local data center.

What are the Key Benefits of Cloud Migration?

Scalability

- Cloud computing can scale to support larger workloads and more users, much more easily than on-premises infrastructure.
- In traditional IT environments, companies had to purchase and set up physical servers, software licenses, storage and network equipment to scale up business services.

Cost

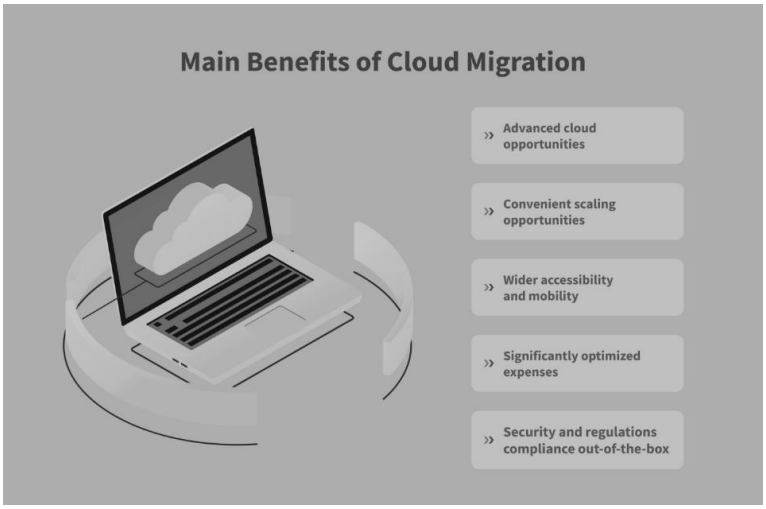
- Cloud providers take over maintenance and upgrades, companies migrating to the cloud can spend significantly less on IT operations.
- They can devote more resources to innovation - developing new products or improving existing products.

Performance

Migrating to the cloud can improve performance and end-user experience. Applications and websites hosted in the cloud can easily scale to serve more users or higher throughput, and can run in geographical locations near to end-users, to reduce network latency.

Digital experience

Users can access cloud services and data from anywhere, whether they are employees or customers. This contributes to digital transformation, enables an improved experience for customers, and provides employees with modern, flexible tools.



What are Common Cloud Migration Challenges?

Cloud migrations can be complex and risky. Here are some of the major challenges facing many organizations as they transition resources to the cloud.

Lack of Strategy

- Many organizations start migrating to the cloud without devoting sufficient time and attention to their strategy.
- Successful cloud adoption and implementation requires rigorous end-to-end cloud migration planning.
- Each application and dataset may have different requirements and considerations, and may require a different approach to cloud migration.
- The organization must have a clear business case for each workload it migrates to the cloud.

Cost Management

- When migrating to the cloud, many organizations have not set clear KPIs to understand what they plan to spend or save after migration.
- This makes it difficult to understand if migration was successful, from an economic point of view. In addition, cloud environments are dynamic and costs can change rapidly as new services are adopted and application usage grows.

Vendor Lock-In

- Vendor lock-in is a common problem for adopters of cloud technology. Cloud providers offer a large variety of services, but many of them cannot be extended to other cloud platforms.
- Migrating workloads from one cloud to another is a lengthy and costly process. Many organizations start using cloud services, and later find it difficult to switch providers if the current provider doesn't suit their requirements.

Data Security and Compliance

- One of the major obstacles to cloud migration is data security and compliance. Cloud services use a shared responsibility model, where they take responsibility for securing the infrastructure, and the customer is responsible for securing data and workloads.
- So, while the cloud provider may provide robust security measures, it is your organization’s responsibility to configure them correctly and ensure that all services and applications have the appropriate security controls.
- The migration process itself presents security risks. Transferring large volumes of data, which may be sensitive, and configuring access controls for applications across different environments, creates significant exposure.

Migrating into a Cloud

- The promise of cloud computing has raised the IT expectations of small and medium enterprises beyond measure. Large companies are deeply debating it.
- Cloud computing is a disruptive model of IT whose innovation is part technology and part business model in short a disruptive techno-commercial model‖ of IT.
- We propose the following definition of cloud computing: —It is a techno-business disruptive model of using distributed large-scale data centers either private or public or hybrid offering customers a scalable virtualized infrastructure or an abstracted set of services qualified by service-level agreements (SLAs) and charged only by the abstracted IT resources consumed
- Several small and medium business enterprises, however, leveraged the cloud much beyond the cautious user. Many startups opened their IT departments exclusively using cloud services very successfully and with high ROI. Having observed these successes, several large enterprises have started successfully running pilots for leveraging the cloud.
- Many large enterprises run SAP to manage their operations. SAP itself is experimenting with running its suite of products: SAP Business One as well as SAP Netweaver on Amazon cloud offerings.

