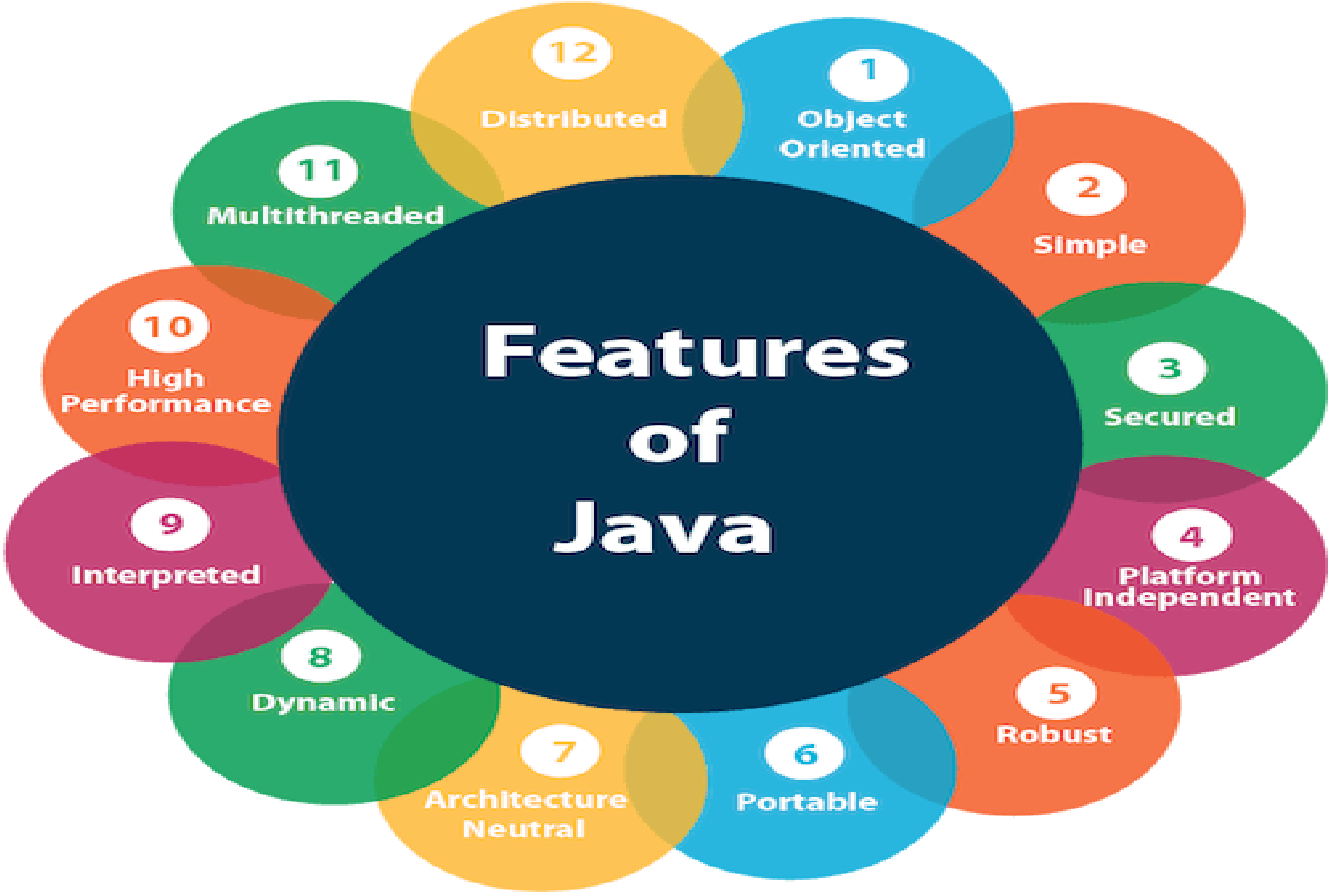
Introduction of Java

Java is a programming language and a platform. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by ***Sun Microsystems*** (which is now the subsidiary of Oracle) in the year 1995. ***James Gosling*** is known as the father of Java. Before Java, its name was ***Oak.*** Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Features Of Java



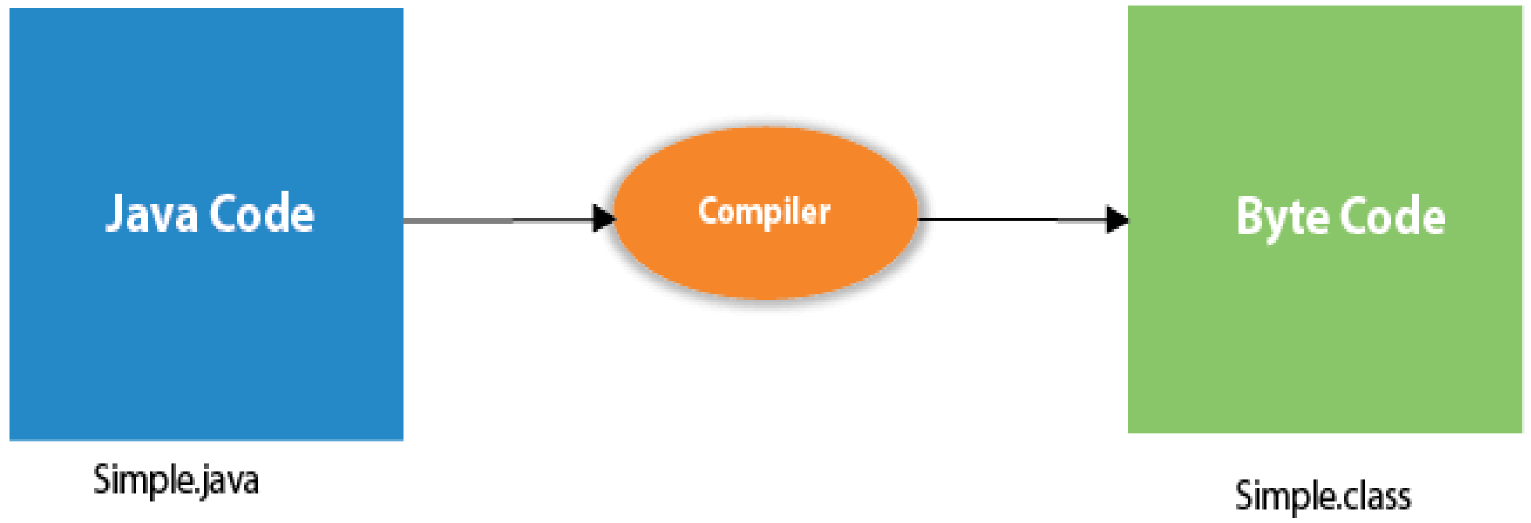
Features Of Java

Java programming were

- Simple,
- Robust,
- Portable,
- Platform-independent,
- Secured,
- High Performance,
- Multithreaded,
- Architecture Neutral,
- Object-Oriented, Interpreted, and Dynamic".

Java Compiler

At compile time, the Java file is compiled by Java Compiler (It does not interact with OS) and converts the Java code into bytecode.



Applications of Java

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

- Desktop Applications such as acrobat reader, media player, antivirus, etc.
- Web Applications such as irctc.co.in, javatpoint.com, etc.
- Enterprise Applications such as banking applications.
- Mobile
- Embedded System
- Smart Card
- Robotics
- Games, etc.

Parameter Used In Java Program

- Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().
- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an 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
- . We will discuss it in coming section.
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println()
- statement in the coming section.

First Java Program

To write the simple program, you need to open notepad by **start menu -> All Programs -> Accessories -> Notepad**

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

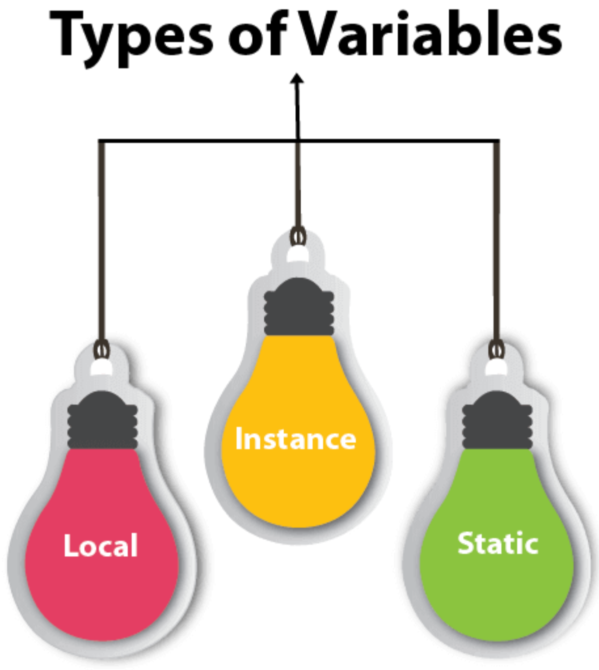
Save the above file as Simple.java.

To compile:javac Simple.java

To execute:java Simple

Variable

- A variable is a container which holds the value while the Java program is executed.
- A variable is assigned with a data type.
- Variable is a name of memory location.



