Object Oriented Programming with Java

AKTU

Unit :- 1

Introduction

BCS-403

Introduction: Why Java, History of Java, JVM, JRE, Java Environment, Java Source File Structure, and Compilation. Fundamental,

What is JAVA

Java is a High level , Class based , Object Oriented Programming (OOP) Language developed by James Gosling in 1991.

Why Java

Java is a high Level programming language and it is also called as a platform. Java is a secured and robust high level object-oriented programming language.

Platform: Any software or hardware environment in which a program runs is known as a platform. Java has its own runtime environment (JRE) and API so java is also called as platform.

Java fallows the concept of **Write Once, Run Anywhere.**

Application of java

1. Desktop Applications
2. Web Applications
3. Mobile
4. Enterprise Applications
5. Smart Card
6. Embedded System
7. Games
8. Robotics etc

1) Core Java (J2SE)

2) Advance Java (J2EE)

3) Android Java (J2ME)

Features of Java

Simple
Java is very easy to learn and its syntax is simple, clean and easy to understand.

```
public class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World!");

    }

}
```

Object-oriented
Java is object-oriented programming language.

Platform Independent

Java is platform independent because it is different from other languages like C, C++ etc. which are compiled into platform specific machines while Java is a write once, run anywhere language.

Secured
Java is best known for its security. With Java, we can develop virus-free systems.

Robust

- Robust simply means strong.

Architecture-neutral

Java is architecture neutral because there is no implementation dependent features e.g. size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

Portable

Java is portable because it facilitates you to carry the java bytecode to any platform. It doesn't require any type of implementation.

High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g. C++). Java is an interpreted language that is why it is slower than compiled languages e.g. C, C++ etc.

Distributed

Java is distributed because it facilitates users to create distributed applications in java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.

Dynamic

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages i.e. C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

History of Java

1) **James Gosling, Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team** at **Sun Microsystem.**
2) Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.
3) Firstly, it was called **"Greentalk"** by James Gosling, and the file extension was .gt. After that, it was called **Oak** and was developed as a part of the Green project.
4) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.
5) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

6) Java is an island in Indonesia where the first coffee was produced (called Java coffee). Java name was chosen by James Gosling while having a cup of coffee nearby his office.

7) Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

8) JDK 1.0 was released on January 23, 1996.

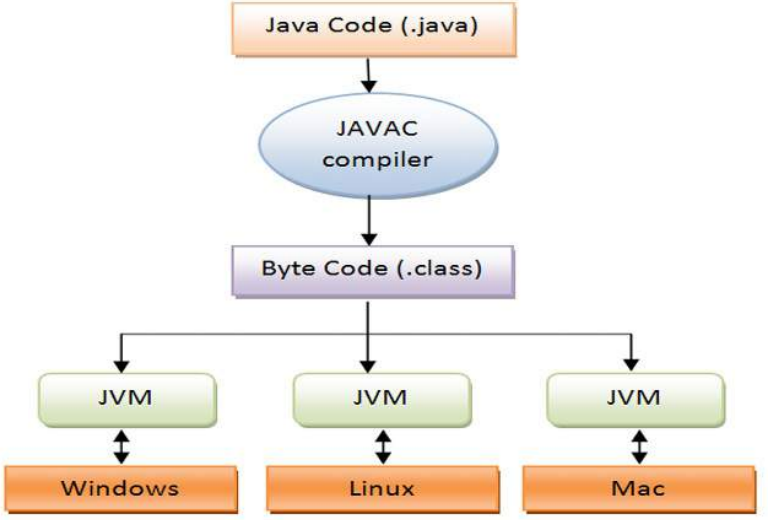
1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan 1996)
3. JDK 1.1 (19th Feb 1997)
4. J2SE 1.2 (8th Dec 1998)

17. Java SE 15 (September 2020)
18. Java SE 16 (Mar 2021)
19. Java SE 17 (September 2021)
20. Java SE 18 (to be released by March 2022)

After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds new features in Java.

JVM (Java Virtual Machine)

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed. JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).



Because of the above feature java is portable

The JVM performs following operation:

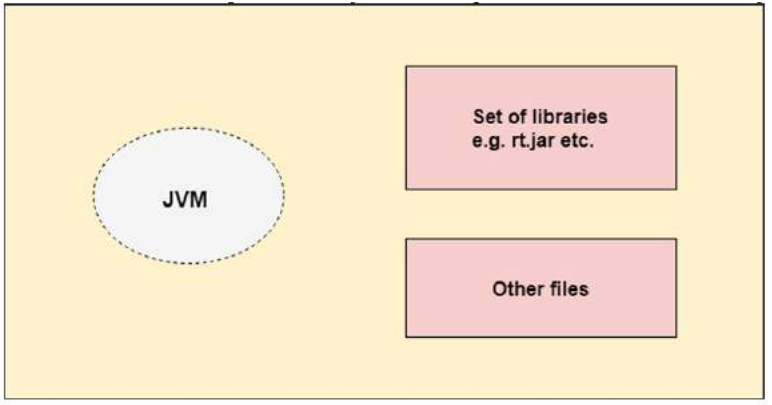
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

Java Runtime Environment (JRE)

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing java applications. It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.



The Java Runtime Environment, is a software layer that runs on top of a computer's operating system software and provides the class libraries and other resources that a specific Java program requires to run.

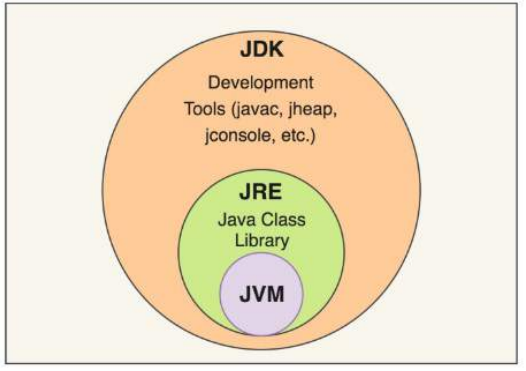
THE JAVA ENVIRONMENT

- Java Environment includes large number of development tools.
- The development tools are part of the system known as Java Development Kit (JDK).

1. Java Development Kit (JDK)

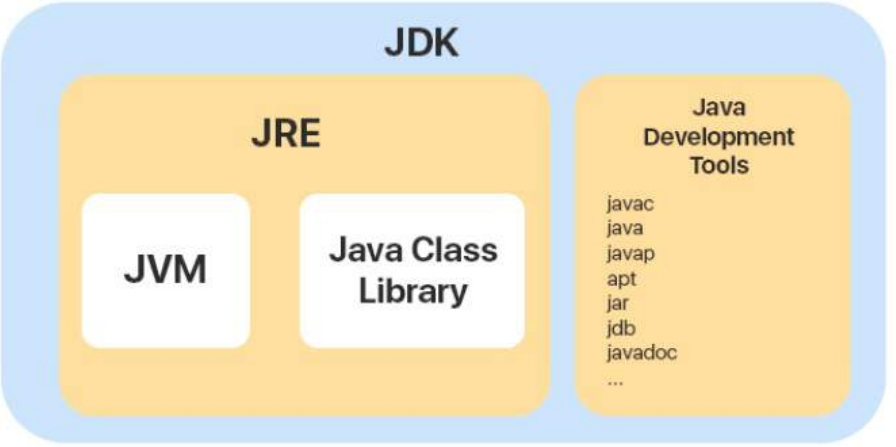
The Java Development Kit (JDK) comes with a collection of tools that are used for developing and running Java programs. They includes:

- ***appletviewer (for viewing java applets) :***
- Applets are small programs that can only run within a web browser.
- ***java (java interpreter):***
- A program that can analyse and executes your program line by line.
- ***javac (java compiler):***
- A program that can analyse and executes your whole program.



- ***javap (java disassembler) :***

- A disassembler is used to translate machine code (0 and 1) into a human readable format.



- ***javah :***

- Used for importing other programming files.

- ***javadoc :***

- Used in web based applications .