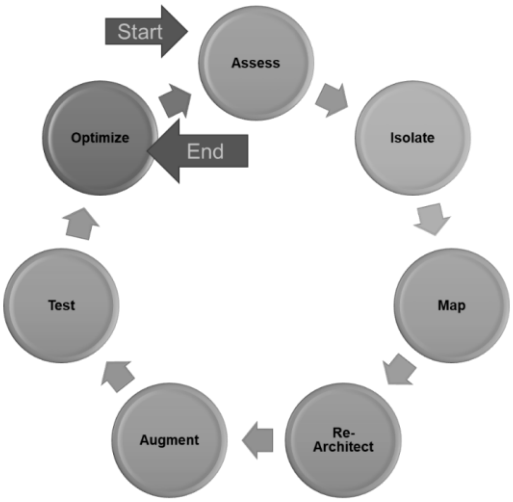
Broad Approaches to Migrating into the Cloud

Cloud Economics deals with the economic rationale for leveraging the cloud and is central to the success of cloud-based enterprise usage. Decision-makers, IT managers, and software architects are faced with several dilemmas when planning for new Enterprise IT initiatives.

The Seven-Step Model of Migration into A Cloud

Typically, migration initiatives into the cloud are implemented in phases or in stages. A structured and process-oriented approach to migration into a cloud has several advantages of capturing within itself the best practices of many migration projects.

1. Conduct Cloud Migration Assessments
2. Isolate the Dependencies
3. Map the Messaging & Environment
4. Re-architect & Implement the lost Functionalities
5. Leverage Cloud Functionalities & Features
6. Test the Migration
7. Iterate and Optimize



Migration Risks and Mitigation

- The biggest challenge to any cloud migration project is how effectively the migration risks are identified and mitigated.
- In the Seven-Step Model of Migration into the Cloud, the process step of testing and validating includes efforts to identify the key migration risks.
- In the optimization step, we address various approaches to mitigate the identified migration risks.
- Migration risks for migrating into the cloud fall under two broad categories:
  - General migration risks
  - Security-related migration risks

- In the former we address several issues including:
  - Performance monitoring and tuning essentially identifying all possible production level deviants
  - The business continuity and disaster recovery in the world of cloud computing service
  - The compliance with standards and governance issues; the IP and licensing issues
  - The quality of service (QoS) parameters as well as the corresponding SLAs committed to
  - The ownership, transfer, and storage of data in the application; the portability and interoperability issues which could help mitigate potential vendor lock-ins
  - The issues that result in trivializing and non-comprehending the complexities of migration that results in migration failure and loss of senior management‘s business confidence in these efforts.

AWS Migration Best Practices

Leverage AWS Tools

AWS offers a wide range of tools designed for the migration process, from the initial planning phase to features for post-migration. Here are several useful tools to consider:

AWS Migration Hub

- A dashboard that centralizes data and helps you monitor and track the progress of migration.
- AWS Application Discovery - collects data needed for pre-migration due diligence.
- TSO Logic - offers data-driven recommendations based on predictive analytics. The recommendations are tailored to help during the planning and strategizing phase.

AWS Server Migration Service

- Provides automation, scheduling, and tracking capabilities for incremental migrations.
- AWS Database Migration Service - keeps the source data store fully-operational while the migration is in process, to minimize downtime.
- Amazon S3 Transfer Acceleration - improves the speed of data transfers made to Amazon S3, to maximize available bandwidth.

Automate Repetitive Tasks

- The migration process typically involves many repetitive tasks. You can perform these tasks manually, and you can automate them.
- The main purpose of automation is to enable you to achieve a higher level of efficiency while reducing costs. In many cases, automation can also help you complete tasks much faster than manually possible.

Outline and Share a Clear Cloud Governance Model

- A cloud governance model defines and specifies the practices, roles, responsibilities, tools, and procedures involved in the governance of your cloud environments.
- Your model needs to be as clear as possible, to ensure all relevant stakeholders understand how cloud resources should be managed and used. Ideally, you should define this information before migrating.
- Here are several questions your cloud governance model should answer:
  - What controls are set in place to meet security and privacy requirements?
  - How many AWS accounts are maintained?
  - What privileges are enabled for each role?

There are many more considerations to address in your cloud governance model, depending on your industry and business needs. Be sure to keep your documentation flexible to allow for change and optimization after the migration process is completed and your workloads settle in the new cloud environment.

