netlink
digital solutions

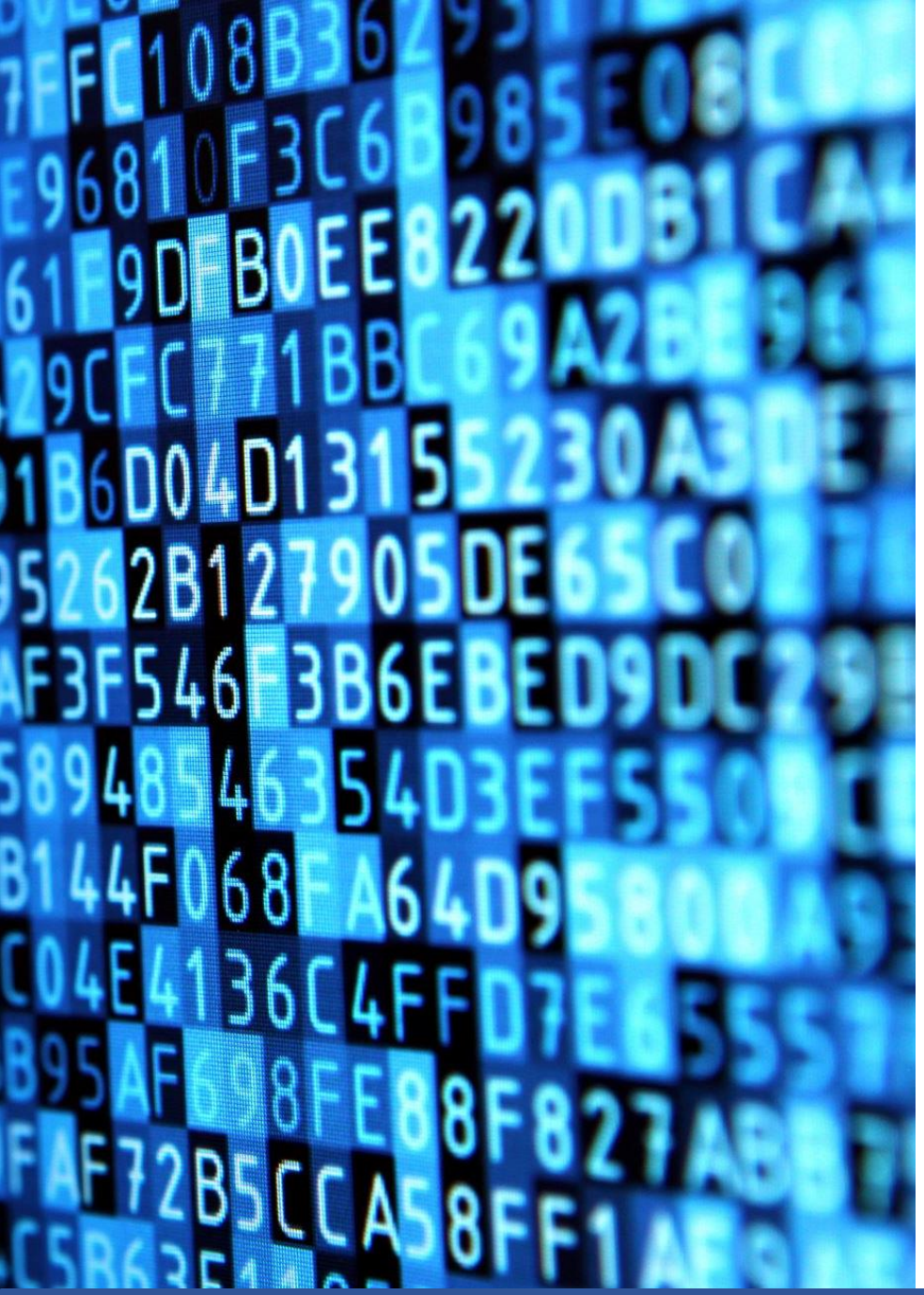
Java

www.netlink.com

Topics

- **www.netlink.com**

INDEX	
S.no.	**Topics**
1	Introduction to Java Programming
2	Introduction to Object Oriented Programming
3	Structure of Java application
4	Class members
5	Static members flow using JVM
6	Non-static members flow Using JVM architecture
7	Data types
8	Access Modifiers
9	Packages
10	Object Oriented Programming
11	This
12	Inheritance
13	Super
14	Final modifier
15	Abstract classes
16	Interfaces
17	Polymorphism
18	Has-a relation
19	Collections
20	Exception handling



Introduction to Types of Computer Language

- **The computer language is defined as code or syntax which is used to write programs or any specific applications.**

1. Machine Language

The machine language is sometimes referred to as machine code or object code which is set of binary digits 0 and 1.

2. Assembly Language:

The assembly language is mostly famous for writing an operating system and also in writing different desktop applications

3. High-Level Language:

The development of high-level language was done when the programmers face the issue in writing programs as the older language has portability issues which mean the code written in one machine cannot be transferred to other machines.

Types of Applications

There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application.
Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

What Is Java?

