Om Sakthi
Adhiparasakthi Engineering College
Melmaruvathur - 603319
OBJECT ORIENTED PROGRAMMING
Subject Code: CS3391

Unit I
INTRODUCTION TO OOP AND JAVA FUNDAMENTALS

1.1 Object Oriented Programming

Object oriented programming is a programming technique in which programs are written on the basis of objects.

Examples:

- C++
- PHP
- Java
- Objective C

1.2 Major Concepts of Object Oriented Programming:

- Classes
- Objects
- Methods or member functions
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Message passing
- Dynamic Binding

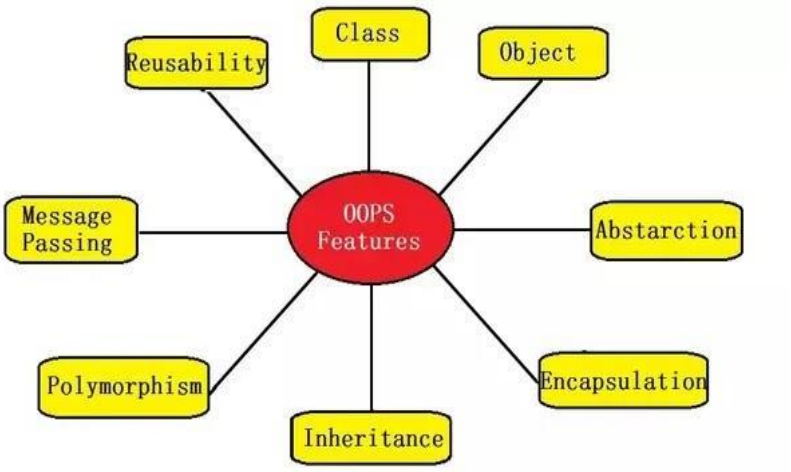


Fig 1. Features of Object Oriented Programming

1.2.1 Classes:

Class
(Defines the characteristics of an object)

=

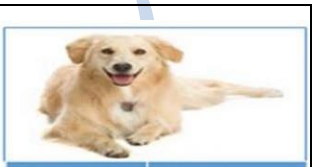
Attributes
Data members
Fields
Properties
State

+

Behaviour
Member functions
Methods
Operations

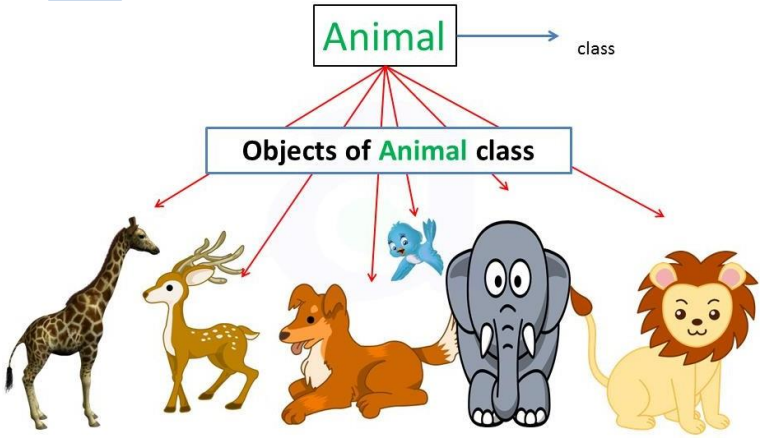
- Class is a blueprint for creating objects by providing member variables, and member functions.
- A class is an abstract data type which consists of set of objects that share common structure and a common behavior.

Representation of the Class:

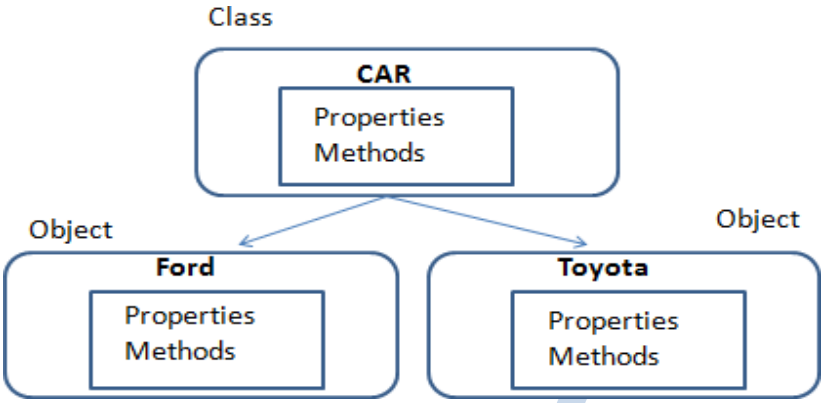
Class Name **Attributes** **Operations**	**Human** Name color Walking(); Talking();	class car methods refuel() getFuel setSpeed() getSpeed() drive() attributes fuel maxspeed	 State: Name, Color, Breed, Hungry Behavior: Barking, Fetching, Wagging Tail

1.2.2. Objects:

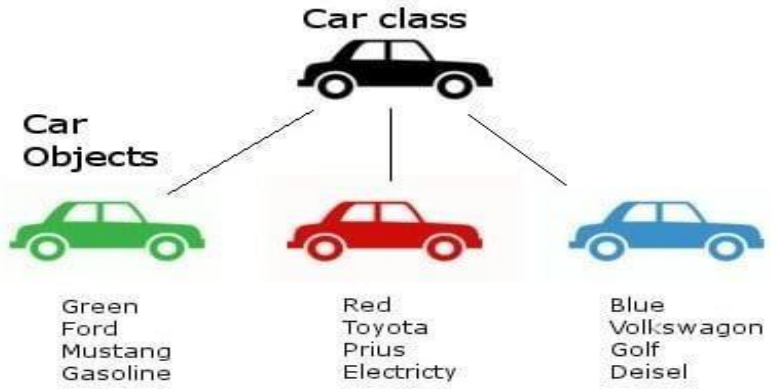
- Object is any real world entity which have physical existence either living or non-living. Examples are chair, car, pen, table, keyboard, bike etc.
- An object is an instance of a class. Creating an object is also known as instantiation
- An object comes from a class. A unique occurrence from a class is called as an instance



- More than one instance of the same class can exist at any one time, which means there can be multiple objects made from the same class.

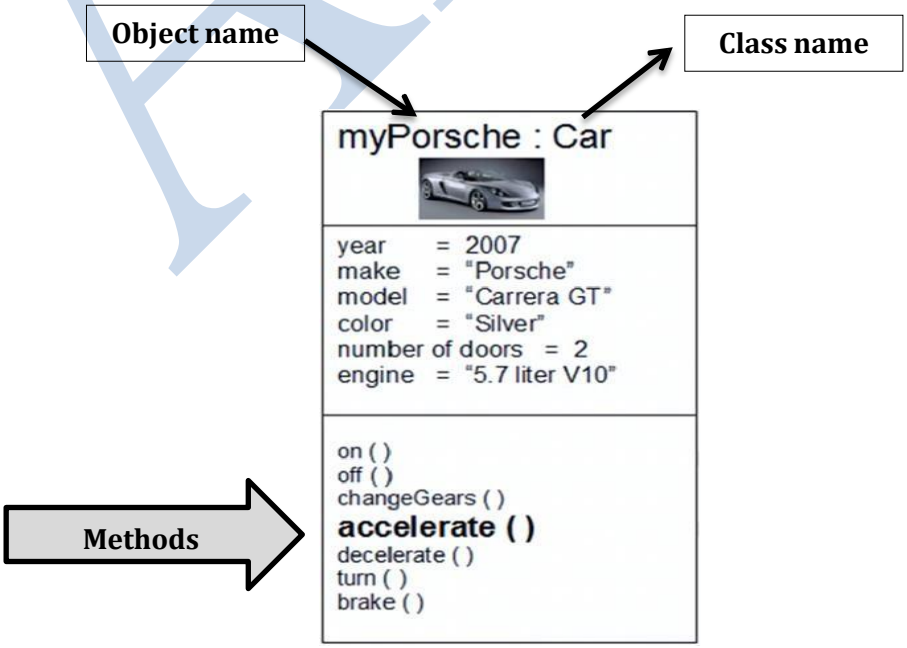


Example for object:



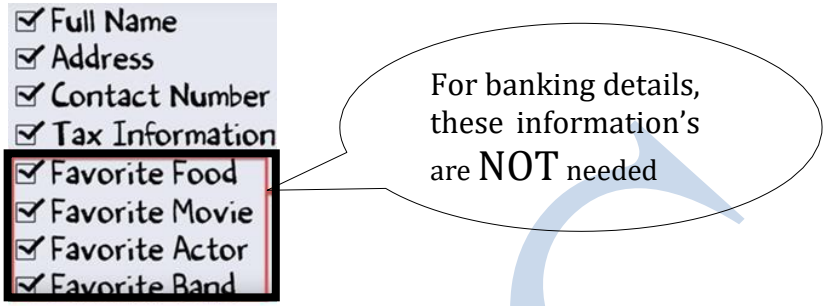
1.2.3 Methods or Member functions

- A method is behaviour of an object. A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. Within the program, a method usually affects only one particular object.



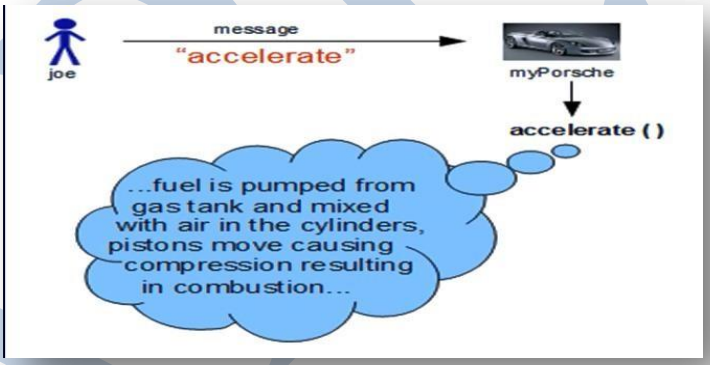
1.2.4 Abstraction

- Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the user.
- Abstraction is a process is to show only "relevant" data and "hide" unnecessary details of an object from the user. It's also called as ***Data Abstraction.***

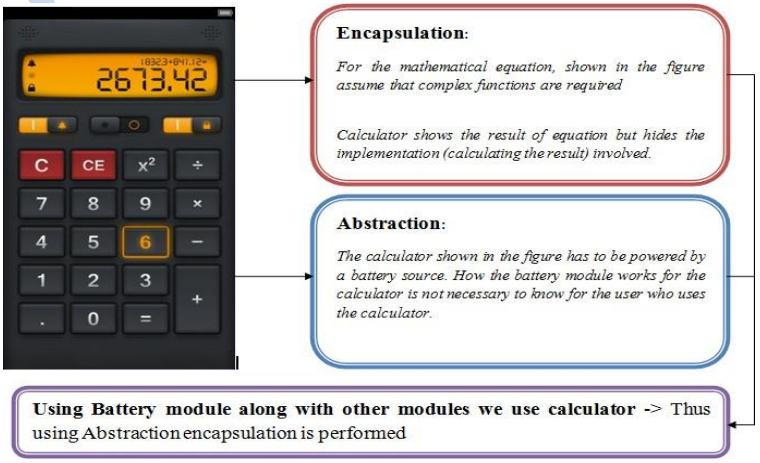


1.2.5 Encapsulation

- The wrapping up of data and the functions into a single unit is called as Encapsulation. This feature keeps the data safe from outside interference and misuse. This leads to the concept of ***Data Hiding***.
- It hides the behaviour of the object from its implementation.

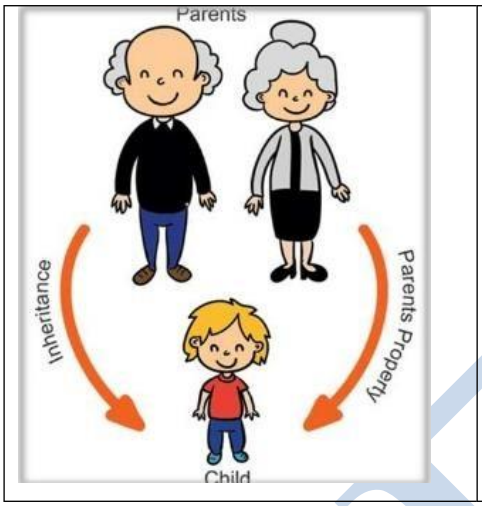


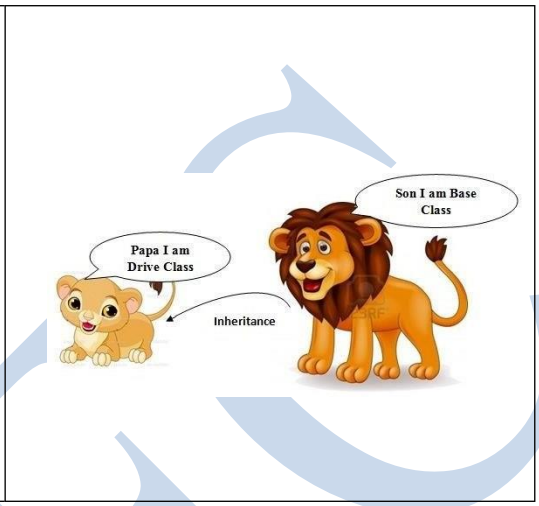
Difference between Abstraction and Encapsulation:

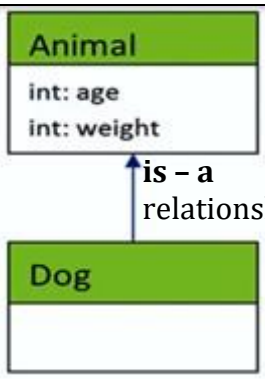


1.2.6 Inheritance

- Inheritance is a mechanism of deriving new class from an old class that is pre-defined.
- The old class is called the base class (or) super class (or) parent class. The new class is called the derived class (or) sub class (or) child class.
- Inheritance concept is mainly used for code reusability.







