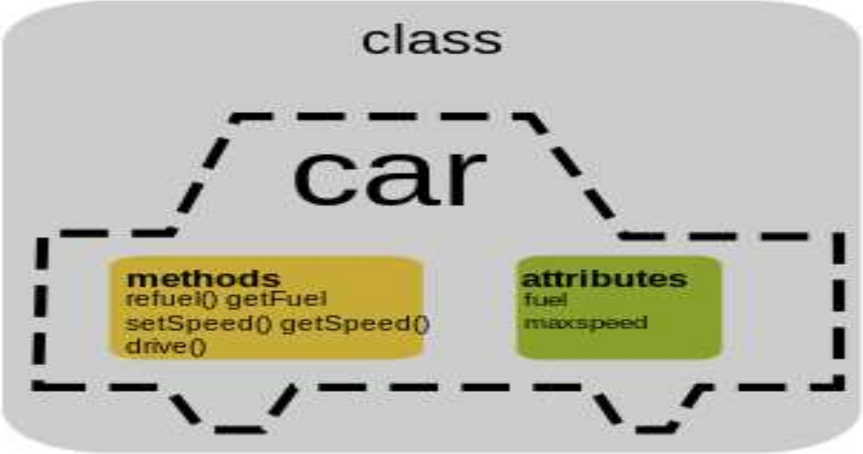
Fundamentals of Object Oriented Programing

Basic Concepts of Object Oriented Programming

- Objects and Classes
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding

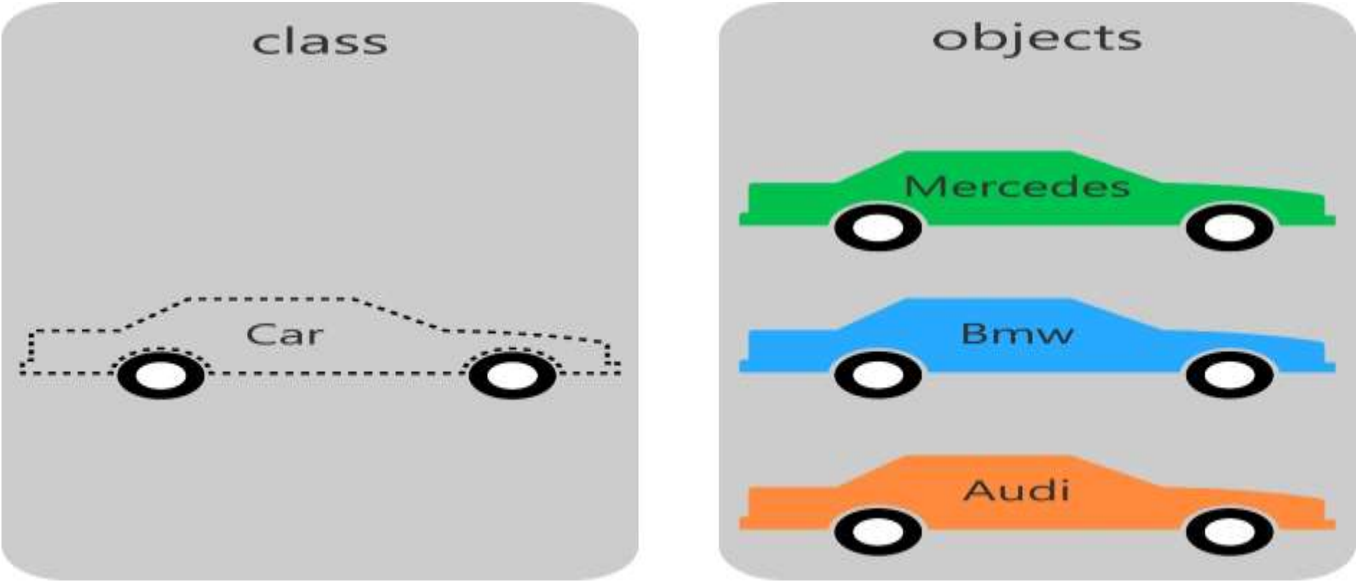
Class

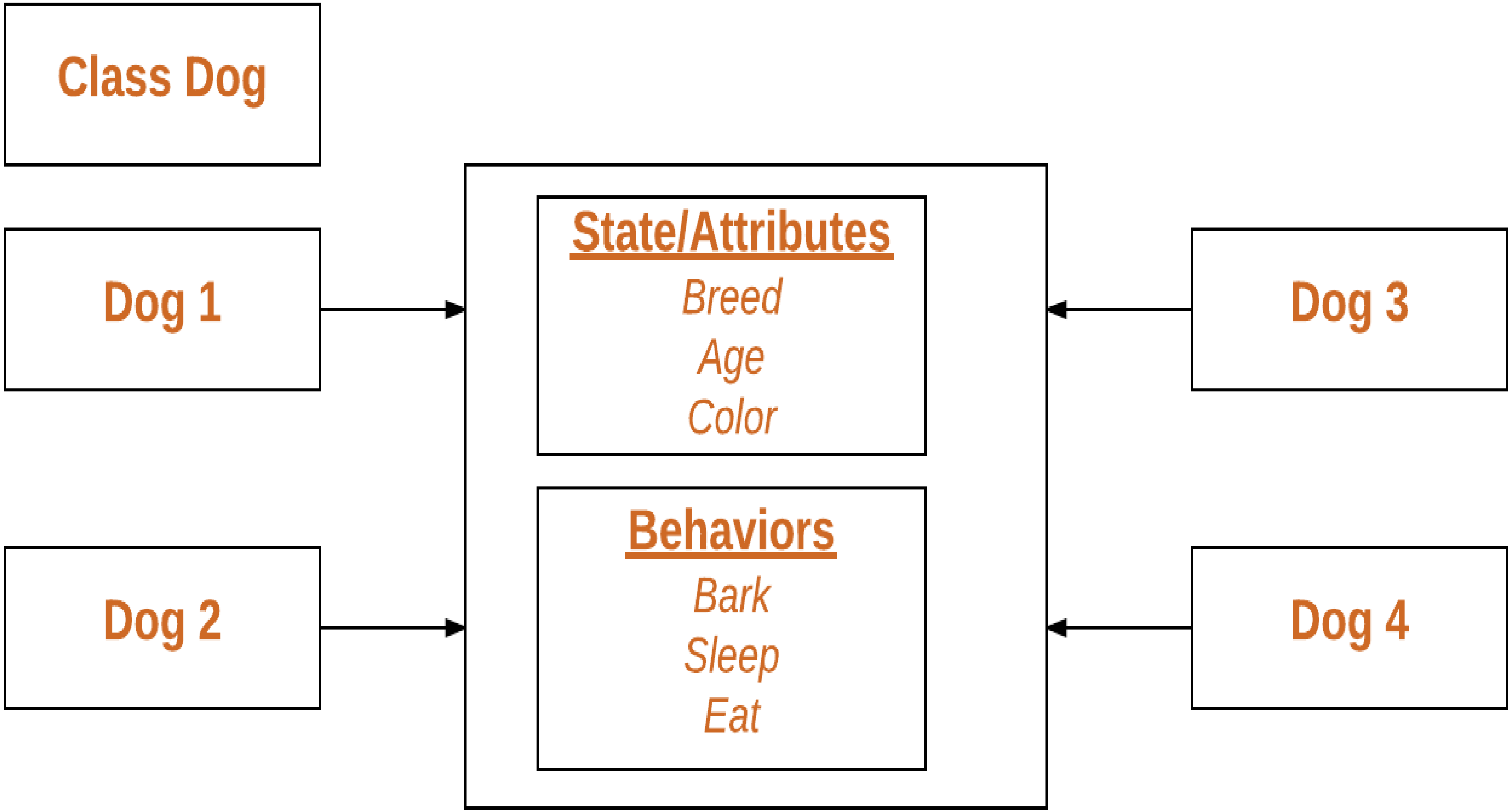
- A class is an entity that determines how an object will behave and what the object will contain.
- In other words, it is a blueprint or a set of instruction to build a specific type of object.
- A class is used to bind data as well as methods together as a single unit.
  - **Attributes** - things that the object stores data in, generally variables.
  - **Methods** - Functions and Procedures attached to an Object and allowing the object to perform actions



Object

- Objects are the basic units of object-oriented programming.
- An **object** is a component of a program that knows how to perform certain actions and how to interact with other elements of the program.
- An object is the instance of the class, which helps programmers to use variables and methods from inside the class.





Class Dog

Dog 1

Dog 2

State/Attributes

Breed

Age

Color

Behaviors

Bark

Sleep

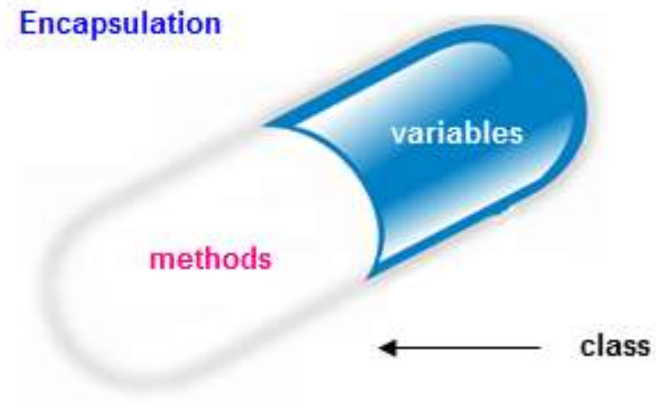
Eat

Dog 3

Dog 4

Data Abstraction and Encapsulation

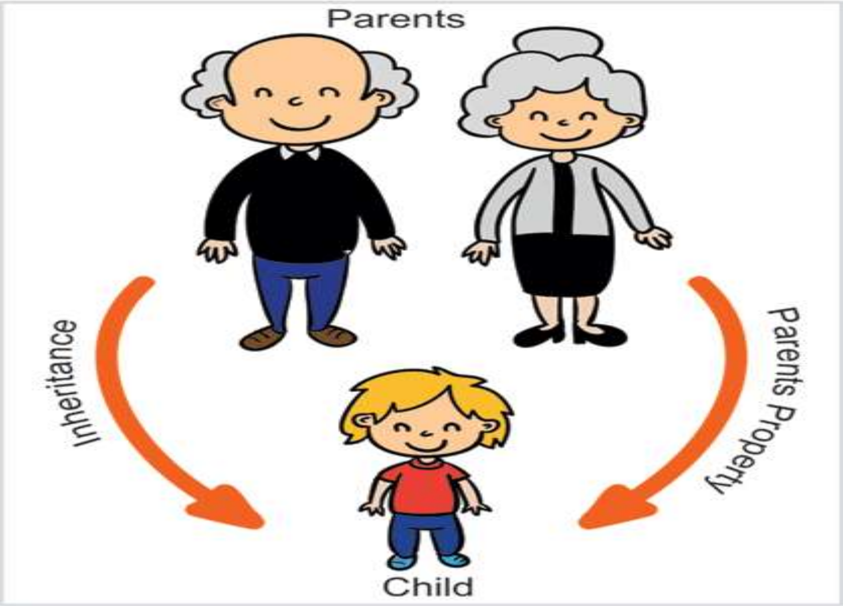
- ***Encapsulation*** : Wrapping up data and member functions (methods) into a single unit i.e. class



- ***Data Abstraction:*** Abstraction is the process of hiding out the working style of an object and showing only the required information of the object in understandable manner.

Inheritance

- Creating a new class from an existing class is called Inheritance.
- Advantage of inheritance is reusability of the code.



Polymorphism

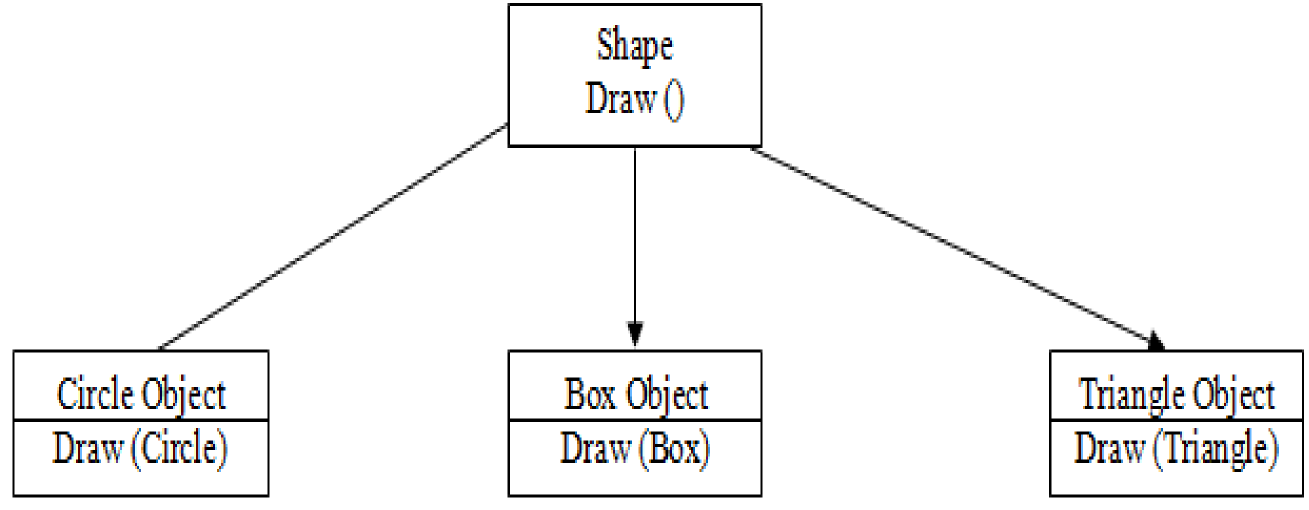
- Polymorphism means having more than one form.
- **Polymorphism** is a concept by which we can perform a *single action by different ways*.
- Polymorphism is achieved with the help of overloading and overriding.