Java is a computer programming language that enforces an **object-oriented** programming model

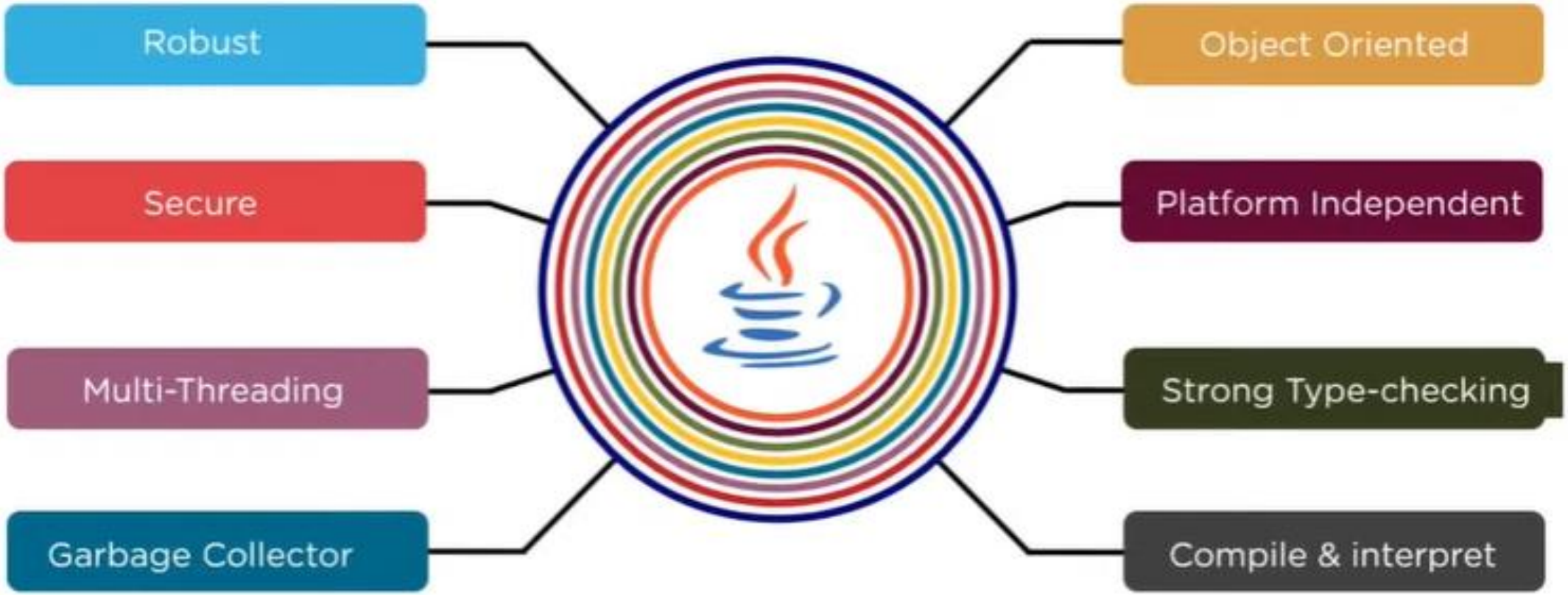
Java is a programming language and computing platform first released by **Sun Microsystems in 1995**

Java was created by a team lead by ***James Gosling***

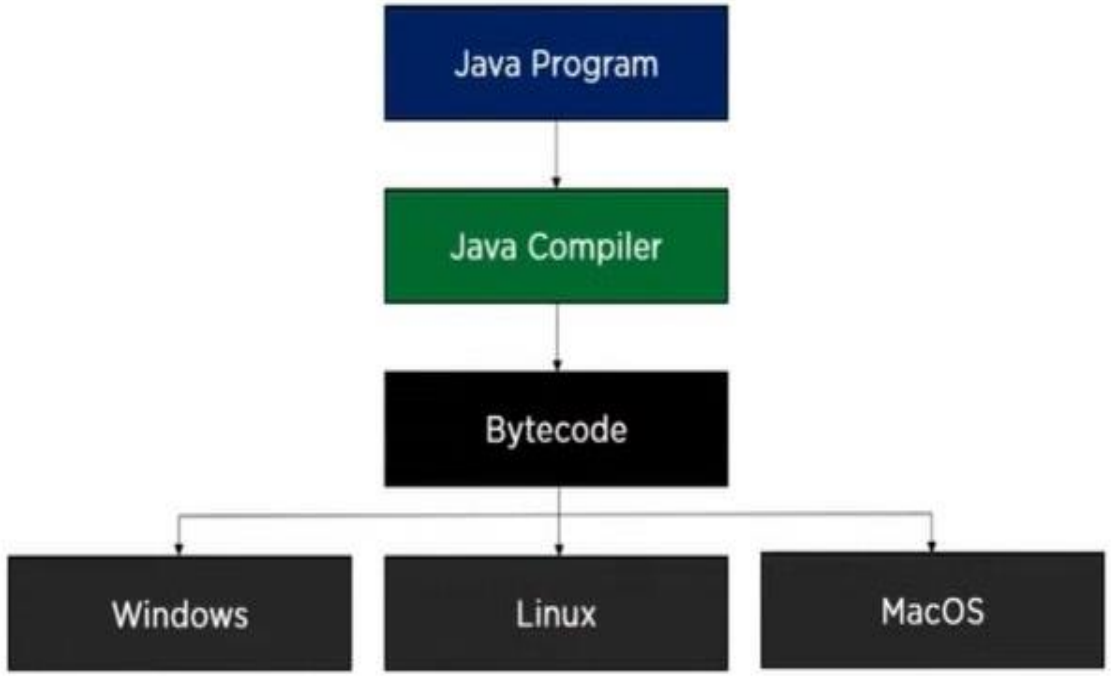
Java is a platform independent programming language that follows the logic of **"Write once, Run anywhere"**

Java can be used to create complete applications that may run on a single computer or can be distributed among servers and clients in a network