- ***jdb (java debugger) :***

- Debugging is the process of detecting and removing of existing errors .

Java Source File Structure

It is used to describe that the Java Source Code file must follow a scheme or structure. The maximum number of classes that may be declared as public in a Java program is one. If a public class exists, the program's name and the name of the public class must match for there to be no compile time errors. There are no limitations when using any name as the name of the Java source file if there is no public class.

Structure of Java Program

- package statement: In Java, a package is a way to gather together classes, sub packages, and interfaces.
- import statements: A package, class, or interface is imported using an import statement.
- class definition: A class is a passive entity that serves as a user-defined blueprint or template from which objects are formed.

```
package example;     //package
import java.util.*;   //import statement

class demo
{                     // class definition
   int x;
}
```

Procedure to write simple java Program

To write a java program First we have install the JDK.

- install the JDK and install it.
- set path of the jdk
- create the java program
- compile and run the java program

STRUCTURE OF JAVA PROGRAM

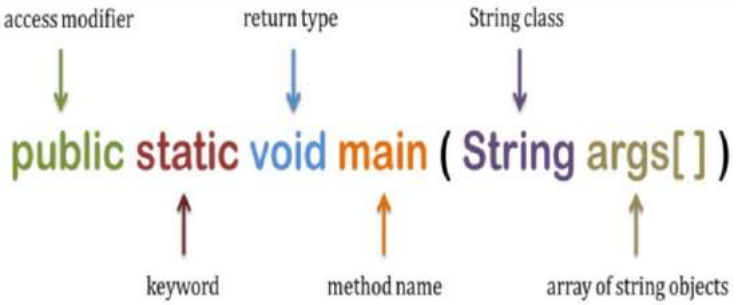
A first Simple Java Program

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

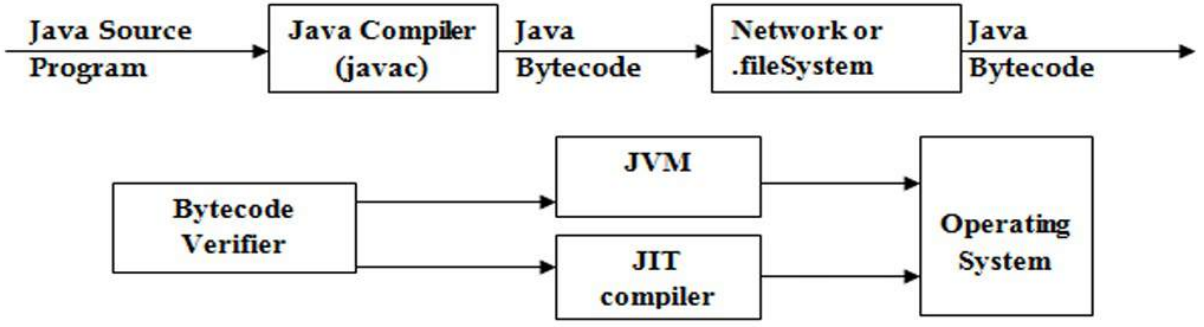
To compile:
javac Simple.java
To execute:
java Simple

class keyword is used to declare a class in java.
public keyword is an access modifier which represents visibility, it means it is visible to all.
static is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
void is the return type of the method, it means it doesn't return any value.
main represents the starting point of the program.
String[] args is used for command line argument.
System.out.println() is used print statement.

Compilation and Fundamentals



A program is written in JAVA, the javac compiles it. The result of the JAVA compiler is the .class file or the bytecode and not the machine native code (unlike C compiler).



The bytecode generated is a non-executable code and needs an interpreter to execute on a machine. This interpreter is the JVM and thus the Bytecode is executed by the JVM.
And finally program runs to give the desired output.

Programming Structures in Java

Programming Structures in Java: Defining Classes in Java, Constructors, Methods, Access Specifies, Static Members, Final Members, Comments, Data types, Variables, Operators, Control Flow, Arrays & String.

DEFINING CLASSES IN JAVA

Class – A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.

The class is at the core of Java .A class is a *template* for an object, and an object is an *instance* of a class. A class is declared by use of the **class** keyword

Syntax of class:

```
class classname
{       type instance-variable;
        type methodname1(parameter-list)
        {
                // body of method
        }
        type methodname2(parameter-list)
        {
                // body of method
        }
}
```

The data, or variables, defined within a class are called instance variables. The code is contained within methods. Collectively, the methods and variables defined within a class are called members of the class. In most classes, the instance variables are acted upon and accessed by the methods defined for that class

CONSTRUCTORS

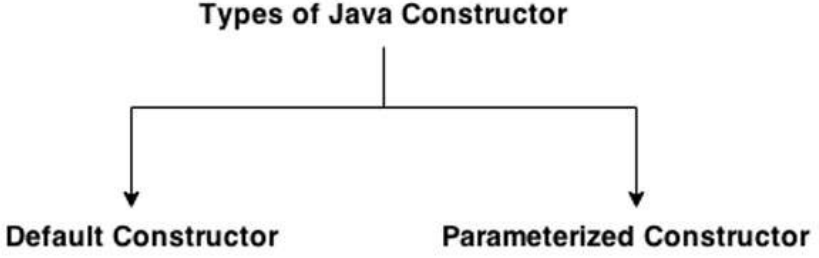
Constructor is special member function ,it has the same name as class name. It is called when an instance of object is created and memory is allocated for the object.

It is a special type of method which is used to initialize the object

Rules for creating java constructor

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type




```
public class MyClass{
    // Constructor
    MyClass(){
        System.out.println("BeginnersBook.com");
    }
    public static void main(String args[]){
        MyClass obj = new MyClass();
        ...
    }
}
```

New keyword creates the object of MyClass & invokes the constructor to initialize the created object.

A constructor is called "Default Constructor" when it doesn't have any parameter.

A constructor which has a specific number of parameters is called parameterized constructor.

Methods

A **method** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

Why use methods? To reuse code: define the code once, and use it many times.

Create a Method

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**. Java provides some pre-defined methods, such as `System.out.println()`, but you can also create your own methods to perform certain actions:

```
public class Main {
  static void myMethod() {
    // code to be executed
  }
}
```

`myMethod()` is the name of the method

`static` means that the method belongs to the Main class and not an object of the Main class. You will learn more about objects and how to access methods through objects later in this tutorial.

`void` means that this method does not have a return value. You will learn more about return values later in this chapter

Access Modifiers

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

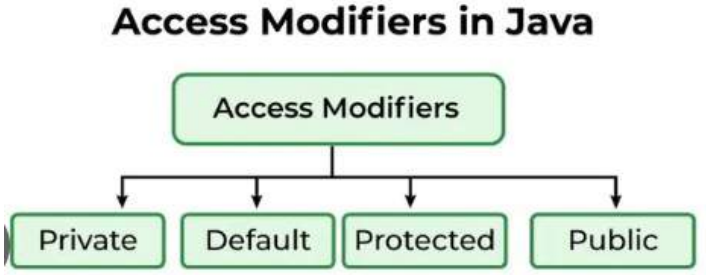
1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

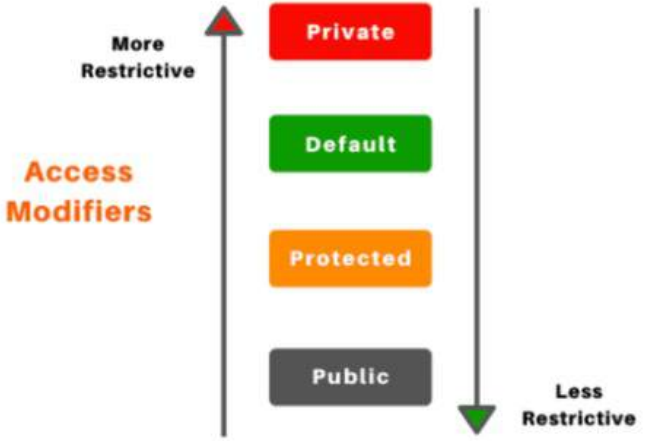
There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y





Static Members and Final Members

In Java, static members are those which belongs to the class and you can access these members without instantiating the class.