If you ask different animal to “speak”, they responds in their own way.

Same Function Different Behavior

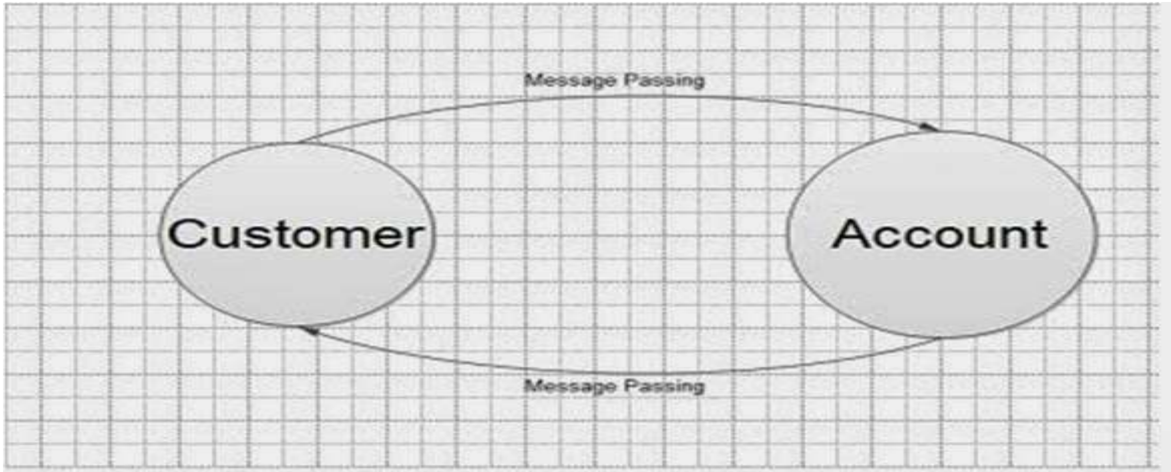
Dynamic Binding

- Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at runtime.
- It is associated with the polymorphism and inheritance.



Message Communication

- An object oriented program consists of a set of objects that communicate with each other.
- Object communicate with one another by sending and receiving information much the same way as people pass messages to one another.
- A message for an object is a request for execution of a procedure(method).



Customer and Account are two objects and they are communication with each other

Benefits of OOP

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- We can built programs from standard working modules that communicate with one another rather than, having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmers to built secure program that can’t be invaded by code in other parts of the program.
- It is possible to have multiple objects to coexist without any interference.

Benefits of OOP

- It is easy to partition the work in a project based on objects.
- The data-centered design approach enables us to capture more details of the model in an implementable form.
- Object-oriented systems can be easily upgraded from small to large system
- Message passing technique for communication between objects make the interface descriptions with external system much simpler.
- Software complexity can be easily managed.

Applications of OOP

- User interface design such as windows, menu ,…
- Real Time Systems
- Simulation and Modelling
- Object oriented databases
- AI and Expert System
- Neural Networks and parallel programming
- Decision support and office automation system

	Procedure Oriented Programming	Object Oriented Programming
Divided Into	In POP, program is divided into small parts calledfunctions.	In OOP, program is divided into parts called objects.
Importance	In POP, Importance is not given to data but to functions as well as sequence of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a real world.
Approach	POP follows Top Down approach.	OOP follows Bottom Up approach.
Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
Expansion	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function, it can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper way for hiding data so it is less secure.	OOP provides Data Hiding so provides more security.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

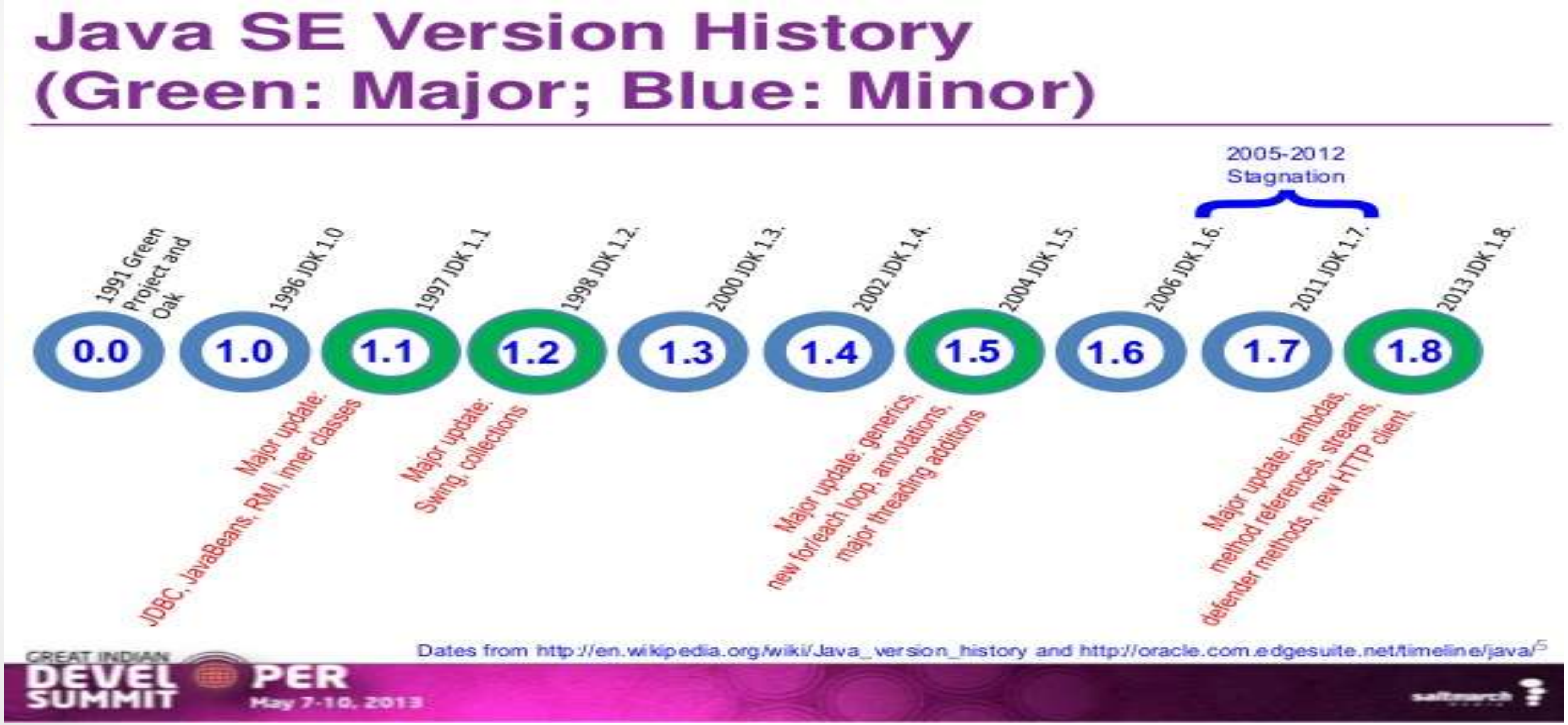
Overview of Java Language

The Java Programming Language

There are four platforms of the Java programming language:

- Java Platform, Standard Edition (Java SE)
- Java Platform, Enterprise Edition (Java EE)
- Java Platform, Micro Edition (Java ME)
- JavaFX

Java History



Java SE Version History (Green: Major; Blue: Minor)

2005-2012
Stagnation

1991 Green Project and Oak — 0.0

1996 JDK 1.0 — 1.0

1997 JDK 1.1 — 1.1
Major update:
JDBC, JavaBeans, RMI, inner classes

1998 JDK 1.2. — 1.2
Major update:
Swing, collections

2000 JDK 1.3. — 1.3

2002 JDK 1.4. — 1.4

2004 JDK 1.5. — 1.5
Major update: generics,
new for/each loop, annotations,
major threading additions

2006 JDK 1.6. — 1.6

2011 JDK 1.7. — 1.7

2013 JDK 1.8. — 1.8
Major update: lambdas,
method references, streams,
defender methods, new HTTP client

Dates from http://en.wikipedia.org/wiki/Java_version_history and http://oracle.com.edgesuite.net/timeline/java/

5

GREAT INDIAN DEVELOPER SUMMIT
May 7-10, 2013

saltmarch

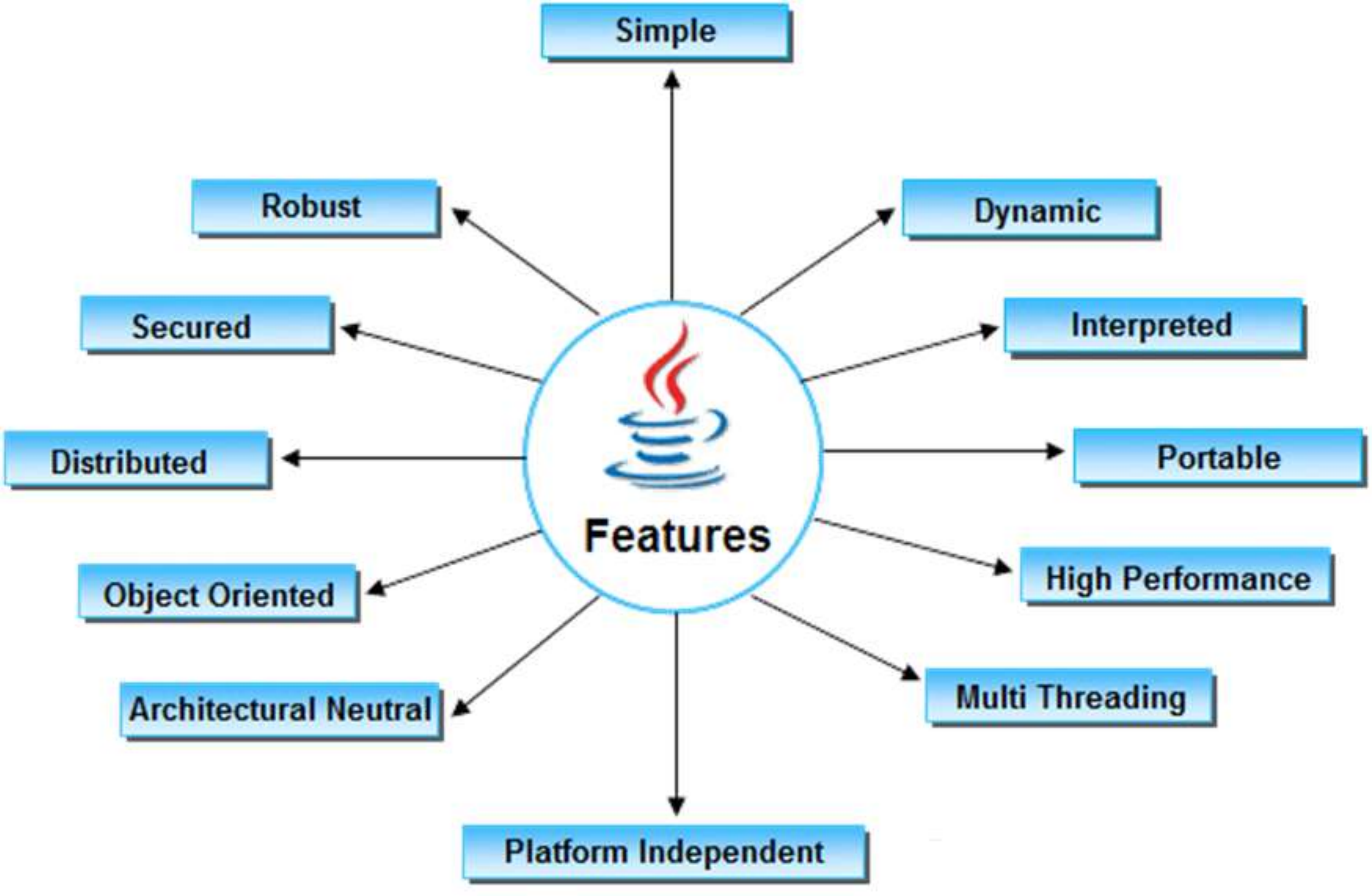
- Java SE 9 (2017)
- Java SE 10 (2018)

  Java SE 10 was released to remove primitive data types and move towards 64-bit addressable arrays to support large data set

- java SE 11 (2019)

  bug fixing

Java Features

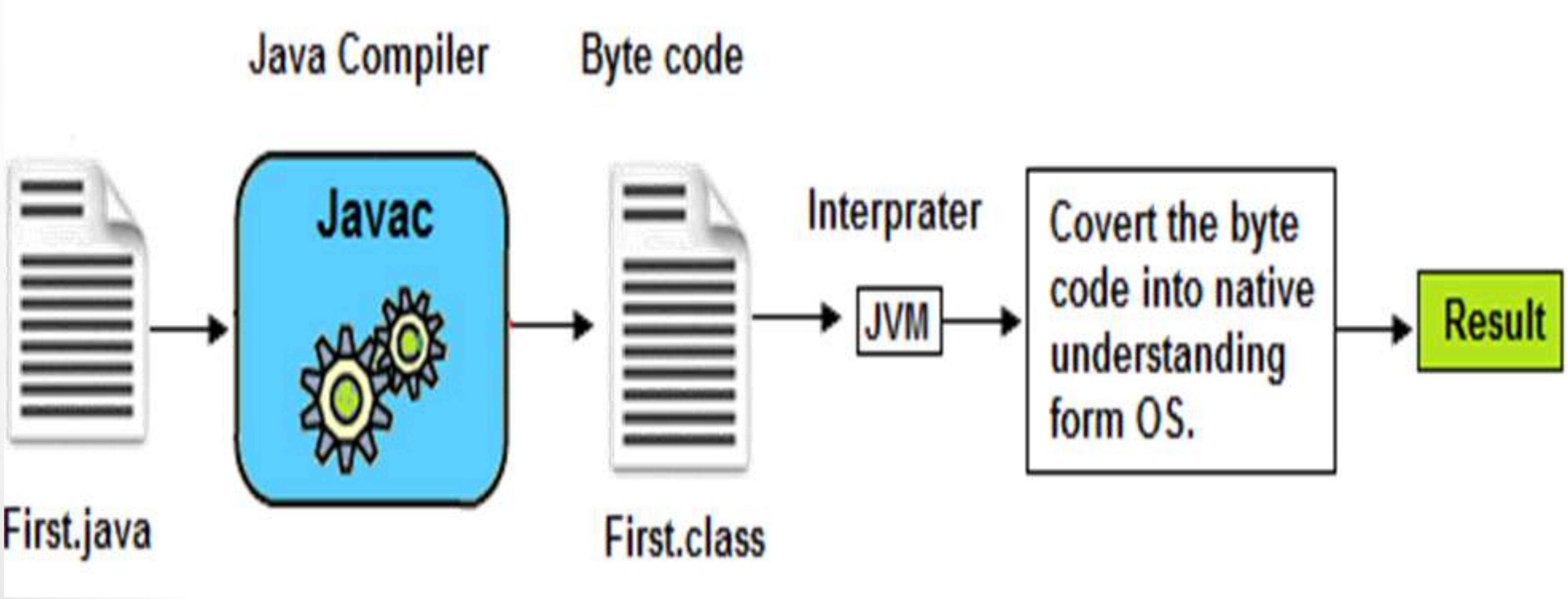


Object Oriented

- Java is a true object oriented language.
- All program code and data reside within objects and classes.
- Java comes with an extensive set of classes, arranged in packages, that we can use in our programs by inheritance.