Azure Migration Best Practices

Azure Migration Tools

Azure offers several migration tools designed to simplify and automate the migration process. Here are three commonly used Azure migration tools:

- **Azure Migrate** —helps you to assess your local workloads, determine the required size of cloud resources, and estimate cloud costs.
- **Microsoft Assessment and Planning**—helps you discover your servers and applications and build an inventory. Additionally, this tool can create reports that determine whether Azure can support your workloads.
- **Azure Database Migration Service**—helps you migrate on-premise SQL Server workloads to Azure.

Cost Management in Azure

- Cloud resources are highly accessible and flexible, but costs can quickly skyrocket if you don't have a cost management strategy in place.
- Here are several tools and techniques you can use to manage your cloud costs:

Tag your resources

- To manage costs, you need visibility into cloud resource consumption.
- You can set this up by tagging resources and monitoring them. Be sure to use standard tags and keep this organized.

Use policies - to automate tagging and monitoring.

- Cloud resources are highly scalable and this can make manual tagging and monitoring incredibly time consuming.
- Use policies to standardize the process and automation to enforce these rules.
- You can leverage either third-party and first-party tools for tagging.
- There are also tools dedicated to cost management and optimization and monitoring. In addition, you can set up role-based access control (RBAC) to ensure resources are properly used by authorized users, and set up several resource groups.

Review Every Policy and Procedure

- Policies and procedures are a foundational component of the migration process and heavily impact the success of the implementation.
- To ensure your migration runs smoothly, you should define and review all policies and then apply them in a cohesive and standardized manner.
- Properly implementing security can ensure all required security measures are set in place. Policies are not only responsible for enforcing security, but also help you achieve and maintain compliance. Data encryption, for example, is a component you can enforce using a policy.
- Once you define your policies and procedures, you should test them before running in production.
- You can automate this process using several tools. Azure Migrate, for example, can help you automatically identify, assess, and migrate your local VMs to the Azure cloud.

Google Cloud Migration Best Practices

- **Moving Data**

Here are several aspects to consider when migrating to Google Cloud:

- **Move your data first** - and then move the rest of the application. This is recommended by Google.

- **Choose the relevant storage** - Google Cloud offers several tiers for hot and warm storage, as well as several archiving options. You can also leverage SSDs and hard discks, or choose a cloud-based database service, such as Bigtable, Datastore, and Google Cloud SQL.

- **Plan the data transfer process** - determine and define how to physically move your data. You can, for example, send your offline disk to a Google data center or opt to stream to persistent disks.

- **Moving Applications**

There are several ways to migrate applications, depending on the application's suitability to the cloud. In some cases, you might need to re-architect the entire application before it can be moved to the cloud. In other cases, you might need to do light modification before the migration. Ideally, when possible, your application can be lifted and shifted to the cloud.

A lift and shift migration means you do not need to make any changes to your application. You can lift it and move it directly to the new cloud environment. For example, you can create a local VM within your on-premise center, and then import it as a Google VM. Alternatively, you can backup your application to GCP - this option lets you automatically create a cloud copy.

- **Optimize**

After the migration process is complete and your application is safely hosted in the cloud, you need to set up measures that help you continuously optimize your cloud environment. Here are several tools offered by Google:

- **Google Cloud operations suite (Stackdriver)** - provides features that enable full observability into your Google cloud environment. The information is centralized in a single database that lets you run queries and leverage root-cause analysis to gain detailed insights.
- **Google Cloud Pub/Sub** - helps you set up communication between any independent applications. You can use Pub/Sub to rapidly scale, decouple applications, and improve performance.
- **Google Cloud Deployment Manager** - lets you automate the configuration of your applications. You specify the requirements and Deployment Manager automatically initiates the deployments.

What Is a Cloud First Strategy?

- White House CIO Vivek Kundra coined the term “cloud-first”, referring to the practice of preferring the cloud as a first option for building programs and applications.
- A cloud-first strategy promotes building software directly in the cloud rather than building on-premises and migrating to the cloud. The goal is to help you create software faster and reduce the overhead associated with on-premises resources and cloud migration.

Why Should a Cloud-First Approach be Considered?

Here are key advantages of a cloud-first approach:

- **Flexibility**- build your systems piece by piece according to business needs.
- **Less overhead**- a cloud-first strategy lowers or eliminates the overhead associated with equipment and maintenance costs incurred when using on-premises server solutions.
- **More resources**- cloud vendors provide access to additional services, which typically require lower or no initial investment.
- **Cost-effective upgrades**- cloud vendors offer various pricing options you can leverage to reduce the costs of upgrades on-demand.
- **Support**- cloud service providers offer support for their services, provided by experts.
- **Quick release**- working directly in the cloud can help you achieve a faster speed of delivery for repairs, improvements, and updates.
- **Collaboration**- cloud services often provide collaboration tools that enable you to work remotely, using numerous device types to access tools, storage, and data from any location.