The static keyword can be used with methods, fields, classes (inner/nested), blocks.

The **final method** in Java is used as a **non-access modifier** applicable only to a **variable**, a **method**, or a **class.** It is used to **restrict a user** in Java.

The following are **different contexts** where the final is used:

1. Variable
2. Method
3. Class

Final Variable → **To Create constant variable**

Final Methods → **Prevent Method Overriding**

Final Classes → **Prevent Inheritance**

Difference between Static and Final Variable in Java

S.No.	Static Variable	Final Variable
1	The static keyword is connected to occupied static classes, variables, methods and blocks.	The final keyword is connected to class, variables and methods.
2	Here, it is not mandatory to initialise the static variable while declaring it..	Here, it is mandatory to initialise the final variable while declaring it.
3	They can be reinitialized.	We can not reinitialize the final keyword.
4	They can only access the static members of the class, and only static methods can call them.	They cannot be inherited.
5	The object of the static class cannot be created, and holds only static members.	It is not possible to inherit the final class from any class.
6	Static block is used to initialise the static variables.	Final keyword does not support such a block.

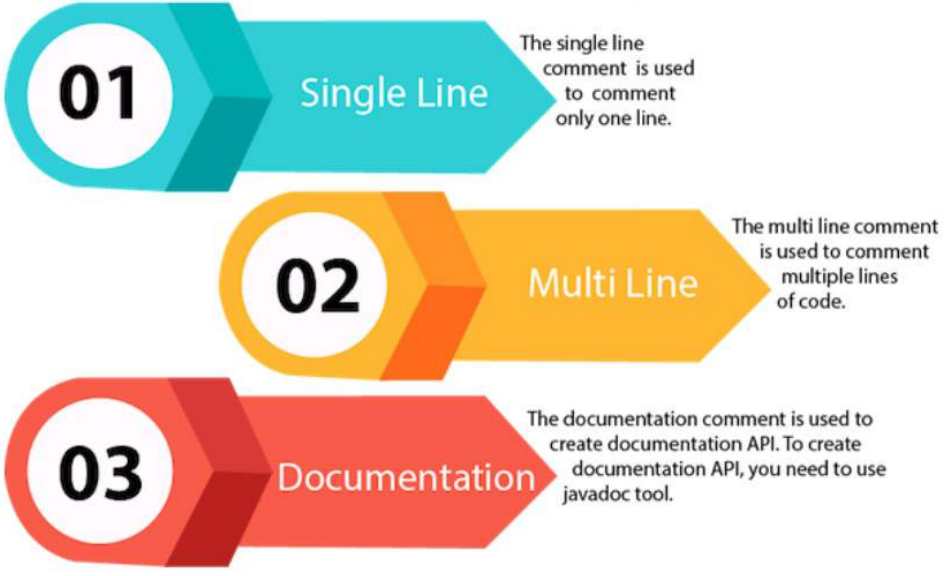
Comments

The Java comments are the statements in a program that are not executed by the compiler and interpreter.

Why do we use comments in a code?

- Comments are used to make the program more readable by adding the details of the code.
- It makes easy to maintain the code and to find the errors easily.
- The comments can be used to provide information or explanation about the variable, method, class, or any statement.
- It can also be used to prevent the execution of program code while testing the alternative code.

Types of Java Comments



Syntax:

```
//This is single line comment
```

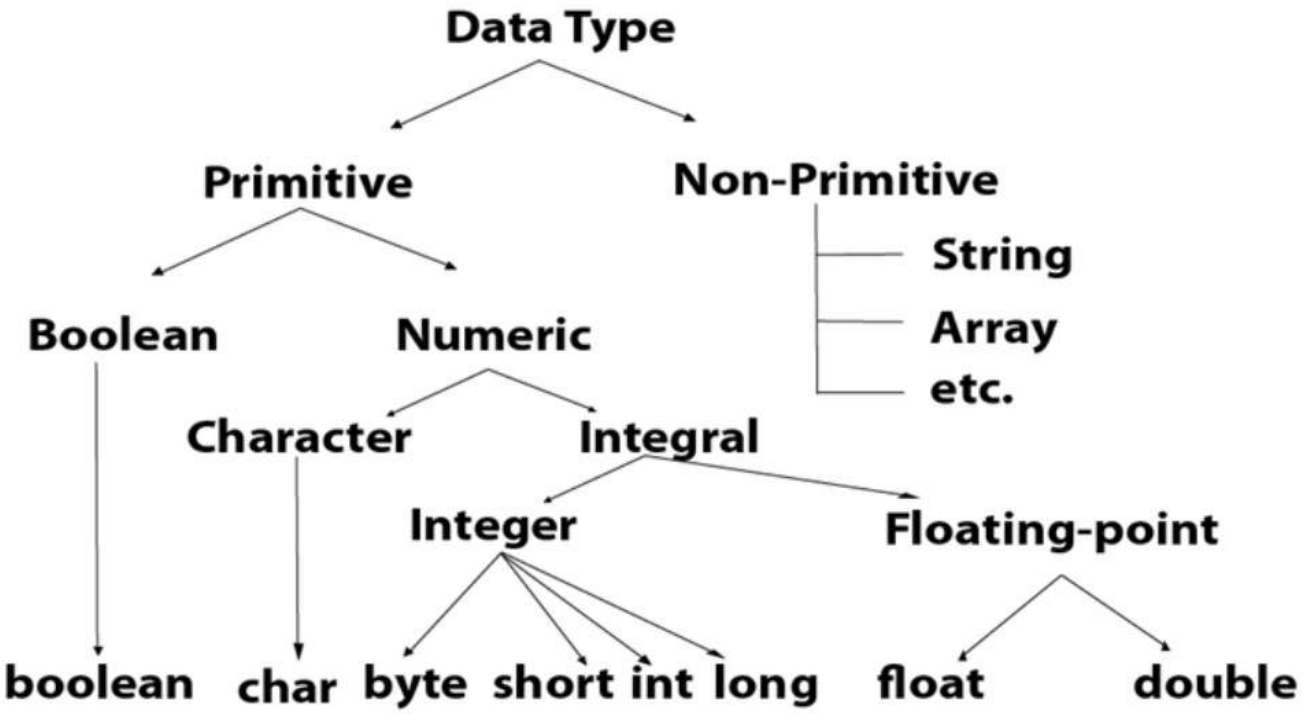
```
/*
This
is
multi line
comment
*/
```

```
/**
*
*We can use various tags to depict the parameter
*or heading or author name
*We can also use HTML tags
*
*/
```

Data Types

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

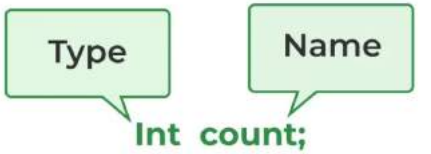
1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.