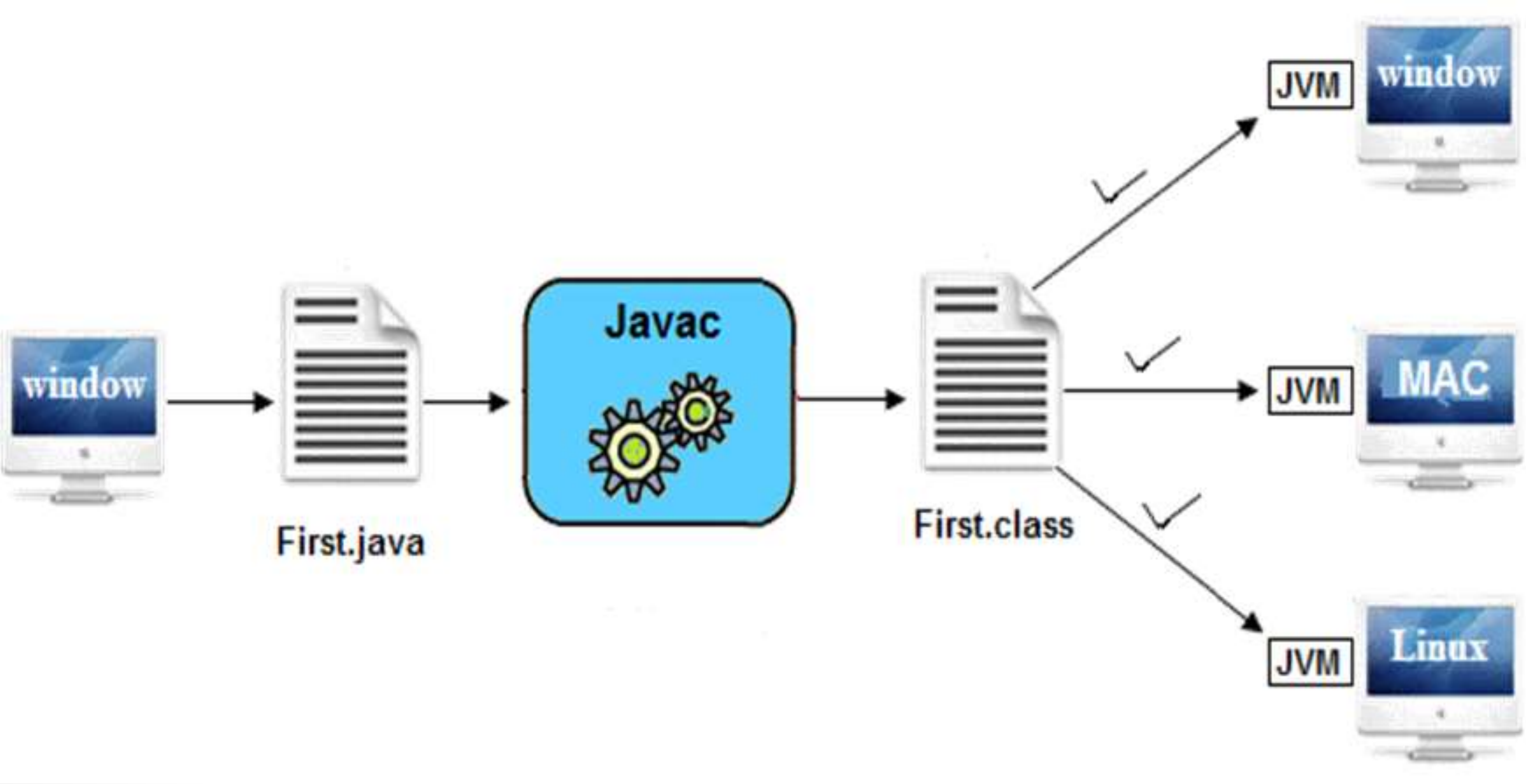
Compiled and Interpreted

1. Java compiler translates source code into bytecode instructions.
2. Bytecodes are not machine instructions therefore java interpreter generates machine code that can be directly executed by the machine i.e. running the java program.



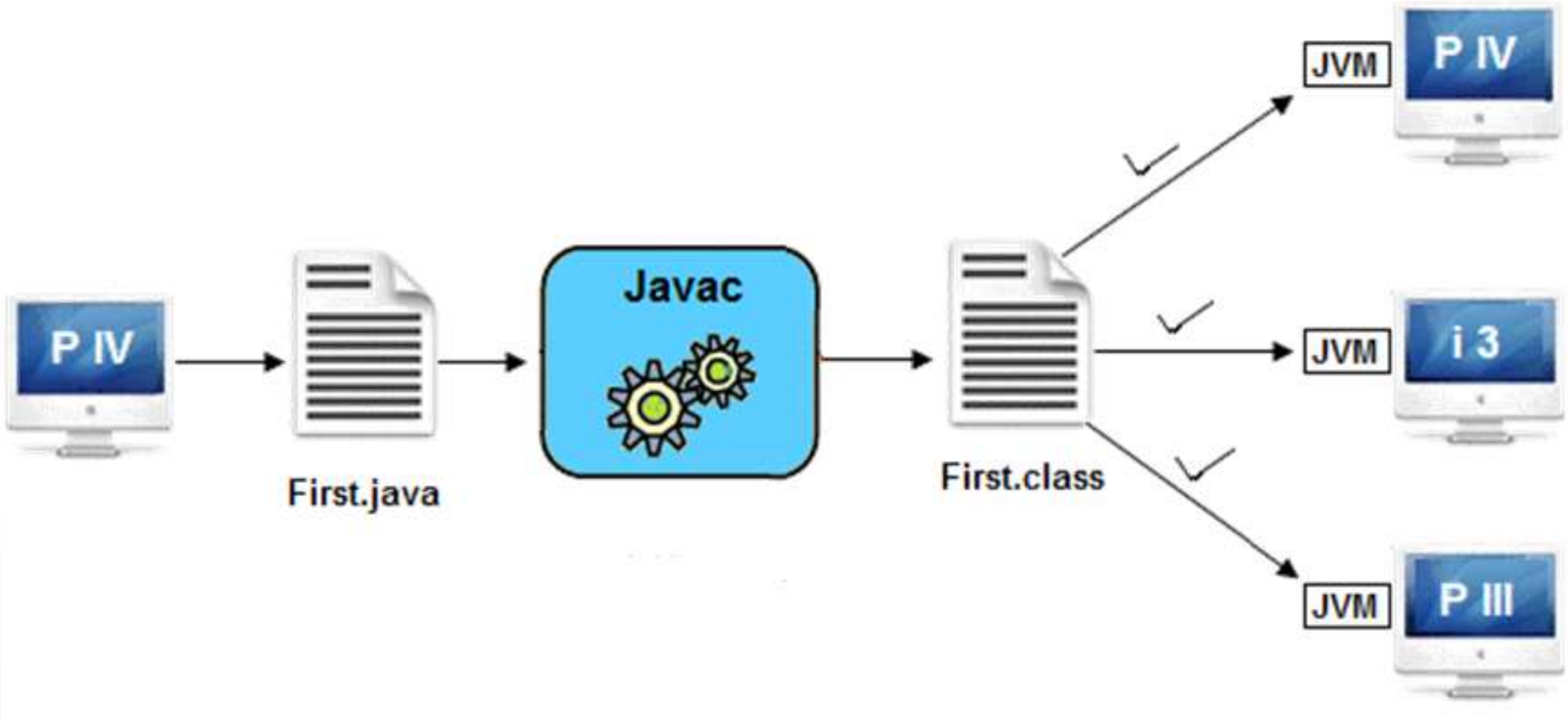
Platform Independent

- A program or technology is said to be platform independent if and only if which can run on all available operating systems with respect to its development and compilation. (Platform represents O.S).



Architectural Neutral

- Architecture represents processor.
- A Language or Technology is said to be Architectural neutral which can run on any available processors in the real world without considering there architecture and vendor (providers) irrespective to its development and compilation.



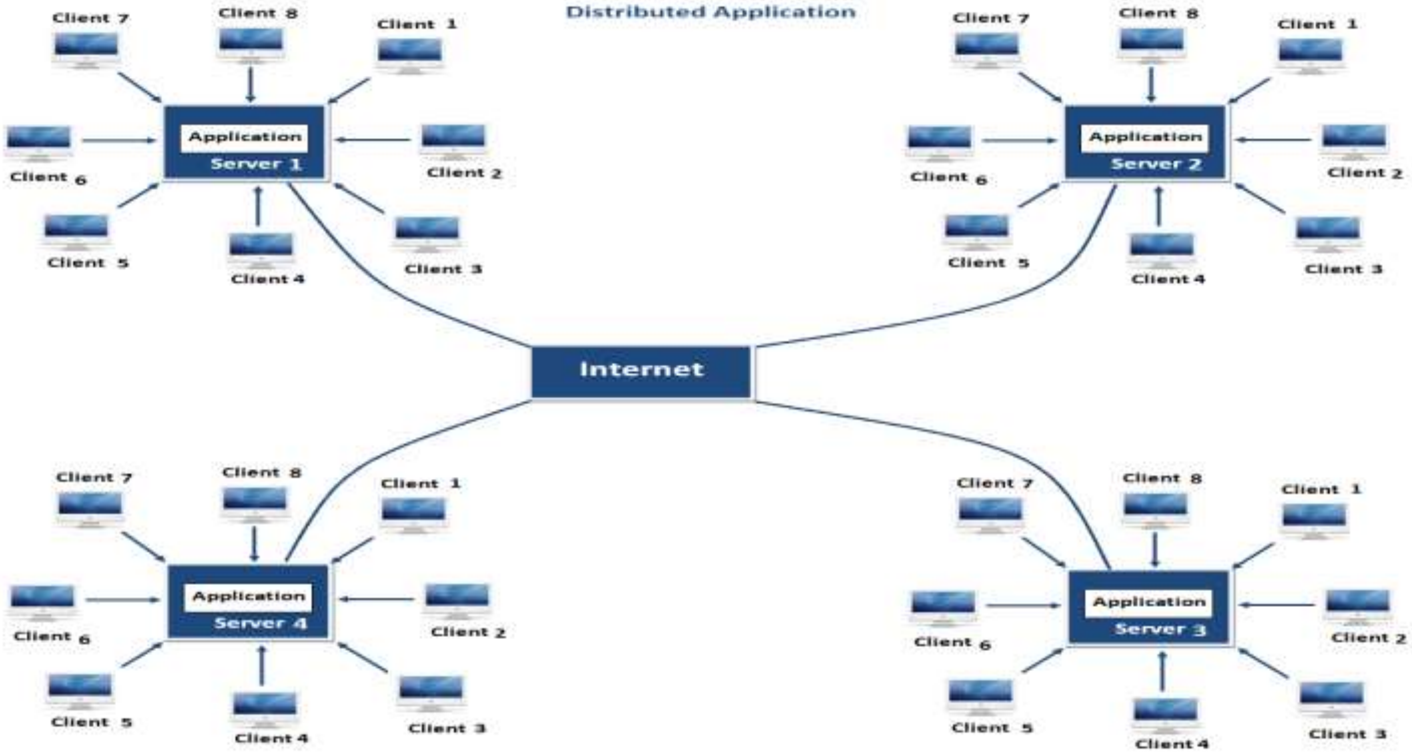
Portable and Multithreaded

- **Portable**
  - If any language supports platform independent and architectural neutral feature known as portable.
  - The languages like C, CPP, Pascal are treated as non-portable language.
  - It is a portable language.

- **Multithreaded**
  - A flow of control is known as thread.
  - When any Language execute multiple thread at a time that language is known as multithreaded Language.
  - It is multithreaded Language.

Distributed

- Java is designed as distributed language for creating applications on network.
- It has ability to share both data and programs.
- Java applications can open and access remote objects on Internet as easily as they can do in a local system.
- This enables multiple programmers at multiple remote location to collaborate and work together on a single project.



Robust and Secure

- **Robust**
  - Simply means of Robust is strong.
  - It is robust or strong Programming Language because of its capability to handle Run-time Error, automatic garbage collection, lack of pointer concept, Exception Handling.
  - All these points makes It robust Language.
- **Secure**
  - It is more secured language compare to other language.
  - In this language all code is covered into byte code after compilation which is not readable by human.