Cloud First Challenges and Considerations

Cloud-First Security Challenges

- Many organizations continue to rely on legacy security protocols established in pre-cloud or sometimes pre-web times. These legacy systems are complex or sometimes impossible to implement successfully in the cloud.
- There are steps your organization can adopt to ensure your cloud-first strategy prioritizes cloud security. Central to these strategies is a focused DevSecOps approach, uniting development securities and operations into a collaborative team to improve testing and efficiency and reduce time-to-market.
- Here are steps you can adopt to secure critical data and resources when using a cloud-first approach:
  - **Foster organizational alignments**- protecting cloud native applications should be the shared responsibility of all project teams and departments.
  - **Secure the application lifecycle**- build security into the integration and deployment stages using practice, including vulnerability remediation and code scanning. You should also automatically apply runtime management with integrations.
  - **Limit privileges**- use a policy of least privilege of your most employees and users and only give access when necessary. This approach will reduce perimeter data leaks caused by human error.
  - **Deploy runtime protection**- next generation firewalls (NGFW) and web application firewalls (WAF) can help monitor request traffic and compare it to normal behavior to identify anomalies and block threats.

End-to-End Application Performance

- In recent years, the cloud has been approaching the edge. Certain use cases demand stringent measures regarding application performance, making it difficult for cloud-based solutions to meet latency demands for some critical applications.
- Storage-intensive applications responsible for processing hundreds of TBs of data every day are an example of performance limitations that affect the suitability of cloud-based solutions.

Vendor Lock-In

- Even when organizations gain effective control over cloud deployments, there are hidden costs of vendor lock-in. Enterprise-grade commercial agreements with cloud providers are rigid and difficult to change over time, as an organization’s requirements change.
- While the market is heading in a good direction, customer protections in cloud agreements are not comparable to those offered by other IT outsourcing contracts. Without good commercial protection, organizations can unknowingly give away future flexibility.

Business Continuity and Disaster Recovery

- Even before the pandemic and the world's mass adoption of remote infrastructure, cloud-based solutions demanded that the industry rethink traditional BC/DR approaches. Cloud providers typically provide data solutions stored and backed up in several locations.
- Cloud-based failover protection is not guaranteed. For example, global-scale cyber-attacks can affect multiple cloud data centers, and locations with a high concentration of cloud data centers can be severely impacted by natural disaster, which can cause ripple effects worldwide.

How to Adopt a Cloud-First Strategy Approach

Learn from Your Peers

- A helpful step in creating a cloud-first strategy is learning from others' experiences. Look towards organizations that have effectively navigated the cloud migration process.
- You can ask questions about how they achieve their goals and their long-term aims for their solution.

Build a Cloud-First Culture

- The success of your organization's cloud-first strategy depends on cooperation from the top down. To make this possible, you will need to initiate a culture shift to the cloud-first approach—emphasizing transparency.
- Don't shy away from employees' apprehensions from the onset. Be approachable so that employees can come to you with questions before, during, and after implementation. Also, it helps employees understand how cloud migration will make their roles simpler.
- Many organizations approach a cloud-first culture shift through educational initiatives and employee engagement. For instance, an organization could create a cloud training program for technical and non-technical employees. Such a program could help employees understand how the technology works and the impact that it will have on their jobs.

Create a Cloud-First Migration Roadmap

- Like any major project, having a cloud migration plan is key. Create a roadmap specific to your organization that has all your solutions. Outline each step in your cloud migration approach.
- Establish a migration path for each application you have, from your most recent applications to your legacy applications: select private, public or hybrid cloud deployment.

9.3 Planning the Migration and Selecting a Vendor

What is a Cloud Strategy Roadmap?

- A cloud strategy roadmap is a visual communication tool that describes how your organization will migrate to the cloud. It includes key tasks, deliverables, and deadlines.
- IT teams use roadmaps to put their cloud migration strategy on track and hold all stakeholders accountable.
- According to Gartner, cloud strategy roadmaps should have at least five parts: aligning objectives, planning, preparing for execution, governance, optimization, and collaboration.
- Because cloud migration projects are complex and involve multiple parts of an organization, developing a cloud strategy roadmap is not a simple task.
- You should follow a structured process to ensure all relevant stakeholders are on board and align your roadmap with available resources and operational considerations.

What Questions Should You Ask When Developing a Cloud Roadmap?