- Every dog is an Animal. Hence "is-a relationship"
- Here,
  Base class → Animal
  Derived class →Dog
- Every derived class has a unique special feature.
  Dog has a unique feature **"Barking".**

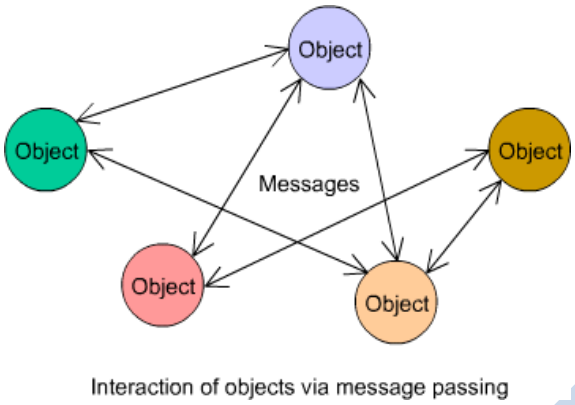
1.2.7 Polymorphism

- Polymorphism means many (poly) shapes (morph).
- The ability to take more than one form is called as polymorphism. An operation may exhibit different behaviour's in different instances. The behaviour depends upon the types of data used in the operation.



1.2.8 Message Passing

- Objects can communicate with each other's by passing message same as people passing message with each other
- Objects can send or receive message or information. Message passing involves specifying the name of the object, the name of the function and the information to be sent.



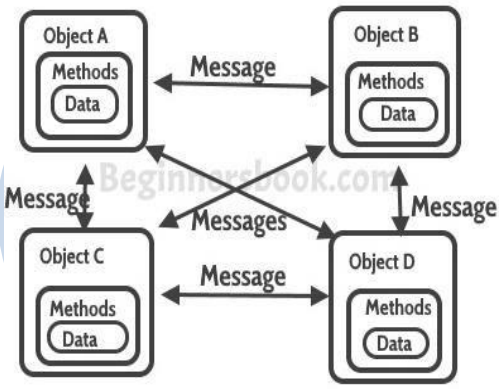
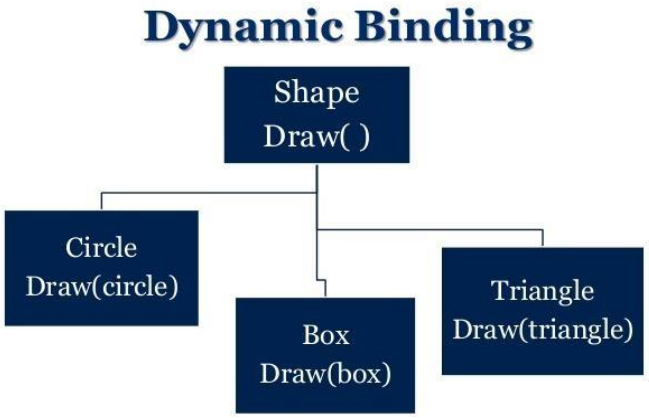


Fig. Message passing in OOPs

1.2.9 Dynamic binding

- Dynamic binding is a process of linking the **code** associated with the **procedure call** at the **run-time**.



1.3 Benefits of OOPs

- Modularity for easier troubleshooting
  - We can build programs from standard working modules rather than, having to start writing the code from scratch. This leads to higher productivity and less development time.
- Reuse of code through inheritance
  - Through inheritance, redundant codes are eliminated.

- Security
  - The principle of data hiding helps the programmers to build secure program that can't be attacked by other parts of the program.
- Flexibility through polymorphism
  - A functionality can behave differently for different instances
- Effective problem solving
  - Large programs are difficult to write, but if the development and designing team follow OOPS concept then they can better design with minimum flaws.

1.4 Object oriented programming in JAVA

Java programming language was originally developed by Sun Microsystems which was initiated by **James Gosling** and released in **1995** as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]- 2018). The latest release of the Java Standard Edition is Java SE 10 (18.3). With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms. For example: J2EE for Enterprise Applications, J2ME for Mobile Applications. Java is guaranteed to be **Write Once, Run Anywhere.**

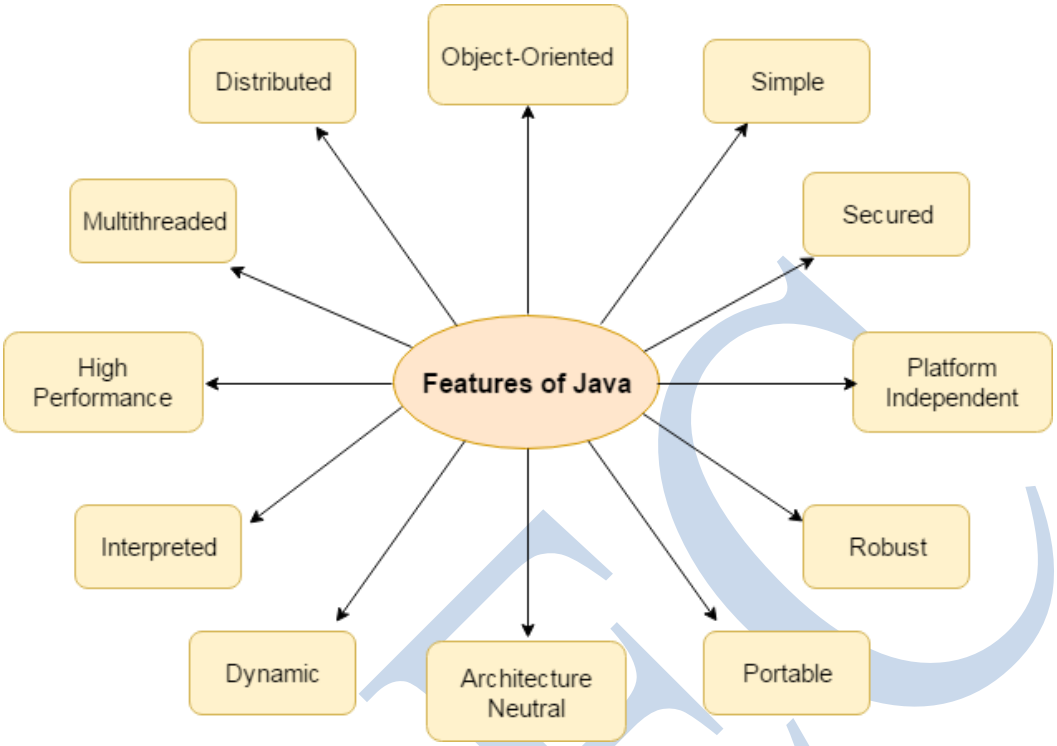
1.4.1 Features of Java

The main objective of Java programming language creation was to make it portable, simple and secure programming language. The features of Java are also known as java buzzwords. Following is a list of most important features of Java language. The Java Features given below are simple and easy to understand.

i. Simple
ii. Object-Oriented
iii. Portable
iv. Platform independent
v. Secured
vi. Robust
vii. Architecture neutral
viii. Dynamic
ix. Interpreted
x. High Performance

xi. Multithreaded

xii. Distributed



Simple

Java is very easy to learn and its syntax is simple, clean and easy to understand. According to Sun, Java language is simple because:

- Syntax is very easy
- Removes features like explicit pointers, operator overloading etc.
- Unreferenced objects need not be removed because there is Automatic Garbage Collection in java.

Object-oriented

Java is Object-oriented programming language. Everything in Java is an object. Object-oriented programming(OOPs) is a methodology that simplifies software development and maintenance by providing some features like Objects, Classes, Inheritance, Polymorphism, Abstraction, Encapsulation.

Platform Independent

Platform independence means the ability to write a program on one platform and run it on any other platform without worrying about the platform specific features. If it works on one platform, it should work on all other platforms. e.g., platforms are Unix, Mac, Windows, Android. Platform independent code runs on variety of CPU's under

different environments. Java uses ***Java Bytecode***, which is an intermediate set of instructions that helps in achieving this platform independence.

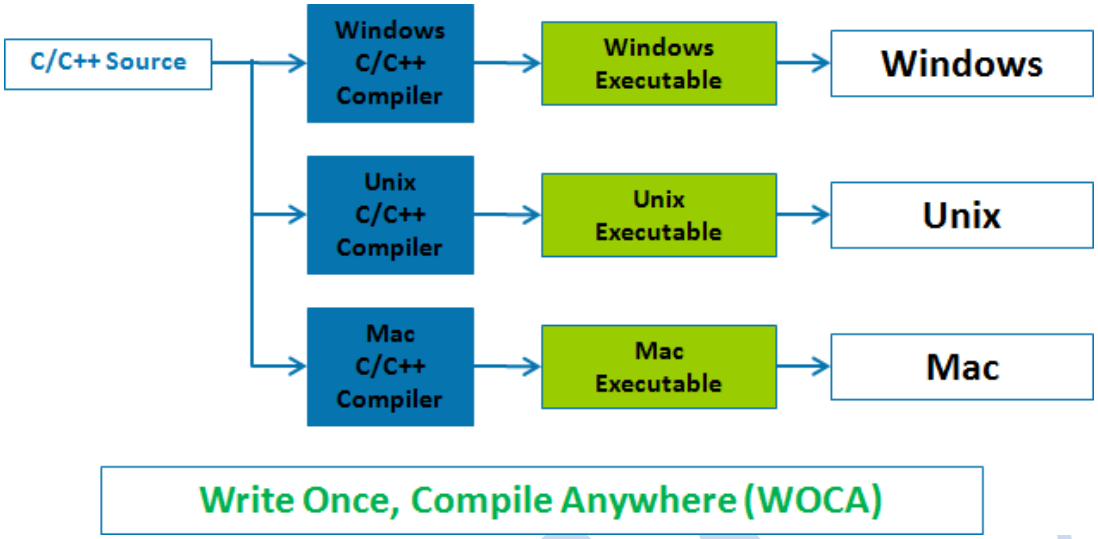


Figure. Other Programming languages are platform dependent

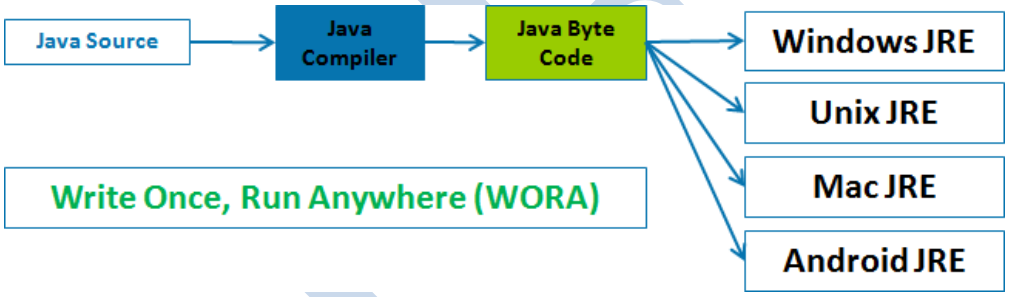
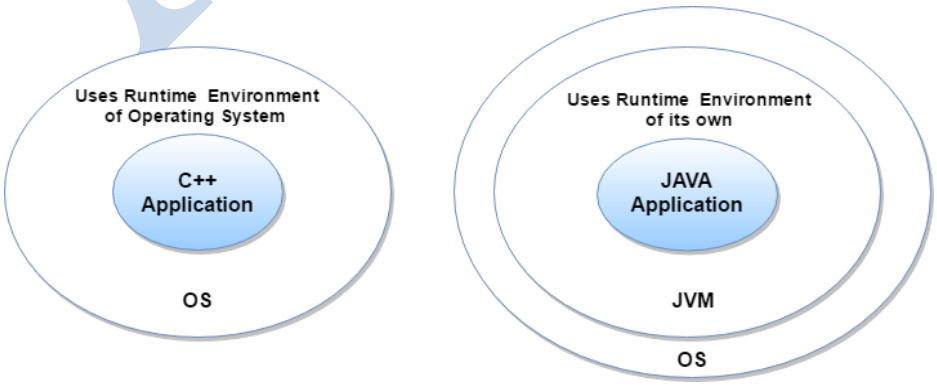


Figure. Java - platform independent Programming language

Secured

Java's secure feature it enables to develop virus-free systems.. Java is secured because Java Programs run inside Java virtual machine (JVM).



Robust

Java is robust because:

- Uses strong memory management
- Lack of pointers avoids security problem
- Automatic garbage collection is available
- Exception handling and type checking mechanisms in java makes java robust.

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.

High Performance

With the use of Just-In-Time compilers, Java enables high performance.

Distributed

Java is designed for the distributed environment of the internet.

Dynamic

Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment.

Multithreaded

With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously.

Interpreted

The programs are compiled into the Java Virtual Machine code called bytecode. The bytecode is machine-independent and can run on any machine that has a Java interpreter, which is part of the Java Virtual Machine (JVM).

1.4.2 Local Environment Setup

Java SE is freely available from the *http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html*. Follow the instructions to download Java and run the .exe to install Java on your machine. Once Java is installed on the machine, you need to set environment variables to point to correct installation directories.