Local : A variable declared inside the body of the method is called local variable.
Instance : A variable declared inside the class but outside the body of the method
Static: A variable that is declared as static is called a static variable

Example of Variables

```
public class A
{
    static int m=100;//static variable
    void method()
    {
        int n=90;//local variable
    }
    public static void main(String args[])
    {
        int data=50;//instance variable
    }
}//end of class
```

Example of Variable

```
public class Simple{
public static void main(String[] args){
int a=10;
int b=10;
int c=a+b;
System.out.println(c);
}
}
```

Output:

```
20
```

Data Types

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

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

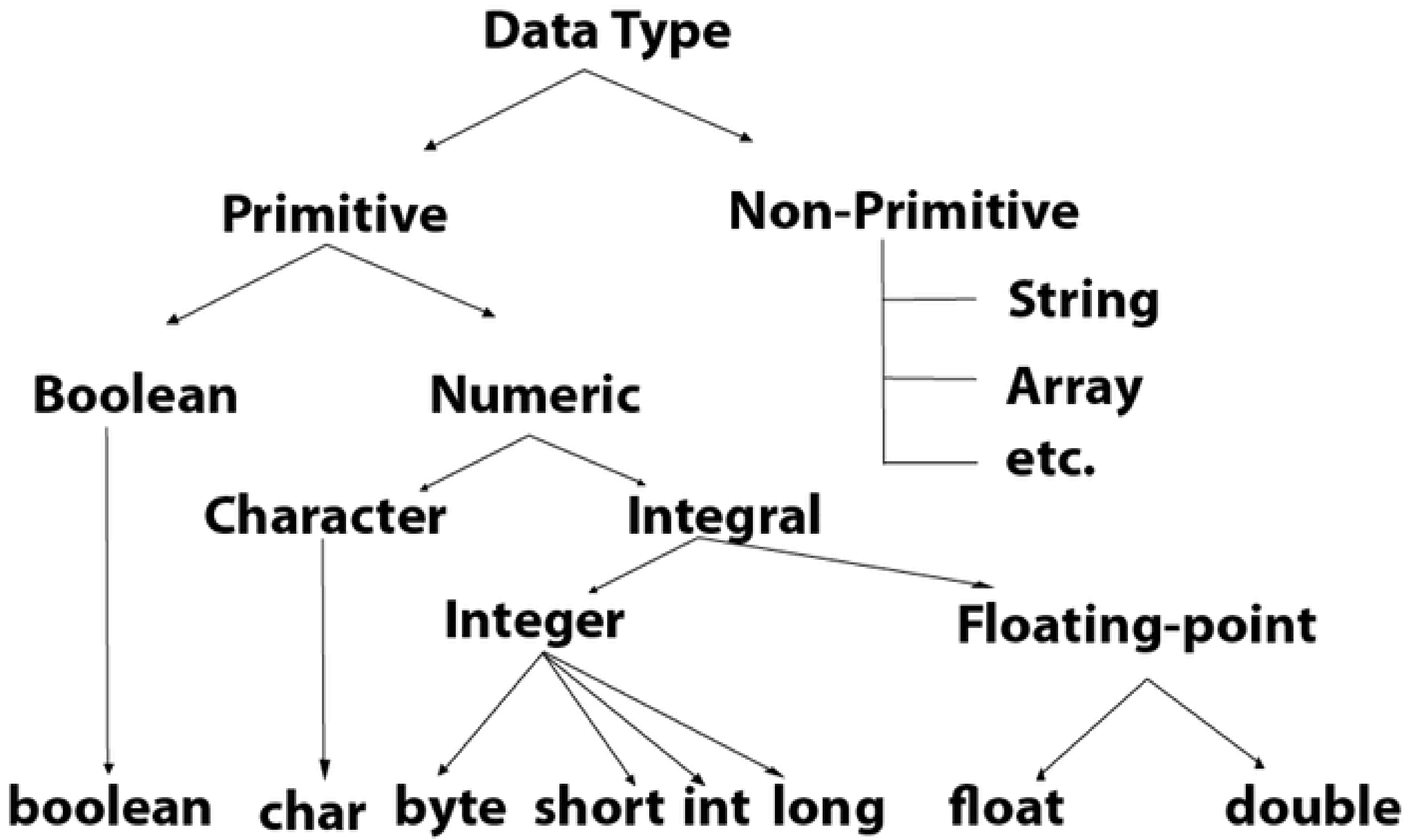
Java Primitive Data Type

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language

There are 8 types of primitive data types :

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type
-

Diagram Of Data Type



Operator in Java

There are many types of operators in Java which are given below:

- Unary Operator : Example: ++ and --
- Arithmetic Operator : +,-,*, /
- Shift Operator : << and >>
- Relational Operator : < > <= >= == !=
- Bitwise Operator :&, ^,|
- Logical Operator :&&, ||
- Ternary Operator : ?:
- Assignment Operator := += ,-=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=

Example of Arithmetic operator

```
public class OperatorExample
{
public static void main(String args[])
{
int a=10;
int b=5;
System.out.println(a+b);    //15
System.out.println(a-b);    //5
System.out.println(a*b);   //50
System.out.println(a/b);    //2
System.out.println(a%b);  //0
}
}
```

Java Control Statements |

Java provides three types of control flow statements.

- Decision Making statements
  - if statements
  - switch statement
- Loop statements
  - do while loop
  - while loop
  - for loop
- Jump statements
  - break statement
  - continue statement

Decision-Making statements

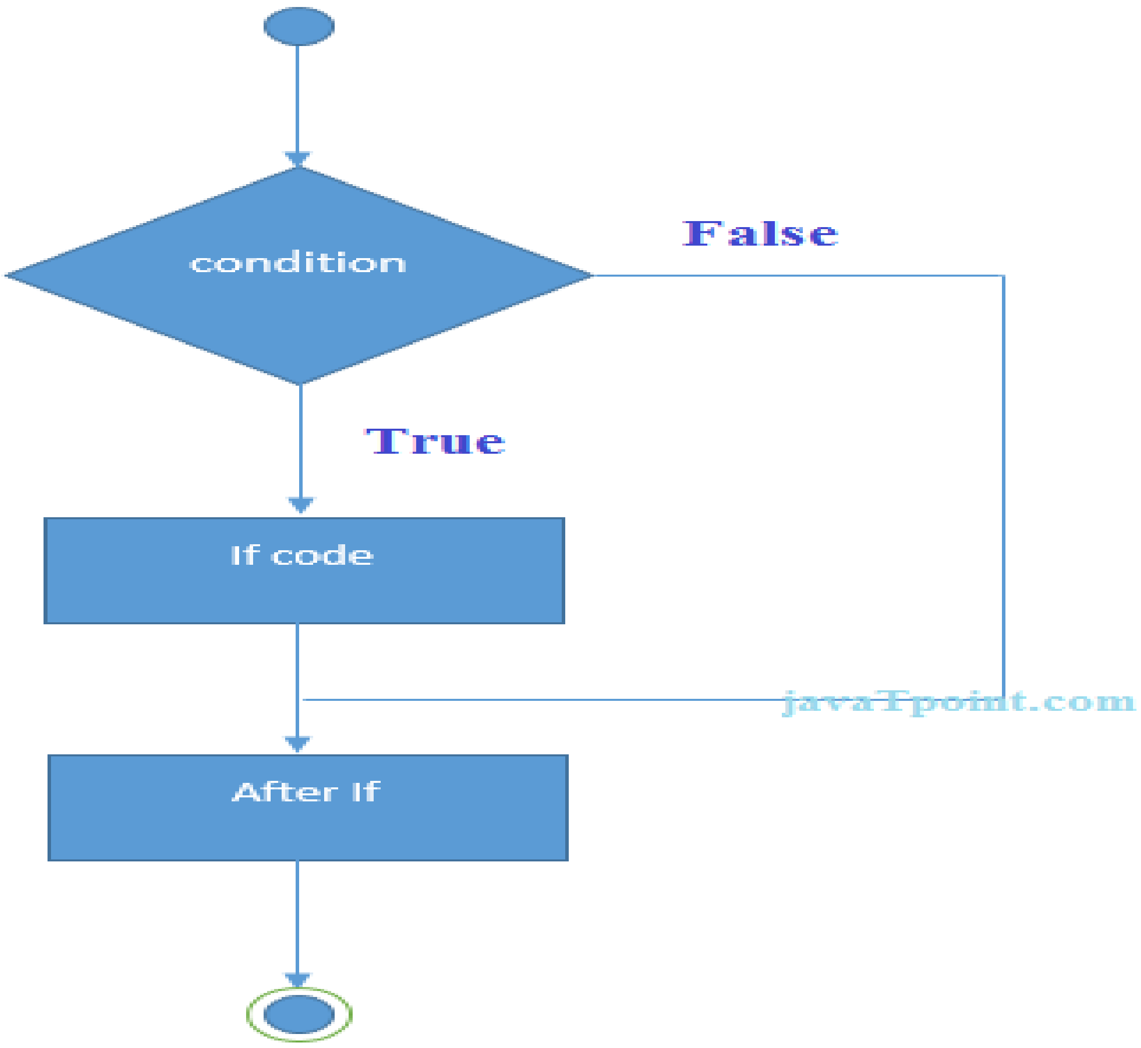
Decision-making statements decide which statement to execute and when.

1) If Statement: In Java, the "if" statement is used to evaluate a condition.