Addressing the "why"

The best way to start building your cloud roadmap is to start with the why. Why are you migrating? What are the benefits? Why should other members of your organization join your cloud vision?

Addressing the "how"

When you approach a cloud roadmap, you'll need concrete answers to the technical challenges of migration. Ask yourself how you'll migrate workloads to the cloud, how they will operate in a hybrid environment, and which cloud migration method is the most appropriate—lift-and-shift, refactoring, or rebuilding.

Addressing cultural factors

- Another category of questions involves the people in your organization. Don't underestimate the importance of culture. Technology is often the easier part of cloud transformation; changing workflows people are accustomed to is more challenging.
- How will you encourage people in your organization to cooperate with the migration, and what will you do to ensure it impacts them positively? Empathy, professional development, and support are as important as choosing between cloud providers or cloud-native technologies.

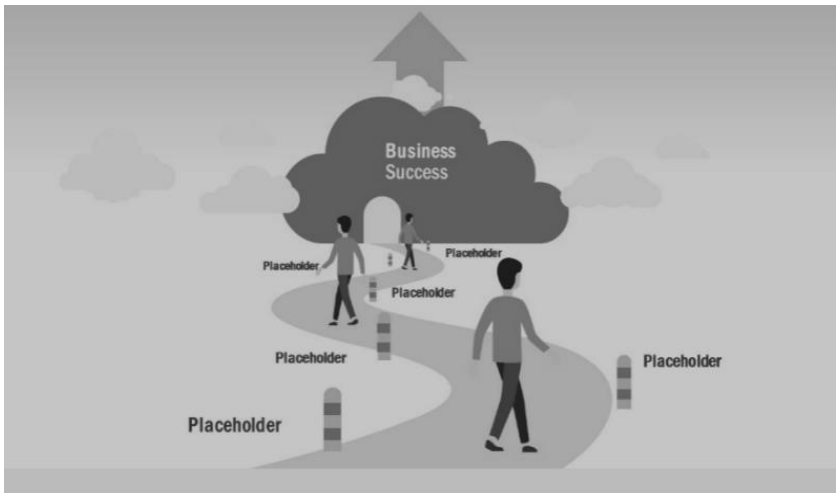
Addressing the "what"

- An essential part of your roadmap is what you will migrate.
- Ask yourself which workloads will move to the cloud, which are easier to migrate, and which are more challenging.
- Define datasets that will move to the cloud and critical aspects like data sensitivity and availability requirements.

Define success

Ask yourself what will make your cloud migration a success. Are you aiming to shut down the on-premises data center or move all new development to the cloud? Define the organization's ultimate goal with specific metrics to measure migration success.

Cloud Roadmap Strategy: Five Must-Have Stages



According to research by Gartner, the ideal cloud migration roadmap consists of five steps.

Align Objectives

- Organizations should create a cloud migration value proposition for business and IT early in the cloud migration roadmap. Start by conducting a survey to understand the use cases for cloud adoption, aligning cloud strategy with IT goals, and defining action steps to achieve your goals.
- Another important aspect is to define migration principles based on application and team readiness, business priorities, and vendor capabilities. Use data available in the organization to define the metrics and key performance indicators (KPIs) for a successful migration.

Develop a Plan of Action

- Choose the right cloud provider and negotiate a successful contract. At this stage, you should build cloud capabilities across the organization, assess alternative service providers, and prepare to mitigate cloud-related risks. Identify the necessary investments in your network, security, identity architecture, and other tools.
- At this stage, determine whether to migrate your entire environment to the cloud, or one workload at a time. Identify if your organization requires a multi-cloud environment or a single cloud provider will suffice. Think about the long term—will the capabilities of your cloud provider fit your needs in the future, and how will costs grow over time given your future growth?

Prepare for Execution

- At this stage, you deploy and optimize workloads in the cloud. Deployment involves identifying workloads for migration, defining your cloud management workflow, adopting implementation best practices, and analyzing how workloads perform in the cloud.
- Managing cloud migration as a structured, well-defined process can help an organization significantly improve the efficiency and effectiveness of cloud workload management.

Establish Governance While Mitigating Risk

- The goal of a successful cloud migration includes setting up robust processes to minimize disruption to your workflow.
- To be successful, you must discover, analyze and monitor sensitive data throughout your cloud deployment. Set up a security control plane using a third-party tool with suitable functionality.
- Take a lifecycle approach to governance, it is important to realize that you must continuously maintain governance to be effective.
- Governance and compliance feedback should be an integral part of your workflow, leveraging automation.