Setting Up the Path for Windows

Assuming you have installed Java in *c:\Program Files\java\jdk* directory –

- Right-click on 'My Computer' and select 'Properties'.
- Click the 'Environment variables' button under the 'Advanced' tab.
- Now, alter the 'Path' variable so that it also contains the path to the Java executable. Example, if the path is currently set to *'C:\WINDOWS\SYSTEM32'*, then change your path to read *'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'.*

Setting Up the Path for Linux, UNIX, Solaris, FreeBSD

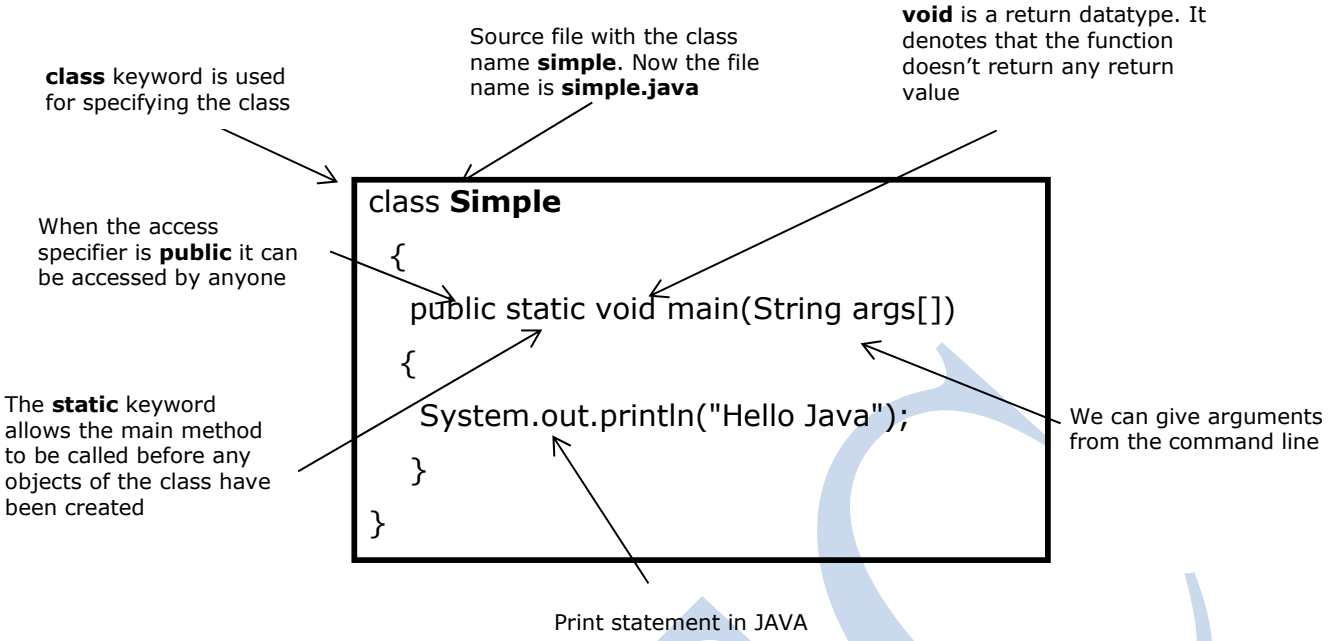
- Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your shell documentation, if you have trouble doing this.
- Example, if you use *bash* as your shell, then you would add the following line to the end of your '.bashrc: export PATH = /path/to/java:$PATH'

Popular Java Editors

To write your Java programs, you will need a text editor. There are even more sophisticated IDEs available in the market. But for now, the following are considered:

- **Notepad** – On Windows machine, you can use any simple text editor like Notepad, TextPad.
- **Netbeans** – A Java IDE that is open-source and free which can be downloaded from *https://www.netbeans.org/index.html.*
- **Eclipse** – A Java IDE developed by the eclipse open-source community and can be downloaded from *https://www.eclipse.org/.*

1.4.3 First program in Java



- To write the simple program, open notepad by **start menu -> All Programs -> Accessories -> notepad** and write simple program and Save the file as Simple.java
- To compile and run this program, you need to open command prompt by **start menu -> All Programs -> Accessories -> command prompt**
    - To compile: javac Simple.java
    - To execute: java Simple

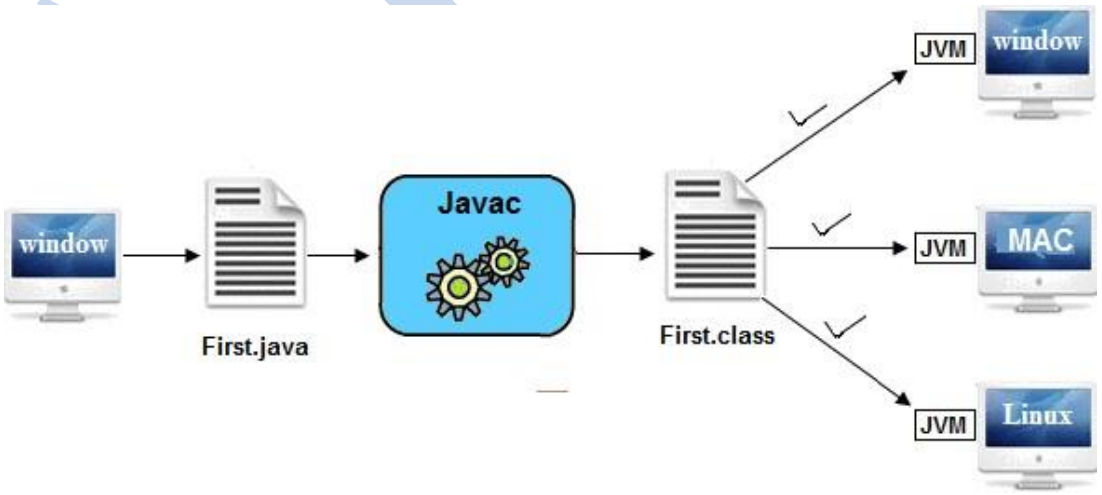
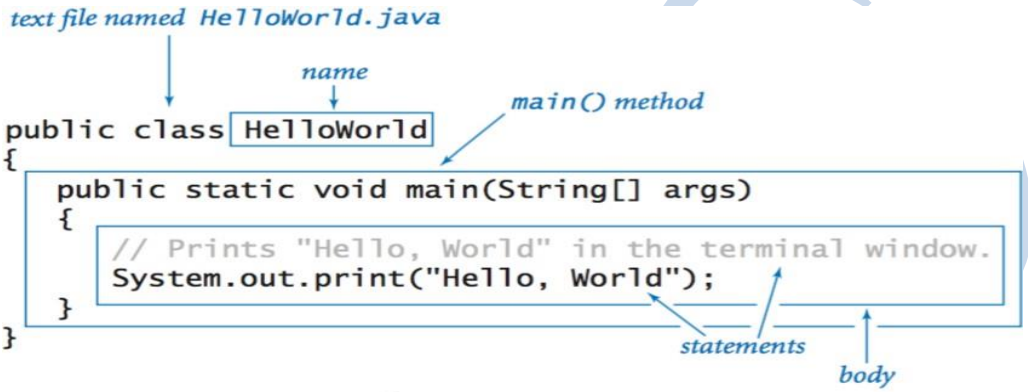


Figure. Working of a Java program

Parameters used in first java program

- **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 creating 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.



1.4.4 Object and Class in Java

In object-oriented programming technique, a program is designed using objects and classes.

Object in Java

Any physical or logical entity that has state and behaviour is known as an object.

- Physical Entity (Tangible → chair, bike, marker, pen, table, car etc.
- Logical Entity (Intangible) →Banking system.

An object has three characteristics:

- State: represents data (value) of an object.
- Behaviour***:*** represents the behaviour (functionality) of an object such as deposit, withdraw etc.
- Identity: Object identity is typically implemented via a unique ID.

Example: **Pen is an object.**

Identity: Reynolds, Marker

State: color (white/black/red/green)

Behaviour: writing, scribbling

Class in Java

A class is a group of objects which have common properties. Class is a blueprint for creating objects by providing member variables, and member functions. It is a logical entity. It can't be physical.

A class in Java contains

- Data members
- Methods
- Constructors
- Blocks
- Nested Class and Interface

Syntax to declare a class:

```
class <class_name>
{
   data members ;
   methods;
}
```

Example

```
class Dog
{
   String breed;
   int age;
   String color;

   void barking()
   {
     }

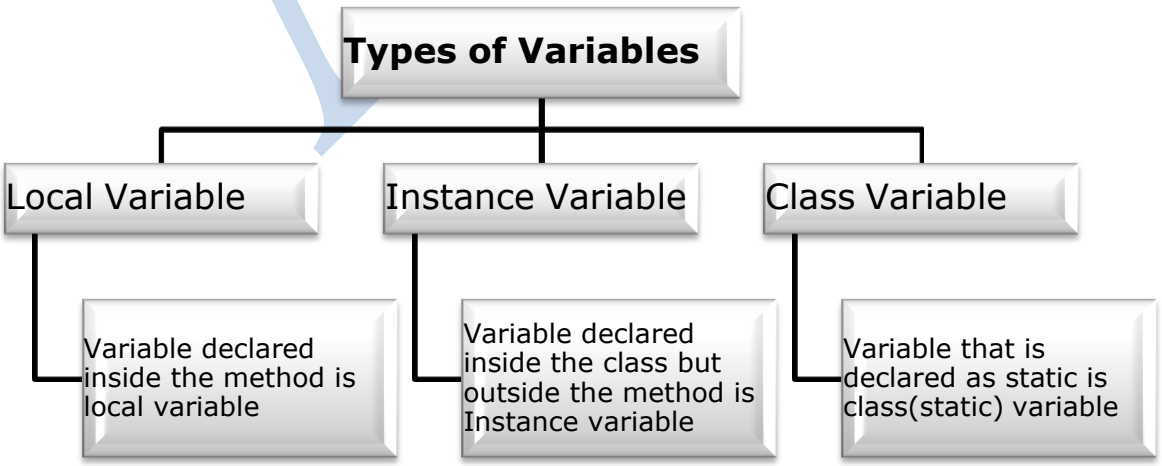
   void hungry()
   {
     }

   void sleeping()
   {
     }
}
```

Data Members:
String breed;
int age;
String color;

Methods:
void barking()
void hungry()
void sleeping()

1.4.4.1 Types of variables in Java



Example to understand types of variables in Java

```
class A
{
   int data = 50;
   static int m = 100;

   void sum()
   {
      int n = 90;
    }
}
```

End of Method

End of Class

Data Members:
int data;
static int m;

Methods:
void sum()

Local Variable:
int n = 90;

Instance variable:
int data = 50;

Class variable:
static int m = 100;

1.4.4.2 Object and Class Example: main() function within class

Filename : Student.java

```
class Student
{
   int id;
   String name;

   public static void main(String args[])
   {
    Student s1=new Student();
    System.out.println(s1.id);
    System.out.println(s1.name);
   }
}
```

Output:
0
null

Class name:
Student

Data members (Both are instance variables)
int id;
String name;

Objects:
s1 is the object of the class Student

Note:
Members and member functions are accessed thro
Objectname. membername
Or
Objectname.methodname

Note:

The **new keyword** is used to allocate memory at run time. All objects get memory in Heap memory area.

1.4.4.3 Object and Class Example: main() function outside the class

In previous program, we are creating main() method inside the class. Now, we create classes and define main() method in another class. In real time development, we create classes and use it from another class. This is a better way than previous one.

File: TestStudent1.java

```
class Student                    // Student is the class name
{
   int id;                       //field or data member or instance variable
   String name;
 }

Class TestStudent1
{
   public static void main(String args[])
  {
    Student s1=new Student();          //s1 is the object of Student
    System.out.println(s1.id);         //accessing member through reference variable
    System.out.println(s1.name);
  }
}
```

Output:
0
null

1.4.4.4 Three Ways to initialize object

There are 3 ways to initialize object in java.

- By reference variable
- By method
- By constructor

Object and Class Example: Initialization through reference

Initializing object simply means storing data into object. Let's see a simple example where we are going to initialize object through reference variable.

File: TestStudent1.java

```
class Student
{
   int id;
   String name;
 }
class TestStudent1
{
 public static void main(String args[])
{
  Student s1=new Student();
  s1.id=101;
  s1.name="Sunil";
  System.out.println(s1.id+" "+s1.name);          //printing members with a white space
 }
}
```

Output:
101 Sunil

We can also create multiple objects and store information in it through reference variable.

File: TestStudent1.java

```
class Student
{
   int id;
   String name;
  }

class TestStudent1
{
     public static void main(String args[])
    {
       //Creating objects
      Student s1=new Student();
      Student s2=new Student();
       //Initializing objects
       s1.id=101;
       s1.name="Sunil";
       s2.id=102;
       s2.name="Amit";
       //Printing data
       System.out.println(s1.id+" "+s1.name);
       System.out.println(s2.id+" "+s2.name);
     }
}
```

s1 and s2 are the objects of the class Student

Output:
101 Sunil
102 Amit

Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insert() method. Output are displayed using the display() functions.

As you can see in the figure, object gets the memory in heap memory area. The reference variable refers to the object allocated in the heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

File: TestStudent1.java

```
class Student
{
    int rollno;
    String name;

    void insert(int r, String n)
    {
        rollno=r;
        name=n;
     }
   void display()
   {
      System.out.println(rollno+" "+name);
     }
}
class TestStudent1
{
    public static void main(String args[])
  {
    Student s1=new Student();
    Student s2=new Student();
    s1.insert(111,"karan");
    s2.insert(222,"Aryan");
    s1.display();
    s2.display();
   }
}
```

Class name:
Student
TestStudent1

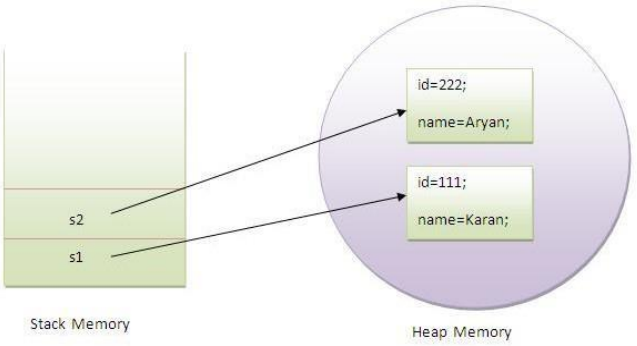
Data members (Both are instance variables)
int rollno;
String name;

Methods:
insert()
display()

Objects:
s1 and s2 are the objects of the class Student

Note:
Main is outside the class "student"

Output:
111 karan
222 Aryan



Write a simple java program to create employee details such as empno, empname, empsalary using objects and classes.

File: TestEmployee.java

```
class Employee
{
   int id;
   String name;
   float salary;

   void insert(int i, String n, float s)
   {
      id=i;
      name=n;
      salary=s;
   }
   void display()
   {
     System.out.println(id+" "+name+" "+salary);
   }
}

 public class TestEmployee
{
   public static void main(String[] args)
  {
   Employee e1=new Employee();
   Employee e2=new Employee();
   Employee e3=new Employee();
   e1.insert(101,"Amir",45000);
   e2.insert(102,"Aryan",25000);
   e3.insert(103,"Suman",55000);
   e1.display();
   e2.display();
   e3.display();
 }
}
```

Class name:
Employee
TestEmployee

Data members (Instance variables)
int id;
String name;
float salary;

Methods:
insert()
display()

Objects:
e1, e2, e3

Note:
Main is outside the class "Employee"

Output:
101 Amir 45000.0
102 Aryan 25000.0
103 Suman 55000.0

APEC

Object and Class Example: Initialization through Constructor

Define Constructor:

A constructor in Java is a block of code similar to a method that's called when an object is created.

Rules for declaring a Constructor:

- A constructor doesn't have a return type.
- The name of the constructor must be the same as the name of the class.
- Unlike methods, constructors are not considered members of a class.
- A constructor is called automatically when an object is created.