There are four types of if-statements

- Simple if statement
- if-else statement
- if-else-if ladder
- Nested if-statement

Flow Chart of If Statement



Program of If Statements

It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

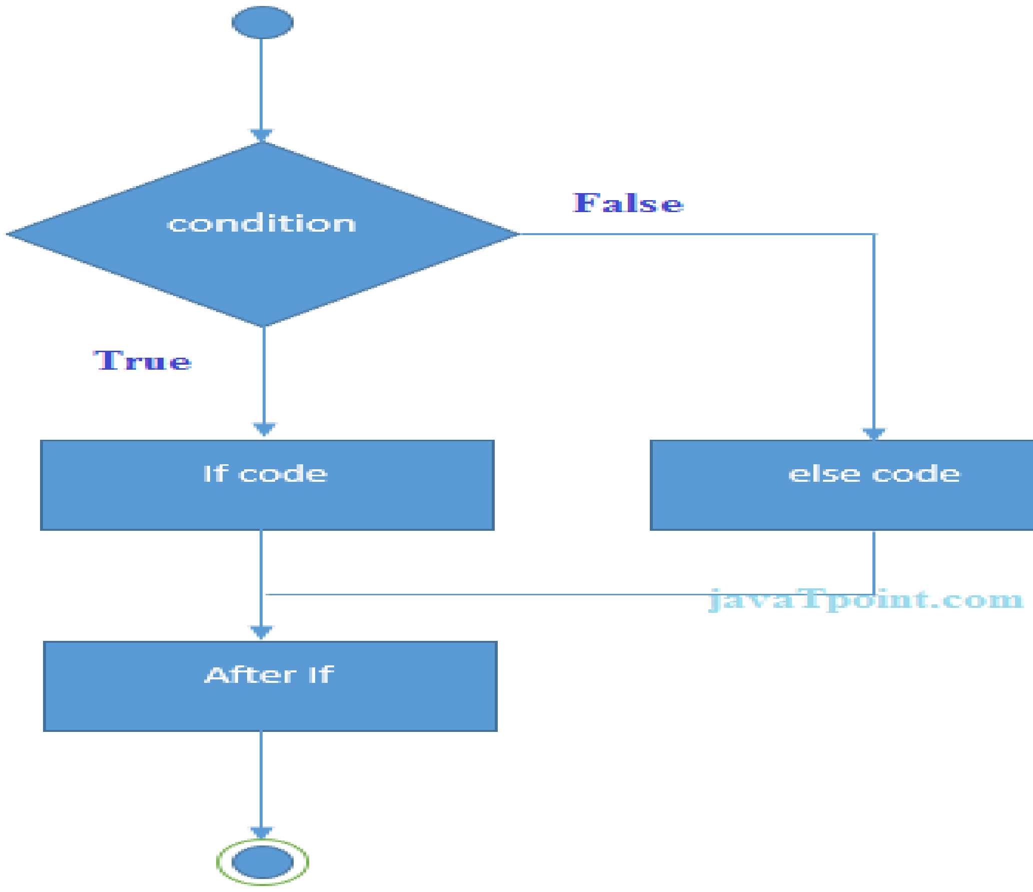
```
if(condition) {
statement 1; //executes when condition is true
}
```

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

Syntax of if else Statements

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

Flow Chart of IF Else Statement

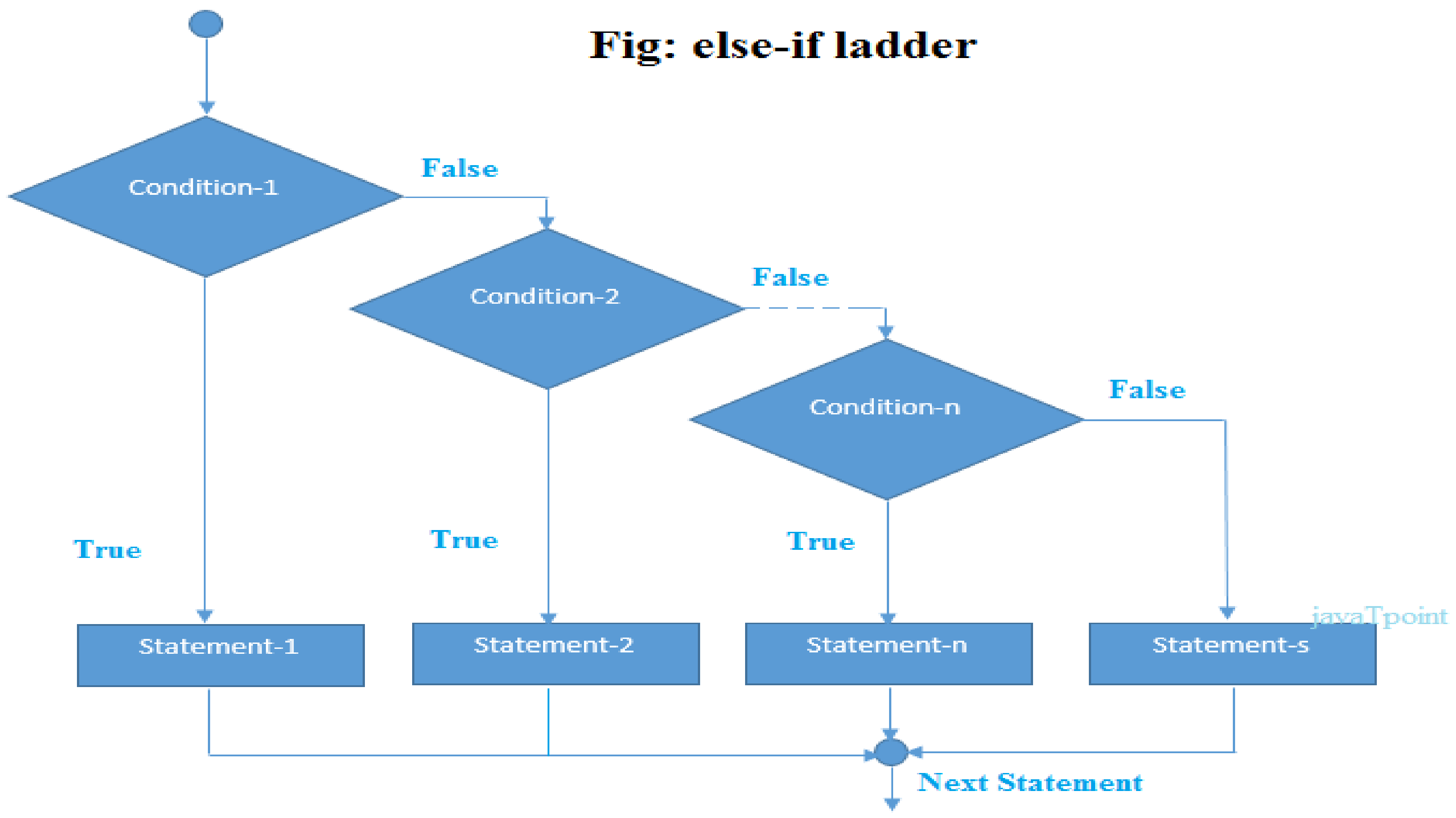


Program of If else Statement

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

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

Flow Chart of IF Else IF Statement



Program of if else if statement

The if-statement followed by multiple else-if statements.

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

Switch Statements

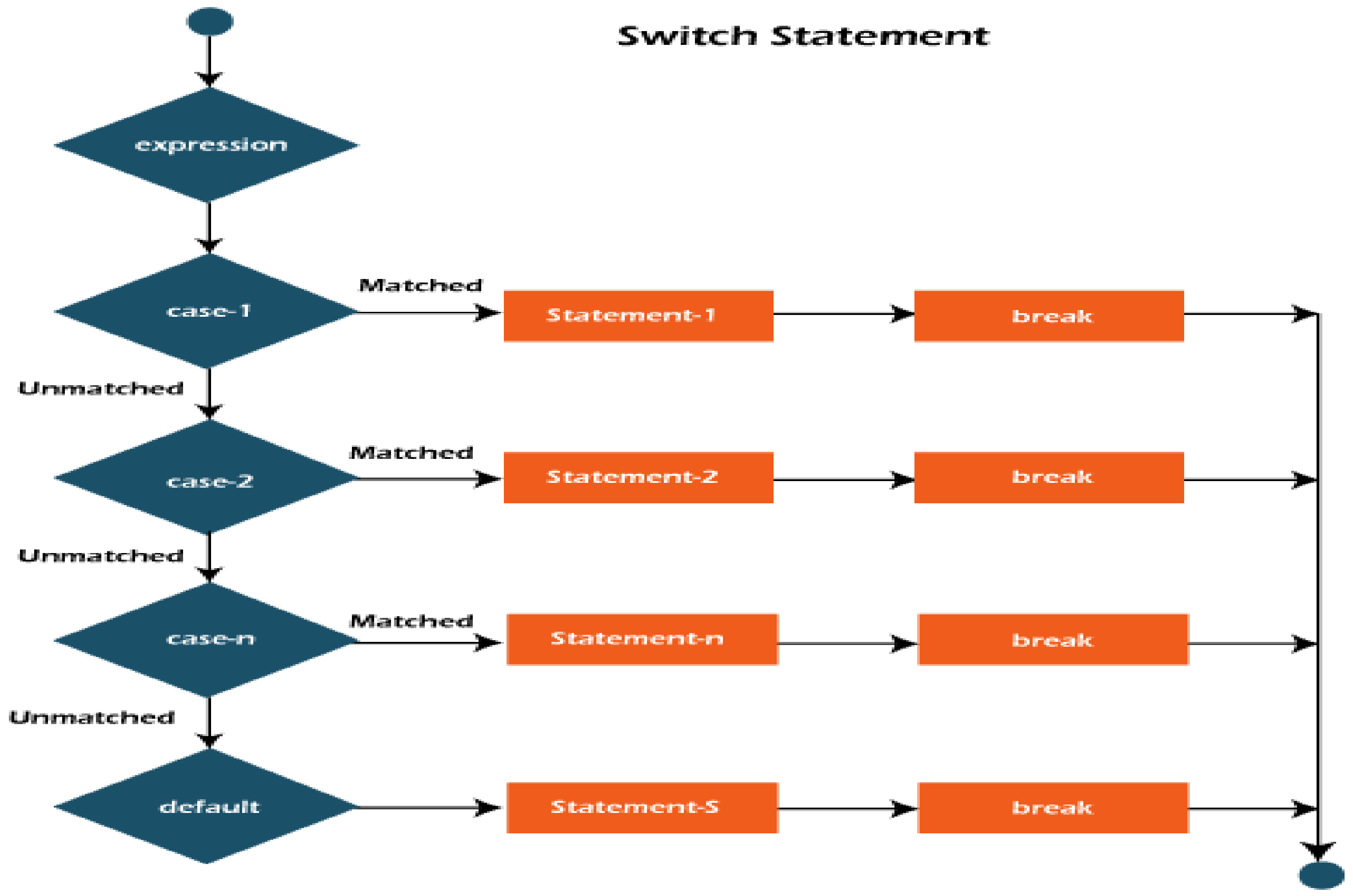
The switch statement contains multiple blocks of code called cases .