Optimize and Scale

- At this stage, workloads are already running successfully in the cloud.
- Consider investments that can improve existing use of the cloud and address operational challenges.
- Define customer-centric goals, communicating to teams how improved cloud use can benefit the organization.
- Align all stakeholders around the need to continuously develop and optimize your cloud presence.

Collaborate

Cloud migration processes can only succeed by achieving cooperation between cross-departmental teams. The following roles should be included in your roadmap and in relevant planning stages:

- **CIO**—provides strategic and planning guidance and can help define the goals of cloud migration. The CIO can help communicate progress to other stakeholders.

- **Development leaders and teams**—provide technical advice and can help establish a vision. They can work with other IT leaders to define specific cloud migration plans using up-to-date progress and planning information.

- **Operations leaders and teams**—provide insight into the infrastructure and operations requirements of cloud migration and determine activities required to implement the strategy. They will typically manage the operational mechanisms needed to enable the migration.

- **Cloud experts**—any cloud migration program will benefit from a team of cloud experts, either in-house or outsourced, who can provide architectural and process plans for the project. They can help evaluate and select the best tools and processes for migrating and refactoring systems and help build the required skills among other teams.

Cloud Migration Strategies

Gartner has identified five cloud migration techniques, known as the “5 Rs”. Organizations looking to migrate to the cloud should consider which migration strategy best answers their needs. The following is a brief description of each:

Rehost

Rehosting, or ‘lift and shift,’ involves using infrastructure-as-a-service (IaaS). You simply redeploy your existing data and applications on the cloud server. This is easy to do and is thus suited for organizations less familiar with cloud environments. It is also a good option for cases where it is difficult to modify the code, and you want to migrate your applications intact.

Refactor

Refactoring, or ‘lift, tinker, and shift,’ is when you tweak and optimize your applications for the cloud. In this case, a platform-as-a-service (PaaS) model is employed. The core architecture of the applications remains unchanged, but adjustments are made to enable the better use of cloud-based tools.

Revise

Revising builds upon the previous strategies, requiring more significant changes to the architecture and code of the systems being moved to the cloud. This is done to enable applications to take full advantage of the services available in the cloud, which may require introducing major code changes. This strategy requires foreplanning and advanced knowledge.

Rebuild. Rebuilding takes the Revise approach even further by discarding the existing code base and replacing it with a new one. This process takes a lot of time and is only considered when companies decide that their existing solutions don’t meet current business needs.

Replace

Replacing is another solution to the challenges that inform the Rebuild approach. The difference here is that the company doesn’t redevelop its own native application from scratch. This involves migrating to a third-party, prebuilt application provided by the vendor. The only thing that you migrate from your existing application is the data, while everything else about the system is new.

Cloud Migration Strategic Process

The cloud migration steps or processes an enterprise follows will vary based on factors such as the type of migration it wants to perform and the specific resources it wants to move. That said, common elements of a cloud migration strategy include the following:

- Evaluation of performance and security requirements
- Selection of a cloud provider
- Calculation of costs
- Any reorganization deemed necessary

At the same time, be prepared to address several common challenges during a cloud migration:

- Interoperability
- Data and application portability
- Data integrity and security
- Business continuity

Without proper planning, a migration could degrade workload performance and lead to higher IT costs -- thereby negating some of the main benefits of cloud computing.

STL academy

A 4-Step Cloud Migration Process

1. Cloud Migration Planning

- One of the first steps to consider before migrating data to the cloud is to determine the use case that the public cloud will serve. Will it be used for disaster recovery? DevOps? Hosting enterprise workloads by completely shifting to the cloud? Or will a hybrid approach work best for your deployment.
- In this stage it is important to assess your environment and determine the factors that will govern the migration, such as critical application data, legacy data, and application interoperability.
- It is also necessary to determine your reliance on data: do you have data that needs to be resynced regularly, data compliance requirements to meet, or non-critical data that can possibly be migrated during the first few passes of the migration?
- Determining these requirements will help you charter a solid plan for the tools you'll need during migration, identifying which data needs to be migrated and when, if the data needs any scrubbing, the kind of destination volumes to use, and whether you'll need encryption of the data both at rest and in transit.

2. Migration Business Case

- Once you have determined your business requirements, understand the relevant services offered by cloud providers and other partners and their costs.
- Determine the expected benefits of cloud migration along three dimensions: operational benefits, cost savings, and architectural improvements.
- Build a business case for every application you plan to migrate to the cloud, showing an expected total cost of ownership (TCO) on the cloud, compared to current TCO.
- Use cloud cost calculators to estimate future cloud costs, using realistic assumptions - including the amount and nature of storage used, computing resources, taking into account instance types, operating systems, and specific performance and networking requirements.
- Work with cloud providers to understand the options for cost savings, given your proposed cloud deployment.
- Cloud providers offer multiple pricing models, and provide deep discounts in exchange for long-term commitment to cloud resources (reserved instances) or a commitment to a certain level of cloud spend (savings plans). These discounts must be factored into your business plan, to understand the true long-term cost of your cloud migration.