Syntax for Constructor:

Syntax:		Example:

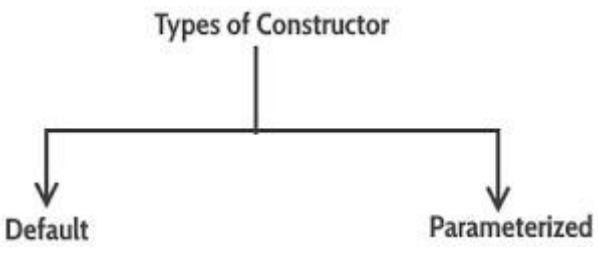
class **classname**
{
…
classname()
{
…
}
}
	Class name and constructor name are same.
	
	Here, Box is the name of the class. Constructor name is also Box
		class **Box**
		{
		…
		Box()
		{
		…
		}
		}

```
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.

Types of Constructor:



Default Constructor:

- A constructor that has no parameters is known as default constructor.
- Default Constructor is also called as no argument constructor

Syntax:

```
class classname
{
   …
   classname()
   {
     …
    }
}
```

Example:

```
class Box
{
   …
   Box()
   {
     …
    }
}
```

Program to explain Default Constructor

```
public class Box
{
   Box ()
   {
     System.out.println("Box is created");
   }

public static void main(String[] args)
{
   Box b1 = new b1();
  }
}
```

Output:
Box is created

Note:
In this example, we are creating the no-arg Constructor in the Box class. It will be invoked at the time of Object Creation.

Example 2:

```
class Rectangle
{
    int length;
    int breadth;

    Rectangle()
    {
       length = 10;
       breadth = 10;
     }

    void setDim()
    {
      length = 20;
      breadth = 20;
}
```

```
public static void main(String args[])
{
    Rectangle r1 = new Rectangle();
    System.out.println("Length of Rectangle : " + r1.length);
    System.out.println("Breadth of Rectangle : " + r1.breadth);

    r1.setDim();
    System.out.println("Length of Rectangle : " + r1.length);
    System.out.println("Breadth of Rectangle : " + r1.breadth);
    }
}
```

Output:

```
Length of Rectangle :10
Breadth of Rectangle :10
Length of Rectangle :20
Breadth of Rectangle :20
```

If there is no constructor in a class, compiler automatically creates a default constructor. The Purpose of Default Constructor is to default values to the object like 0, null etc. depending on the datatype.



Filename : Student.java

```
class Student
{
   int id;
   String name;
   void display()
   {
    System.out.println(id + " " + name);
    }
   public static void main(String args[])
  {
    Student s1=new Student();
    s1.display();
  }
}
```

Output:
0 null

Note:
- In the above class, we are not creating any constructor so compiler provides you a default constructor.
- Here 0 and null values are provided by default constructor.

Parameterized Constructor

Constructors that have parameters are known as parameterized constructor. They are used to provide different values to the distinct objects.

Syntax:

```
class classname
{
  …
  classname(datatype arg1, datatype arg2,…)
 {
   …
   }
}
```

Example:

```
class Box
{
   int length;
   int breadth;

  Box(int l, int b)
 {
    length = l;
    breadth = b;
  }
}
```

Note: Constructor can have any number of arguments. In this example, its 2 argument Constructor.

Example 2:

```
class Rectangle
{
   int length;
   int breadth;

   Rectangle(int  l, int b)
   {
      length = l;
      breadth = b;
     }

   public static void main(String args[])
   {
      Rectangle r1 = new Rectangle(20, 10);
      System.out.println("Length of Rectangle : " + r1.length);
      System.out.println("Breadth of Rectangle : " + r1.breadth);
   }
}
```

Output:
Length of Rectangle :20
Breadth of Rectangle :10

Here, an object r1 is created for the class Rectangle. The arguments are passed to the constructor as 20, 10

Overloading Constructors in Java

Constructor overloading is a concept of having more than one constructor with different parameters list, in such a way so that each constructor performs a different task.

Syntax:	Example:

class **classname**
{
classname(datatype arg1, datatype arg2,…)
{ }
classname(datatype arg1)
{ }
classname(datatype arg1, datatype arg2)
{ }
classname(datatype arg1, datatype arg2,…)
{ }
}
	class **Box**
	{
	Box()
	{ }
	Box(int l)
	{ }
	Box(int l, int b)
	{ }
	Box(float l, float b)
	{ }
	}

Note: In this example, 4 constructors are there with different number of parameters and different types of parameters.	

Example :

Filename : Rectangle.java

```
public class Rectangle
{
   int length;
   int breadth;
   double weight;

   Rectangle()
   { System.out.println("This is the constructor with zero parameters");
     }
   Rectangle(int  l)
   { length = l;
      System.out.println("This is the constructor with 1 parameter");
      System.out.println("Length of the Box is " + length);
     }

   Rectangle(int  l, int b)
   { length = l;
      breadth = b;
      System.out.println("This is the constructor with 2 parameters");
      System.out.println("Length of the Box is " + length);
      System.out.println("Breadth of the Box is " + breadth);
     }
  Rectangle(int  l, double w)
   { length = l;
      weight = w;
      System.out.println("This is the constructor with mixed datatype");
      System.out.println("Length of the Box is " + length);
      System.out.println("Breadth of the Box is " + weight);
     }

   public static void main(String args[])
  { Rectangle r1 = new Rectangle();
     Rectangle r2 = new Rectangle(20);
     Rectangle r3 = new Rectangle(55, 67);
     Rectangle r4 = new Rectangle(22, 13.5);
   }
}
```

Output:

```
This is the constructor with zero parameters
This is the constructor with 1 parameter
Length of the Box is 20
This is the constructor with 2 parameters
Length of the Box is 55
Breadth of the Box is 67
This is the constructor with float datatype
Length of the Box is 22.9
Breadth of the Box is 13.5
```

AREC

Rectangle r1 = new Rectangle();

Here r1 is the object of the class Rectangle. It automatically invokes the default constructor and the output got is "This is the constructor with zero parameters".

Rectangle r1 = new Rectangle(20);

This statement invokes the one argument constructor of integer datatype.

The output got is:

This is the constructor with 1 parameter

Length of the Box is 20

Rectangle r1 = new Rectangle(55, 67);

This statement invokes the two arguments constructor of integer datatype.

The output got is:

This is the constructor with 2 parameters

Length of the Box is 55

Breadth of the Box is 67

Rectangle r1 = new Rectangle(22.9, 13.5);

This statement invokes the two arguments constructor of float datatype.

The output got is:

This is the constructor with float datatype

Length of the Box is 22.9

Breadth of the Box is 13.5

Use of **"this"** keyword in constructors

this is a reference variable that refers to the current object. It is a keyword in java language represents current class object.

- It can be used to refer current class instance variable.
- this() can be used to invoke current class constructor.
- It can be used to invoke current class method (implicitly)
- It can be passed as an argument in the method call.
- It can be passed as argument in the constructor call.
- It can also be used to return the current class instance.

Why use this keyword in java ?

The main purpose of using this keyword is to differentiate the formal parameter and data members of class, whenever the formal parameter and data members of the class are same then java virtual machine get ambiguity. To differentiate between formal parameter and data member of the class, the data member of the class must be preceded by "this".

"this" keyword can be use in two ways.

- this .
- this()

WITHOUT "this" keyword

Filename : Rectangle.java

```
class Rectangle
{
   int length;
   int breadth;

   Rectangle(int  length, int breadth)
   {
     length = length;
     breadth = breadth;
     }

   public static void main(String args[])
   {
      Rectangle r1 = new Rectangle(20, 10);
      System.out.println("Length of Rectangle : " + r1.length);
      System.out.println("Breadth of Rectangle : " + r1.breadth);
   }
}
```

Instance variables and formal parameters are same. There is a ambiguity between Instance variable and local variables

The compiler understand "Assign to itself"

Due to ambiguity the compiler prints the default values of the Instance variables

Output:

Length of Rectangle :0
Breadth of Rectangle :0

Using "this" keyword

Filename : Rectangle.java

```
class Rectangle
{
   int length;
   int breadth;

   Rectangle(int  length, int breadth)
   {
     this.length = length;
     this.breadth = breadth;
     }

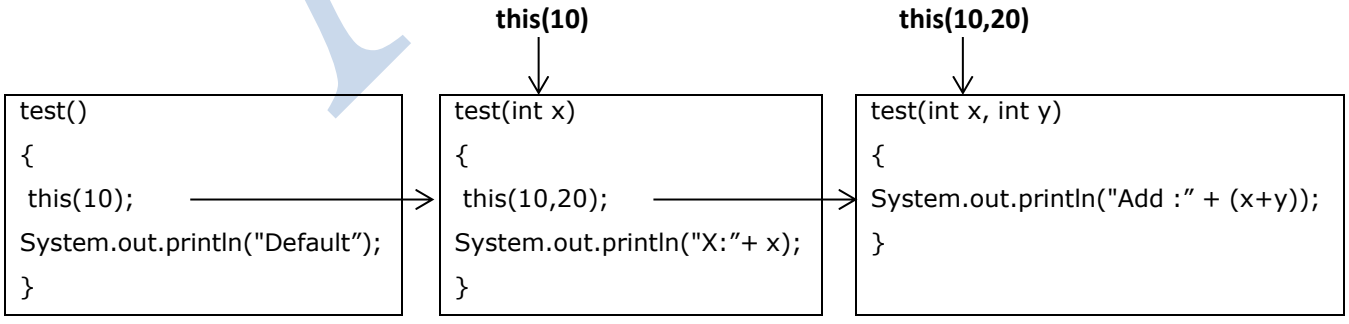
   public static void main(String args[])
  {
      Rectangle r1 = new Rectangle(20, 10);
      System.out.println("Length of Rectangle : " + r1.length);
      System.out.println("Breadth of Rectangle : " + r1.breadth);
   }
}
```

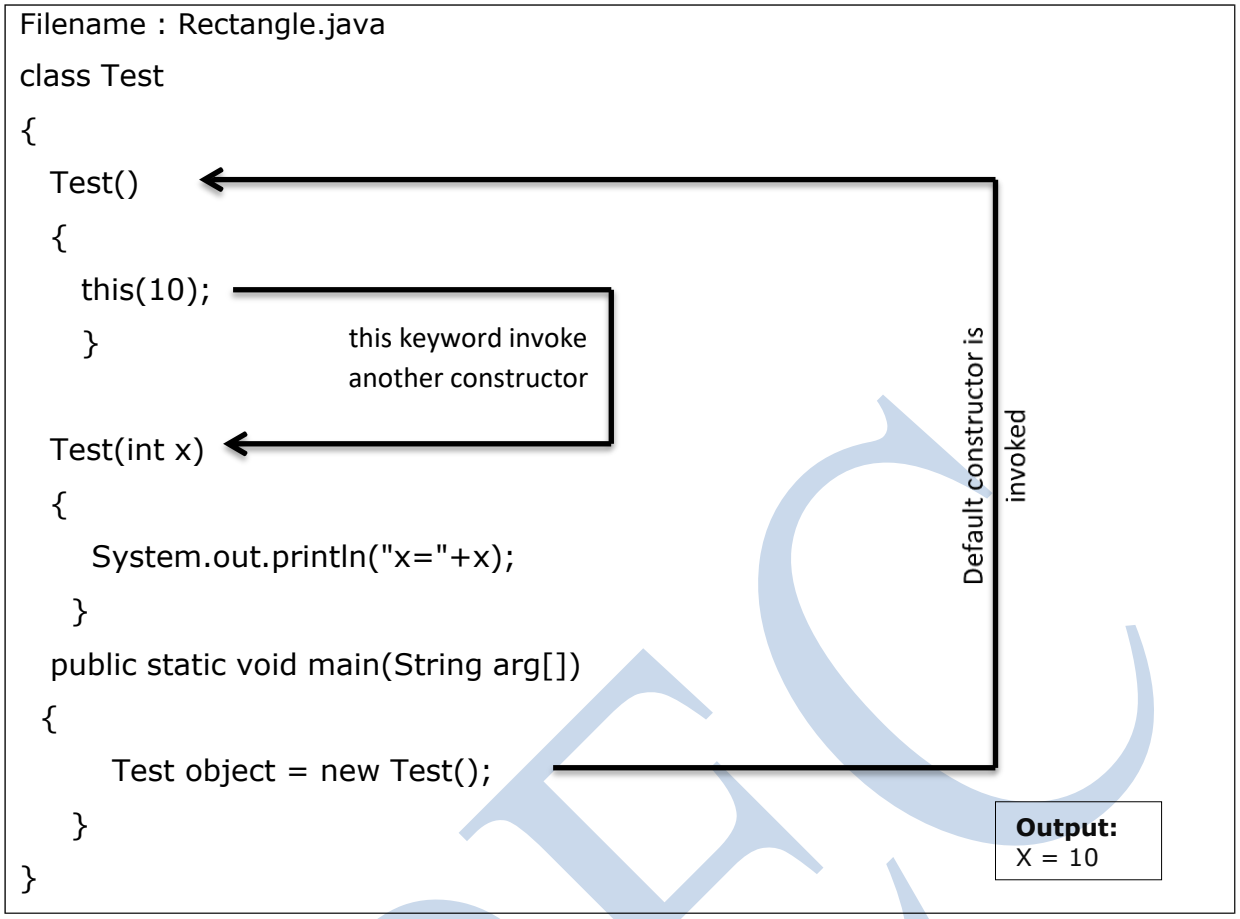
Formal parameters and Instance Variables are same. We are using **this** keyword to distinguish between local variable and instance variable.

Output:
Length of Rectangle :20
Breadth of Rectangle :10

Constructor chaining in Java

Constructor chaining is a phenomena of calling one constructor from another constructor of same class. Since constructor can only be called from another constructor in Java, constructor chaining is used for this purpose.





Java Copy Constructor

There is no copy constructor in java. But, we can copy the values of one object to another. There are many ways to copy the values of one object into another in java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

Java Copy Constructor: By constructor

A copy constructor is a constructor that creates a new object using an existing object of the same class. In this example, we are going to copy the values of one object into another using java constructor.

Filename : Student.java

```
class Student
{
   int id;
   String name;