Dynamic

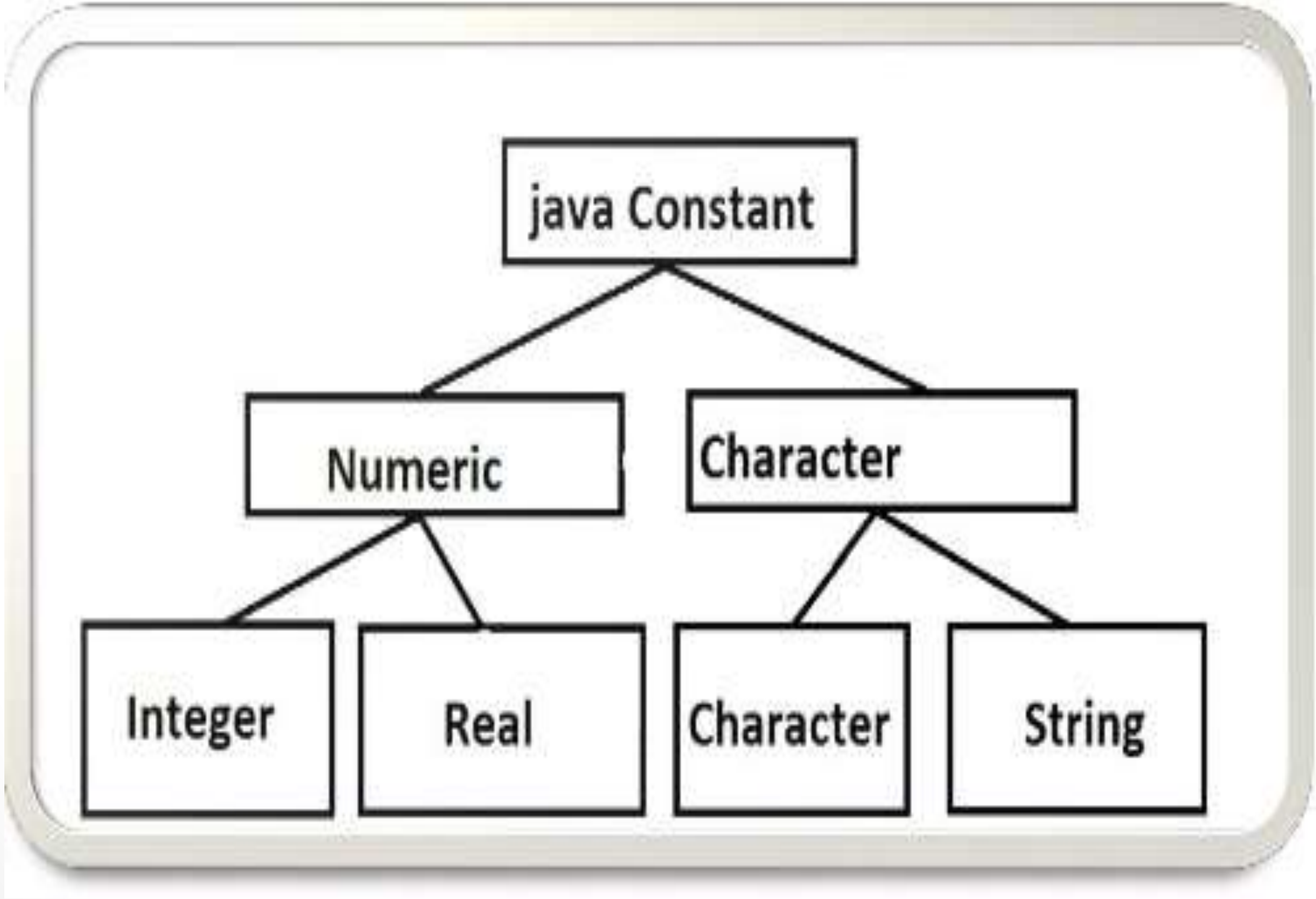
- It support Dynamic memory allocation due to this memory wastage is reduce and improve performance of application.
- The process of allocating the memory space to the input of the program at a run-time is known as dynamic memory allocation.

High performance

- This language **uses Bytecode** which is more faster than ordinary code so Performance of this language is high.
- **Garbage collector**, collect the unused memory space and improve the performance of application.
- It have **no pointers** so that using this language we can develop an application very easily.
- It **support multithreading**, because of this time consuming process can be reduced to execute the program.

Constants, variables and data types

Constants


java Constant
Numeric
Character
Integer
Real
Character
String

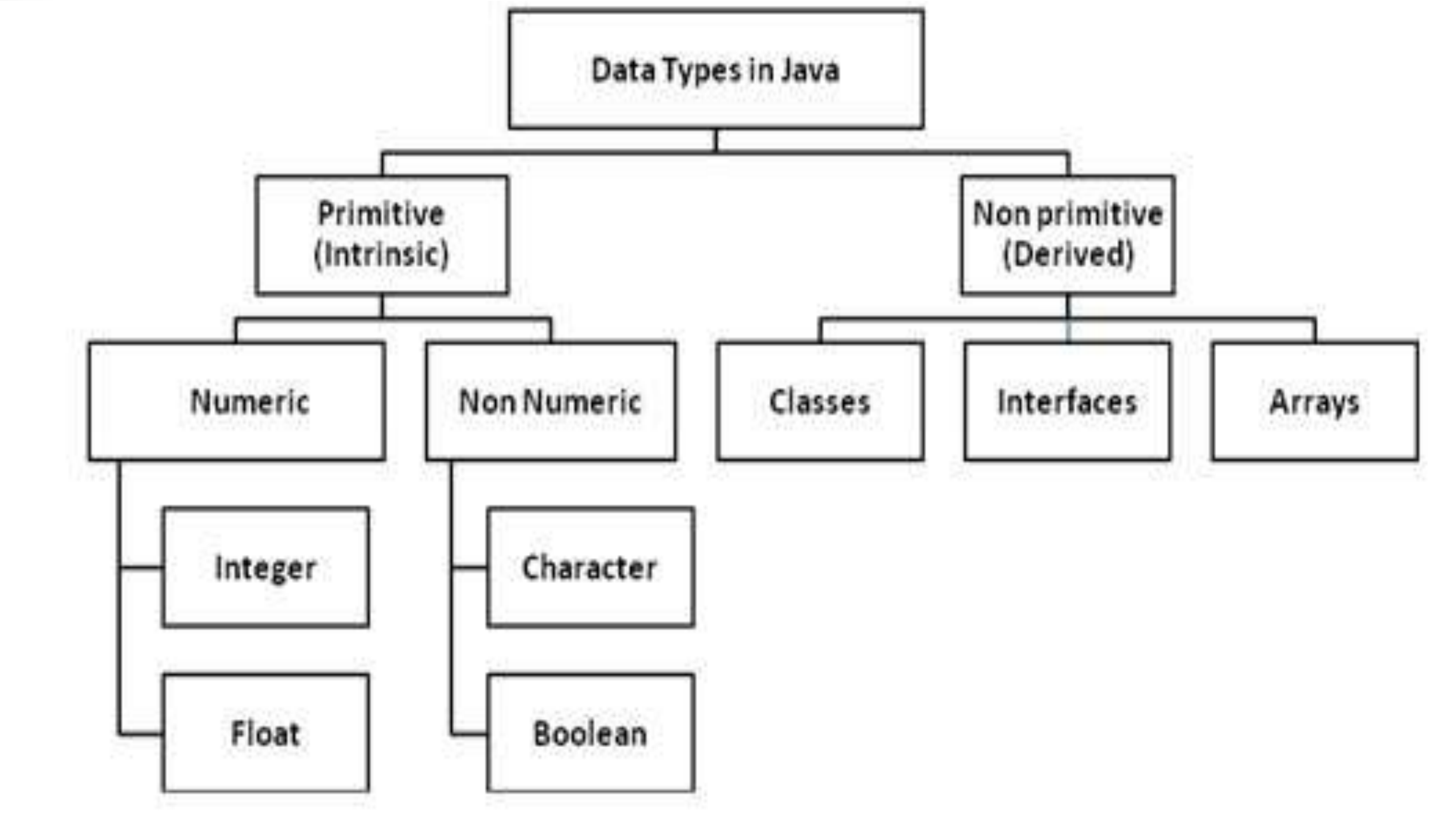
Variables

A variable is an identifier that denotes a storage location used to store data value.

Conditions for using variable names:

- They must not begin with a digit.
- Uppercase and Lowercase are distinct.
- It should not be a keyword.
- White space is not allowed.
- Variable names can be of any length.

Data Types



Data Types in Java

Primitive (Intrinsic)

Non primitive (Derived)

Numeric

Non Numeric

Classes

Interfaces

Arrays

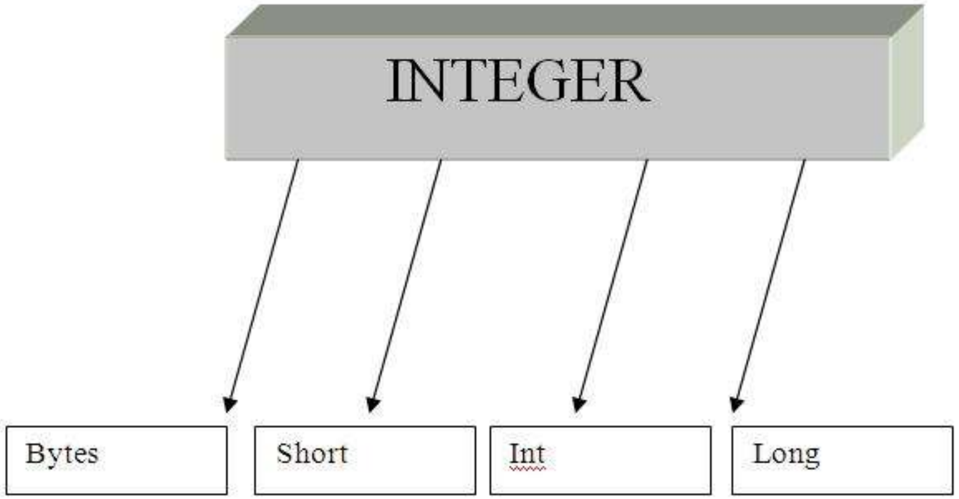
Integer

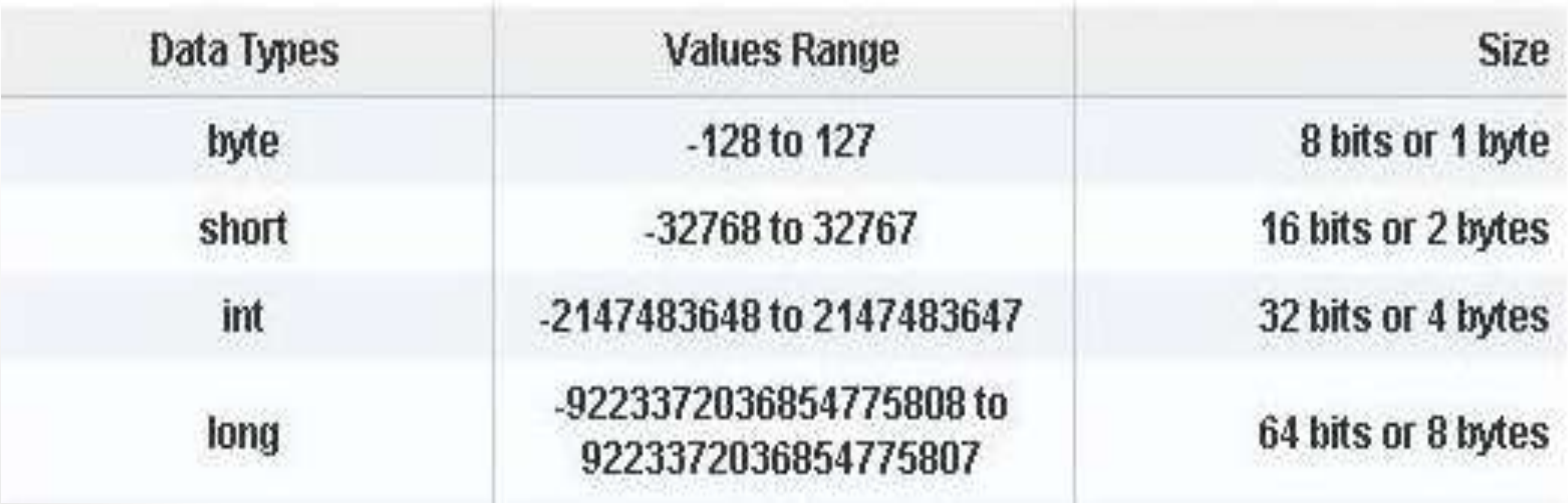
Float

Character

Boolean

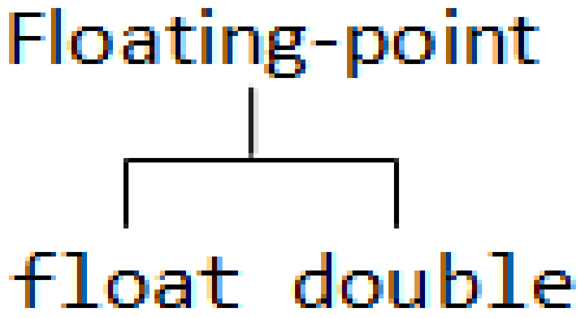
Integer types

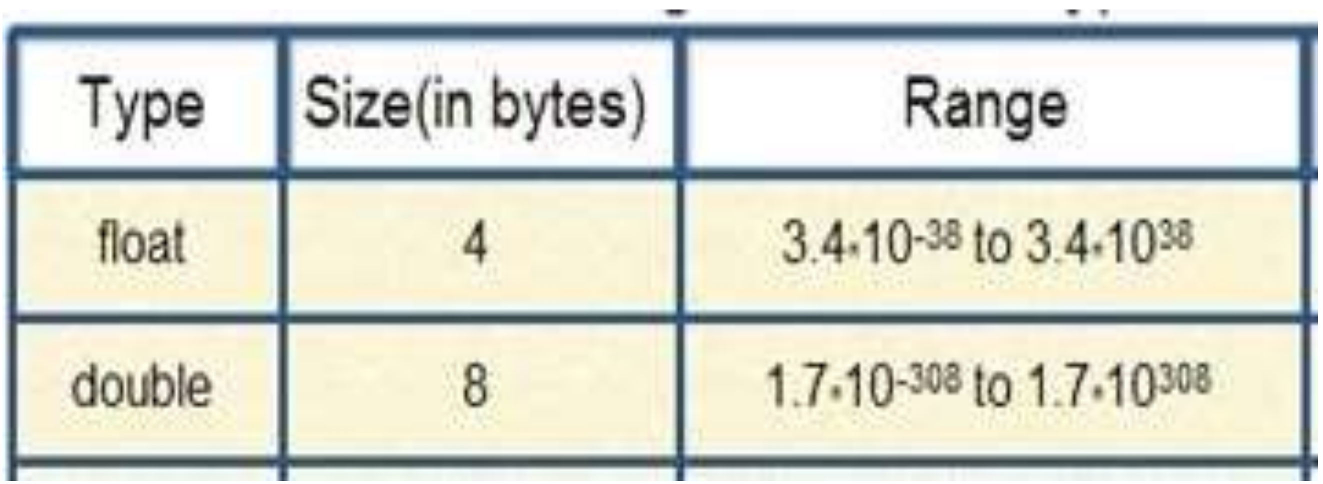




Data Types	Values Range	Size
byte	-128 to 127	8 bits or 1 byte
short	-32768 to 32767	16 bits or 2 bytes
int	-2147483648 to 2147483647	32 bits or 4 bytes
long	-9223372036854775808 to 9223372036854775807	64 bits or 8 bytes