Default statement is executed when any of the case doesn't match the value of expression. It is optional.

Break statement terminates the switch block when the condition is satisfied. It is optional, if not used, next case is executed.

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
.......
........
default:
// code to be executed if all cases are not matched;
}
```

Flow Chart of Switch Statement



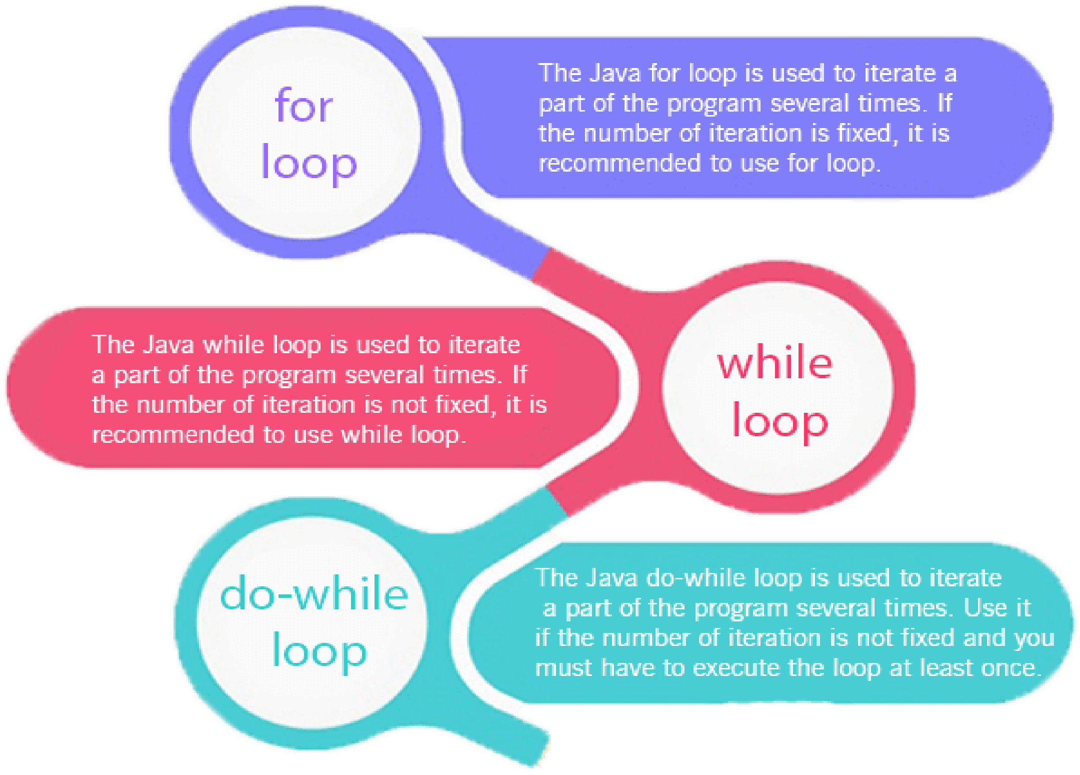
Program of Switch Statement

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

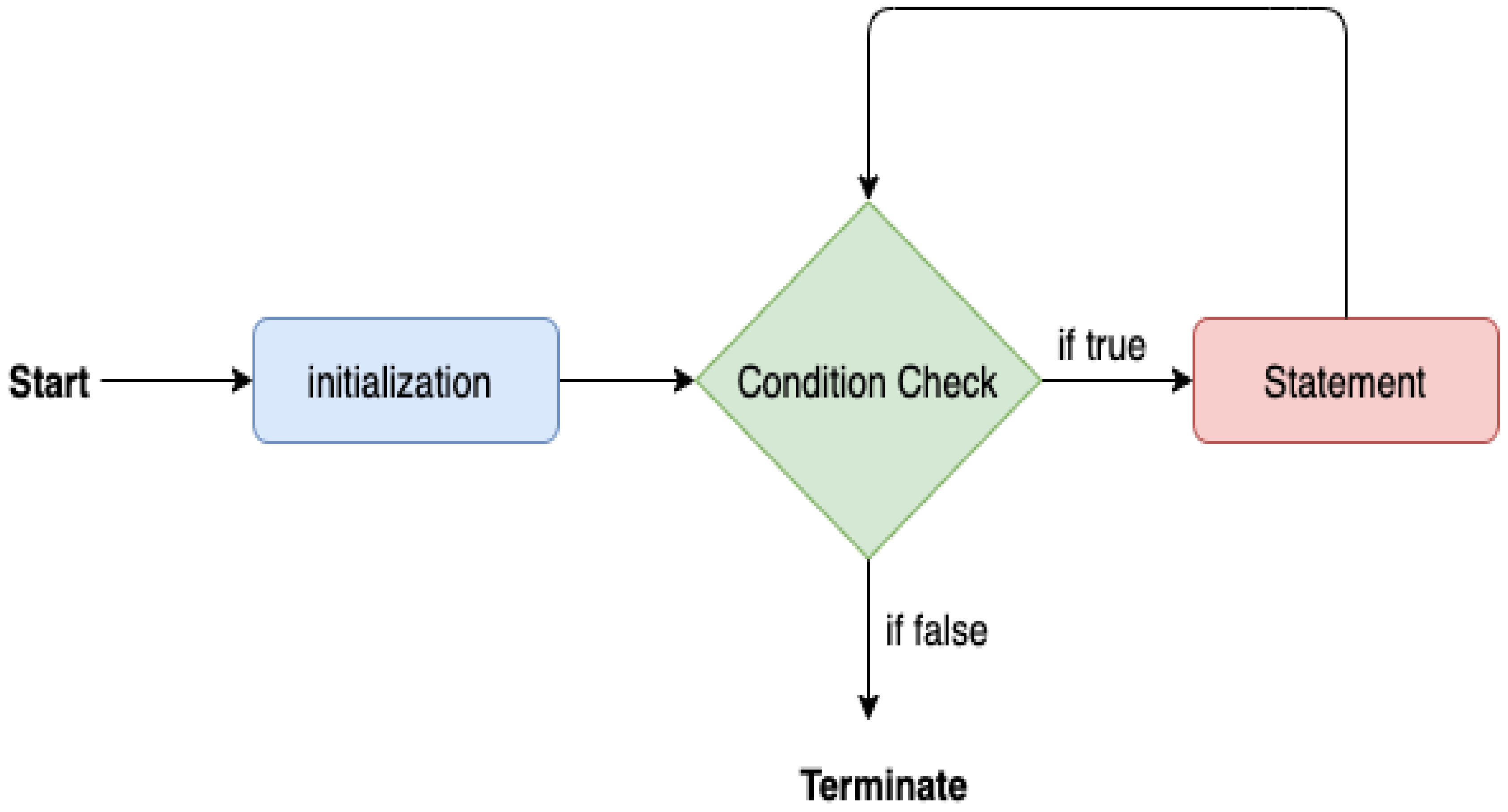
Loops In Java

The Java *for loop* is used to iterate a part of program several times

There are three types of for loops in Java.


for loop
The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.
while loop
The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.
do-while loop
The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

Flow Chart Of For Loop



For loop Program

It consists of four parts:

- Initialization:
- Condition
- Increment/Decrement
- Statement:

```
for(initialization; condition; increment/decrement){
//statement or code to be executed
}
```

//which prints table of 1

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

Nested for loop

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely wenever outer loop executes.

```
public class NestedForExample {
public static void main(String[] args) {
//loop of i
for(int i=1;i<=3;i++){
//loop of j
for(int j=1;j<=3;j++){
    System.out.println(i+" "+j);
}//end of i
}//end of j
}
}
```

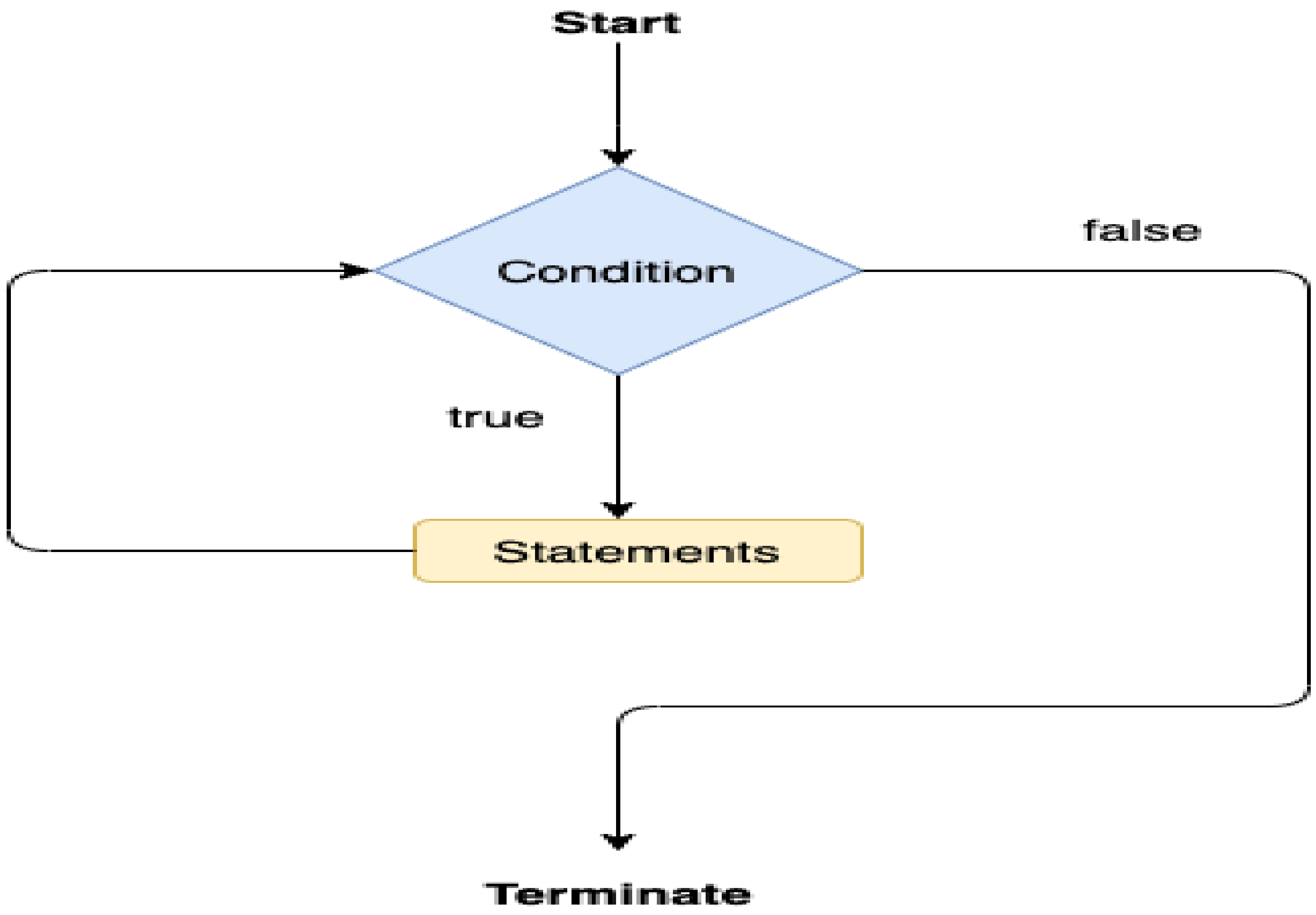
While Loop

The Java *while loop* is used to iterate a part of the program repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops.

Syntax:

```
while (condition){
//code to be executed
Increment / decrement statement
}
```

Flow Chart Of While loop



Program of while loop

```
public class WhileExample {
public static void main(String[] args) {
    int i=1;
    while(i<=10){
        System.out.println(i);
    i++;
    }
}
}
```

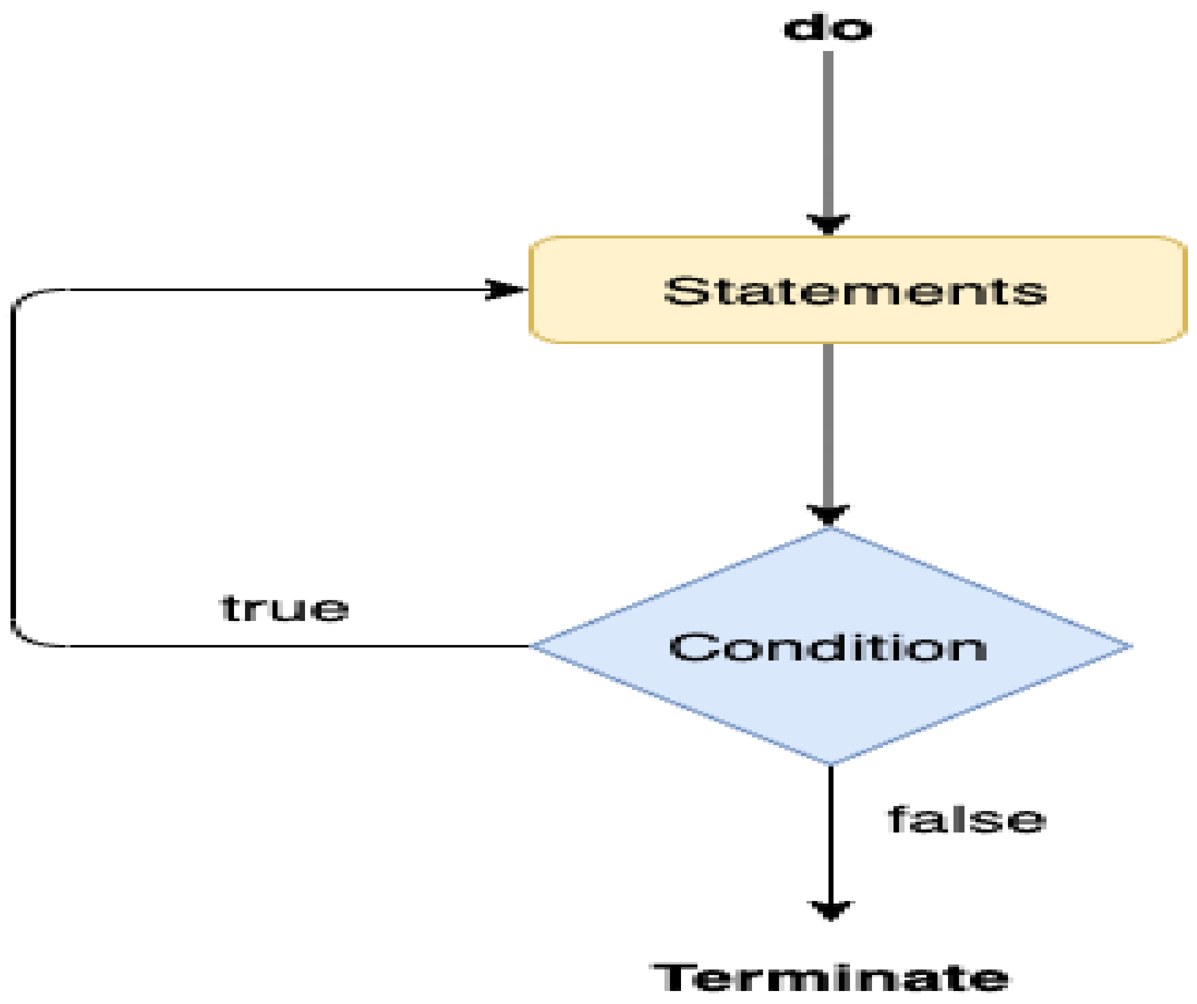
Do While Loop

The Java *do-while loop* is used to iterate a part of the program repeatedly, until the specified condition is true. If the number of iteration is not fixed and you must have to execute the loop at least once. Java do-while loop is called an exit control loop.

Syntax:

```
do{
//code to be executed / loop body
//update statement
}while (condition);
```

Flow Chart of Do While Loop



Program of do while loop

```
public class DoWhileExample {
public static void main(String[] args) {
    int i=1;
    do{
        System.out.println(i);
    i++;
    }while(i<=10);
}
}
```

Java Break Statement

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

Synyax:

```
jump-statement;
break;
public class BreakExample {
public static void main(String[] args) {
   //using for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //breaking the loop
         break;
      }
      System.out.println(i);
   }
}
}
```

Arrays

- An array is a collection of similar type of elements which has contiguous memory location.
- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- We can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.
- Array is an object of a dynamically generated class

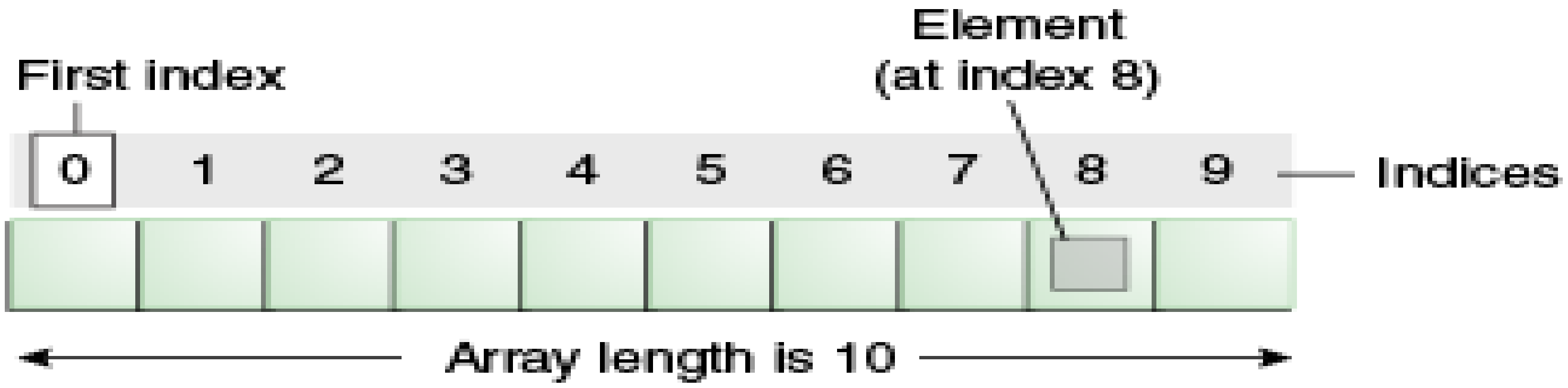
Types Of Arrays

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

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

- dataType[] ;
- dataType[][];



Single Dimension Array

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

Syntax of Array :

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

Example Of Single Dimensional Array