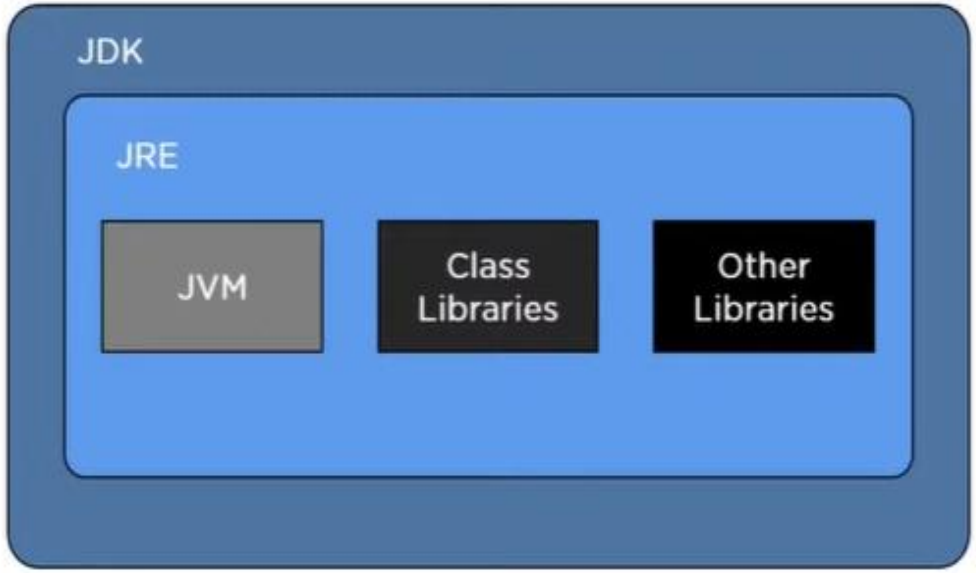
Features of Java



How a Java Program is Run



Anatomy of Java



Difference between JDK, JRE, and JVM

JVM

It is a specification that provides a runtime environment in which Java bytecode can be executed.

The JVM performs the following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment
- It is not physically exists.

JRE

- The Java Runtime Environment is a set of software tools which are used for developing Java applications.
- It is used to provide the runtime environment.
- It is the implementation of JVM.

JDK

- JDK is an instance for Java Development Kit.
- The Java Development Kit (JDK) is a software development environment
- which is used to develop Java applications.
- It physically exists.

Memory Management in Java

Method Area :-

- Method Area is a part of the heap memory which is shared among all the threads.
- It creates when the JVM starts up.
- **It is used to store class structure, superclass name, interface name, and constructors.**

Heap Area :-

- Heap stores the actual objects.
- It creates when the JVM starts up.
- When you use a new keyword, the JVM creates an instance for the object in a heap.

Stack Area :-

- Stack Memory in Java is used for static memory allocation and the execution of a thread.

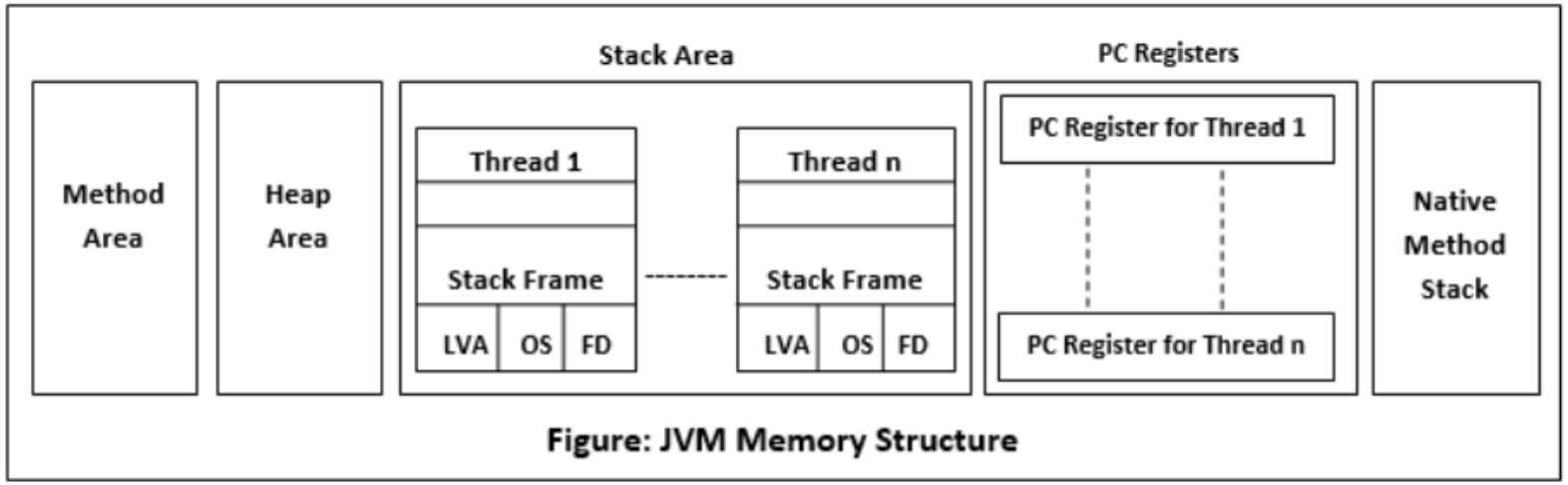
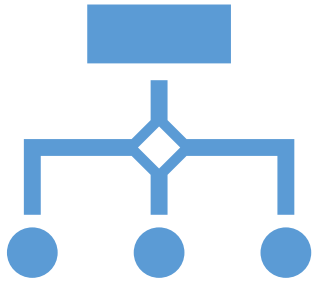