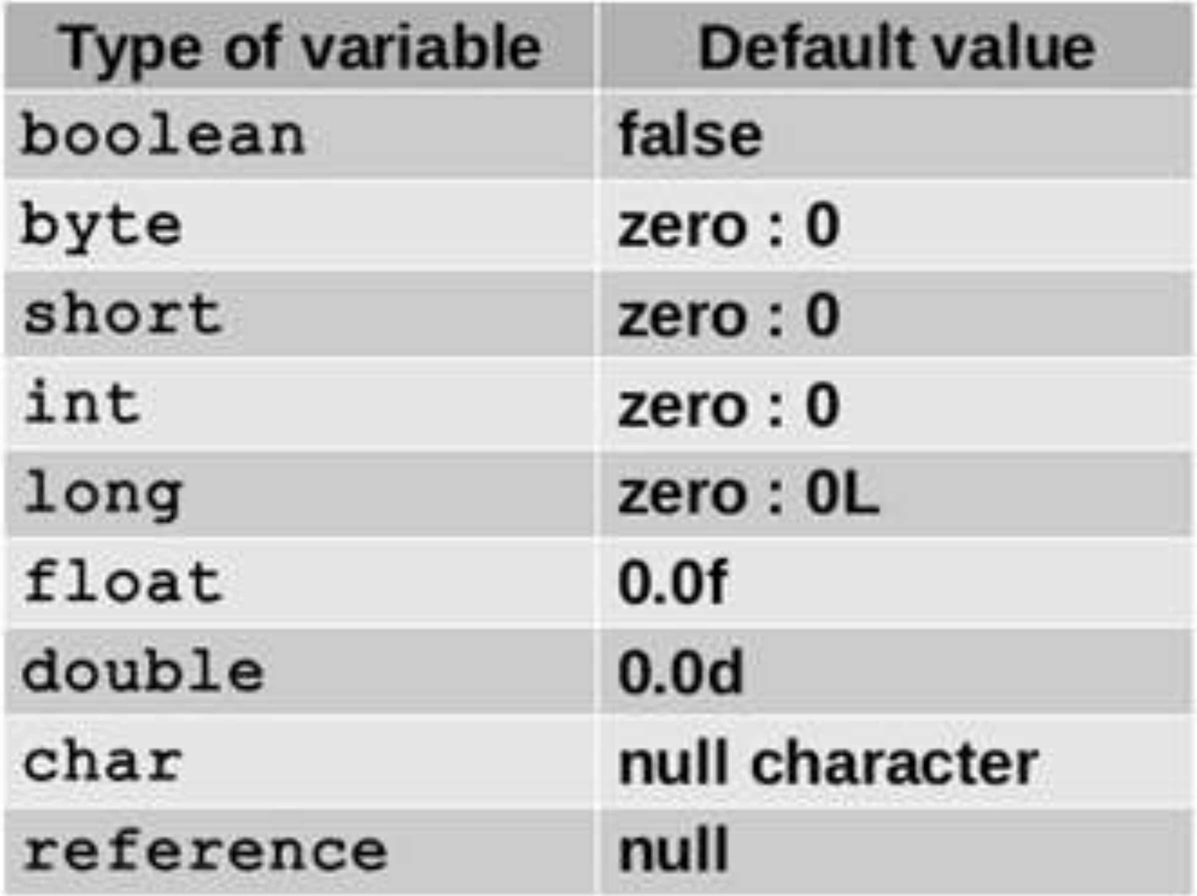
Floating Point type





Type	Size(in bytes)	Range
float	4	$3.4 \cdot 10^{-38}$ to $3.4 \cdot 10^{38}$
double	8	$1.7 \cdot 10^{-308}$ to $1.7 \cdot 10^{308}$

standard default values

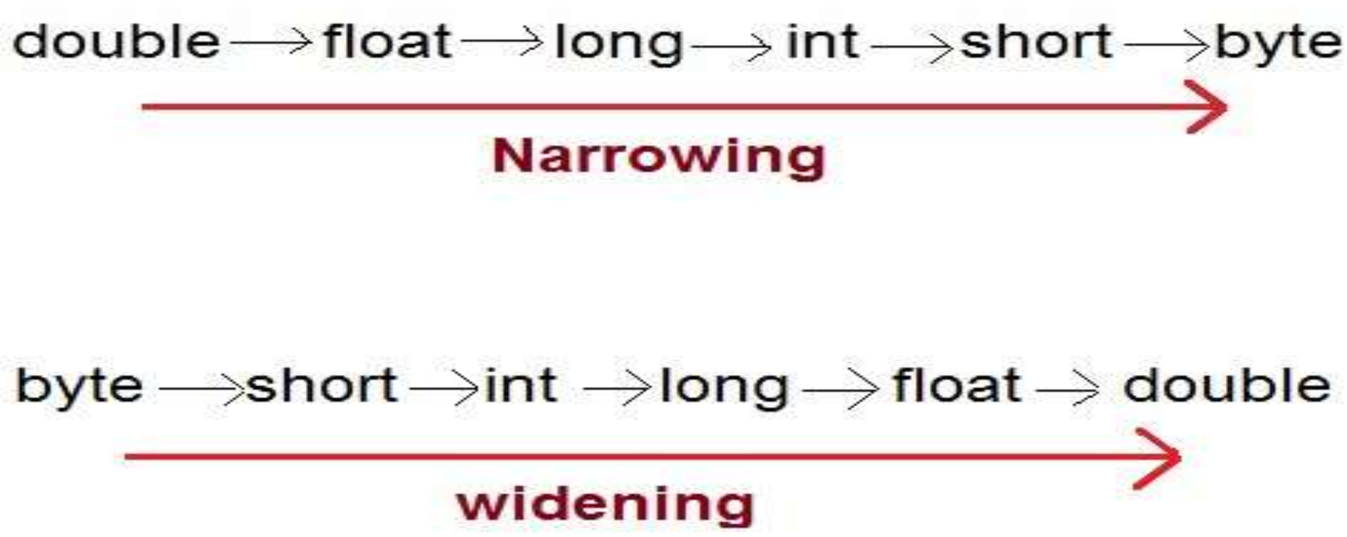


Type of variable	Default value
`boolean`	false
`byte`	zero : 0
`short`	zero : 0
`int`	zero : 0
`long`	zero : 0L
`float`	0.0f
`double`	0.0d
`char`	null character
`reference`	null

Type Casting

- The process of converting one data type to another is called casting.
- Casting is often necessary when a method returns a type different than the one we require.

Syntax : type variable1=(type) variable2;


double → float → long → int → short → byte
Narrowing
byte → short → int → long → float → double
widening

Operators

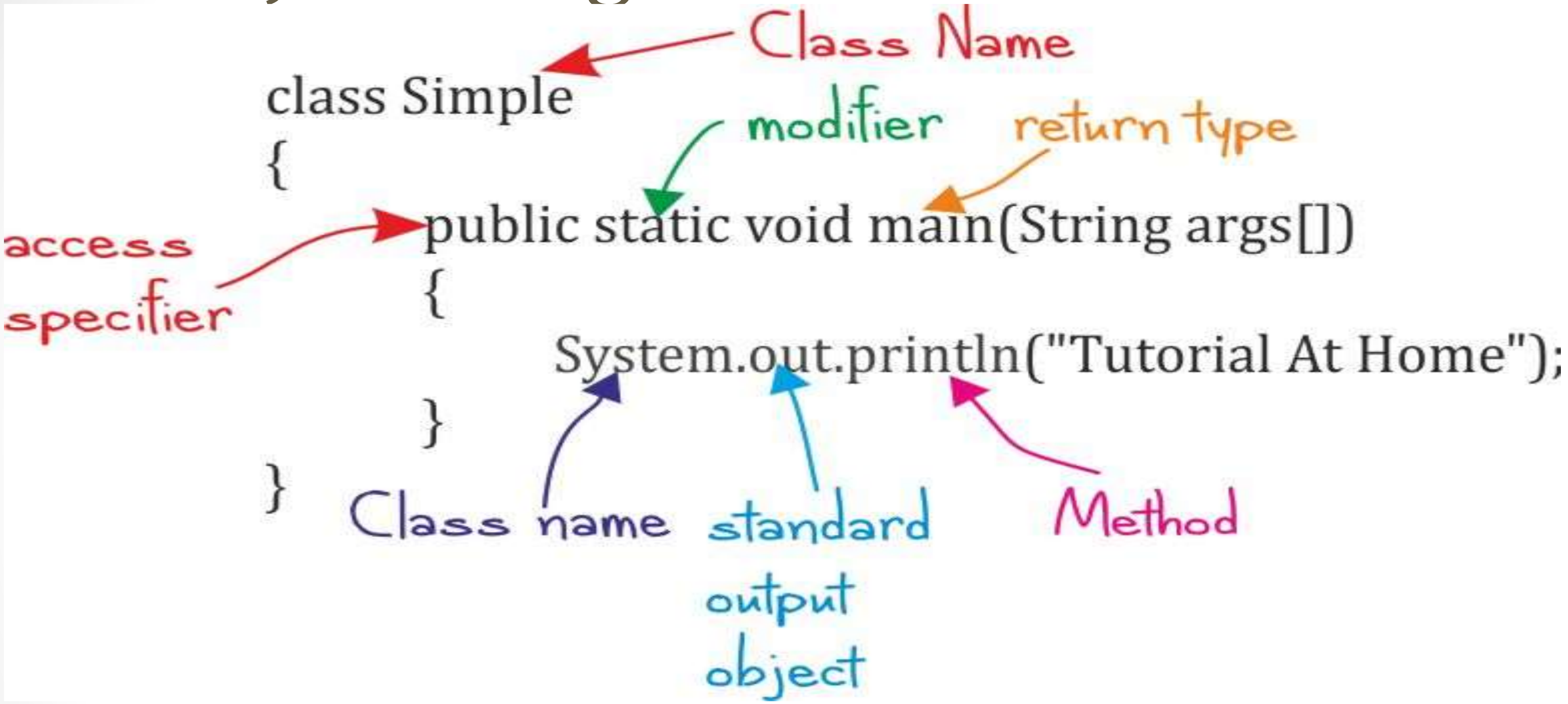
- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Increment / decrement operator
- Conditional operators
- Bitwise operators
- Special operators

Special operators

- **Instanceof** operators
  - The instanceof is an object reference operator and returns true if the object on the left hand side is an object of the class given on the right hand side.
  - This operator allows us to determine whether the object belongs to a particular class or not.
  - E.g. person instanceof student

- **Dot** operator
  - The dot operator is used to access the instance variable and methods of class objects.
  - E.g. person.age

Basic Java Program



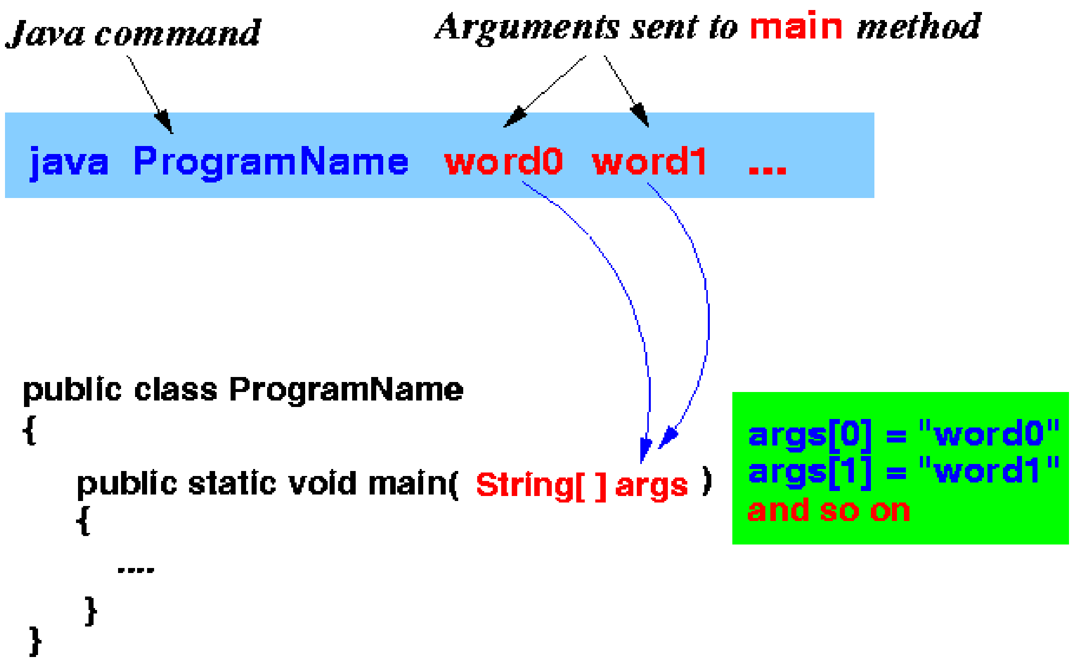
```
class Simple
{
    public static void main(String args[])
    {
        System.out.println("Tutorial At Home");
    }
}
```