DATA TYPES	SIZE	DEFAULT	EXPLAINATION
boolean	1 bit	false	Stores true or false values
byte	1 byte/ 8bits	0	Stores whole numbers from -128 to 127
short	2 bytes/ 16bits	0	Stores whole numbers from -32,768 to 32,767
int	4 bytes/ 32bits	0	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes/ 64bits	0L	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes/ 32bits	0.0f	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes/ 64bits	0.0d	Stores fractional numbers. Sufficient for storing 15 decimal digits
char	2 bytes/ 16bits	'\u0000'	Stores a single character/letter or ASCII values

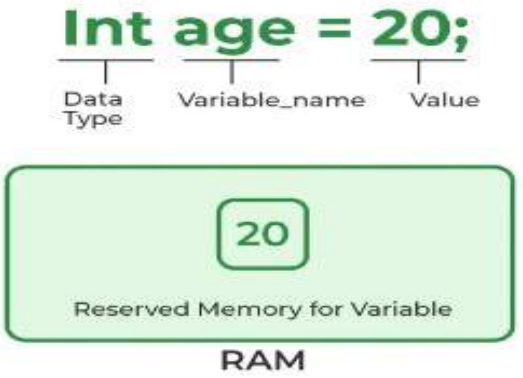
Variables

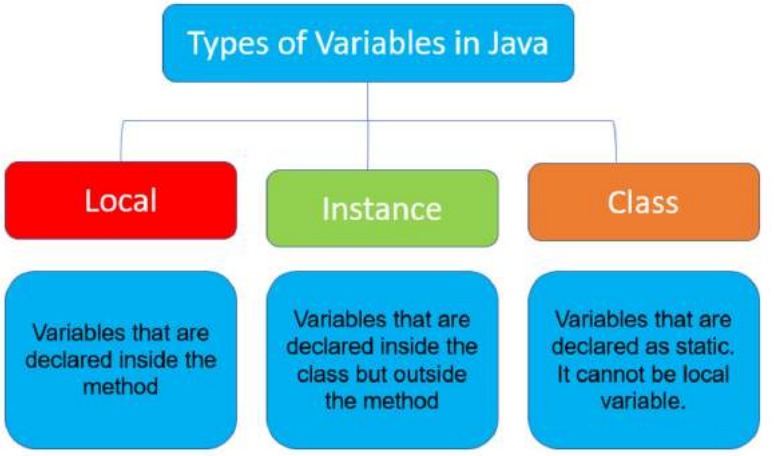
In Java, Variables are the data containers that save the data values during Java program execution. Every Variable in Java is assigned a data type that designates the type and quantity of value it can hold. A variable is a memory location name for the data.



Java variable is a name given to a memory location. It is the basic unit of storage in a program.

- The value stored in a variable can be changed during program execution.
- Variables in Java are only a name given to a memory location. All the operations done on the variable affect that memory location.
- In Java, all variables must be declared before use.





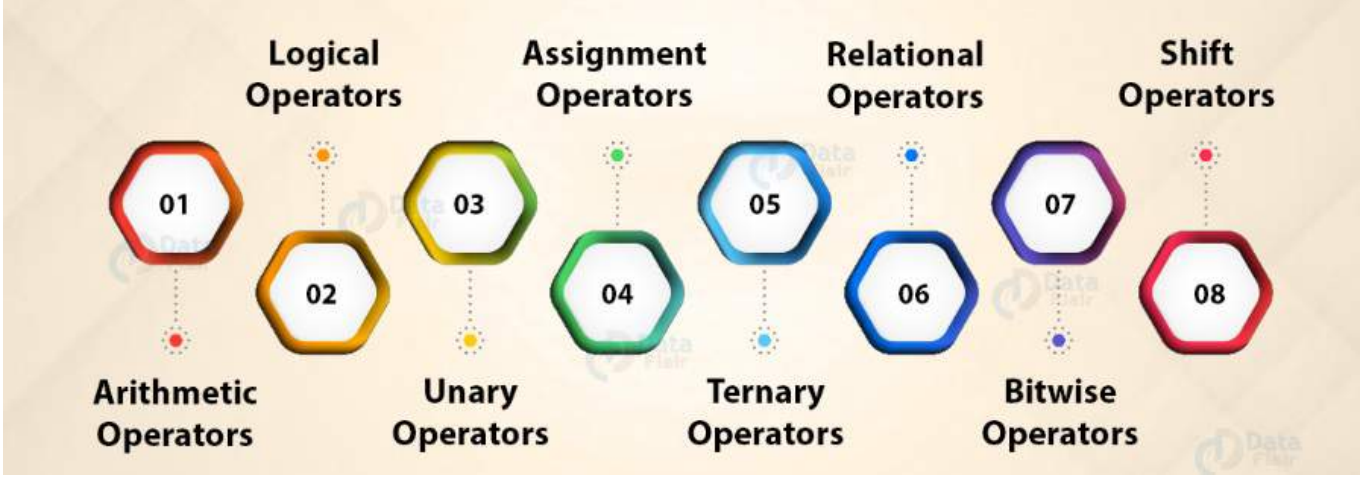
Operators

Operators in Java are the symbols used for performing specific operations in Java. Operators make tasks like addition, multiplication, etc which look easy although the implementation of these tasks is quite complex.

Types of Operators in Java

There are multiple types of operators in Java all are mentioned below:

1. Arithmetic Operators
2. Unary Operators
3. Assignment Operator
4. Relational Operators
5. Logical Operators
6. Ternary Operator
7. Bitwise Operators
8. Shift Operators
9. instance of operator



1. Arithmetic Operators

They are used to perform simple arithmetic operations on primitive data types.

```java
// Java Program to implement
// Arithmetic Operators
import java.io.*;

// Drive Class
class GFG {
      // Main Function
    public static void main (String[] args) {

        // Arithmetic operators
        int a = 10;
        int b = 3;

        System.out.println("a + b = " + (a + b));
        System.out.println("a - b = " + (a - b));
        System.out.println("a * b = " + (a * b));
        System.out.println("a / b = " + (a / b));
        System.out.println("a % b = " + (a % b));

    }
}
```

- * : Multiplication
- / : Division
- % : Modulo
- + : Addition
- – : Subtraction

Output

```
a + b = 13
a - b = 7
a * b = 30
a / b = 3
a % b = 1
```

2. Unary Operators

Unary operators need only one operand. They are used to increment, decrement, or negate a value.

- **Unary minus (-): It used to convert into a negative number**
- **Unary plus (+): It used to convert into a positive number**
- **Logical not operator (!)**
- **Increment operator (++): It increments the value by 1.**
- **Decrement operator (–):** It decrements the value by 1.

3. Assignment Operator

'=' Assignment operator is used to assign a value to any variable. It has right-to-left associativity, i.e. value given on the right-hand side of the operator is assigned to the variable on the left, and therefore right-hand side value must be declared before using it or should be a constant.

The general format of the assignment operator is:

```
variable = value;
```

Shorthand operator	Description
*=	Multiplying left operand with right operand and then assigning it to the variable on the left.
/=	Dividing left operand with right operand and then assigning it to the variable on the left.
%=	Assigning modulo of left operand with right operand and then assigning it to the variable on the left.
+=	Adding left operand with right operand and then assigning it to the variable on the left.
-=	Subtracting left operand with right operand and then assigning it to the variable on the left.

4. Relational Operators

These operators are used to check for relations like equality, greater than, and less than. They return boolean results after the comparison and are extensively used in looping statements as well as conditional if-else statements. The general format is,

```
variable relation_operator value
```

Operator	Meaning
==	Equals to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

5. Logical Operators

These operators are used to perform “logical AND” and “logical OR” operations, i.e., a function similar to AND gate and OR gate in digital electronics. One thing to keep in mind is the second condition is not evaluated if the first one is false, i.e., it has a short-circuiting effect. Used extensively to test for several conditions for making a decision. Java also has “Logical NOT”, which returns true when the condition is false and vice-versa

Conditional operators are:

- **&&, Logical AND:** returns true when both conditions are true.
- **||, Logical OR:** returns true if at least one condition is true.
- **!, Logical NOT:** returns true when a condition is false and vice-versa

6. Ternary operator

The ternary operator is a shorthand version of the if-else statement. It has three operands and hence the name Ternary.

The general format is:

```
condition ? if true : if false
```

The above statement means that if the condition evaluates to true, then execute the statements after the ‘?’ else execute the statements after the ‘:’.

Operator	Meaning
&	Logical AND
|	Logical OR
^	Logical exclusive OR (XOR)
!	Logical NOT
&&	Short-circuit AND
||	Short-circuit OR
==	Equal to
!=	Not equal to
&=	AND assignment
|=	OR assignment
^=	XOR assignment
?:	Ternary if-then-else

7. Bitwise Operators

These operators are used to perform the manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of the Binary indexed trees.