   Student(int i,String n)
   { id = i;
     name = n;
    }

   Student (Student s)
   {  id = s.id;
     name =s.name;
   }

   void display()
  {  System.out.println(id+" "+name);
    }

   public static void main(String args[])
  { Student s1 = new Student6(1901,"Gayu");
    Student s2 = new Student(s1);
     s1.display();
     s2.display();
     }
}
```

Output:
1901 Gayu
1901 Gayu

APEC

Java Copy Constructor: By Assigning the Values

Here, we copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

Filename : Student.java

```
class Student
{
   int id;
   String name;

   Student(int i,String n)
   { id = i;
     name = n;
    }

   Student ()
   {    }

   void display()
  {  System.out.println(id+" "+name);
     }

   public static void main(String args[])
  {  Student s1 = new Student6(1901,"Gayu");
     Student s2 = new Student();
     s2.id = s1.id;
     s2.name = s1.name;
     s1.display();
     s2.display();
     }
 }
```

Output:
1901 Gayu
1901 Gayu

Assign values of object s1 to object s2

APEC

Difference between Java Constructor and Java Method

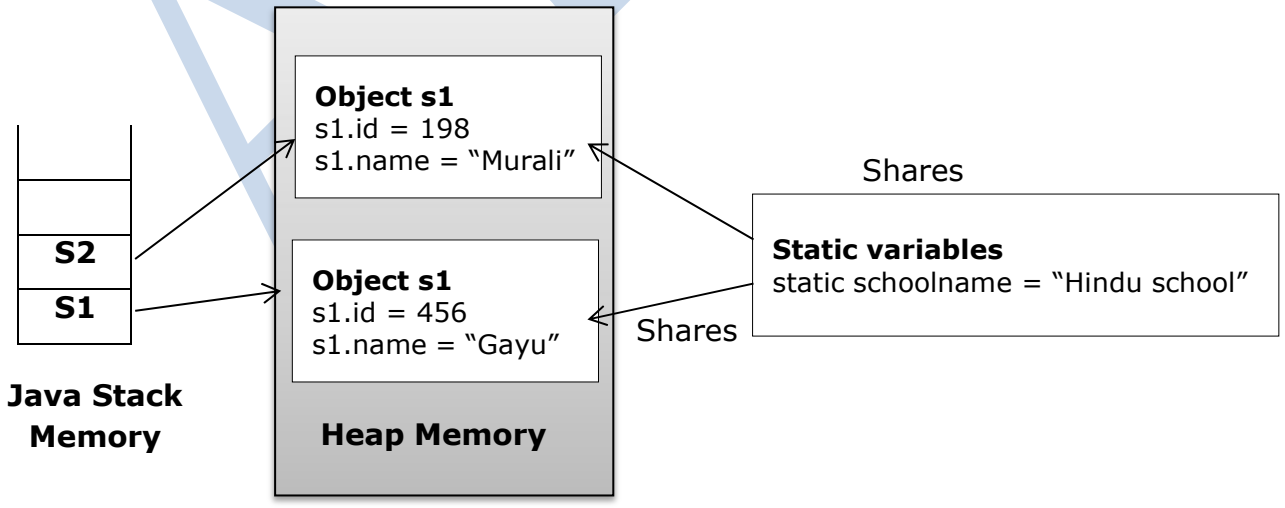
Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

1.4.5 "Static" keyword in Java

The static keyword in java is mainly used for memory management. Static keyword can be applied to variables, methods, blocks and nested class. In java language static keyword can be used for following:

- Variable (also known as class variable)
- Method (also known as class method)
- Block
- Nested class

Static Variable:



If a field is declared static, then exactly only one copy of that field is created and shared among all instances of that class. The value of this static field will be shared across all objects of the class.

Simple program to explain "static" variable in Java:

```
Filename : Student.java

class Student
{
   int id;
   String name;
   static String schoolname = "Hindu school";

   Student(int i,String n)
   { id = i;
     name = n;
    }

   void display()
  {  System.out.println(id+" "+name+" " schoolname);
     }

   public static void main(String args[])
  {
     Student s1 = new Student(456,"Gayu");
     Student s2 = new Student(198,"Murali");
     s1.display();
      s2.display();
      }
 }
```

schoolname is shared by both objects s1 and s2.

Arguments passed only for id and name

```
Output:
456  Gayu Hindu school
198  Murali Hindu school
```

*Counter program **WITHOUT** "static" variable*

```
Filename : Counter.java

class Counter
{
   int count=0;          //Whenever the object is created, count value is initialized

   Counter()
  {
      count++;
      System.out.println("Count = " + count);
   }

  public static void main(String args[])
  {
    Counter c1=new Counter();
    Counter c2=new Counter();
    Counter c3=new Counter();

 }
```

```
Output:
Count =1
Count =1
Count =1
```

In this example, an instance variable named count is created which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable. If it is incremented, it won't reflect to other objects.

*Counter program **with** "static" variable*

Filename : Counter.java

```
class Counter
{
   static int count=0;
    Counter()
   {
       count++;
       System.out.println("Count = " + count);
    }

   public static void main(String args[])
   {
     Counter c1=new Counter();
     Counter c2=new Counter();
     Counter c3=new Counter();
 }
```

Static variable will get memory only once and it retains its value

Output:
Count =1
Count =2
Count =3

Static method:

- When "static" keyword is applied to any method, it is known as static method. It belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class. These methods can access static data member and can change the value of it.
- Static members are automatically initialized with the value of zero.

Restrictions for using static method in Java

- Static methods can call other static methods only
- Static methods can access static data only.
- Static methods cannot use "this" or "super"(Super is used in Inheritance)

Syntax:

```
static void  methodname
{
  …
  …
}
```

Program to explain static method

Filename : Calculate.java

```
class Calculate
{
 static int x;

 static void cube(int x)
 {
   System.out.println(x+9);
  }

 public static void main(String args[])
 {
    cube();
   }
}
```

Static method is invoked without creating instance of the class

Output:

9

Note:
Static method can invoke only static data members. If the data member is not static, the compiler raises an error.

Java static block:

Java static block is executed before main method at the time of loading the class.

Filename : small.java

```
class small
{
 static
  {
   int x = 25;
   System.out.println("Static block is invoked");
   System.out.println("Value of x =" + x);
  }

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

Java Static block is invoked before the main()

Output:
Static block is invoked
Value of x = 25
Hello main

Filename : Student.java

```
class Student
 {
    int rollno;
    String name;
    static String college = "IIT";

    static void change()
   {
      college = "BITS";
    }

    Student(int r, String n)
   {
      rollno = r;
      name = n;
    }

    void display ()
   {
     System.out.println(rollno+" "+name+" "+college);
     }

   public static void main(String args[])
   {
   change();

   Student s1 = new Student (1901,"Gayu");
   s1.display();
   }
}
```

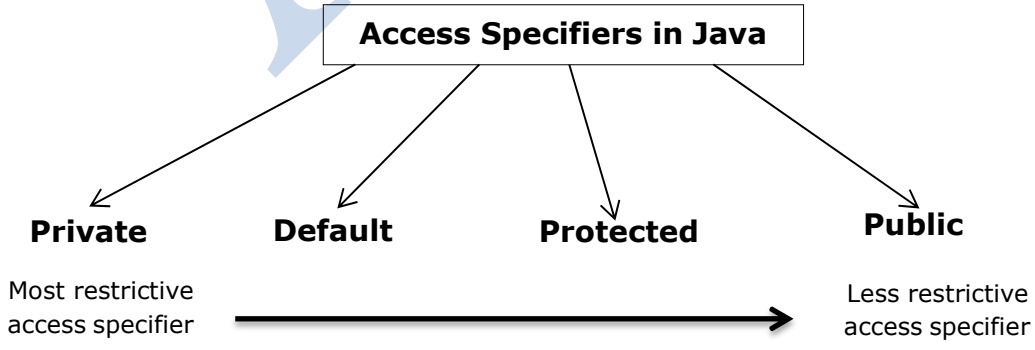
Static method changes the value of the static data member from IIT to BITS

Output:
1901 Gayu BITS

1.4.6 Access Specifiers in Java

In Java, methods and data members of a class/interface can have one of the following four access specifiers. The access specifiers are listed as:

- Private
- Default
- Protected
- Public



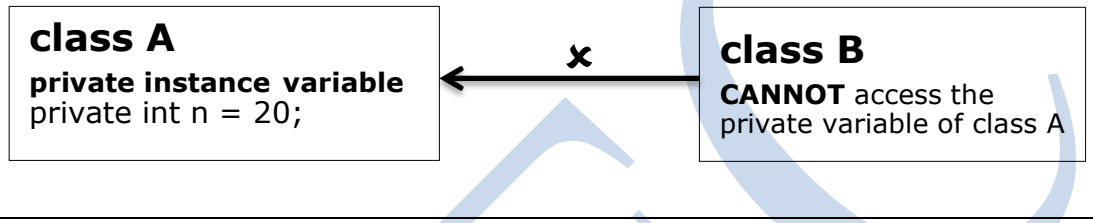
Access modifiers provide features accessing and controlling mechanism among the classes and interfaces.

Accessibility details:

Private

- Used for most instance variables
- private variables and methods are accessible only to methods of the class in which they are declared
- Declaring instance variables private is known as data hiding

Example:



File: B.java

```
class A
{
   private int n = 20;        // private instance variable
  }

public class B                // public class
{
  public static void main(String args[])
  {
      A obj =new A();
      System.out.println(obj.n);
   }
}
```

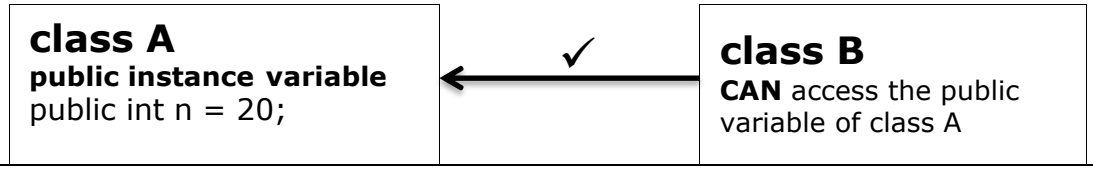
class B **CANNOT** invoke the private variable of class A

Output:
error: n has private access in A

Public

- Exposes variables or methods outside the Class.
- Declaring public methods is known as defining the class' public interface.

Example:



File: B.java

```
class A
{
    public int n = 20;          // public instance variable
  }

public class B                  // public class
{
   public static void main(String args[])
   {
       A obj =new A();
       System.out.println(obj.n);
    }
}
```

class B invoke the public variable of class A

Output:
20

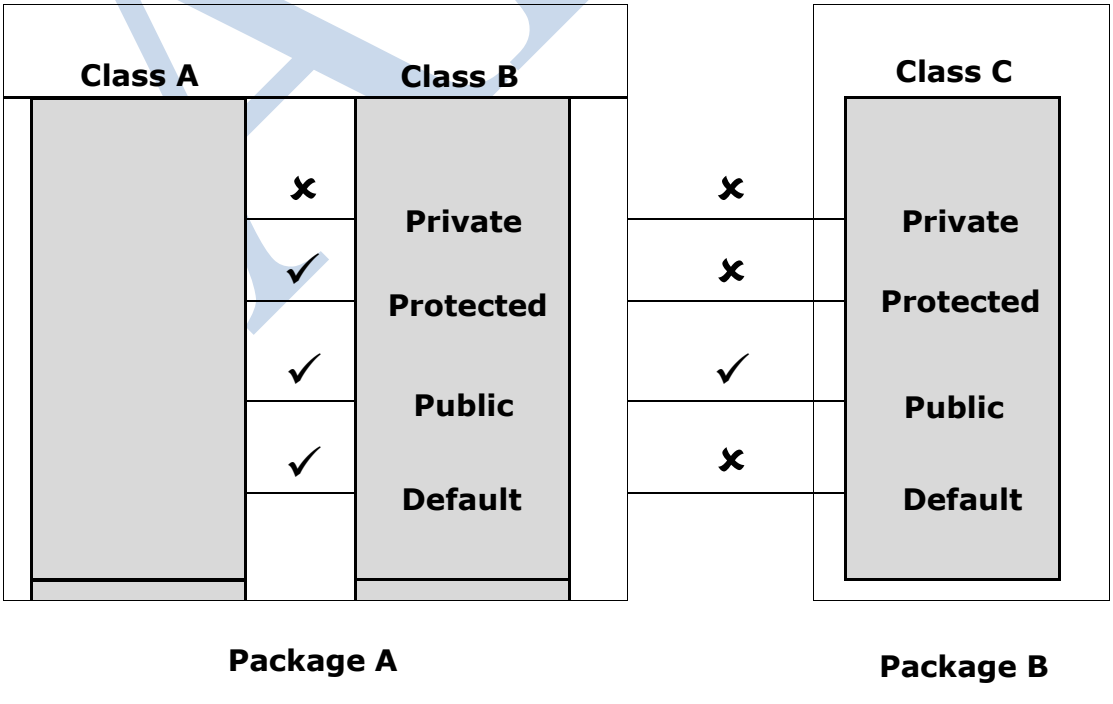
Protected

- Provides limited access
- Provides access to sub classes but not to the classes outside a package

Note:

protected keyword will be explained in detail in Unit II

Rules for access specifiers:



- Anything declared default can be accessed only within the classes of same package.
- Anything declared public can be accessed from anywhere
- Anything declared private cannot be accessed outside the class
- Anything declared protected can be accessed by the sub classes or other classes inside the same package but not accessed outside the package.

Access Specifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

Comment statement in Java

Single line comments

- These comments start with //
- Example: // this is comment line

Multi line comments

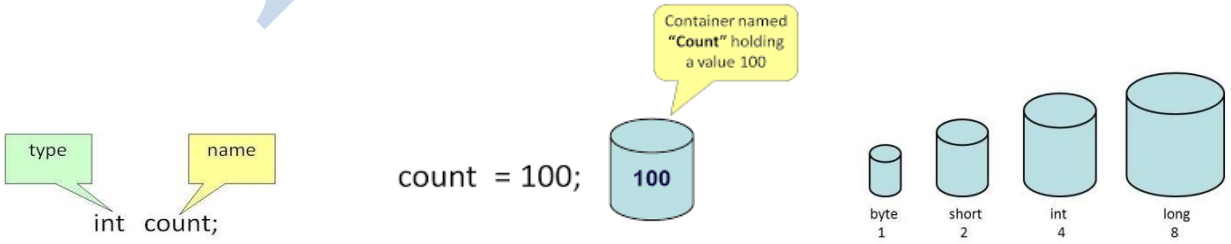
- These comments start with /* and end with */
- Example: /* this is comment line*/

Documentation comment

- /** documentation */
- It is generally called as doc comment. JDK javadoc tool uses doc comments when preparing automatically generated documentation.