Figure: JVM Memory Structure

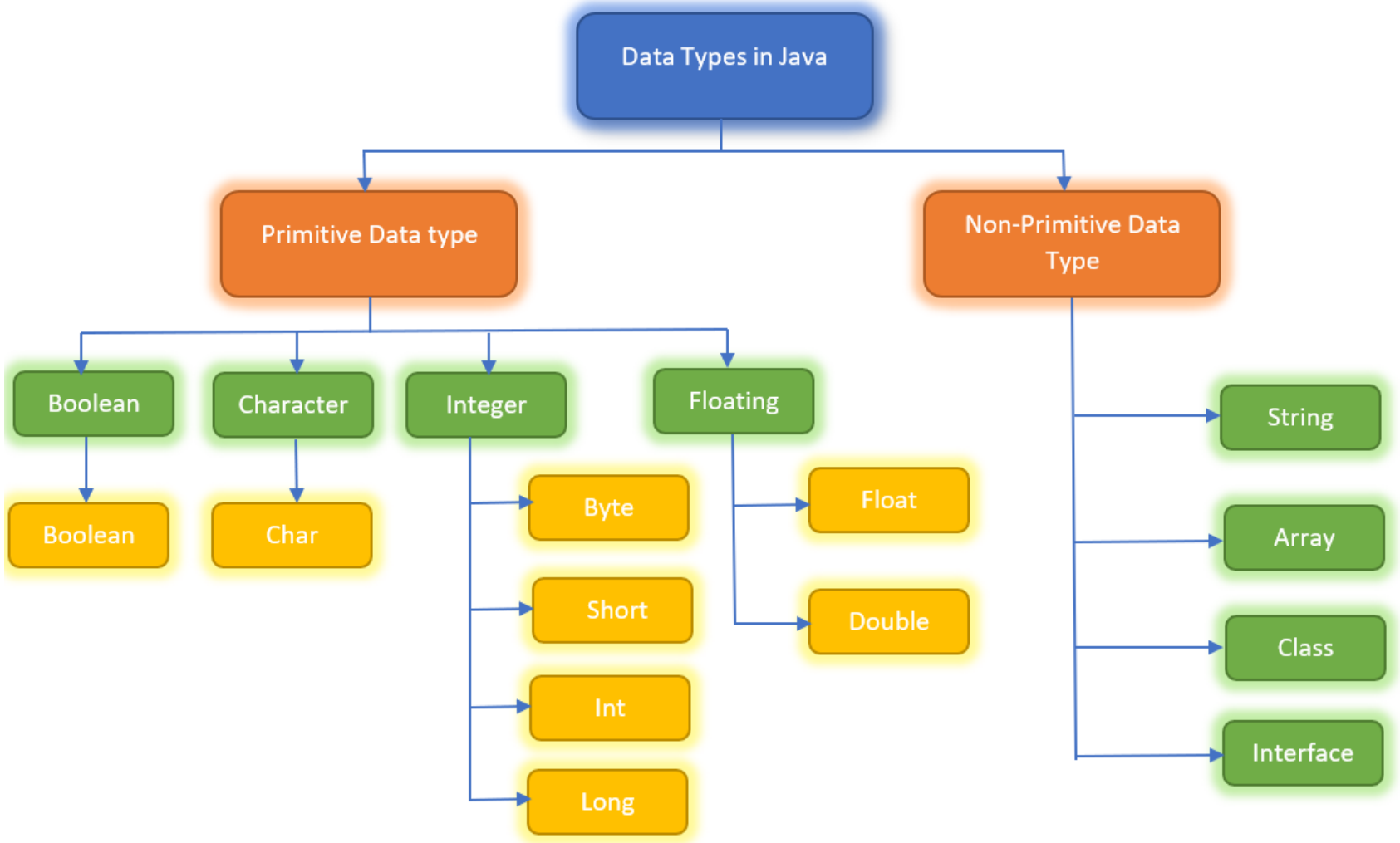
Compile time vs Runtime



Compile-time is the time at which the source code is converted into an executable code.

while the run time is the time at which the executable code is started running.

Data Types In JAVA

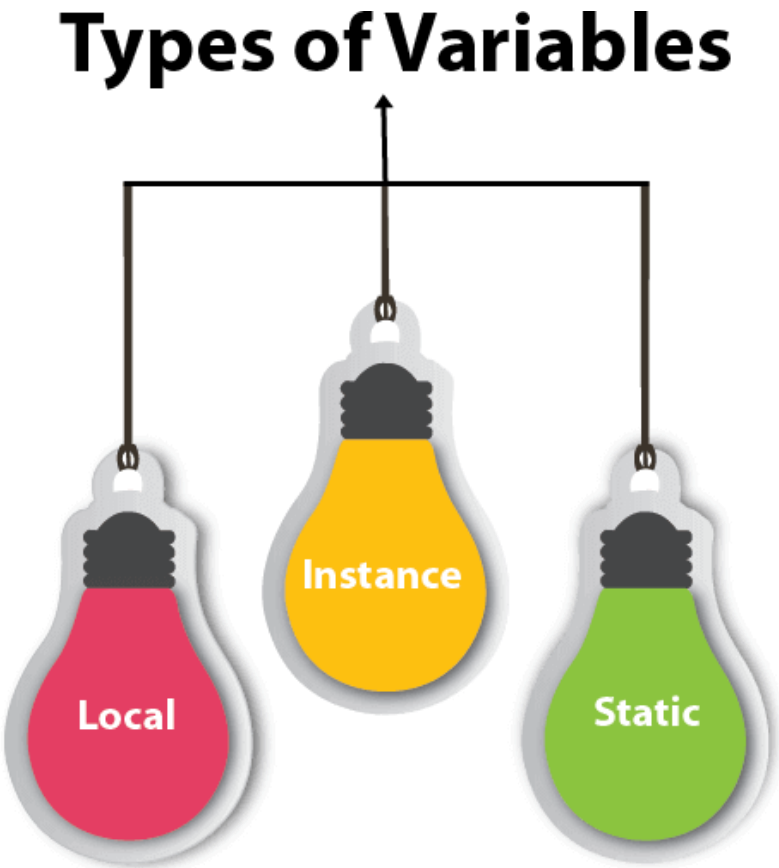


Data Types In JAVA

```
public class City {
    private int id;          //Premetive Data Type
    private String name;   // Reference Data type
    private State state; // Reference Data type
}

public class State {
    private int id;
    private String name;   // Reference Data type
}
```

Java Variables

- 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.



Java Variables

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "**static**" keyword.

2) Instance Variable // Object Creation

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

It is not declared as static.

3) Static variable //

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class.

Memory allocation for static variables happens only once when the class is loaded in the memory.