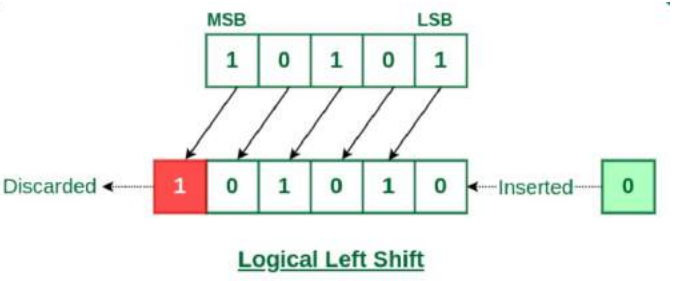
- **&, Bitwise AND operator:** returns bit by bit AND of input values.
- **|, Bitwise OR operator:** returns bit by bit OR of input values.
- **^, Bitwise XOR operator:** returns bit-by-bit XOR of input values.
- **~, Bitwise Complement Operator:** This is a unary operator which returns the one's complement representation of the input value, i.e., with all bits inverted.

Operator	Function
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR (Exclusive OR)
<<	LEFT SHIFT
>>	RIGHT SHIFT

8. Shift Operators

These operators are used to shift the bits of a number left or right, thereby multiplying or dividing the number by two, respectively. They can be used when we have to multiply or divide a number by two. General format-

```
number shift_op number_of_places_to_shift;
```



Logical Left Shift

9. instanceof operator

The instance of the operator is used for type checking. It can be used to test if an object is an instance of a class, a subclass, or an interface. General format-

```
object instance of class/subclass/interface
```

Control Flow

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

1. Decision Making statements
   - if statements
   - switch statement
2. Loop statements
   - do while loop
   - while loop
   - for loop
   - for-each loop
3. Jump statements
   - break statement
   - continue statement

Decision-Making statements:

As the name suggests, decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in Java, i.e., If statement and switch statement.

1) If Statement:

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

1. Simple if statement
2. if-else statement
3. if-else-if ladder
4. Nested if-statement

Let's understand the if-statements one by one.

1) Simple if statement:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax of if statement is given below.

```
if(condition) {
statement 1; //executes when condition is true
}
```

```
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y > 20) {
System.out.println("x + y is greater than 20");
}
}
}
```

Output:

```
x + y is greater than 20
```

2) if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block.
The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

```
if(condition) {
statement 1; //executes when condition is true
}
else{
statement 2; //executes when condition is false
}
```

```
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y < 10) {
System.out.println("x + y is less than 10");
} else {
System.out.println("x + y is greater than 20");
}
}
}
```

Output:

```
x + y is greater than 20
```

3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false
}
```

```
public class Student {
public static void main(String[] args) {
String city = "Delhi";
if(city == "Meerut") {
System.out.println("city is meerut");
}else if (city == "Noida") {
System.out.println("city is noida");
}else if(city == "Agra") {
System.out.println("city is agra");
}else {
System.out.println(city);
}
}
}
```

Output:

```
Delhi
```

4. Nested if-statement

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

```
public class Student {
public static void main(String[] args) {
String address = "Delhi, India";

if(address.endsWith("India")) {
if(address.contains("Meerut")) {
System.out.println("Your city is Meerut");
}else if(address.contains("Noida")) {
System.out.println("Your city is Noida");
}else {
System.out.println(address.split(",")[0]);
}
}else {
System.out.println("You are not living in India");
}
}
}
```

Output:

```
Delhi
```

```
if(condition 1) {
statement 1; //executes when condition 1 is true
if(condition 2) {
statement 2; //executes when condition 2 is true
}
else{
statement 2; //executes when condition 2 is false
}
}
```

Switch Statement:

In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

- The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java
- Cases cannot be duplicate
- Default statement is executed when any of the case doesn't match the value of expression. It is optional.
- Break statement terminates the switch block when the condition is satisfied.
  It is optional, if not used, next case is executed.
- While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

```
switch (expression){
   case value1:
    statement1;
    break;
   .
   .
   .
   case valueN:
    statementN;
    break;
   default:
    default statement;
}
```

```
public class Student implements Cloneable {
public static void main(String[] args) {
int num = 2;
switch (num){
case 0:
System.out.println("number is 0");
break;
case 1:
System.out.println("number is 1");
break;
default:
System.out.println(num);
}
}
}
```

Output:

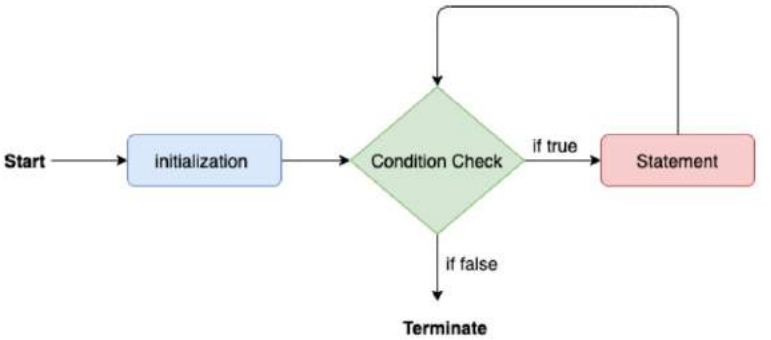
```
2
```

Loop Statements

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

1. for loop
2. while loop
3. do-while loop



Java for loop

In Java, for loop is similar to C and C++. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

```
for(initialization, condition, increment/decrement) {
//block of statements
}
```

```
public class Calculattion {
public static void main(String[] args) {
// TODO Auto-generated method stub
int sum = 0;
for(int j = 1; j<=10; j++) {
sum = sum + j;
}
System.out.println("The sum of first 10 natural numbers is " + sum);
}
}
```

Output:

```
The sum of first 10 natural numbers is 55
```

Java while loop

The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.

It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

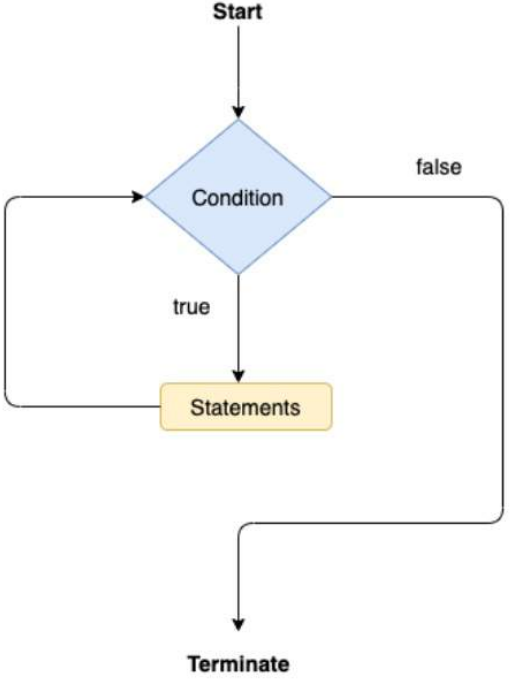
The syntax of the while loop is given below.

```
while(condition){
//looping statements
}
```

```
public class Calculation {
public static void main(String[] args) {
// TODO Auto-generated method stub
int i = 0;
System.out.println("Printing the list of first 10 even numbers \n");
while(i<=10) {
System.out.println(i);
i = i + 2;
}
}
}
```

```
Printing the list of first 10 even numbers

0
2
4
6
8
10
```



Java do-while loop

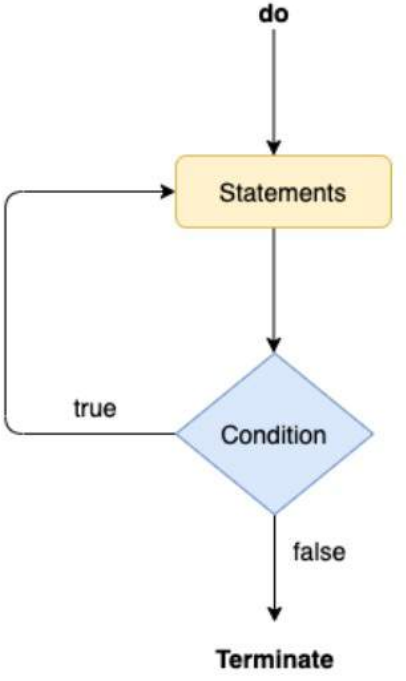
The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