Compilation: javac *filename*.java

Execution: java *classname*

Command line Arguments

- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.
- So, it provides a convenient way to check the behaviour of the program for the different values. You can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.



```
java  ProgramName  word0  word1  ...
```

```
public class ProgramName
{
     public static void main( String[ ] args )
     {
          ....
     }
}
```

```
args[0] = "word0"
args[1] = "word1"
and so on
```

example

```
class CommandLine
{
    public static void main(String args[])
   {
     for(int i=0;i<args.length;i++)
     System.out.println(args[i]);
   }
}
```

Execution:

```
Java CommandLine c c++ java
```

Output:

```
C
C++
java
```

Java program for addition of two number using command line argument

```
class CommandLine
{
  public static void main(String args[])
  {
   int a,b,c=0;
   a=Integer.parseInt(args[0]);
   b=Integer.parseInt(args[1]);
   c=a+b;
   System.out.println("c= "+c);
  }
}
```

Using Scanner class for input

```
import java.util.Scanner;
class ScannerTest
{
 public static void main(String args[])
{
   Scanner sc=new Scanner(System.in);
    System.out.println("Enter your rollno");
   int rollno=sc.nextInt();
   System.out.println("Enter your name");
   String name=sc.next();
   System.out.println("Enter your fee");
   double fee=sc.nextDouble();
   System.out.println("Rollno:"+rollno+" name:"+name+"
fee:"+fee);
    }
}
```

- Output

```
Enter your rollno
111
Enter your name
Ratan
Enter your fee
450000
Rollno:111 name:Ratan fee:450000
```

Using BufferedReader for input

```
import java.io.*;
public class BufferedReaderExample
{
   public static void main(String args[])throws Exception
  {
   InputStreamReader r=new InputStreamReader(System.in);
   BufferedReader br=new BufferedReader(r);
   System.out.println("Enter your name");
   String name=br.readLine();
   System.out.println("Enter your Roll number");
   int ro_num=Integer.parseInt(br.readLine());
   System.out.println("name: "+name);
   System.out.println("Roll number: "+ro_num);
}
}
```

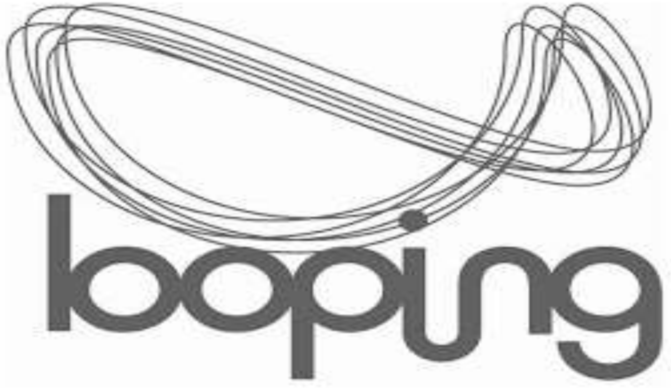
Output:

```
Enter your name
Ramesh
Enter your Roll number
1
name: Ramesh
Roll number: 1
```

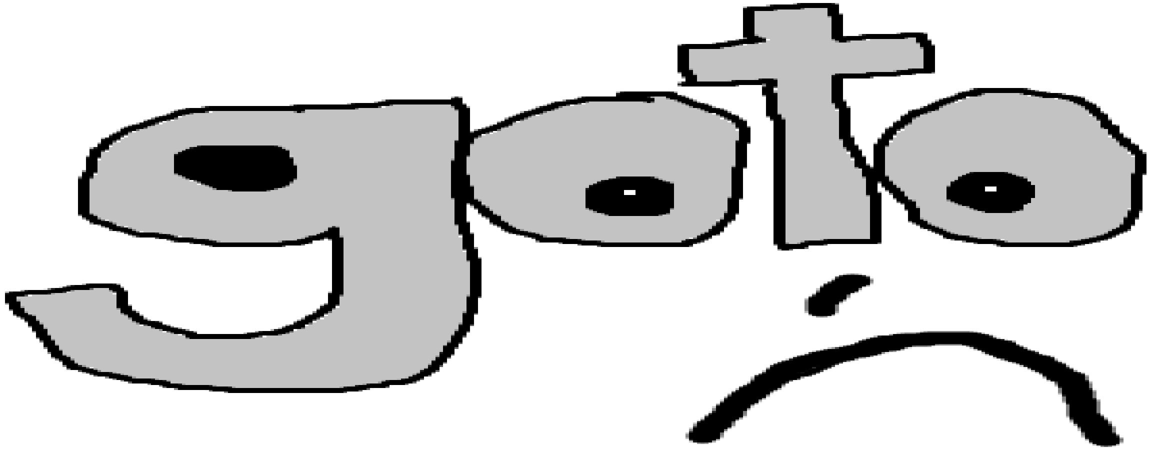
BufferedReader class

- The **Java.io.InputStreamReader** class is a bridge from byte streams to character streams.It reads bytes and decodes them into characters using a specified charset.

- The **Java.io.BufferedReader** class reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.



Decision making , Branching and Looping



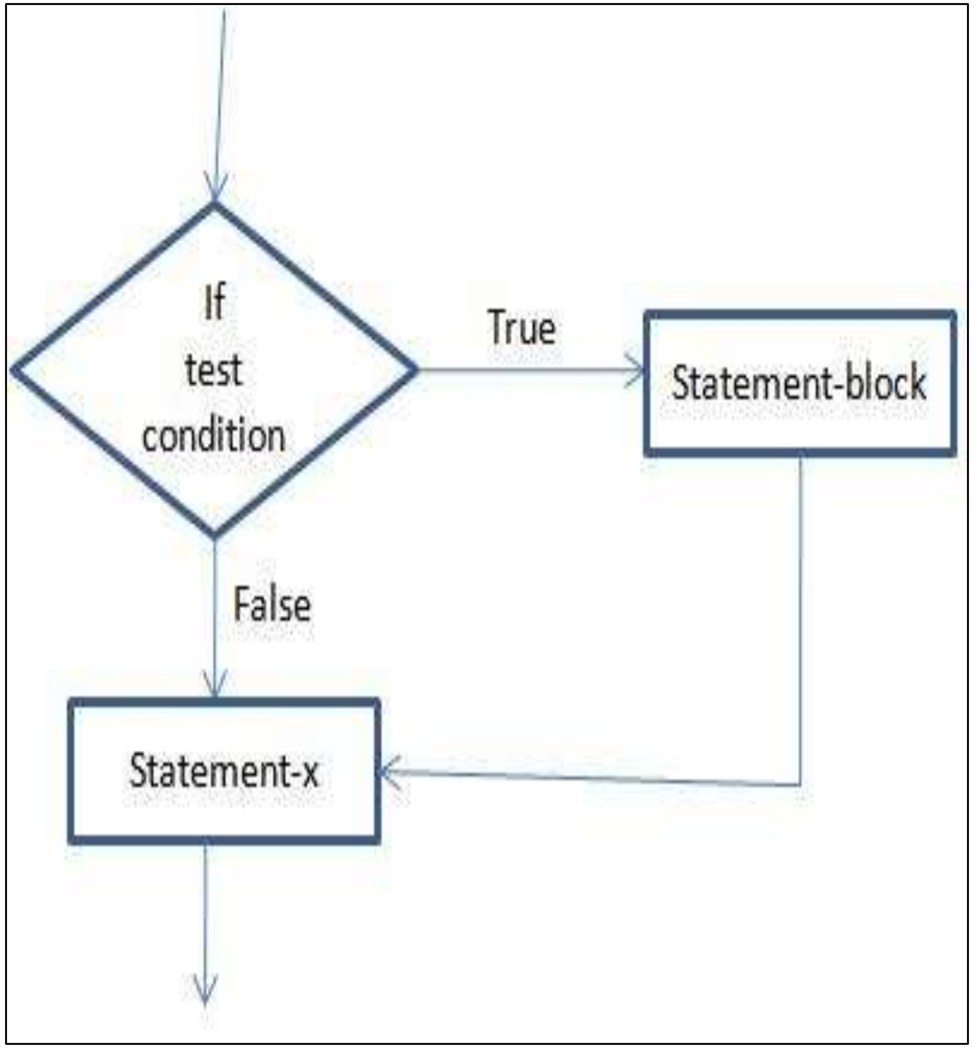
Forms of if….

- if statement
- if-else statement
- nested if-else statement
- else-if ladder
- switch statement

Simple If statement

Syntax:

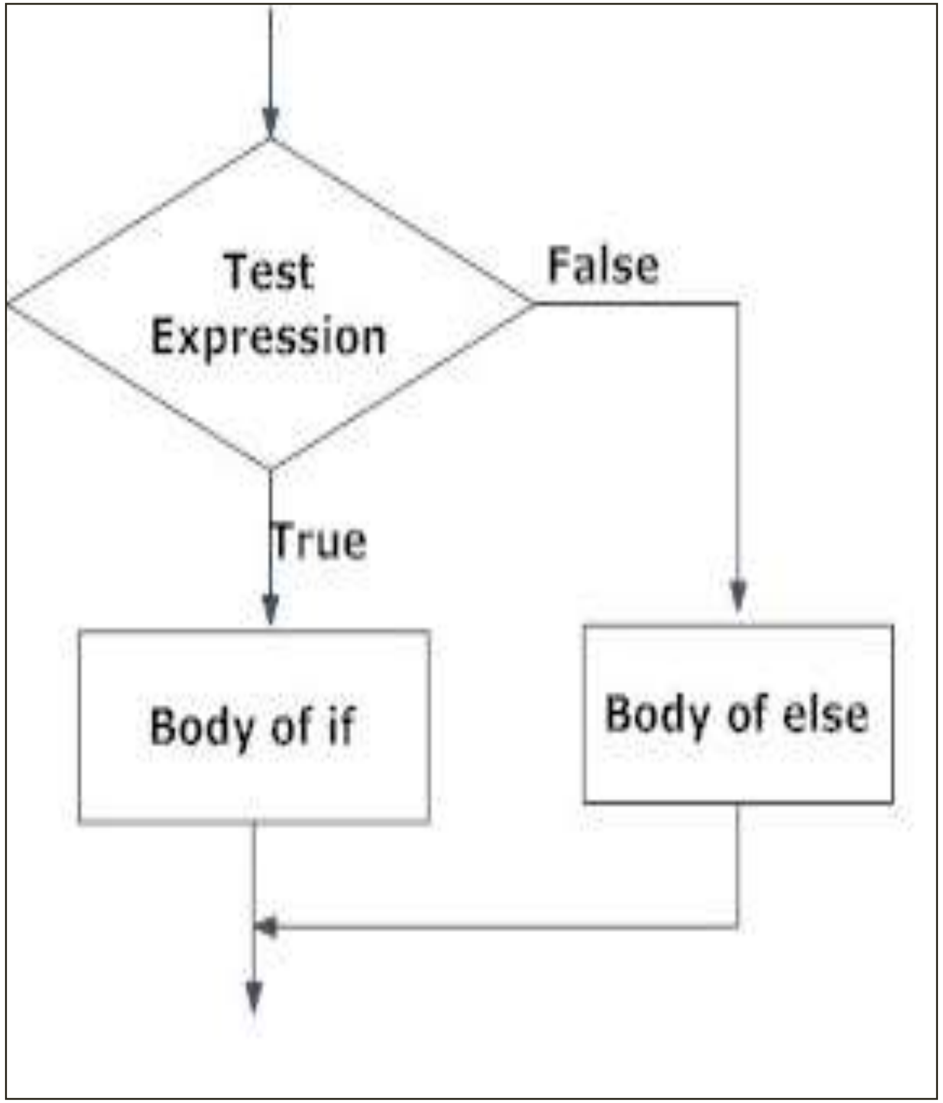
```
if(condition)
{
   statements
}
statement_x;
```



If else statement

Syntax:

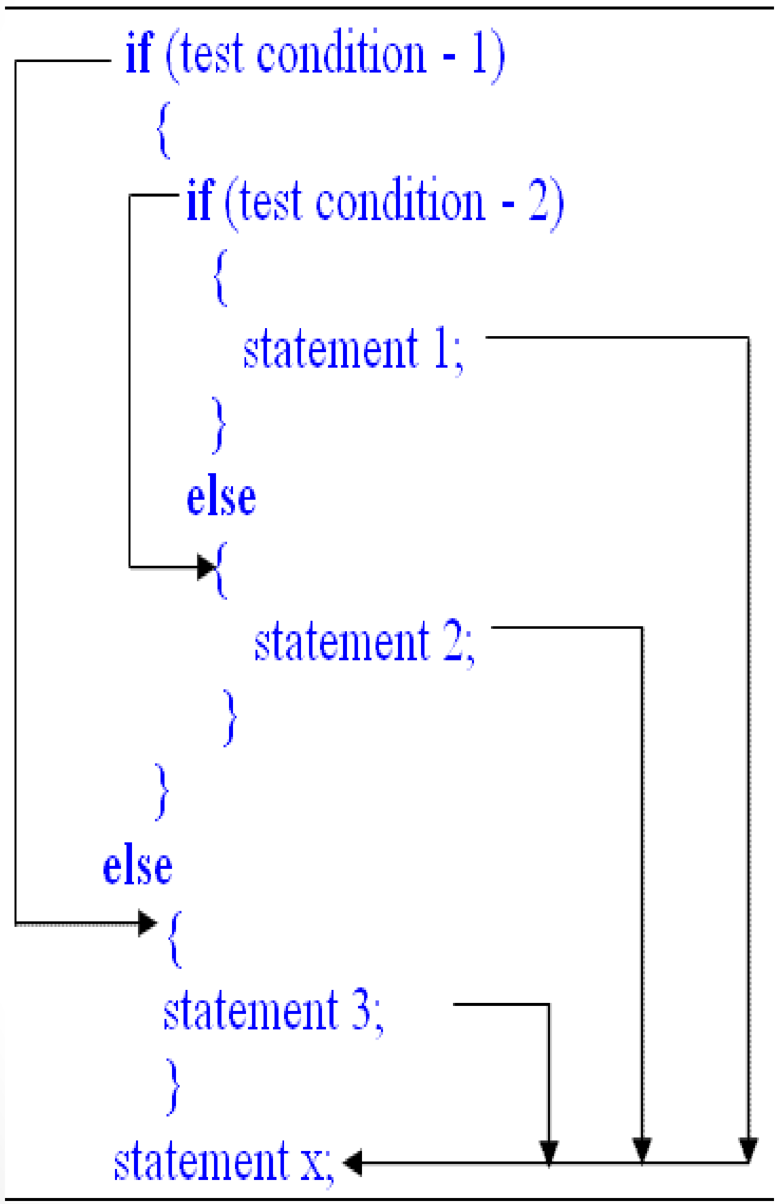
```
if(condition)
{
  statement1;
}
else
{
   statement2;
}
statement x;
```



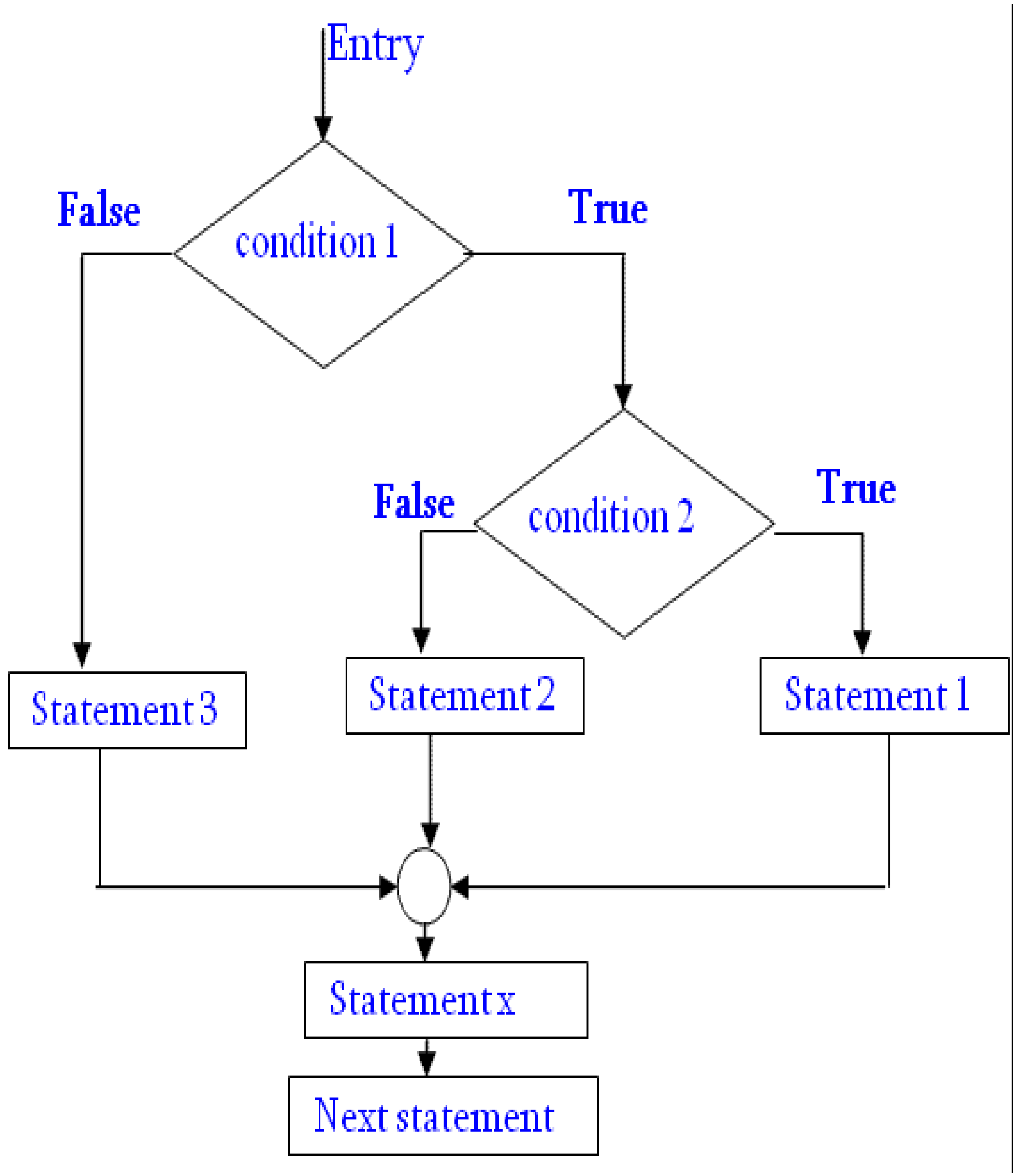
Example

```
class Test
{
  public static void main(String args[])
  {
     int x = 30;
     if( x < 20 )
     {
        System.out.println("This is if statement");
     }
     else
     {
        System.out.println("This is else statement");
     }
  }
}
```

Nested if else



```
if (test condition - 1)
  {
  if (test condition - 2)
    {
      statement 1;
    }
  else
    {
      statement 2;
    }
  }
else
  {
  statement 3;
  }
statement x;
```



Example

```
class Test
{
   public static void main(String args[])
   {
      int x = 30;
      int y = 10;
      if( x == 30 )
     {
         if( y == 10 )
        {
             System.out.println("X = 30 and Y = 10");
          }
        else
        {
           System.out.println("something else");
         }
       }
    }
}
```

Else-if ladder

Syntax:

```
if(test_condition1)
   {
      statement 1;
    }
 else if(test_condition2)
   {
      statement 2;
    }
else if(test_condition3)
   {
      statement 3;
    }
 else if(test_condition4)
   {
      statement 4;
    }
  else
     {
        statement x;
     }
```

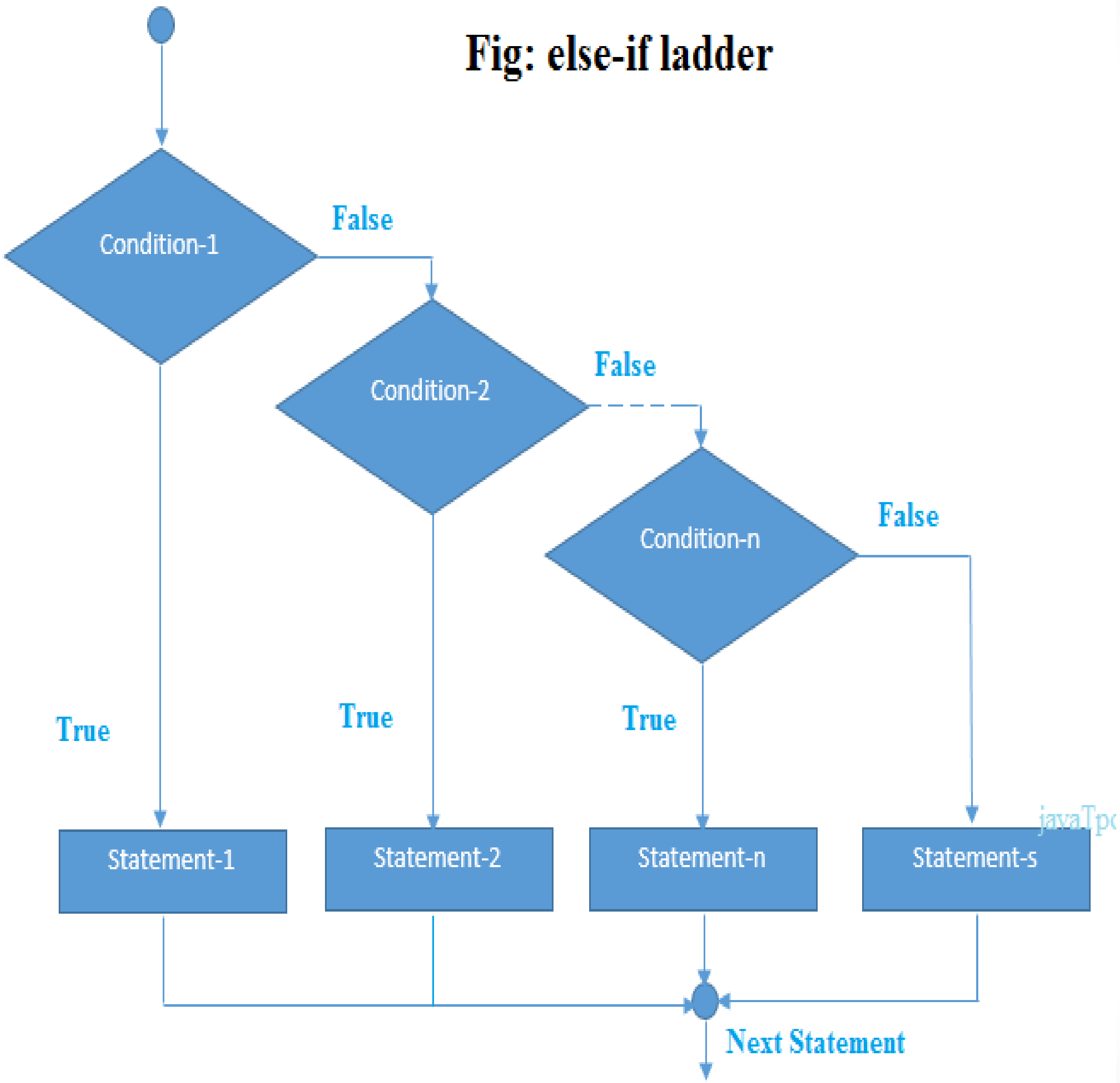


Fig: else-if ladder

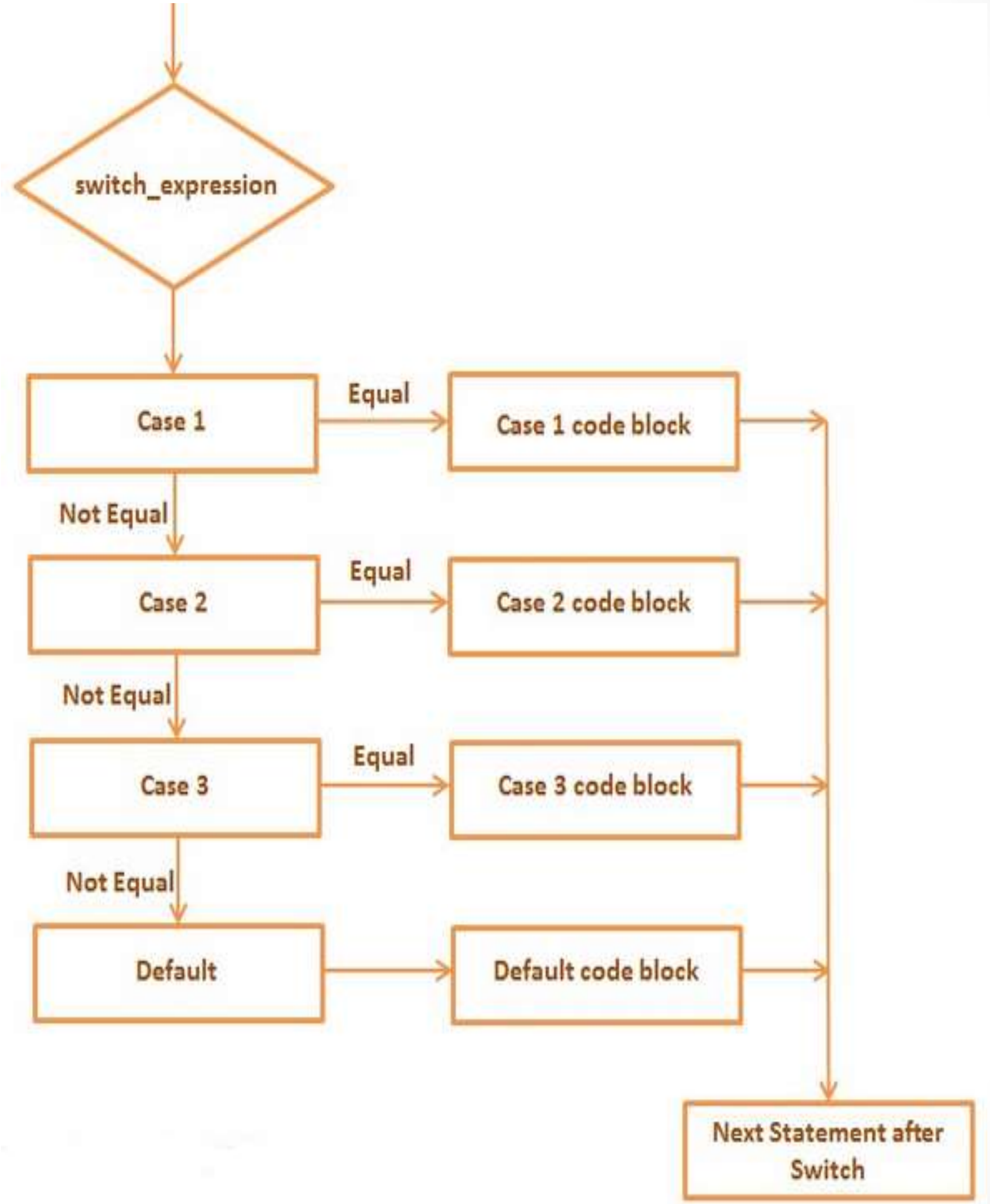
Example

```
class Test
 {
   public static void main(String args[])
   {
      int x = 30;
      if( x == 10 )
      {          System.out.println("Value of X is 10");        }
      else if( x == 20 )
     {           System.out.println("Value of X is 20");        }
      else if( x == 30 )
      {          System.out.println("Value of X is 30");         }
      else
     {
         System.out.println("This is else statement");
      }
   }
}
```