```
class Testarray1{
public static void main(String args[]){
int a[]={33,3,4,5};//declaration, instantiation and ini
    tialization
//printing array
for(int i=0;i<a.length;i++)//length is the property of
    array
System.out.println(a[i]);
}}
```

Multi Dimensional Array

- A multidimensional array is an array of arrays.
- Multidimensional arrays are useful when you want to store data as a tabular form, like a table with rows and columns.
- To create a two-dimensional array, add each array within its own set of **curly braces**:

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

Example Of Multidimensional Array

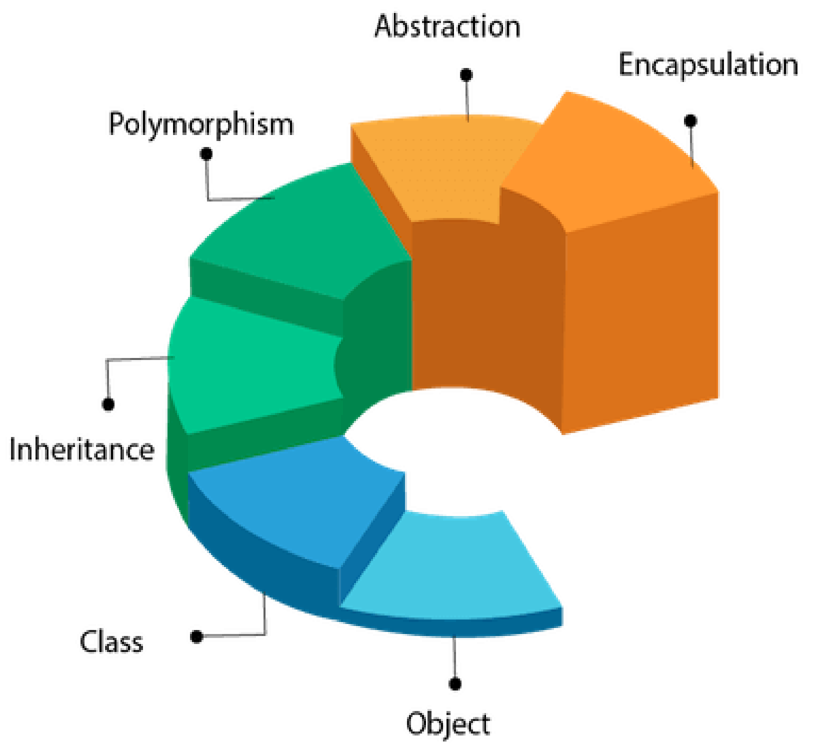
```
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}}
```

Opps concept

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

OOPs (Object-Oriented Programming System)



Object in Java

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

An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely
- **An object is an instance of a class.** A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**

Syntax to declare a class:

```
class <class_name>
{
   field;
   method;
}
```

Program of class

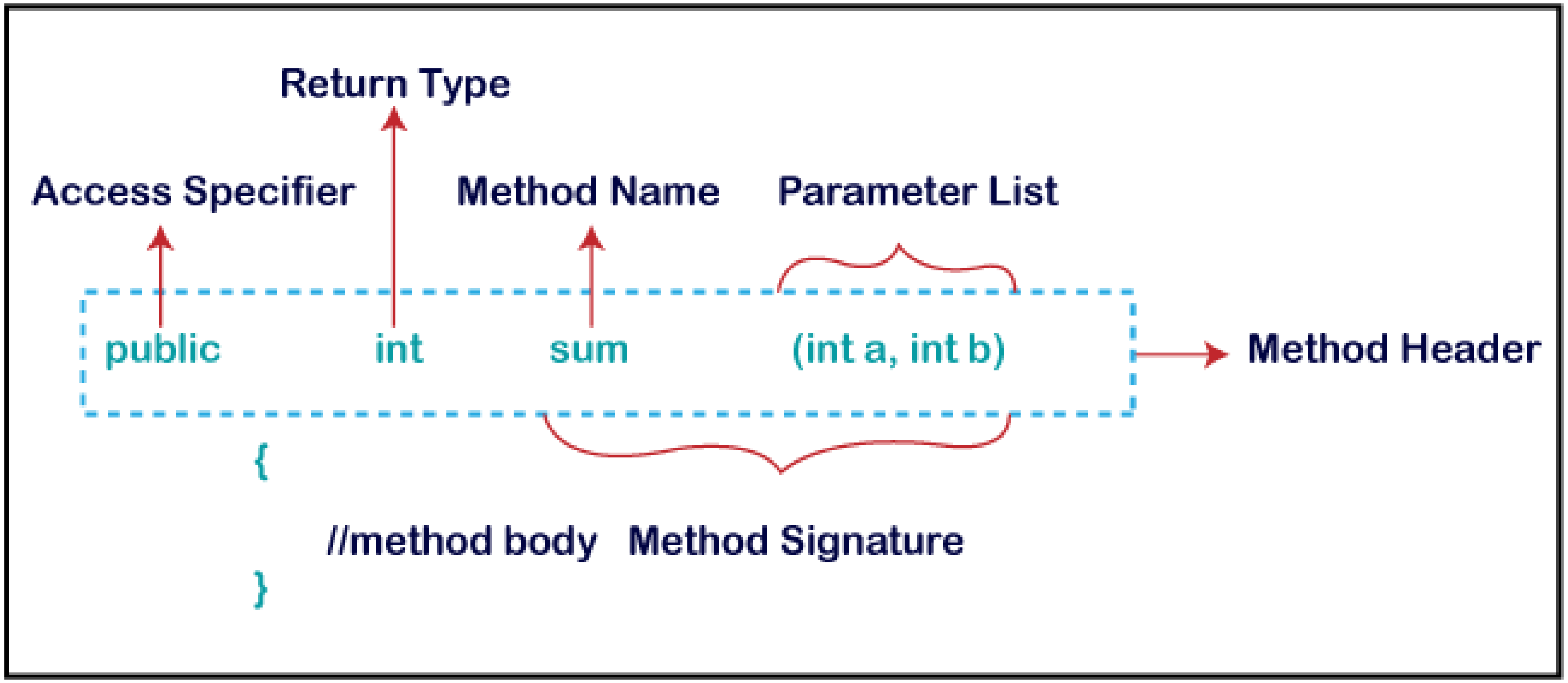
Defining a Student class.

```
class Student{
 //defining fields
 int id;//field or data member or instance variable
 String name;
 //creating main method inside the Student class
 public static void main(String args[]){
  //Creating an object or instance
  Student s1=new Student();//creating an object of Student
  //Printing values of the object
  System.out.println(s1.id);//accessing member through reference varia
    ble
  System.out.println(s1.name);
 }
}
```

Method in Java

Method is a way to perform some task .A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.

Method Declaration



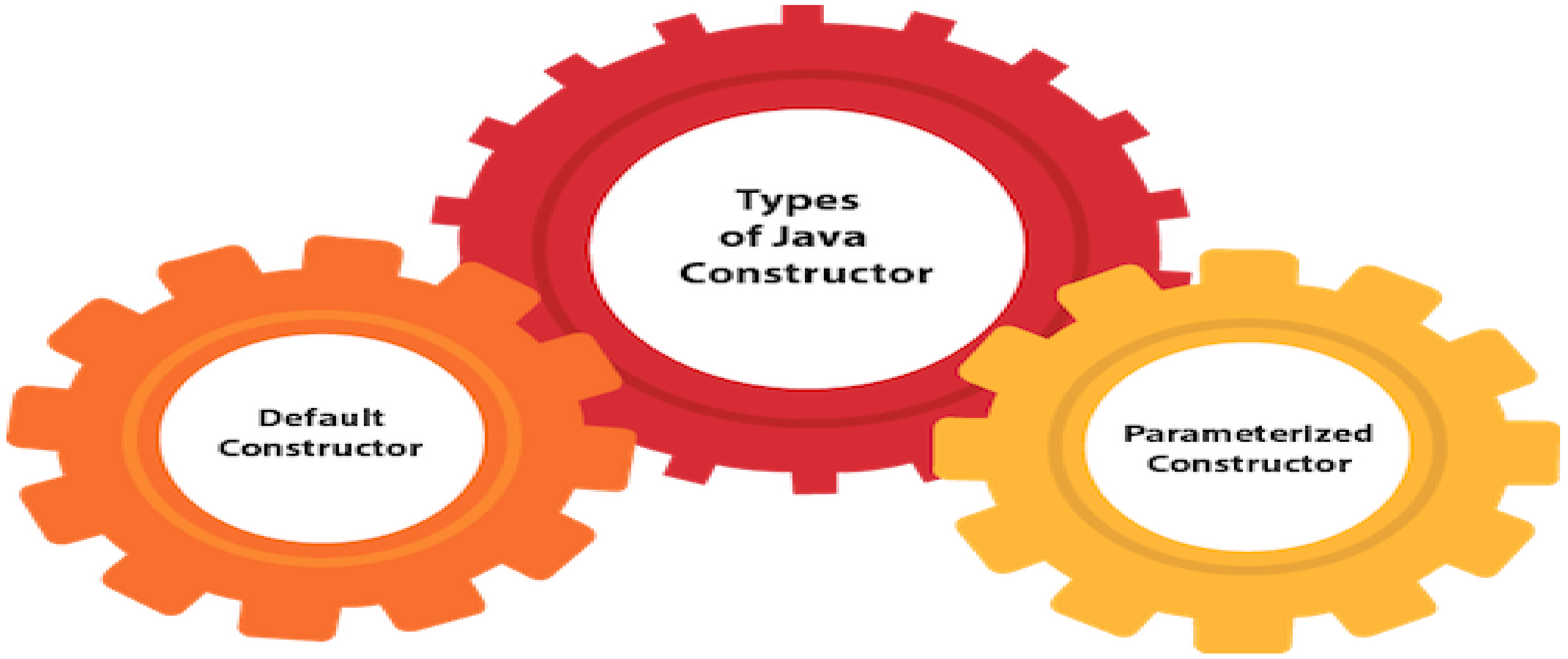
Constructor

- constructor is a block of codes similar to the method. It is called when an instance of the class is created.
- Every time an object is created using the new() keyword, at least one constructor is called. It calls a default constructor .
- It is a special type of method which is used to initialize the object.

Types of Constructor

There are two types of constructors in Java:

- Default constructor (no-arg constructor)
- Parameterized constructor



Example Of Constructor