Java Variables Example:

```java
package com.test;

public class VariableTest {

  private  int num ; // instance variable , object label variable
  private static int  num1;//// static variable , class label variable

    public static void main(String[] args) {

        VariableTest variableTest=new VariableTest();//crate the instance of the oblect
        isPrimeNumber(10);
        VariableTest.num1=10;
        variableTest.num=10;

        System.out.println("instance value "+variableTest.num +"static value"+variableTest.num1);  // 10 , 10

        VariableTest variableTest1=new VariableTest();//

        VariableTest.num1=20;
        variableTest1.num=20;

        System.out.println("instance value "+variableTest1.num +"static value"+variableTest1.num1); // 20 ,10  // 20 20

        VariableTest variableTest2=new VariableTest();//

        VariableTest.num1=30;
        variableTest2.num=30;

        System.out.println("instance value "+variableTest2.num +"static value"+variableTest2.num1); // 30, 10  // 30 30


        System.out.println("instance value "+variableTest.num +"static value"+variableTest.num1);  // 10 ,30
        System.out.println("instance value "+variableTest1.num +"static value"+variableTest1.num1); //20 ,30
```

Java Keywords

List of Java Keywords

Primitive Types and void	Modifiers	Declarations	Control Flow	Miscellaneous
1. boolean	1. public	1. class	1. if	1. this
2. byte	2. protected	2. interface	2. else	2. new
3. char	3. private	3. enum	3. try	3. super
4. short	4. abstract	4. extends	4. catch	4. import
5. int	5. static	5. implements	5. finally	5. instanceof
6. long	6. final	6. package	6. do	6. null
7. float	7. transient	7. throws	7. while	7. true
8. double	8. volatile		8. for	8. false
9. void	9. synchronized		9. continue	9. strictfp
	10. native		10. break	10. assert
			11. switch	11. _ (underscore)
			12. case	12. goto
			13. default	13. const
			14. throw	
			15. return	

JAVA KeyWords

- **abstract**: Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
- **boolean:** Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
- **break**: Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.
- **byte**: Java byte keyword is used to declare a variable that can hold 8-bit data values.
- **case**: Java case keyword is used with the switch statements to mark blocks of text.
- **catch**: Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
- **char**: Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
- **class**: Java class keyword is used to declare a class.
- **continue**: Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.

JAVA KeyWords

- **default:** Java default keyword is used to specify the default block of code in a switch statement.
- **do:** Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.
- **double:** Java double keyword is used to declare a variable that can hold 64-bit floating-point number.
- **else:** Java else keyword is used to indicate the alternative branches in an if statement.
- **enum:** Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
- **extends:** Java extends keyword is used to indicate that a class is derived from another class or interface.
- **final:** use final keyword – variable, method, Class
- **finally:** Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.
- **float:** Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
- **for:** Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.
- **if:** Java if keyword tests the condition. It executes the if block if the condition is true.

JAVA KeyWords

- **implements:** Java implements keyword is used to implement an interface.
- **import:** Java import keyword makes classes and interfaces available and accessible to the current source code.
- **instanceof:** Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
- **int:** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
- **interface:** Java interface keyword is used to declare an interface. It can have only abstract methods.
- **long:** Java long keyword is used to declare a variable that can hold a 64-bit integer.
- **native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
- **new:** Java new keyword is used to create new objects.
- **null:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
- **package:** Java package keyword is used to declare a Java package that includes the classes.
- **synchronized:** Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.

JAVA KeyWords

- **private**: Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
- **protected**: Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.
- **public**: Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
- **return**: Java return keyword is used to return from a method when its execution is complete.
- **short**: Java short keyword is used to declare a variable that can hold a 16-bit integer.
- **static**: Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.
- **strictfp**: Java strictfp is used to restrict the floating-point calculations to ensure portability.
- **super**: Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.
- **switch**: The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.

JAVA KeyWords

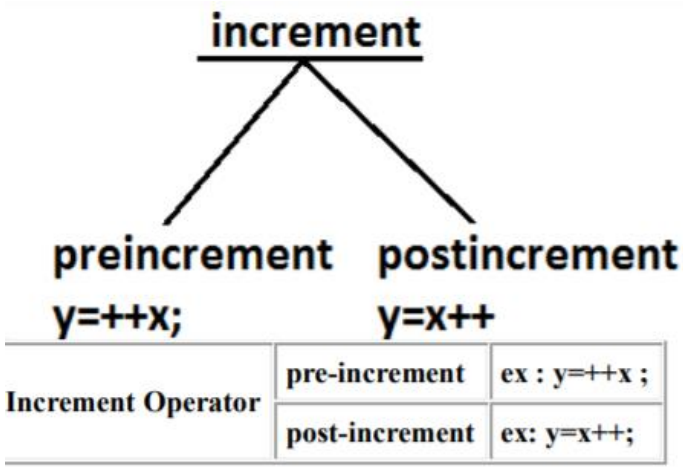
- **throws:** The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.
- **transient:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
- **try:** Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
- **void :** Java void keyword is used to specify that a method does not have a return value.
- **volatile:** Java volatile keyword is used to indicate that a variable may change asynchronously.
- **while:** Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.

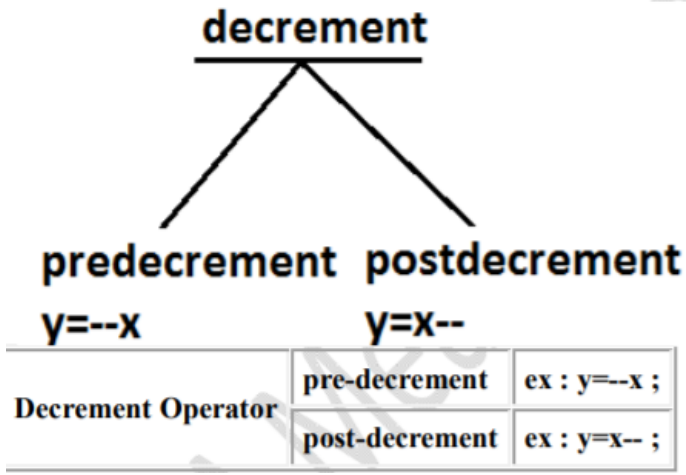
Java Operators

1. increment & decrement operators
2. arithmetic operators
3. string concatenation operators
4. Relational operators
5. Equality operators
6. instanceof operators
7. Bitwise operators
8. assignment operator
9. new operator
10. [ ] operator
11. Java Ternary Operator
12. new Vs newInstance( )