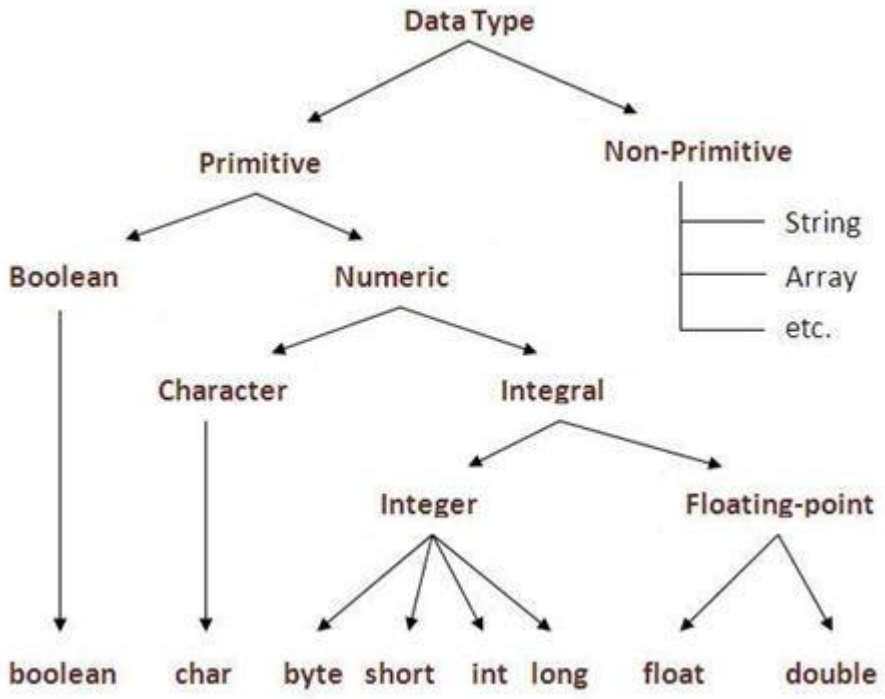
Variables and Data Types in Java

Variables are nothing but reserved memory locations to store values. It means when a variable is created, some space in the memory is reserved.



Data types represent the different values to be stored in the variable. In java, there are two types of data types:

- Primitive data types
- Non-primitive data types



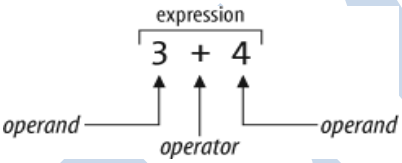
Data Type	Description	Default value	Size	Range	Example
byte	2's complement integer. byte is 4 times smaller than an integer	0	1 byte	-2^7 to $2^7 -1$	byte a = 100
short	2's complement integer. short is 2 times smaller than an integer	0	2 bytes	-2^{15} to $2^{15} -1$	short s = 10000
int	2's complement integer	0	4 bytes	-2^{31} to $2^{31} -1$	int a = 100
long	2's complement integer	0L	8 bytes	-2^{63} to $2^{63} -1$	long a = 1000L
float	single-precision 32-bit IEEE 754 floating point	0.0f	4 bytes	-2^{31} to $2^{31} -1$	float f1 = 234.5f
double	double-precision 64-bit IEEE 754 floating point	0.0d	8 bytes	-2^{63} to $2^{63} -1$	double d1 = 123.4
char	Unicode character	'\u0000'	2 bytes	\u0000 to \uFFFF	char b = 'A'
boolean	Contain values true or false	false	1 bit	NA	boolean xxx = true

Examples on datatypes:

Widening	Narrowing and Typecasting	Overflow	Byte addition

int a=10;
float f=a;
System.out.println(f);
 	
Output:
10.0
	float f=10.5f;
	//int a=f; Compile error
	
	int a=(int)f;
	
	System.out.println(f);
	System.out.println(a);
	
	Output:
	10.5
	10
		int a=130;
		byte b=(byte)a;
		System.out.println(a);
		System.out.println(b);
		
		Output:
		130
		-126
			byte a=10;
			byte b=10;
			
			//byte c=a+b;
			//Compile Error: because a+b=20 will be int
			
			byte c=(byte)(a+b);
			System.out.println(c);
			
			Output:
			20

Operators in Java



An operator is a symbol that tells the compiler to perform mathematical or logical calculations on operands (variables).

Binary Operators:

Operators that have two operands are called binary operators.

Unary Operators:

C provides two unary operators for which only one variable is required.

List of operators in Java are:

- Arithmetic Operators
- Relational or comparison Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Increment and decrement Operators

Arithmetic operators:

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Operator	Name	Expression	Example	Output
+	Addition	x+y	15 + 4	19
-	Subtraction	x-y	15 - 4	11
*	Multiplication	x*y	15 * 4	60
/	Division	x/y	15 / 4	3.75
%	Modulus	x%y	15 % 4	3
++	Unary plus	x++	20++	21
--	Unary minus	x--	15--	14

Relational operators

Comparison operators are also called Relational operators. They are used to compare values and return True or False condition.

Operator	Name	Expression	Example	Output
>	Greater than	x>y	54 > 18	True
<	Lesser than	x<y	54 < 18	False
==	Equal to	x==y	54 == 18	False
!=	Not Equal to	x!=y	54 != 18	True
>=	Greater than or equal to	x>=y	54 >= 18	True
<=	Lesser than or equal to	x<=y	54 <= 18	False

Assignment operators

Assignment operators are used to assign a value or an expression or a value of a variable to another variable.

Operator	Name	Example	Equivalent to
=	Equal to	x = 10	x = 10
+=	Plus Equal to	x += 10	x = x + 10
-=	Minus Equal to	x -= 10	x = x - 10
*=	Multiplication Equal to	x *= 10	x = x * 10
/=	Division Equal to	x /= 10	x = x / 10
%=	Modulus Equal to	x %= 10	x = x % 10
&=	AND Equal to	x &= 10	x = x &10
|=	OR Equal to	x |= 10	x = x | 10
^=	XOR Equal to	x ^= 10	x = x ^ 10
>>=	Right shift Equal to	x >>= 2	x = x >>2
<<=	Left shift Equal to	x >>= 2	x = x >>2

Logical operators

Logical operators are used to combine the results of two or more conditions.

Operator	Name	Expression	Explanation
&&	Logical and	x && y	True if both the operands are true
||	Logical or	x || y	True if either of the operands is true
!	Logical not	!x	True if operand is false

Bitwise Operators

Bitwise operators are used to manipulate the data at the bit level. Numbers are represented as bits (0's and 1's).

A	B	A | B	A & B	A^ B	~A
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

Bitwise AND

- Both inputs = 1, then output = 1.
- If one or both bits are 0, the output = 0 for that bit

Bitwise OR

- One or both inputs = 1, the output is 1.
- If both bits are 0, the output = 0 for the bit.

For Bitwise XOR

- One input is 1 and the other 0, the output = 1.
- If both bits are 1 and both are 0, the output is 0 for the bit.

Operator	Name	Expression	Explanation
&	Bitwise AND	X & y	Both operands are true, then the result is true
|	Bitwise OR	x | y	If either of the operands are true, then the result is true
~	Bitwise NOT	~x	It complements the bits
^	Bitwise XOR	x ^ y	Result is true, when either of the operands are true, but not both
>>	Bitwise right shift	x>> 2	Operands are shifted right by the number of times specified
<<	Bitwise left shift	x<< 2	Operands are shifted left by the number of times specified

Difference between logical && and bitwise &:

- Logical && - Do not evaluate the second operand if the first operand is false
- Logical || - Do not evaluate the second operand if the first operand is true
- But, bitwise & and |, always evaluate their operands

Increment and decrement operators

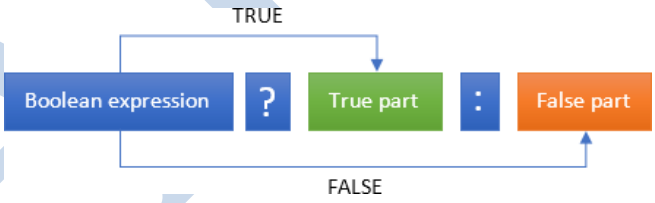
Java programming has two operators: auto increment ++ and auto decrement -- to change the value of an operand (constant or variable) by 1. Auto increment ++ increases the value by 1 whereas auto decrement -- decreases the value by 1. These two operators are unary operators, as they only operate on a single operand.

i = 4;	i is 4	
j = ++i;	j is 5, i is 5	(i was incremented before use)
j = i++;	j is 5, i is 6	(i was incremented after use)
j = --i;	j is 5, i is 5	(i was decremented before use)
j = i--;	j is 5, i is 4	(i was decremented after use)
i++;	i is 5	(i was incremented)

Ternary operator

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

Syntax: $result = Boolean\ expression\ ?\ True\ statement\ :\ False\ statement$




```
File: A.java
public class A
{
  public static void main(String args[])
  {
   int x = 5;
   System.out.println((x>10)?"Hello":"Goodbye");
   System.out.println((x++<6)?"Hello":"Goodbye");
   System.out.println((++x<6)?"Hello":"Goodbye");
    }
}
```

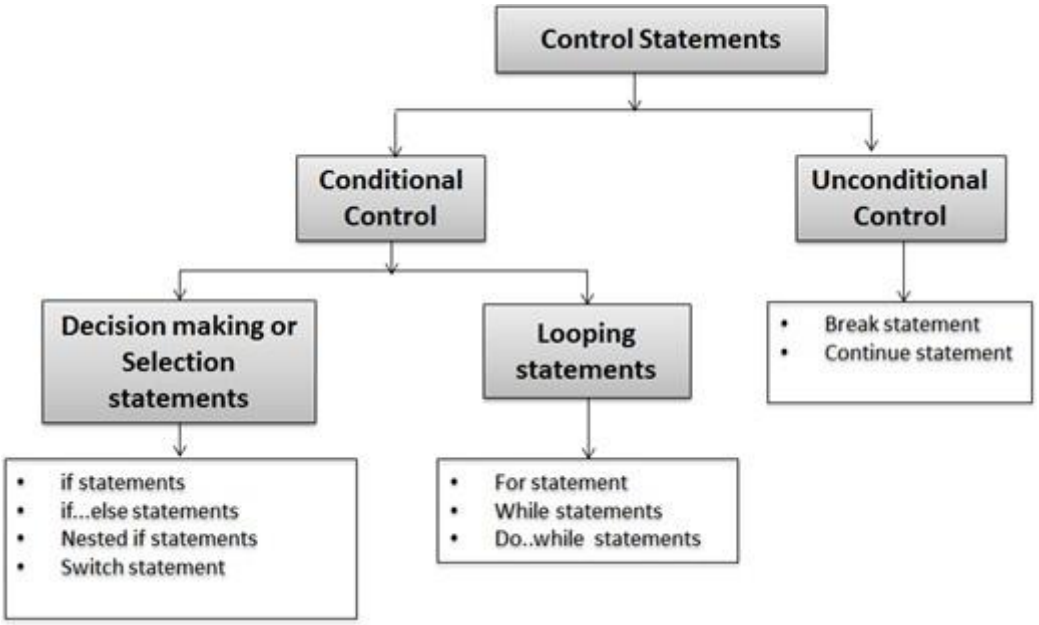
Output:
Goodbye
Hello
Goodbye

Java Precedence and Associativity of Operators:

If more than one operator is involved in an expression, C language has a predefined rule of priority for the operators. This rule of priority of operators is called operator precedence. Operator associativity is used to determine the order of operators with equal precedence evaluated in an expression. In c programming language, when an expression contains multiple operators with equal precedence, we use associativity to determine the order of evaluation of those operators.

Level	Operators	Description	Associativity
15	() [] -> . ++ --	Function Call Array Subscript Member Selectors Postfix Increment/Decrement	Left to Right
14	++ -- + - ! ~ (type) * & sizeof	Prefix Increment / Decrement Unary plus / minus Logical negation / bitwise complement Casting Dereferencing Address of Find size in bytes	Right to Left
13	* / %	Multiplication Division Modulo	Left to Right
12	+ -	Addition / Subtraction	Left to Right
11	>> <<	Bitwise Right Shift Bitwise Left Shift	Left to Right
10	< <= > >=	Relational Less Than / Less than Equal To Relational Greater / Greater than Equal To	Left to Right
9	== !=	Equality Inequality	Left to Right
8	&	Bitwise AND	Left to Right
7	^	Bitwise XOR	Left to Right
6	|	Bitwise OR	Left to Right
5	&&	Logical AND	Left to Right
4	||	Logical OR	Left to Right
3	?:	Conditional Operator	Right to Left
2	= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operators	Right to Left
1	,	Comma Operator	Left to Right

Control flow statements in Java



DECISION MAKING STATEMENTS

Decision making is about deciding the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met. Following is the general form of decision making structure found in most of the programming languages.

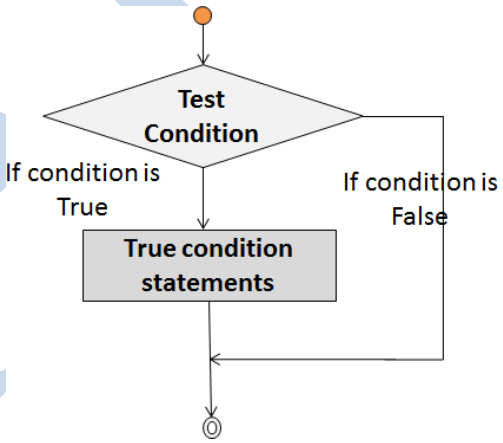


Figure Decision making structure

Java supports the following types of conditional checking statements.

- *if statements*
- *if...else statements*
- *Nested if statements*

- *Switch statement*

if statements

The program evaluates the test condition and executes the statement(s) only if the text condition is True.