```
do
{
//statements
} while (condition);
```

```
public class Calculation {
public static void main(String[] args) {
// TODO Auto-generated method stub
int i = 0;
System.out.println("Printing the list of first 10 even numbers \n");
do {
System.out.println(i);
i = i + 2;
}while(i<=10);
}
}
```

```
Printing the list of first 10 even numbers
0
2
4
6
8
10
```



Jump Statements

Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

Java break statement

As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

Java continue statement

Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

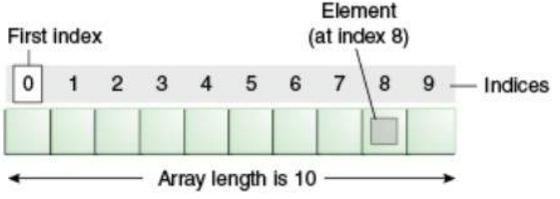
Java Arrays

Normally, an array is a collection of similar type of elements which has contiguous memory location.

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator.



Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Syntax to Declare an Array in Java

```
dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];
```

Instantiation of an Array in Java

```
arrayRefVar=new datatype[size];
```

Java String

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

```
char[] ch={'j','a','v','a','t','p','o','i','n','t'};
String s=new String(ch);
```

is same as:

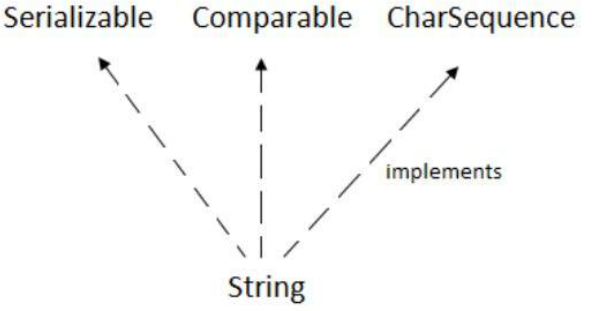
```
String s="javatpoint";
```

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.

What is String in Java?

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.



How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1. By string literal

```
String s="welcome";
```

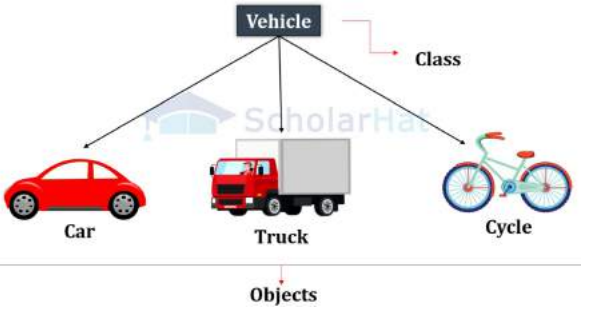
2. By new keyword

```
String s=new String("Welcome");//creates two objects and one reference variable
```

Object Oriented Programming

Object Oriented Programming: Class, Object, Inheritance Super Class, Sub Class, Overriding, Overloading, Encapsulation, Polymorphism, Abstraction, Interfaces, and Abstract Class.

A Class is a template or blueprint for creating Objects; an Object is an Instance of a Class.



Java Classes

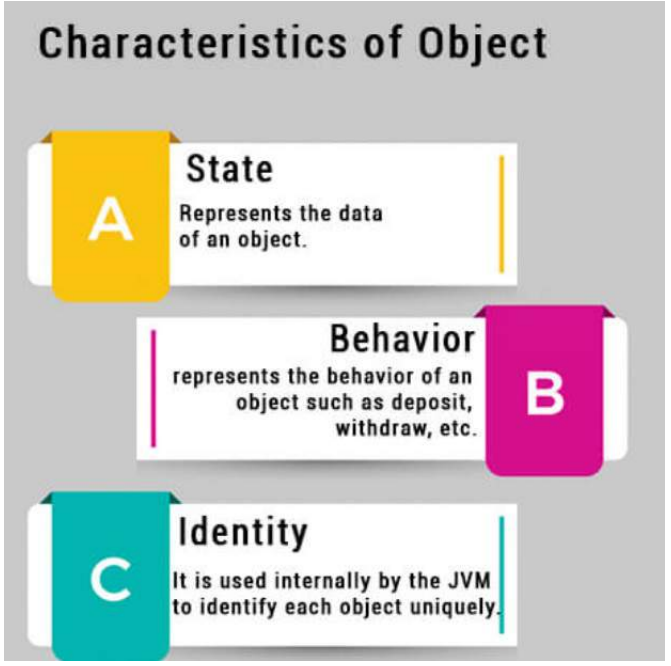
1. Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
2. Class does not occupy memory.
3. Class is a group of variables of different data types and a group of methods.
4. A Class in Java can contain:
   - Data member
   - Method
   - Constructor
   - Nested Class
   - Interface

```
access_modifier class <class_name>
{
    data member;
    method;
    constructor;
    nested class;
    interface;
}
```

object in Java

An entity that has state and behavior is known as an object e.g., chair, bike, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system

An object has three characteristics:

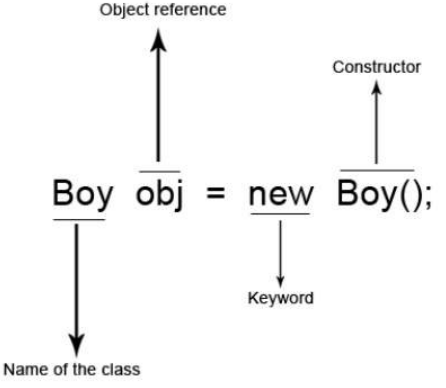


For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

- An object is *a real-world entity.*
- An object is *a runtime entity.*
- The object is *an entity which has state and behavior.*
- The object is *an instance of a class.*



Inheritance

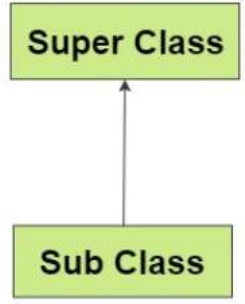
Java, Inheritance is an important pillar of OOP. It is the mechanism in Java by which one class is allowed to inherit the features(fields and methods) of another class. In Java, Inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class. In addition, you can add new fields and methods to your current class as well.

Why Do We Need Java Inheritance?

- Code Reusability:
- Method Overriding:
- Abstraction:

Important Terminologies Used in Java Inheritance

- **Super Class/Parent Class:** The class whose features are inherited is known as a superclass(or a base class or a parent class).
- **Sub Class/Child Class:** The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.



How to Use Inheritance in Java?

The **extends keyword** is used for inheritance in Java. Using the extends keyword indicates you are derived from an existing class. In other words, "extends" refers to increased functionality.

Syntax :

```
class DerivedClass extends BaseClass
{
   //methods and fields
}
```

Example :

```
// Java Program to illustrate Inheritance (concise)

import java.io.*;

// Base or Super Class
class Employee {
    int salary = 60000;
}

// Inherited or Sub Class
class Engineer extends Employee {
    int benefits = 10000;
}