```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Inheritance in Java

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

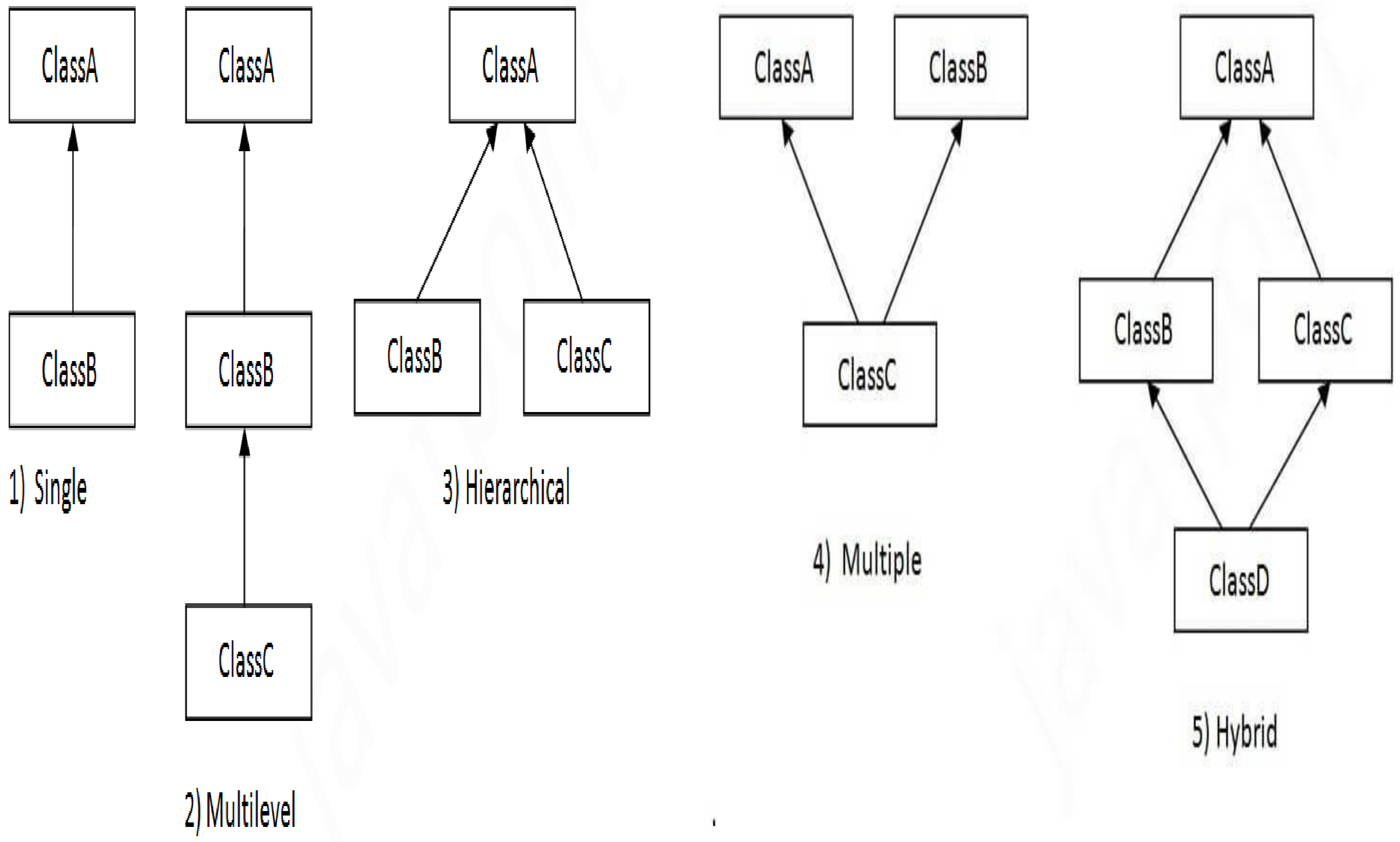
Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class.

Types of inheritance in java



Program of inheritance

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus)
    ;
}
}
```

Polymorphism

Polymorphism is considered one of the important features of Object-Oriented Programming.

The word “poly” means many and “morphs” means forms, So it means many forms.

We can achieve polymorphism by the use of :

- Method Overloading
- Method Overriding
- Super Keyword
- Final Keyword

Method Overloading

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

Method Overriding in Java

if subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

```
/Java Program to illustrate the use of Java Method Overriding
//Creating a parent class.
class Vehicle{
  //defining a method
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){System.out.println("Bike is running safely");}

  public static void main(String args[]){
  Bike2 obj = new Bike2();//creating object
  obj.run();//calling method
  }
}
```

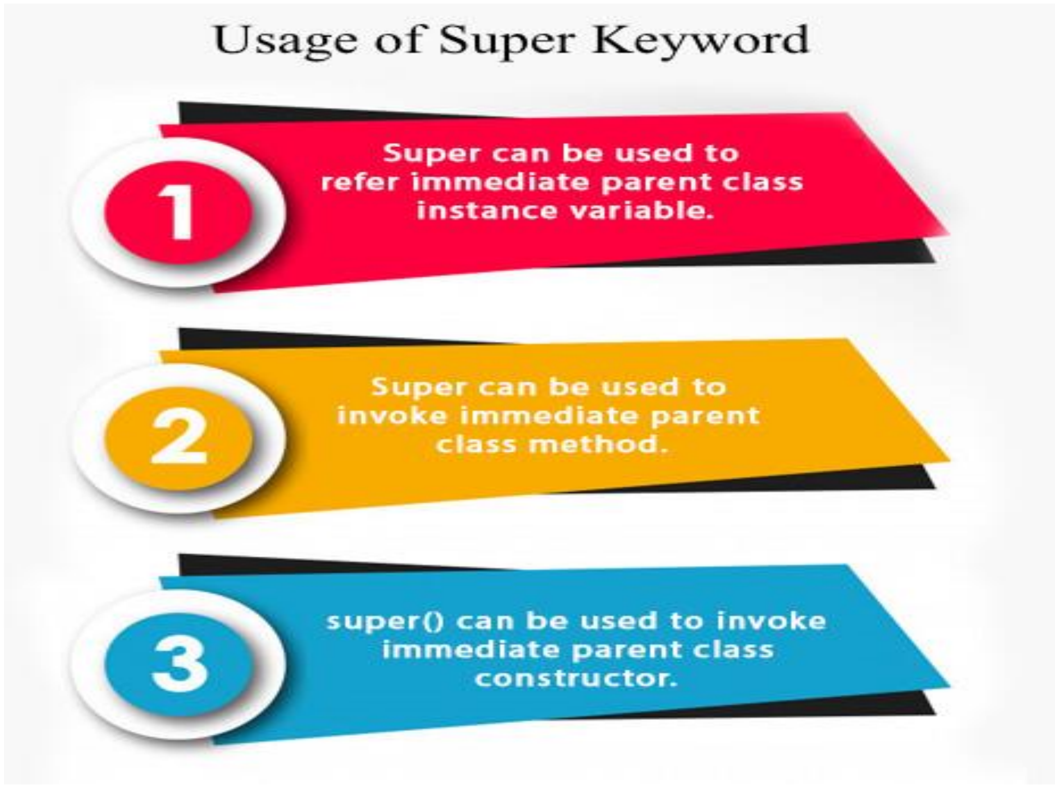
Output :Vehicle is running

Super Keyword

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword


Usage of Super Keyword
1
Super can be used to refer immediate parent class instance variable.
2
Super can be used to invoke immediate parent class method.
3
super() can be used to invoke immediate parent class constructor.

Example of Super Keyword

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

Output : black

```
black
white
```

Final Keyword

- The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:
- Variable : you cannot change the value of final variable(It will be constant).
- Method :If you make any method as final, you cannot override it.
- Class :If you make any class as final, you cannot extend it.

Example Of Final Keyword

```
class Bike{
  final void run(){System.out.println("running");}
}
    class Honda extends Bike{
   void run(){System.out.println("running safely with 100k
    mph");
}
      public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

Output : compile time error

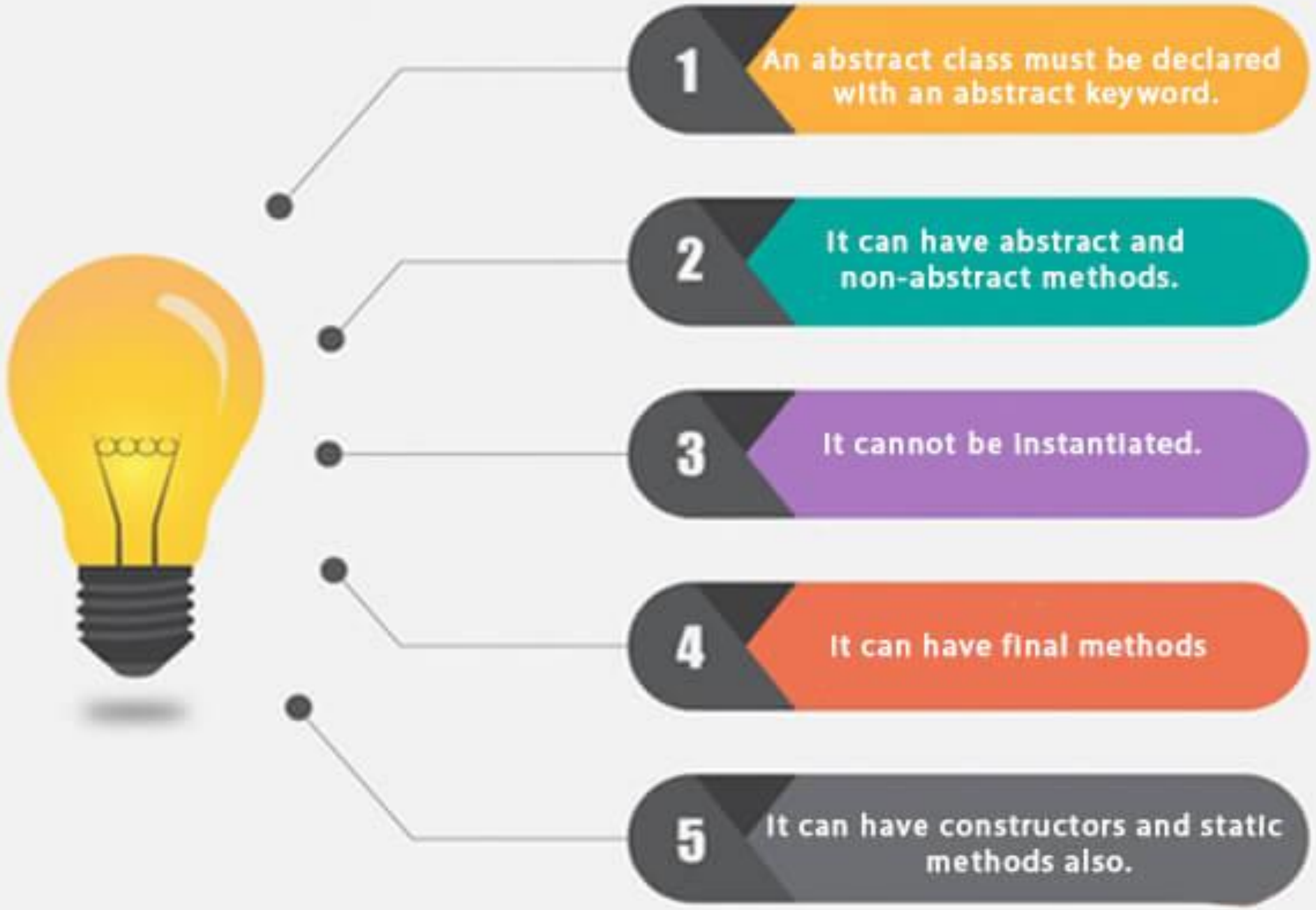
Abstraction

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- Interface (100%)