3. Cloud Data Migration Execution

- Once your environment has been assessed and a plan has been mapped out, it's necessary to execute your migration.
- The main challenge here is carrying out your migration with minimal disruption to normal operation, at the lowest cost, and over the shortest period of time.
- If your data becomes inaccessible to users during a migration, you risk impacting your business operations.
- The same is true as you continue to sync and update your systems after the initial migration takes place.
- Every workload element individually migrated should be proven to work in the new environment before migrating another element.
- You'll also need to find a way to synchronize changes that are made to the source data while the migration is ongoing.
- Both AWS and Azure provide built-in tools that aid in AWS cloud migration and in Azure data migration, and later in this article we'll see how NetApp users benefit from migrating with services and features that come with Cloud Volumes ONTAP.

4. Ongoing Upkeep

- Once that data has been migrated to the cloud, it is important to ensure that it is optimized, secure, and easily retrievable moving forward. It also helps to monitor for real-time changes to critical infrastructure and predict workload contentions.
- Apart from real-time monitoring, you should also assess the security of the data at rest to ensure that working in your new environment meets regulatory compliance laws such as HIPAA and GDPR.
- Another consideration to keep in mind is meeting ongoing performance and availability benchmarks to ensure your RPO and RTO objectives should they change.

Cloud migration deployment models

- Enterprises today have more than one cloud scenario from which to choose:
  - The public cloud lets many users access compute resources through the internet or dedicated connections.
  - A private cloud keeps data within the data center and uses a proprietary architecture.
  - The hybrid cloud model mixes public and private cloud models and transfers data between the two.
  - In a multi-cloud scenario, a business uses IaaS options from more than one public cloud provider.
- As you consider where the application should live, consider how well it will perform once it's migrated. Ensure there is adequate bandwidth for optimal application performance. Also, determine whether an application's dependencies may complicate a migration.
- Review what's in the stack of the application that will make the move.
- Local applications may contain a lot of features that go unused, and it is wasteful to pay to migrate and support those nonessential items.
- Stale data is another concern with cloud migration. Without a good reason, it's probably unwise to move historical data to the cloud, which typically incurs costs for retrieval.
- As you examine the application, it may be prudent to reconsider its strategic architecture to set it up for what could potentially be a longer life.
- A handful of platforms support hybrid and multi-cloud environments, including the following:
  - Microsoft Azure Stack;
  - Google Cloud Anthos;
  - AWS Outposts;
  - VMware Cloud on AWS; and
  - A container-based PaaS, such as Cloud Foundry or Red Hat OpenShift.

Best Practices to ensure Cloud Migration Success

- There are many reasons why an organization chooses to migrate an app or workload to the cloud, and each project will be unique depending on resource allocations, integrations with other services and multiple other factors.
- Here are some general guidelines for a cloud migration that streamline the process and improve changes for success:

Get organizational buy-in

The transition is much smoother when all stakeholders are on board and know their roles, from management to technical practitioners to end users.

Define cloud roles and ownership

Determine right upfront who is responsible to manage various aspects of the cloud workload. Is it a shared environment? How is identity confirmed and access granted, or limited? This includes proper documentation of setups and processes.

Pick the right cloud services

- Cloud providers have a vast menu of services to pick from.
- Be clear with which ones your workload will tap into, or you risk running extraneous services some of which may be interdependent and become problematic to manage.

Understand security risks

Cloud environments can be susceptible to mischief from internet attacks. Misconfigurations are arguably a bigger problem, given the complexity of cloud environments.

Calculate cloud costs

The cloud's pay-as-you-go model may seem attractive and simpler to organizations used to large infrastructure investments. But it's a double-edged sword: Pay close attention to service selections and usage, or you'll get a shock at the end of the month.

Devise a long-term cloud roadmap

- If a cloud migration is successful, organizations likely will look to replicate that success for other workloads.
- Identify the criteria to follow, from project timelines to different deployment options, such as a hybrid cloud setup.