1. increment & decrement operators

```
private static void getIncDecOperator() {
        int x=10;
        System.out.println(x++);//10 (11)
        System.out.println(++x);//12
        System.out.println(x--);//12 (11)
        System.out.println(--x);//10
}
```





2. String Concatenation operator

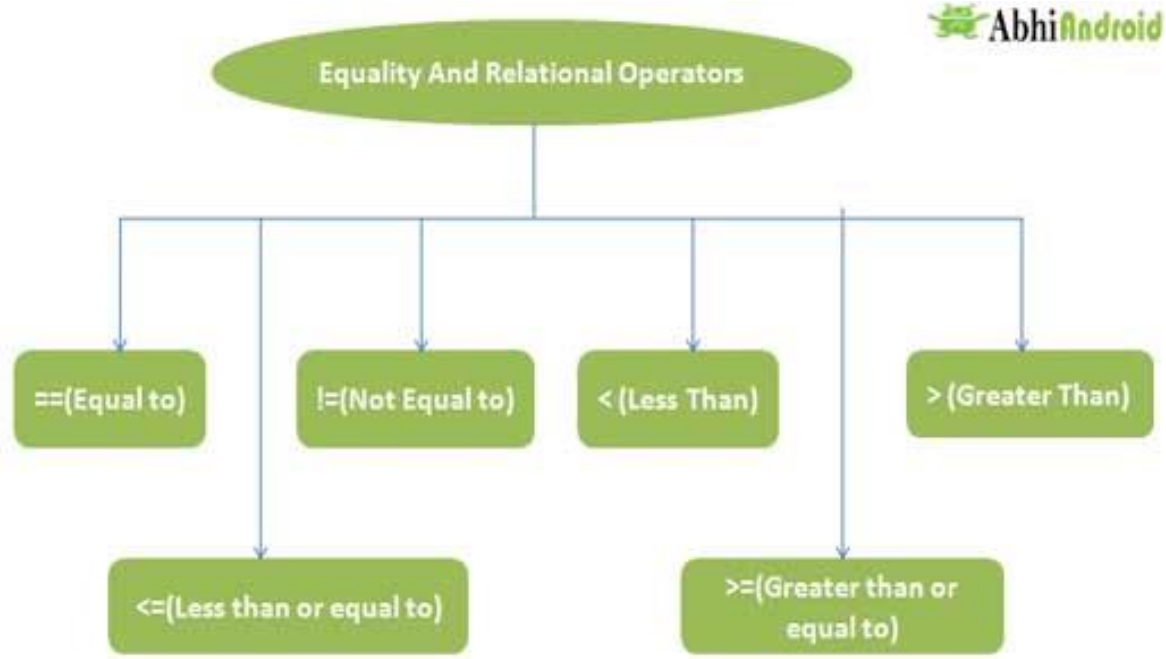
- The only overloaded operator in java is ' + ' operator some times it access arithmetic addition operator & some times it access String concatenation operator.
- If acts as one argument is String type , then '+' operator acts as concatenation and If both arguments are number type , then operator acts as arithmetic operator

```
public static void getStringConcatation(){
    String a="ashok";
    int b=10 , c=20 , d=30 ;
    System.out.println(a+b+c+d);  // output : ashok102030
    System.out.println(b+c+a+d);  // output : 30ashok30
}
```

3. Relational Operators and Equality Operators

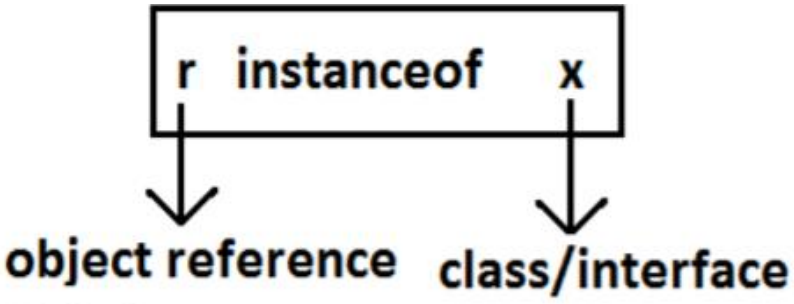
- We can apply relational operators for every primitive type except boolean .

```
public static void getEqualityAndRelational(){
    int x = 12, y = 11;

    System.out.println("x is " + x + " and y is " + y);
    System.out.println(x == y);  // false  //equal
    System.out.println(x != y);  // true   //not equal
    System.out.println(x > y);  // false
    System.out.println(x < y);  // true
    System.out.println(x >= y);  // false
    System.out.println(x <= y);  // true
}
```



4. instanceof operator

- We can use the instanceof operator to check whether the given an object is perticular type or not.




```
class Animal {}
class Mammal extends Animal {}

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

    Mammal mobj = new Mammal();
    Animal aobj = new Animal();

        if (mobj instanceof Mammal)
          System.out.println("mobj is instance of Mammal");
        else
          System.out.println("mobj is NOT instance of Mammal");

        if (mobj instanceof Animal)
          System.out.println("mobj is instance of Animal");
        else
          System.out.println("mobj is NOT instance of Animal");

        if (aobj instanceof Mammal)
           System.out.println("mobj is instance of Animal");
        else
           System.out.println("mobj is NOT instance of Animal");
        }
    }
```

5. Short-circuit AND Bitwise Operator

```
public static void bitWiseAndShortCircuit(){

// & operator
    System.out.println("a&b = " + (5 & 7));
// | operator
    System.out.println("a|b = " + (5 | 7));
// ^ operator
    System.out.println("a^b = " + (5 ^ 7));
// && operator
    System.out.println((5 > 3) && (8 > 5));   // true
    System.out.println((5 > 3) && (8 < 5));   // false
// || operator
    System.out.println((5 < 3) || (8 > 5));   // true
    System.out.println((5 > 3) || (8 < 5));   // true
    System.out.println((5 < 3) || (8 < 5));   // false
// ! operator
    System.out.println(!(5 == 3));   // true
    System.out.println(!(5 > 3));   // false
}
```

& , |	&& , ||
Both arguments should be evaluated always.	Second argument evaluation is optional.
Relatively performance is low.	Relatively performance is high.
Applicable for both integral and boolean types.	Applicable only for boolean types but not for integral types.