// Driver Class
class Gfg {
    public static void main(String args[])
    {
        Engineer E1 = new Engineer();
        System.out.println("Salary : " + E1.salary
                           + "\nBenefits : " + E1.benefits);
    }
}
```

In the below example of inheritance, class Employee is a base class, class Engineer is a derived class that extends the Employee class and class Test is a driver class to run the program.

Output

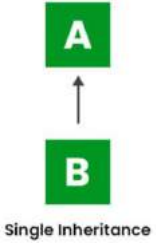
```
Salary : 60000
Benefits : 10000
```

Types of inheritance in java

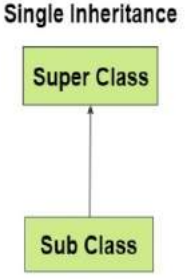
On the basis of class, there can be three types of inheritance in java: **single**, **multilevel** and **hierarchical**.

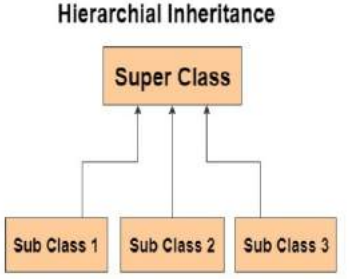
In java programming, **multiple** and **hybrid** inheritance is supported through **interface** only. We will learn about interfaces later.

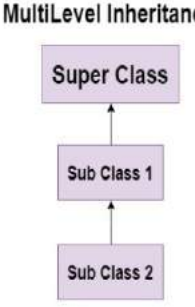
1. Single Inheritance



In single inheritance, subclasses inherit the features of one superclass. In the image below, class A serves as a base class for the derived class B

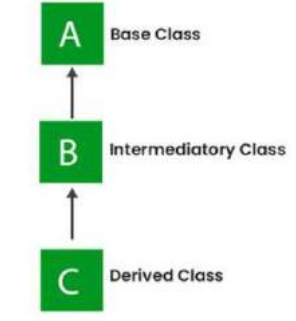


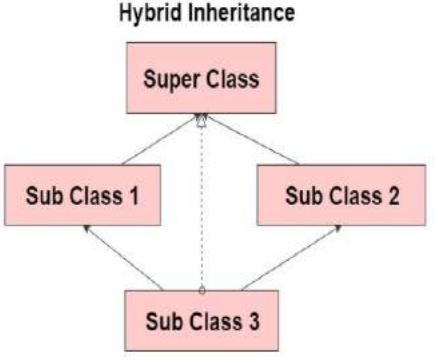


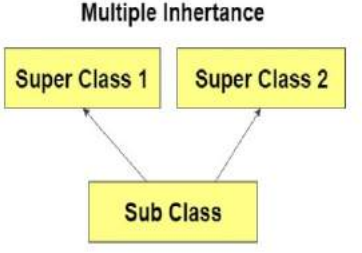


2. Multilevel Inheritance

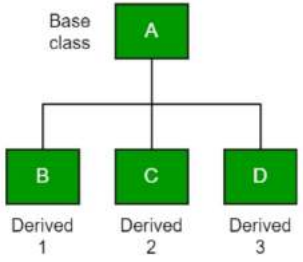
In Multilevel Inheritance, a derived class will be inheriting a base class, and as well as the derived class also acts as the base class for other classes. In the below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the **grandparent's members**.







3. Hierarchical Inheritance



In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In the below image, class A serves as a base class for the derived classes B, C, and D.

Overloading And Overriding

Overloading is like ordering a coffee at a coffee shop. You can order a “coffee” (method name), but you need to specify the type – an espresso, a latte, a cappuccino, etc. (parameters). They are all “coffee” but prepared differently based on your order.

Overriding, on the other hand, is like visiting a coffee shop that has a new way of making espresso. Maybe they use a different coffee bean or a new brewing method. When you order an “espresso” (method name) here, you get their version of espresso, not the usual one you get at other coffee shops. The new coffee shop has “overridden” the standard way of making espresso.

So, the main difference is that overloading is about using the same name (like “coffee”) for different things (like espresso, latte, cappuccino), while overriding is about changing the way something is done (like making espresso) in a new context (like a different coffee shop).

What is Overloading?

Overloading definition: ***Overloading occurs when two or more methods in the same class have the same name but different parameters.***

> Method overloading is also called compile-time polymorphism, static polymorphism, or early binding. In Method overloading, the child argument takes precedence over the parent argument.

What is Overriding?

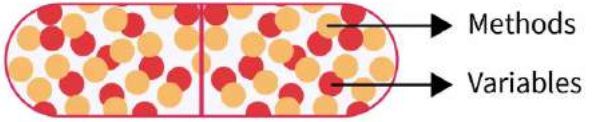
Overriding definition: ***Method overriding is the act of declaring a method in a subclass that is already present in a parent class.***

Conclusion

When two or more methods in the same class have the same method name but different parameters, this is called overloading. In contrast, overriding occurs when two methods have the same name and parameters.

Encapsulation in JAVA

Encapsulation in Java is a fundamental concept in object-oriented programming (OOP) that refers to the bundling of data and methods that operate on that data within a single unit, which is called a class in Java. Java Encapsulation is a way of hiding the implementation details of a class from outside access and only exposing a public interface that can be used to interact with the class.



Class

Encapsulation in Java is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, that it is a protective shield that prevents the data from being accessed by the code outside this shield.

Encapsulation can be achieved by Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

It is more defined with the setter and getter method.

Below is the example with Java Encapsulation:

```java
// Java Program to demonstrate
// Java Encapsulation

// Person Class
class Person {
    // Encapsulating the name and age
    // only approachable and used using
    // methods defined
    private String name;
    private int age;

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }

    public void setAge(int age) { this.age = age; }
}
```

```java
// Driver Class
public class Main {
    // main function
    public static void main(String[] args)
    {
        // person object created
        Person person = new Person();
        person.setName("John");
        person.setAge(30);

        // Using methods to get the values from the
        // variables
        System.out.println("Name: " + person.getName());
        System.out.println("Age: " + person.getAge());
    }
}
```

Output

```
Name: John
Age: 30
```

Advantages

Disadvantages

- Can lead to increased complexity, especially if not used properly.
- Can make it more difficult to understand how the system works.
- May limit the flexibility of the implementation.

Polymorphism

The word polymorphism means having many forms. In simple words, we can define Java Polymorphism as the ability of a message to be displayed in more than one form.

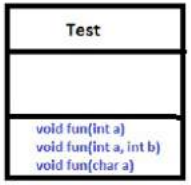
Real-life Illustration of Polymorphism in Java: A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, and an employee. So the same person possesses different behaviors in different situations. This is called polymorphism.

What is Polymorphism in Java?

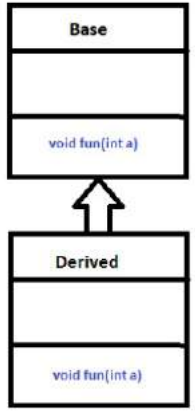
Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.

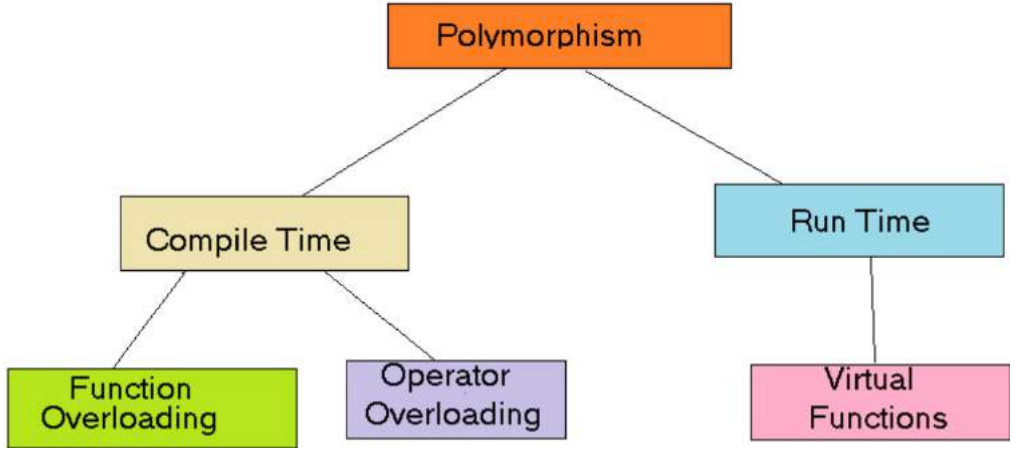
Types of Java Polymorphism

In Java Polymorphism is mainly divided into two types:



Overloading





- Compile-time Polymorphism
- Runtime Polymorphism

Advantages of Polymorphism in Java

1. Increases code reusability by allowing objects of different classes to be treated as objects of a common class.
2. Improves readability and maintainability of code by reducing the amount of code that needs to be written and maintained.
3. Supports dynamic binding, enabling the correct method to be called at runtime, based on the actual class of the object.
4. Enables objects to be treated as a single type, making it easier to write generic code that can handle objects of different types.

Disadvantages

1. Can make it more difficult to understand the behavior of an object, especially if the code is complex.
2. This may lead to performance issues, as polymorphic behavior may require additional computations at runtime.

Abstraction

Abstraction in Java is the process in which we only show essential details/functionality to the user. The non-essential implementation details are not displayed to the user.

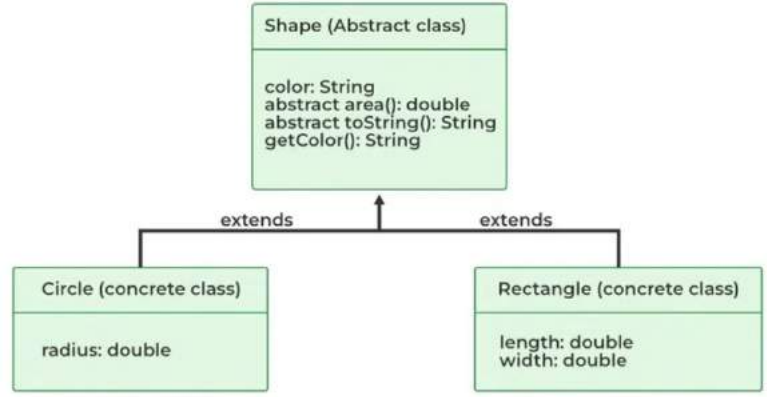
Real Life Example of Abstraction

In Java, abstraction is achieved by **interfaces** and **abstract classes**. We can achieve 100% abstraction using interfaces.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Java Abstract classes and Java Abstract methods

1. An abstract class is a class that is declared with an abstract keyword.
2. An abstract method is a method that is declared without implementation.
3. An abstract class may or may not have all abstract methods. Some of them can be concrete methods
4. A method-defined abstract must always be redefined in the subclass, thus making overriding compulsory or making the subclass itself abstract.



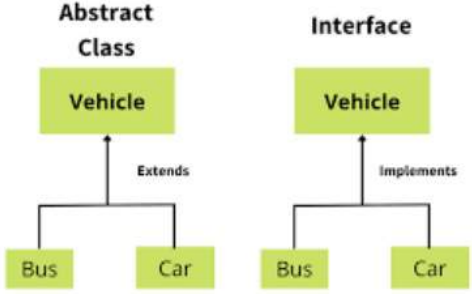
Interfaces

The interface in Java is *a* mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not the method body. It is used to achieve abstraction and multiple inheritances in Java using Interface. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship**.

Syntax:

```
interface <interface_name>{

   // declare constant fields
   // declare methods that abstract
   // by default.
}
```



How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Output:

```
Hello
```

Abstract Class

- Abstract classes are classes that contain one or more abstract methods(methods without body)
- Abstract class is declared with abstract modifier.
- Abstract class can contain concrete methods as well.
- **Concrete methods** are complete methods with method heading and body

```
abstract class Shape
{
...
}
```

Abstract Class	Interface
1. *abstract* keyword	1. *interface* keyword
2. Subclasses *extends* abstract class	2. Subclasses *implements* interfaces
3. Abstract class can have implemented methods and 0 or more abstract methods	3. Java 8 onwards, Interfaces can have default and static methods
4. We can extend only one abstract class	4. We can implement multiple interfaces

Packages

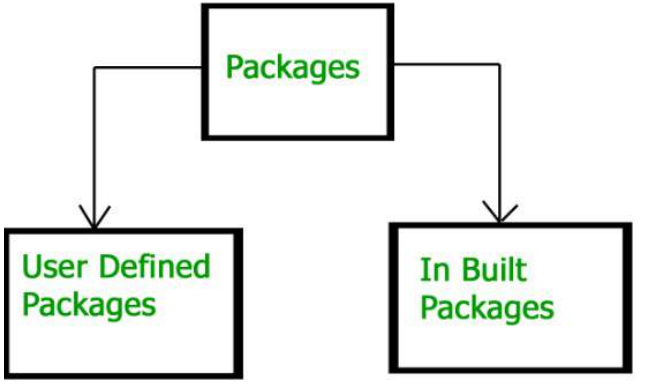
Packages: Defining Package, CLASSPATH Setting for Packages, Making JAR Files for Library Packages, Import and Static Import Naming Convention For Packages

Package

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

- Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
- Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.
- Packages can be considered as data encapsulation (or data-hiding).

Types of packages:



Classpath in Java?

CLASSPATH describes the location where all the required files are available which are used in the application. Java Compiler and JVM (Java Virtual Machine) use CLASSPATH to locate the required files. If the CLASSPATH is not set, Java Compiler will not be able to find the required files and hence will throw the following error. `Error: Could not find or load main class <class name>`

The above error is resolved when CLASSPATH is set.

Set the CLASSPATH in JAVA in Windows

```
set PATH=.;C:\Program Files\Java\JDK1.6.20\bin
```

Note: Semi-colon (;) is used as a separator dot (.) is the default value of **CLASSPATH**

Set the CLASSPATH on Linux

Command Line:

Find out where you have installed Java, basically, it's in **/usr/lib/jvm** path. Set the CLASSPATH in **/etc/environment** using `sudo <editor name> /etc/environment`

Add the following lines,

```
JAVA_HOME = "/usr/lib/jvm/<java folder (eg. java-1.8.0-openjdk-amd64>)/bin"
export JAVA_HOME
CLASSPATH=".:/usr/lib/jvm/<java folder>/lib:/home/name/Desktop"
export CLASSPATH
```

To check the current CLASSPATH, run

```
echo ${CLASSPATH}
```

JAR files in Java

A JAR (Java Archive) is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file to distribute application software or libraries on the Java platform.
In simple words, a JAR file is a file that contains a compressed version of .class files, audio files, image files, or directories. We can imagine a .jar file as a zipped file(.zip) that is created by using WinZip software. Even, WinZip software can be used to extract the contents of a .jar . So you can use them for tasks such as lossless data compression, archiving, decompression, and archive unpacking.

1.1 Create a JAR file

In order to create a .jar file, we can use ***jar cf command*** in the following ways as discussed below:

Syntax: `jar cf jarfilename inputfiles`

Here, cf represents to create the file. For example , assuming our package pack is available in C:\directory , to convert it into a jar file into the pack.jar , we can give the command as:

```
C:\> jar cf pack.jar pack
```

Package Naming Conversion in Java

In Java, package plays an important role in preventing naming conflicts, controlling access, and making searching and usage of classes, enumeration, interfaces, and annotation easier.

Naming Conventions

For avoiding unwanted package names, we have some following naming conventions which we use in creating a package.

- The name should always be in the lower case.
- They should be period-delimited.
- The names should be based on the company or organization name.

In order to define a package name based on an organization, we'll first reverse the company URL. After that, we define it by the company and include division names and project names.

Static import in Java

In Java, static import concept is introduced in 1.5 version. With the help of static import, we can access the static members of a class directly without class name or any object.

For Ex: we always use sqrt() method of Math class by using Math class i.e. **Math.sqrt()**, but by using static import we can access sqrt() method directly. According to SUN microSystem, it will improve the code readability and enhance coding. But according to the programming experts, it will lead to confusion and not good for programming. If there is no specific requirement then we should **not** go for static import.

Advantage of static import:

If user wants to access any static member of class then less coding is required.

```
// Java Program to illustrate
// calling of predefined methods
// without static import
class Geeks {
    public static void main(String[] args)
    {
        System.out.println(Math.sqrt(4));
        System.out.println(Math.pow(2, 2));
        System.out.println(Math.abs(6.3));
    }
}
```

Output:

```
2.0
4.0
6.3
```

Disadvantage of static import:

Static import makes the program unreadable and unmaintainable if you are reusing this feature.

```
// Java Program to illustrate
// calling of predefined methods
// with static import
import static java.lang.Math.*;
class Test2 {
    public static void main(String[] args)
    {
        System.out.println(sqrt(4));
        System.out.println(pow(2, 2));
        System.out.println(abs(6.3));
    }
}
```