Cloud Migration Tools and Services

The big IaaS providers -- AWS, Microsoft and Google -- offer various cloud migration services as well as free tiers. Here are a few examples:

	AWS	Azure	Google Cloud
Database migration	AWS Database Migration Service	Azure Database Migration Service	Database Migration Service (preview)
Data transfer appliance	Snow Family	Data Box	Transfer Appliance
Disaster recovery	CloudEndure Disaster Recovery	Azure Site Recovery	N/A
Online data transfer	AWS DataSync, AWS Transfer Family	Azure File Sync	BigQuery Data Transfer Service, Cloud Data Transfer
On-premises application analysis	AWS Application Discovery Service, Migration Evaluator	Azure Migrate, Movere, Azure Resource Mover	N/A
On-premises and cloud storage integration	Storage Gateway	StorSimple	N/A (offered by partner Cloudian)
Migration tracker	AWS Migration Hub	Azure Migrate	N/A
Server migration	AWS App2Container, AWS Server Migration Service, CloudEndure Migration	Azure Migrate	Migrate for Anthos, Migrate for Compute Engine, VM migration

Ready to migrate to the cloud? Answer these questions

Cloud computing ultimately frees an enterprise IT team from the burden of managing uptime. Placing an application in the cloud is often the most logical step for growth. A positive answer to some or all of these questions may indicate your company's readiness to move an app to the cloud.

Should your application stay or go?

Legacy applications, or workloads that require low latency or higher security and control, probably should stay on premises or move to a private cloud.

What's the cost to run an application in the cloud?

One of the primary benefits of a cloud migration is workload flexibility. If a workload suddenly needs more resources to maintain performance, its cost to run may escalate quickly.

Which cloud model fits best?

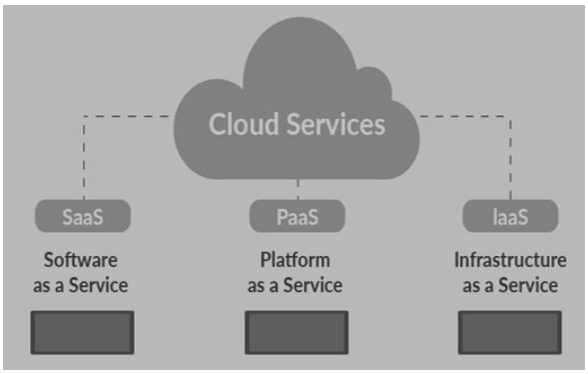
Public cloud provides scalability through a pay-per-usage model. Private or on-premises cloud provides extra control and security. A hybrid cloud model provides the best of both, although performance and connectivity may suffer.

How do I choose the right cloud provider?

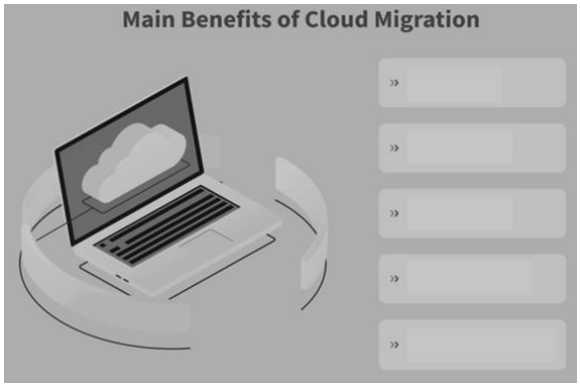
The top three cloud providers -- AWS, Microsoft and Google -- generally offer comparable services to run all kinds of workloads in the cloud, as well as tools to help you efficiently move apps there. Gauge your specific needs for availability, support, security and compliance, and pricing to find the best fit.

Section 3: Exercises

Exercise 1: Write down the Purpose of Use in Single word for Different Cloud Services in below Diagram.



Exercise 2: Write Down the Main Benefits of Cloud Migration.



Exercise 3: Participate in group discussion on following topics:

a) Re-architecting Applications for the Cloud
b) Cloud Migration Strategies
c) 4-Step Cloud Migration Process

Section 4: Assessment Questionnaire

d) Cloud Migration Tools and Services

1. Explain Cloud migration?
2. List few challenges while migrating to cloud?
3. List some advantages and disadvantages of Cloud Migration?
4. What are the tools for cloud migration Services?

----------End of the Module----------

STL academy

Empowering Youth!

STL is one of the industry's leading integrators of digital networks providing All-in 5G solutions. Our capabilities across optical networking, services, software, and wireless connectivity place us amongst the top optical players in the world. These capabilities are built on converged architectures helping telcos, cloud companies, citizen networks, and large enterprises deliver next-gen experiences to their customers. STL collaborates with service providers globally in achieving a green and sustainable digital future in alignment with UN SDG goals. STL has a global presence in India, Italy, the UK, the US, China, and Brazil

ISO 21001 | Guinness World Records Record Holder | Asia Book of Records | Skill & Assessment Partner NASSCOM®

stl.tech | stlacad.tech