6. Assignment Operator

There are 3 types of assignment operators

1. Simple assignment:

Example:

```
int x=10;
```

2. Chained assignment:

Example:

```
int a,b,c,d;
a=b=c=d=20;
System.out.println(a+"---"+b+"---"+c+"---"+d);   //20---20---20---20
```

3. Compound assignment:

Sometimes we can mixed assignment operator with some other operator to form compound assignment operator.

Example:

```
int a=10 ;
a +=20 ;
System.out.println(a); //30
```

7. Java Ternary Operator

The ternary operator (conditional operator) is shorthand for the if-then-else statement.

For example,

variable = Expression ? expression1 : expression2

Here's how it works :-

- If the Expression is true, expression1 is assigned to the variable.
- If the Expression is false, expression2 is assigned to the variable.

```
public static void Ternary(){
    System.out.println("Ternary Operator");
    int februaryDays = 28;
    String result;

    // ternary operator
    result = (februaryDays == 28) ? "Not a leap year" : "Leap year";
    System.out.println(result);
```

8. [ ] operator

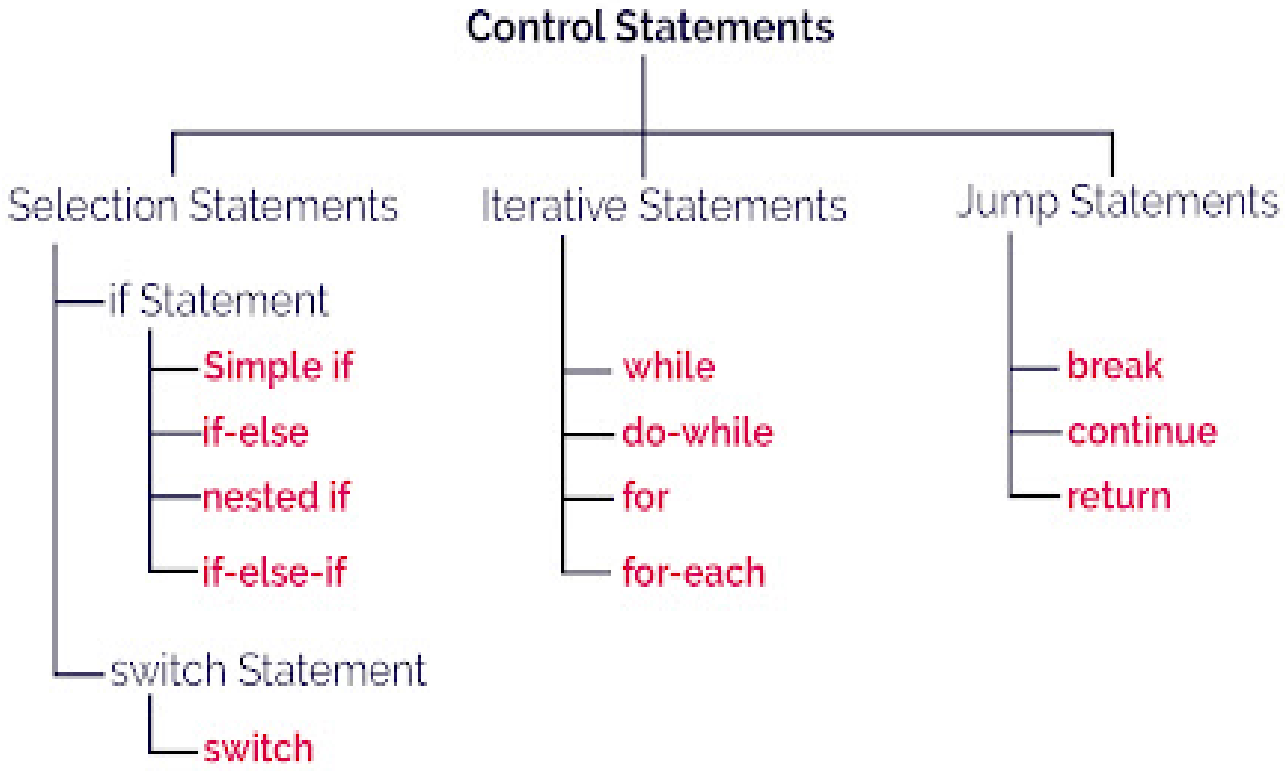
We can use this operator to declare under construct/create arrays.

- **Arrays** are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- Array fixed in Size
- Array Homogenous Data

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

```
public static void arrayOpe(){
    System.out.println("array Operator");

    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
    System.out.println(cars[0]);
}
```

Control Flow in Java



Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Decision-Making statements

- **1) Simple if statement:**

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

```
public static  void ifStatement(){
    int x = 10;
    int y = 12;
    if(x+y > 20) {
        System.out.println("x + y is greater than 20")
    }
}
```

2) if else statement:

➢ The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block.