Rules for Java Abstract class



1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. It cannot be instantiated.
4. It can have final methods
5. It can have constructors and static methods also.

Example of Abstraction Class and Method

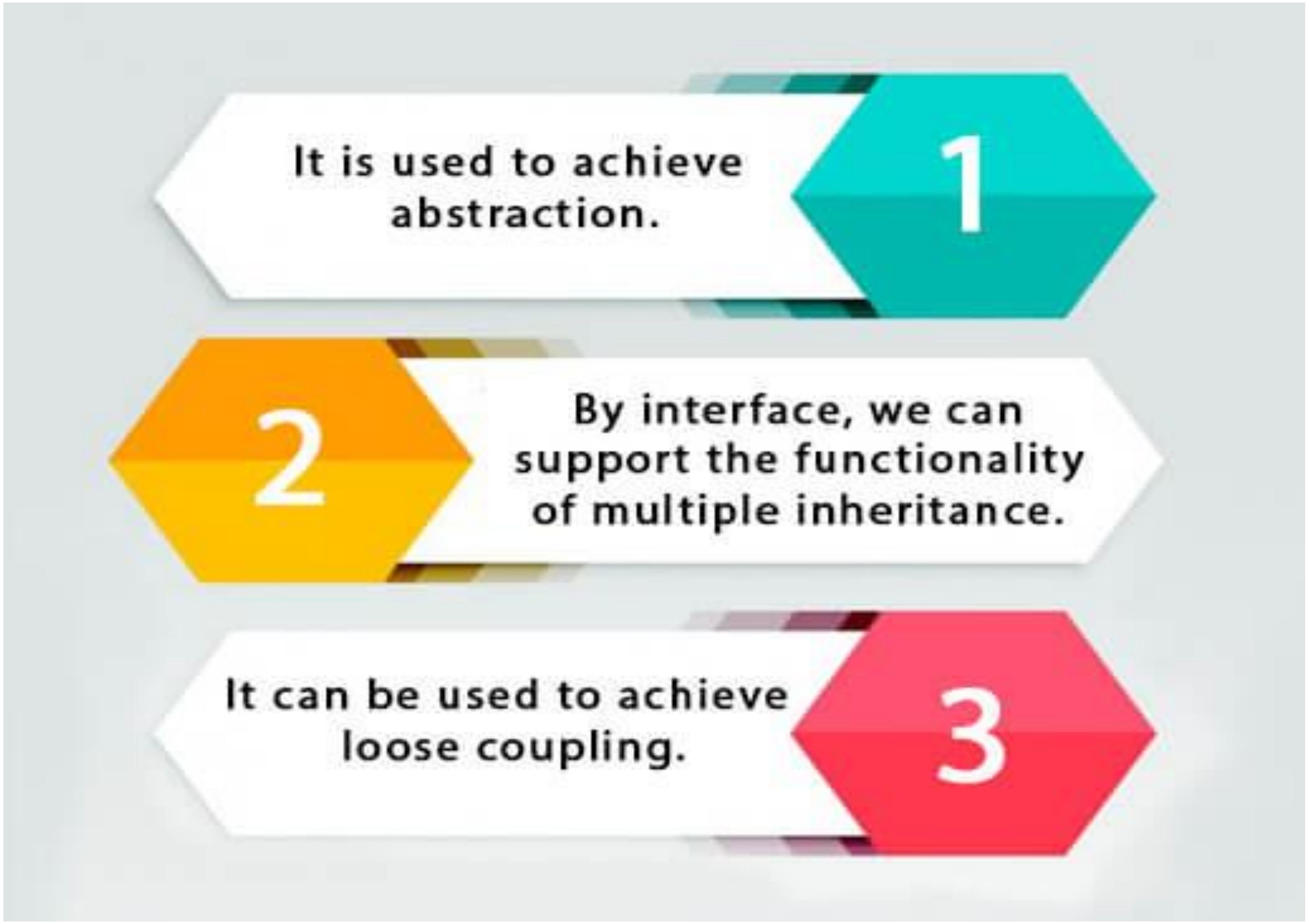
```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., getSh
    ape() method
s.draw();
}
}
```

Output: drawing circle

Interface

- An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.
- The interface in Java is *a mechanism to achieve abstraction*
- There can be only abstract methods in the Java interface, not method body.

Use of Interface


1
It is used to achieve abstraction.
2
By interface, we can support the functionality of multiple inheritance.
3
It can be used to achieve loose coupling.

Example of Interface

```
/Interface declaration: by first user
interface Drawable{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided by method e.g.
    getDrawable()
d.draw();
}}
```

Output: drawing circle

Package In Java

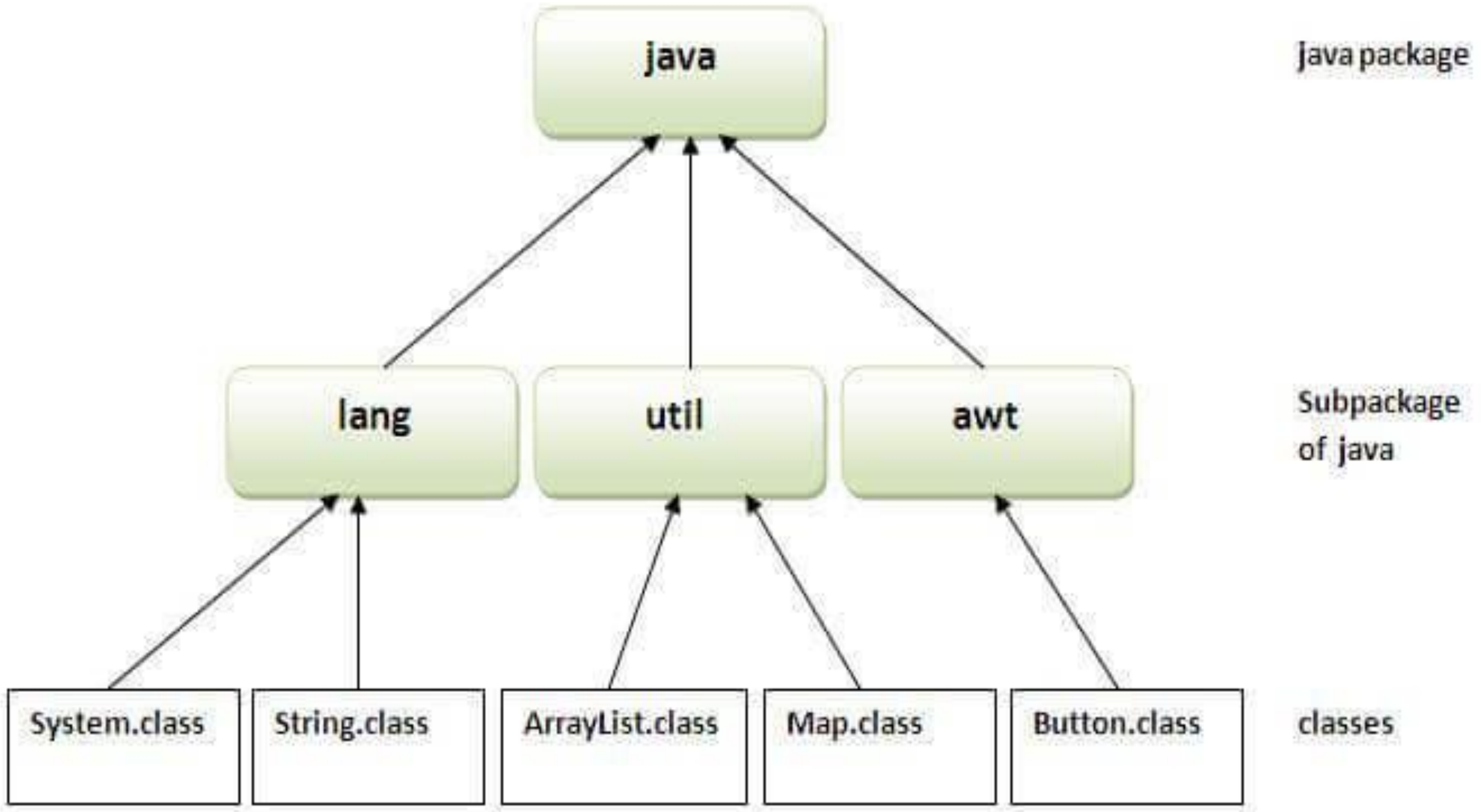
A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form,

- Built-in Package
- User-defined Package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Package And Sub Package



Example Of Package

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.A;
  class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output : Hello

END