Syntax:

$if(test\ condition)$

{

$True\ condition\ statement(s);$

}

Flowchart for if statement:

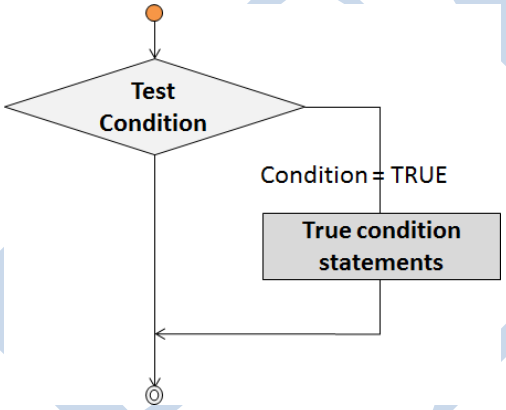


Figure Flowchart for if statement

File: Test.java

```
public class Test
{
   public static void main(String[] args)
   {
      int x = 5;
      if (x<10)
        System.out.println("Hello");
   }
}
```

Output:
Hello

if..else statements

The program evaluates the test condition and executes the statement(s) only if the text condition is True. If the text condition is False, the else statement(s) are executed.

Syntax:

```
if (test condition)
{
    True condition statement(s);
     }
else
  {
    False condition statement(s);
     }
```

Flowchart for if...else statements:

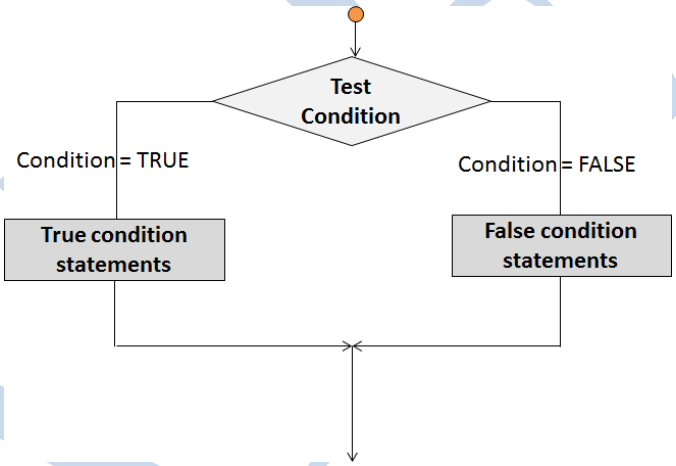


Figure Flowchart for if...else statements

```
File: Test.java
public class Test
{
   public static void main(String[] args)
   {
      int x = 5;
      if (x>10)
         System.out.println("Hello");
      else
         System.out.println("Goodbye");
   }
}
```

Output:
Goodbye

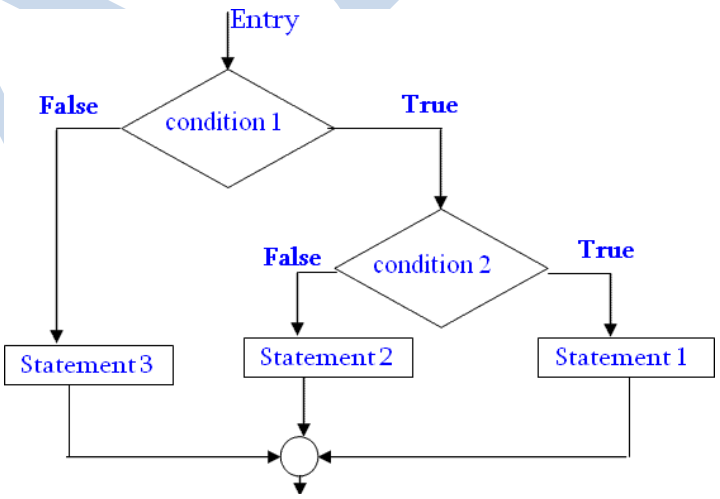
Nested if statements

if..else statement can be used inside another if..else statement. This is called nesting in computer programming.

Syntax:

```
if( outer test condition )
{
    if( inner test condition )
    {
        True statement(s);
        }
    else
   {
      False statement(s);
     }
else
{
    False statement(s);
}
```

Flowchart for nested...if statements:



File: Test.java

```
public class Test {
public static void main(String[] args)
{
   int marks=65;

   if(marks<50){
      System.out.println("fail");
   }
   else if(marks>=50 && marks<60){
      System.out.println("D grade");
   }
   else if(marks>=60 && marks<70){
      System.out.println("C grade");
   }
   else if(marks>=70 && marks<80){
      System.out.println("B grade");
   }
   else if(marks>=80 && marks<90){
      System.out.println("A grade");
   }else if(marks>=90 && marks<100){
      System.out.println("A+ grade");
   }else{
      System.out.println("Invalid!");
   }
  }
}
```

Output:
C grade

Switch statements

The switch statement is a multi-way branch statement. Switch statement is a control statement that allows us to choose only one choice among the many given choices. It executes the block of code which matches the case value. If there is no match, then default block is executed.

Syntax:

$switch(expression)$

{

$case\ value\ 1$:

$block\ 1$;

$break$;

$case\ value\ 2$:

$block\ 2$;

$break$;

$case\ value\ 3$:

$block\ 3$;

$break$;

.

.

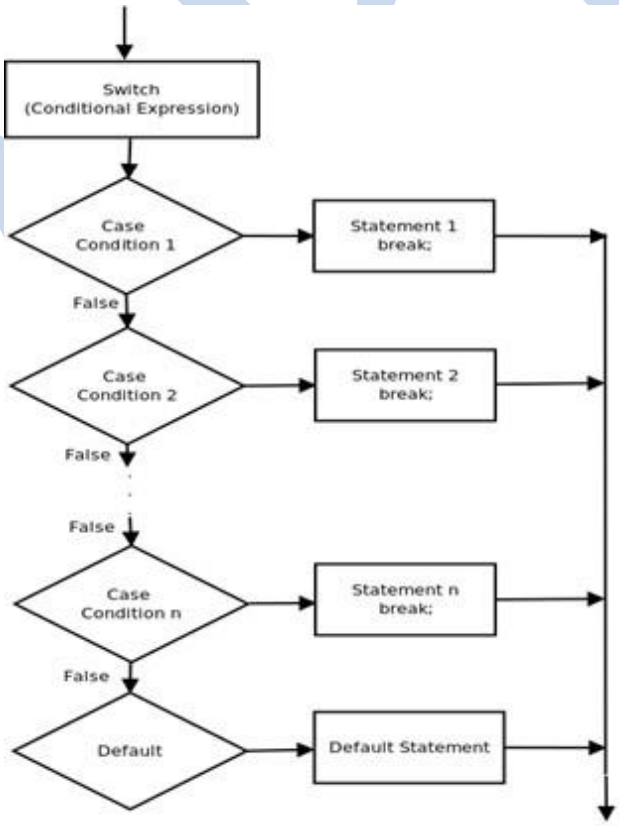
$default$:

$default - block$;

$break$;

}

Flowchart for switch statements:



```
File: Test.java
public class Test{
public static void main(String[] args) {
   int number=20;
   switch(number){
   case 10: System.out.println("10");break;
   case 20: System.out.println("20");break;
   case 30: System.out.println("30");break;
   default:System.out.println("Not in 10, 20 or 30");
   }
}
}
```

Output:
20

LOOPING STATEMENTS

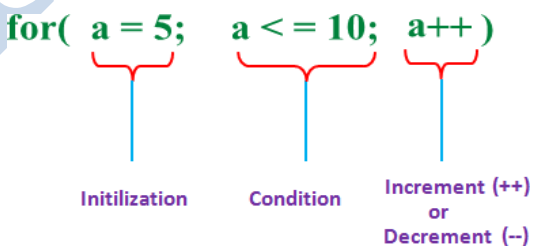
Computers often do repetitive tasks. There are situations when programmers need to execute a block of code several number of times. Repeated execution of a set of statements is called iteration or looping.

Looping statement execute one or more statement repeatedly several number of times. There are three types of loops.

- *for statement*
- *while statement*
- *do-while statement*

FOR loop

For loop executes a block of statements repeatedly until a given condition is true.



Syntax:

$$for\ (initialization;\ condition\ test;\ increment\ or\ decrement)$$
$$\{$$
$$//Statements\ to\ be\ executed\ repeatedly$$
$$\}$$

Flowchart for FOR loop:

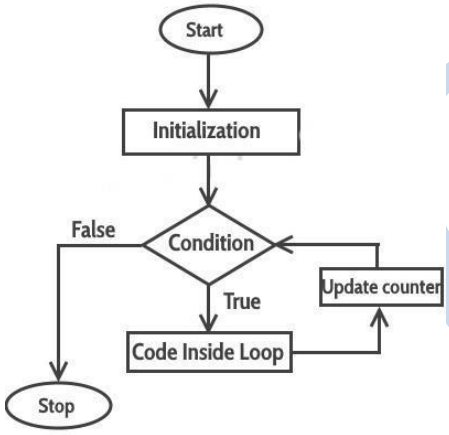


Figure Flowchart – For loop

File: Test.java

```
public class Test
{
   public static void main(String[] args)
   {
      for(int i=1;i<=5;i++)
         System.out.println(i);
   }
}
```

Output:
1
2
3
4
5

WHILE loop

While loop repeatedly executes a set of statement as long as the condition is true.

Syntax:

$$while(condition)$$
$$\{$$
$$statement(s);$$
$$\}$$

Flowchart for While loop:

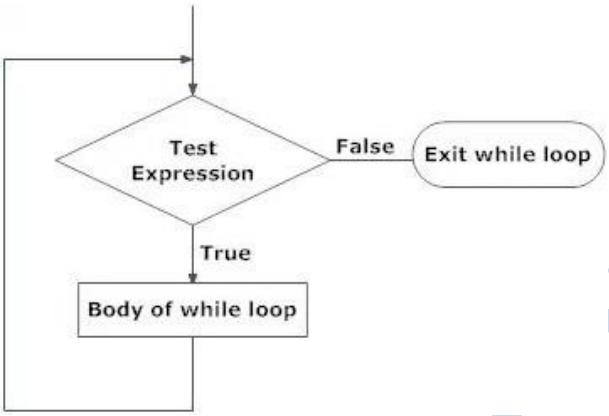


Figure Flowchart – While loop

File: Test.java

```
public class Test
{
   public static void main(String[] args)
   {
      int i=5;
      while(i<10)
      {
        System.out.println(i);
        i++;
      }
   }
}
```

Output:
5
6
7
8
9

DO-WHILE loop

Unlike for and while loops, the do...while loop checks its condition at the bottom of the loop.

Syntax:

$$do$$
$$\{$$
$$statement(s);$$
$$\}\, while(condition);$$

Flowchart for do..while loop:

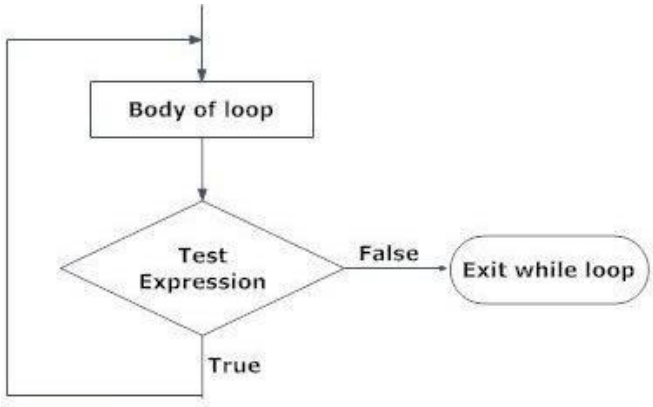


Figure Flowchart – Do While loop

File: Test.java

```
public class Test
{
   public static void main(String[] args)
   {
      int i=6;
      do
      {
        System.out.println(i);
        i++;
      } while(i<10);
   }
}
```

Output:
6
7
8
9

Difference between while and do..while loop	
while loop	**do..while loop**
While loop is an entry controlled loop.	Do..while loops is an exit controlled loop
Since condition is checked first, the looping statements may or may not execute.	Since condition is checked later, the looping statements will execute at least once.

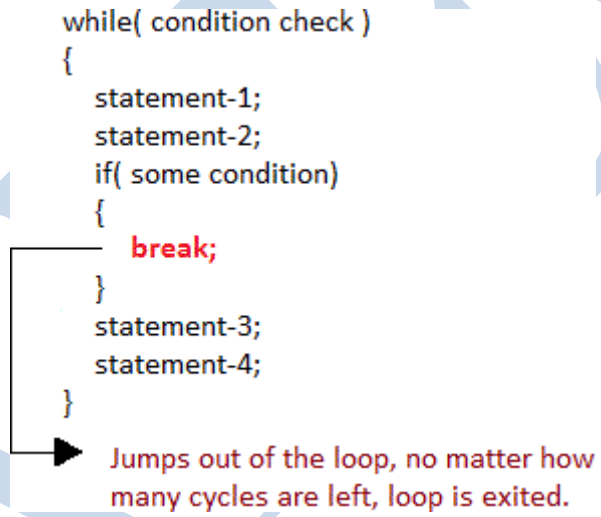
Unconditional control statements

For loop, *while* loop, and *do. . while* loop statements execute a block until a certain condition is met. Sometimes it is desirable to skip some statements inside the loop or terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used.

The break statement:

When the break statement is encountered inside a loop, the loop is immediately terminated and program control jumps to the next statement following the loop.

- It is used along with the Switch statement and looping statements
- Syntax:



File: Test.java

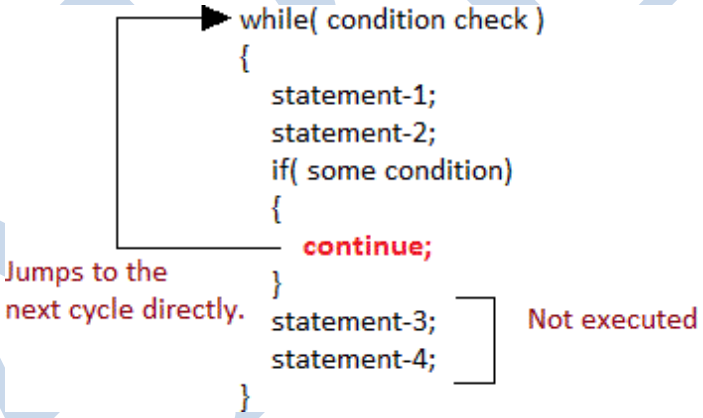
```
public class Test {
public static void main(String[] args)
{
   for(int i=1;i<=10;i++)
   {
      if(i==5)
         break;
      System.out.println(i);
   }
 }
}
```

Output:
1
2
3
4

The continue statement:

The continue statement skips the rest of the loop statements and causes the next iteration of the loop.