Switch statement

Syntax:

```
switch(expression)
{
case constant-expression : statement(s);
                                                                   break;
case constant-expression : statement(s);
                                                                   break;
default : default_block;            //optional
}
```



Example

```
class Test
{
  public static void main(String args[])
{
        char grade = 'C';
    switch(grade)
    {
      case 'A' :
        System.out.println("Excellent!");
        break;
      case 'B' :
      case 'C' :
        System.out.println("Well done");
        break;
```

contd….

contd….

```
case 'D' :
        System.out.println("You passed");
      case 'F' :
        System.out.println("Better try again");
        break;
      default :
        System.out.println("Invalid grade");
    }
    System.out.println("Your grade is " + grade);
  }
}
```

Types of control statements

- The while statement

- The do-while statement

- The for statement

While loop

Syntax:

```
Initialize the variable;
while(condition)
{
  statements;
  increment/decrement;
}
```

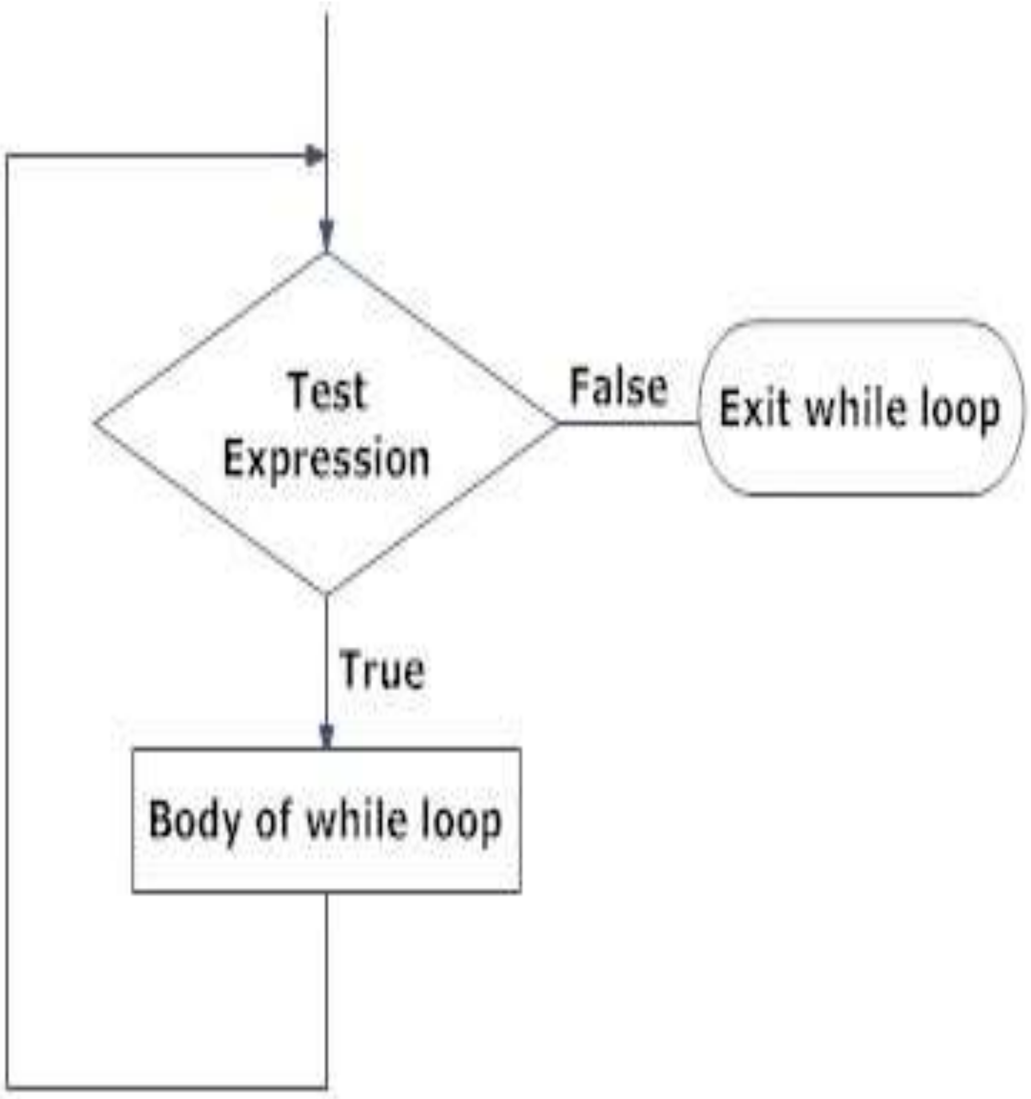


Figure: Flowchart of while loop

Example

```
class Test
{
   public static void main(String args[])
   {
      int x = 10;
      while( x < 20 )
      {
         System.out.println("value of x : " + x );
         x++;
         System.out.println("\n");
      }
   }
}
```

Do-while loop

Syntax:

```
initialize the variable;
do
{
  statements;
  increment/decrement;
}
while(condition);
```

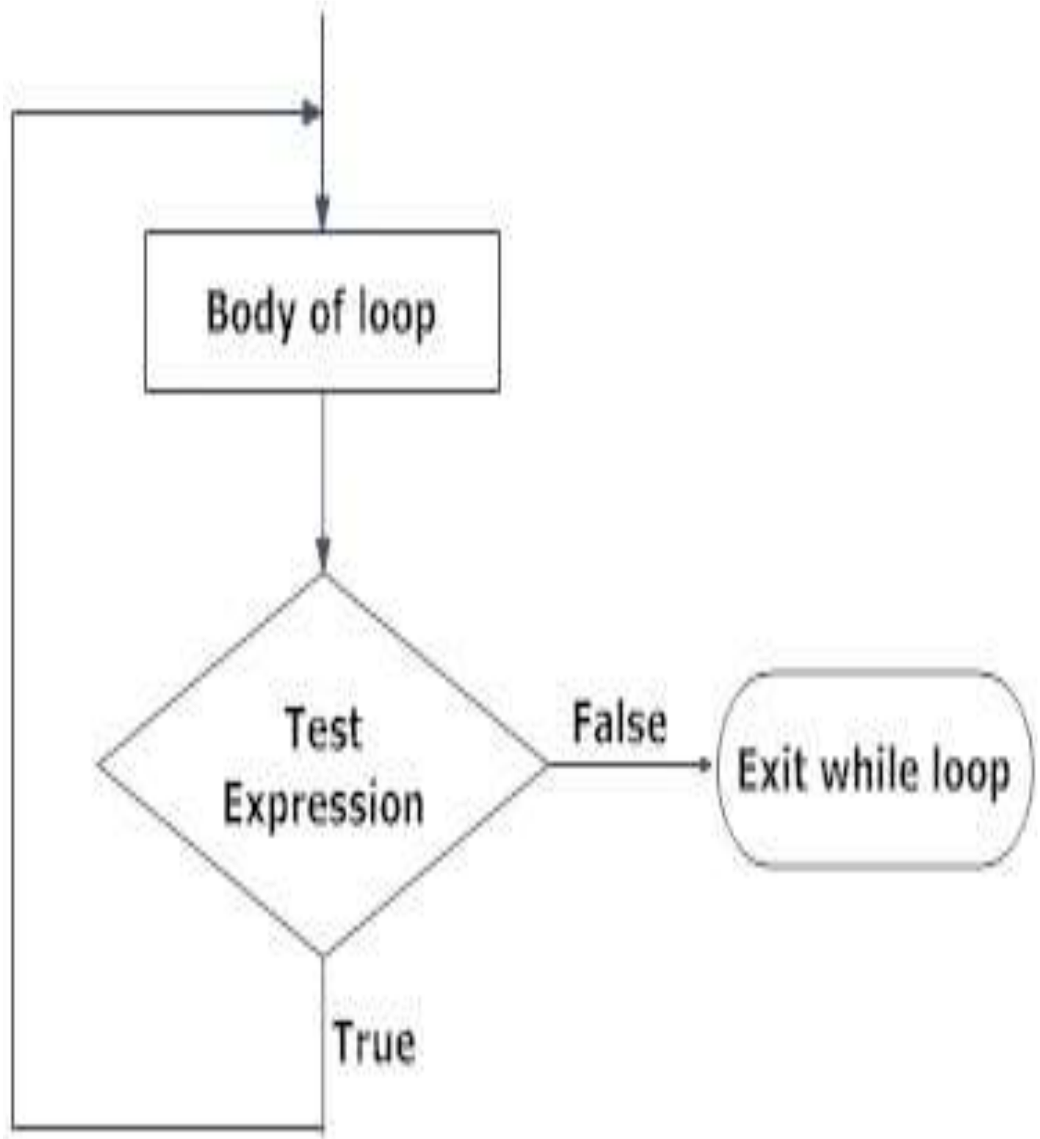


Figure: Flowchart of do...while loop

Example

```
class Test
{
   public static void main(String args[])
   {
      int x = 10;
      do
      {
         System.out.println("value of x : " + x );
         x++;
         System.out.println("\n");
      }while( x < 20 );
   }
}
```

For loop

Syntax:

```
for(initialization; test_condition; increment/decrement)
 {
   code to be executed;
 }
```

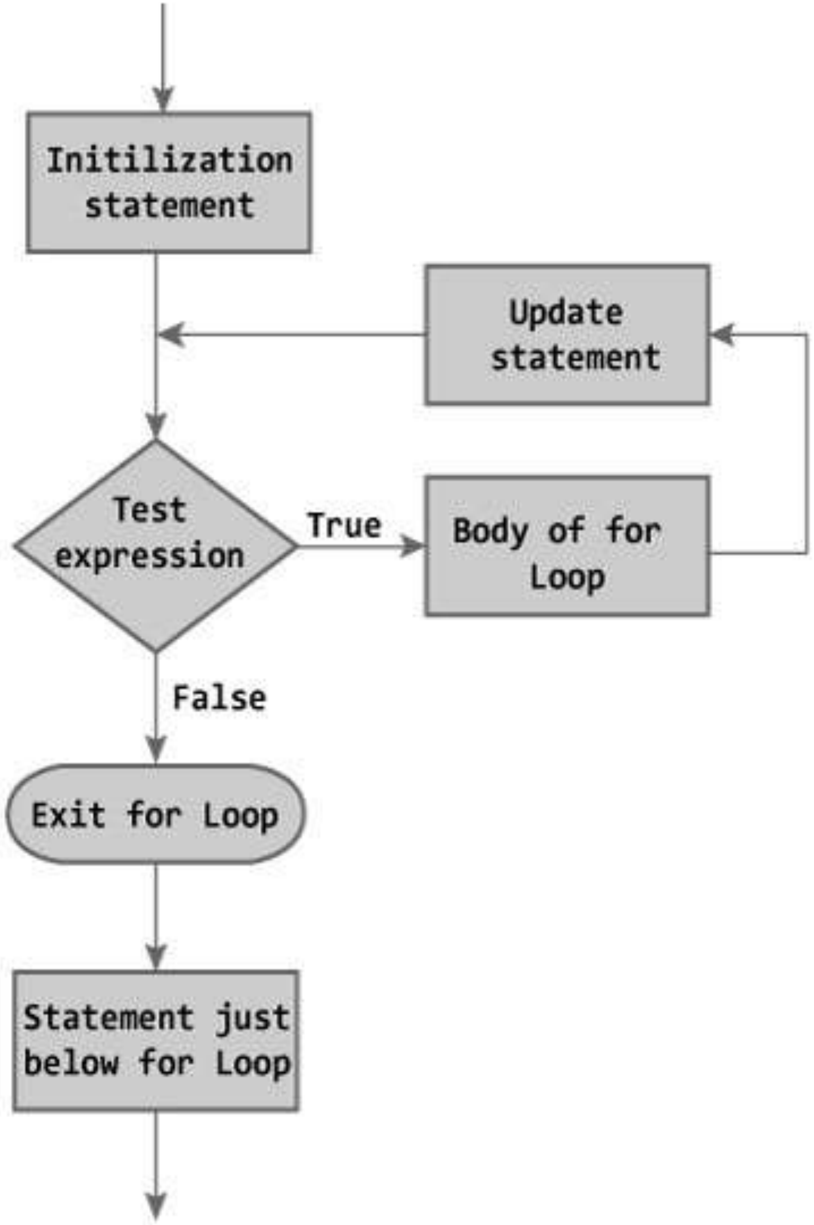


Figure: Flowchart of for Loop

example

```
class Test
{
   public static void main(String args[])
   {
      for(int x = 10; x < 20; x = x+1)
      {
         System.out.println("value of x : " + x );
         System.out.println("\n");
      }
   }
}
```

Jumps in the loops

- Break

- continue