Syntax:



File: Test.java

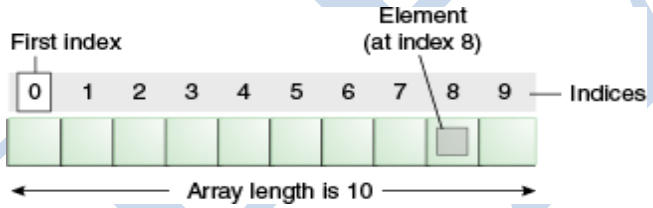
```
public class Test {
public static void main(String[] args)
{
   for(int i=1;i<=10;i++)
   {
      if(i==5)
          continue;
      System.out.println(i);
   }
 }
}
```

Output:

```
1
2
3
4
6
7
8
9
```

Arrays in Java

Array is a collection of similar type of elements that have contiguous memory location. Java array is an object that contains elements of similar data type. Array in java is index based, first element of the array is stored at 0 index.



Types of Arrays:

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array

Syntax:

$$dataType[]\ Var\ =\ new\ dataType[arraySize];$$
$$dataType\ []Var\ =\ new\ dataType[arraySize];$$
$$dataType\ Var[]\ =\ new\ dataType[arraySize];$$

Arrays are referenced with the help of the index values. [] notation to represent the variable as an array. Assume that the array contains "n" integers, then the first element are indexed with the "0" value and the last element are indexed with "n-1" value.

Example:

One Dimensional Array

Example(Array Initialization):

int[] a = new int[5];

a→ array name
5 → Size of the array. It means, it can hold values from 0 to 4

Each individual element in the array is accessed as :

a[0] = 100
a[1] = 200
a[2] = 300
a[3] = 400
a[4] = 500

The above declaration allocates, 5 continuous memory locations of 2 bytes each with the name "a"

Index →	a[0]	a[1]	a[2]	a[3]	a[4]
Values →	100	200	300	400	500

Simple program to explain single dimensional array concept:

File: Test.java

```
public class Test{
public static void main(String args[]) {
   int[] a=new int[3];          //declaration and instantiation
   a[0]=10;                     //initialization
   a[1]=20;
   a[2]=30;

   //printing array
   for(int i=0;i<a.length;i++)        //length is the property of array
        System.out.println(a[i]);
 } }
```

Output:
10
20
30

File: Test.java

```
public class Testarray{
public static void main(String args[]) {
   int[] a= { 100, 200, 300};
//declaration,instantiation and initialization
   for(int i=0;i<a.length;i++)
      System.out.println(a[i]);
   } }
```

Output:
100
200
300

Multi-Dimensional Array

Syntax:

$$dataType[][]\ Var\ =\ new\ dataType[arraySize];$$
$$dataType\ [][]Var\ =\ new\ dataType[arraySize];$$
$$dataType\ Var[][]\ =\ new\ dataType[arraySize];$$

Example

Two Dimensional Array

Example (Array initialization):

int[][] a = new int[2][2];

a → array name
2 → size of the array. It means, 2 rows and 2 columns.

Each individual element in the array is accessed as:

a[0][0] = 100
a[0][1] = 200
a[1][0] = 300
a[1][1] = 400

a	[0]	[1]
[0]	100	200
[1]	300	400

File: Test.java

```
public class Test{
public static void main(String args[])
{
  //declaring and initializing 2D array
  int arr[][]={{100,200},{300, 400}};

  //printing 2D array
   for(int i=0;i<2;i++) {
      for(int j=0;j<2;j++)  {
        System.out.print(arr[i][j]+" ");
          }
  System.out.println();
}
}}
```

Output:
100 200
300 400

Jagged Arrays in Java

Jagged array is array of arrays such that member arrays can be of different sizes, i.e., we can create a 2-D arrays but with variable number of columns in each row. These types of arrays are also known as Jagged arrays.

Program to demonstrate 2-D jagged array in Java

Filename : Test.java

```
public class Test
{
    public static void main(String[] args)
    {
        int arr[][] = new int[4][];

        // Making the above array Jagged

        arr[0] = new int[1];

        arr[1] = new int[2];

        arr[2] = new int[3];
        arr[3] = new int[4];
        arr[4] = new int[5];

        int count = 0;
        for (int i=0; i<arr.length; i++)
            for(int j=0; j<arr[i].length; j++)
                arr[i][j] = count++;

        System.out.println("Contents of 2D Jagged Array");
        for (int i=0; i<arr.length; i++)
        {
            for (int j=0; j<arr[i].length; j++)
                System.out.print(arr[i][j] + " ");
            System.out.println();
        }
    }
}
```

Declaring 2-D array with 5 rows

First row has 1 columns

Second row has 2 columns

Similarly 3rd, 4th and 5th rows have 3,4 and 5 columns respectively

Initializing the array

Displaying the values of 2D Jagged array

Output:

```
Contents of 2D Jagged Array
0
1  2
3  4  5
6  7  8  9
10 11 12 13 14
```

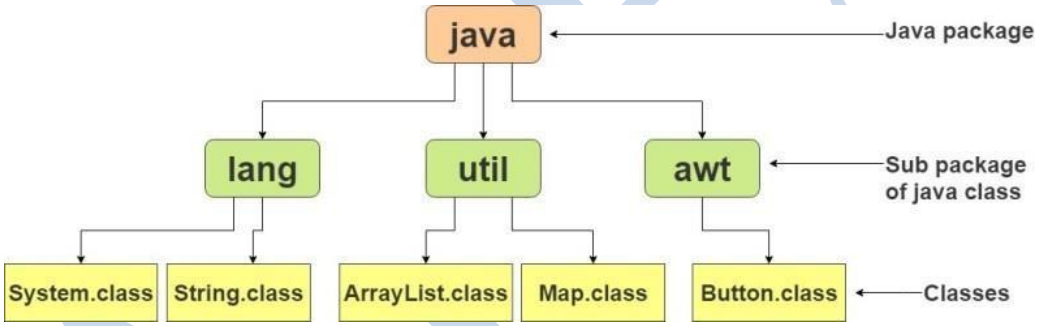
Advantages of using arrays

- Code Optimization
  - Beneficial for retrieving or sorting the data easily.
- Random Access
  - We can randomly get any data located at any index position.

Disadvantages of using arrays

- Size limit
  - Arrays are limited in size (the size is declared only once and it doesn't grow its size during runtime). To solve this problem, collections are used in Java.

Packages in Java



Package in java is a pack (group) of classes, interfaces and other packages. Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, easier, etc.

Example from the Real World

Think of animal kingdom. We basically divide all the animals in the world in to two categories, Vertebrates and non-vertebrates. Each of these two again is divided to many categories. Vertebrates divide to 5 major sub categories: Amphibians, reptiles, birds, fish and mammals. Each of these types will again divide in to their own sub categories.

Just imagine, without dividing the animals like this, we would never be able to remember the features of each and every individual animal. Similar thing is happening in Java packages. It's a way to organize the classes we write and prevent any collision between two Classes.

Syntax:

$package\ nameOfPackage;$

Advantage of package

- Package is used to categorize the classes and interfaces so that they can be easily maintained
- packages reduce memory space and execution time
- Redundancy (repetition) of code is minimized
- Package provides access protection.
- Package removes naming collision.

Types of Java Package

Built-in Packages

These packages consist of a large number of classes which are a part of Java API. Some of the commonly used built-in packages are:

- ***java.lang:*** Contains language support classes (e.g classed which defines primitive data types, math operations). This package is automatically imported.
- ***java.io:*** Contains classed for supporting input/output operations.
- ***java.util:*** Contains utility classes which implement data structures like Linked List, Dictionary etc.
- ***java.applet:*** Contains classes for creating Applets.
- ***java.awt:*** Contain classes for implementing graphical user interfaces (like button, menus etc).
- ***java.net***: Contain classes for supporting networking operations.

User-defined packages

These are the packages that are defined by the user. First we create a directory mypack (name should be same as the name of the package). Then create a class inside the directory.

Creating a package

Creating a package in java is quite easy. Simply include a package command followed by name of the package as the first statement in java source file.

```
// Name of the package must be same as the directory
// under which this file is saved
package mypack;
public class Test
{
   statement;
}
```

Test.java is saved inside the mypack directory since it is a part of the package.

Example 1

File: Test.java

```
package Testpackage;
public class Test{
   public static void main(String args[]) {
      System.out.println("Welcome to java package");
   }
}
```

Output:
Welcome to java package

Steps to Create User Defined Package:

Step 1: Create a folder named **myPackage** and place your java file **MyClass.java**

```
// Name of the package must be same as the directory
// under which this file is saved
package myPackage;

public class MyClass
{
   public void getNames(String s)
   {
      System.out.println(s);
   }
}
```

Step 2: Compile the Java File as follows

D:\ Packages\ javac myPackage \MyClass.java

Make sure that your myPackage folder contains MyClass.class

Step 3: import your myclass from my package

Create a java file named **PrintName** to invoke the package

```
/* import 'MyClass' class from 'names' myPackage */
import myPackage.MyClass;

public class PrintName
{
  public static void main(String args[])
  {
    // Initializing the String variable with a value
    String name = "Welcome to Java Packages";

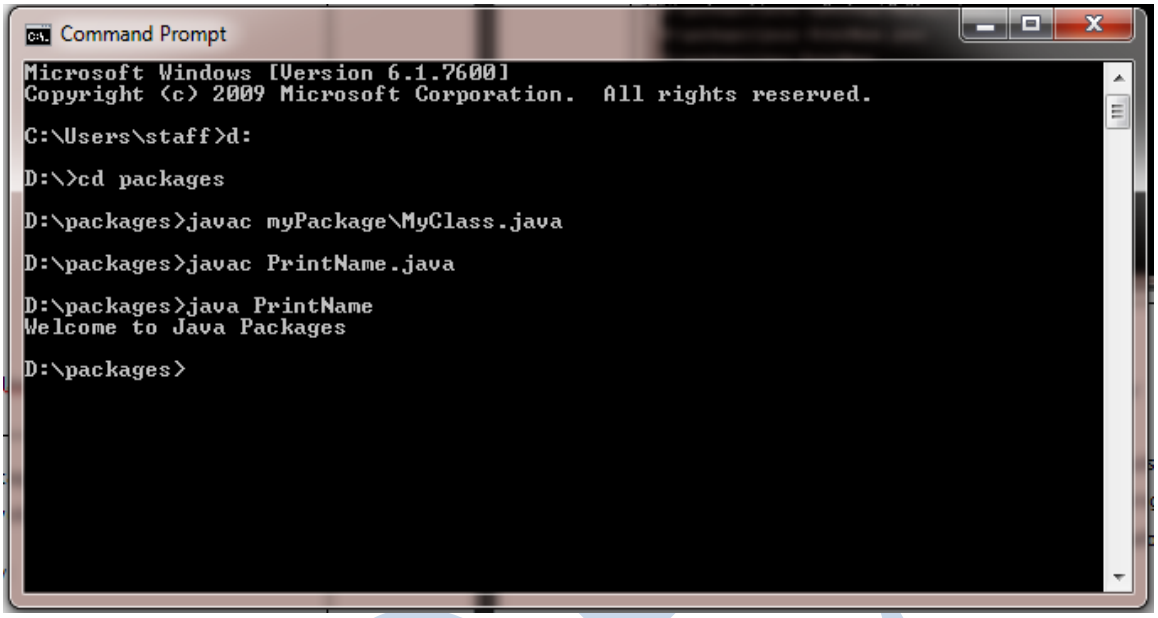
    // Creating an instance of class MyClass in the package.
    MyClass obj = new MyClass();
    obj.getNames(name);
  }
}
```

Step 4: Compile the program

D:\packages\ javac PrintName.java

D:\ packages\ java PrintName

Output:



Note:

- Package is always defined as a separate folder
- Store all the classes in that package folder.
- All classes of the package which is accessed outside are declared public.
- Package statement should be first line in the program.
- All classes of the package must be compiled before use.

Java doc

Javadoc is a documentation tool for the Java programming language. The document it creates from the Java sources is in HTML format and describes the application programming interface.

Javadoc generates output from four different types of source files namely:

- Source code files
- Package comment files
- Overview comment files
- Miscellaneous unprocessed files.

Various tags and parameters are used for different types of documentation are:

- Describing an author ("@author")
- the version of a class or interface ("@version")
- Method parameter ("@param")
- Return value ("@return")
- Link to other documentation ("@see"), etc.

Comment lines used in Javadoc are:

- /** … */ is a documentation comment and in general it's called doc comment
- /*….*/ are Java multi-line comments
- // is Java single-line comment

Example program to implement Javadoc:

```
/**
* Simple program to implement an application that
* displays "Hello World!" to the standard output.
*
* @author Dr. C. DHAYA
* @version 2.0
* @since   2018-05-31
*/

public class HelloWorld {
  public static void main(String[] args) {
    /* Prints Hello, World! on standard output. */
    System.out.println("Hello World!");
  }
}
```

Documentation section

Single line comment

Various tags in Javadoc

Various tags used in Javadoc are:

TAG	USED WHERE	PURPOSE
`@author` *name*	Interface and Classes	Indicates the author of the code.
`@since` *version*	Interfaces and Classes	Indicates the version item was introduced.
`@version` description	Interfaces and Classes	Indicates the version of the source code.
`@deprecated`	Interfaces, Classes and Methods	Indicates a deprecated API item.
`@param` *name* description	Methods	Indicates the method's parameters.
`@return` description	Methods	Indicates the method's return value.
`@throws` *name* descripion	Methods	Indicates exceptions the method throws.
`@see` Classname	All	Indicates additional class to see.
`@see` Classname#member	All	Indicates additional member to see.

Sample Javadoc with methods and parameters

```
/**
 * This is where the description of the class goes.
 * HTML tags can be included here.
 *
 * @author      Author's name
 * @version     Version number
 *
 */
public class ClassName
{
   /**
    * This is where the description of the method goes.
    *
    * @param param1    Description of the first parameter
    * @param param2    Description of the second parameter
    * @return          Description of what is returned
    */
    public ReturnType methodName(Type1 param1, Type2 param2)
   {

   }
}
```

APEC