➢ The else block is executed if the condition of the if-block is evaluated as false.

```
public static  void ifElseStatement(){
    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");
    }
}
```

3) if-else-if ladder

- The **if-else-if** statement contains the if-statement followed by multiple else-if statements. We can also define an else statement at the end of the chain.

```
public static  void ifElseIfStatement(){

    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);
    }
}
```

4) Nested if-statement

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

```
public static void nestedIfStatement() {
    System.out.println("nestedIfStatement");
    String address = "Delhi, India";

    if (address.endsWith("India")) {
        if (address.contains("Meerut")) {
            System.out.println("Your city is Meerut");
        } else if (address.contains("Noida")) {
            System.out.println("Your city is Noida");
        } else {
            System.out.println(address.split(",")[1]);
        }
    } else {
        System.out.println("You are not living in India");
    }
}
```

5) Switch Statement:

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

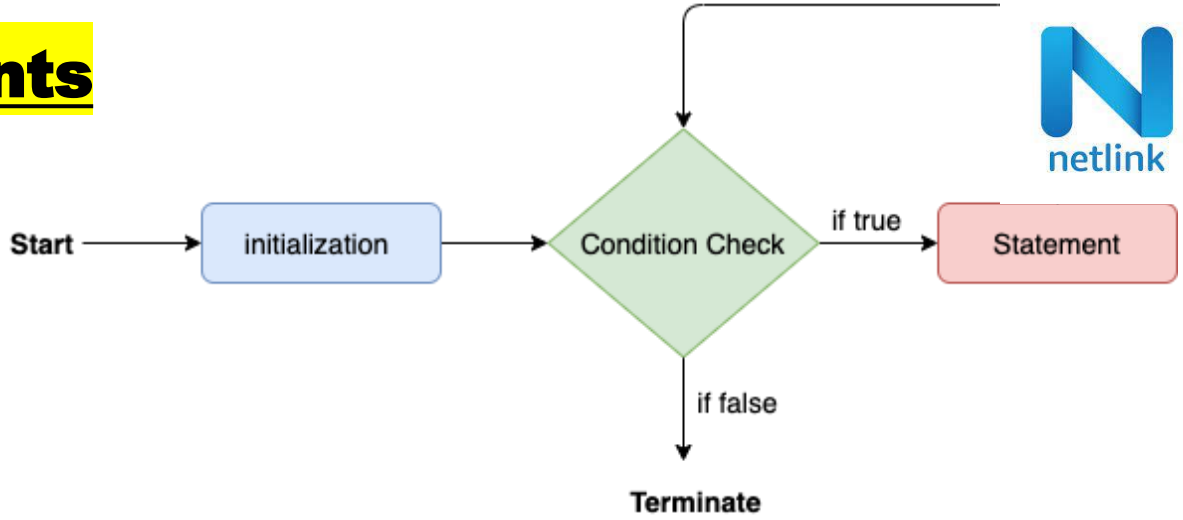
- The switch statement is easier to use instead of if-else-if statements.
- It also enhances the readability of the program.

```
public static void switchStatemenyt(){
    int num = 2;
    switch (num){
        case 0:
            System.out.println("number is 0");
            break;
        case 2:
            System.out.println("number is 2");
            break;
        default:
            System.out.println(num);
    }
}
```

Loop statements

1) for loop

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




```
public static void forLoop(){
    int sum = 0;
       for(int j = 1; j<=10; j++) {
          sum = sum + j;
       }
     System.out.println("The sum of first ten natural numbers is " + sum);
}
```

Click to add text

2) for-each loop

Java provides an enhanced for loop to traverse the data structures like array or collection. In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

```
public  static void forEachLoop(){
    String[] names = {"Java","C","C++","Python","JavaScript"};

    System.out.println("Printing the content of the array names:\n");
    for(String name:names) {
        System.out.println(name);
    }
}
```

3) while loop

The while loop loops through a block of code as long as a specified condition is true:

```
public static  void  whileLoop(){
    int i = 0;
    while (i < 5) {
        System.out.println(i);
        i++;
    }
}
```

4) Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
public static void doWhileLoop(){
    int i = 0;
    do {
        System.out.println(i);
        i++;
    }
    while (i < 10);
}
```

Brake statements

1) Brake

It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a **loop**.

```
public static void BrakeStatement(){
    for (int i = 0; i < 10; i++) {
        if (i == 4) {
            break;
        }
        System.out.println(i);
    }
}
```

2) Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

```
public static void continueStatement(){
    for (int i = 0; i < 10; i++) {
        if (i == 4) {
            continue;
        }
        System.out.println(i);
    }
}
```

Declaration and Access Modifiers

Agenda

- **Java source file structure**
  - **o Package**
  - **o Import statement**
- **Class Modifiers**
  1. Public
  2. Default
  3. Final
  4. Abstract
  5. Strictfp

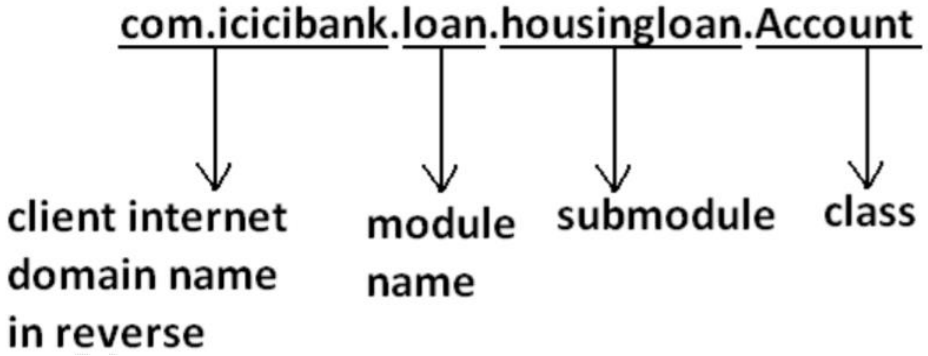
  1. Private
  2. Protected
  3. Static
- **Member modifiers**
- **Interfaces**

Package :

It is an encapsulation mechanism to group related classes and interfaces into a single module.

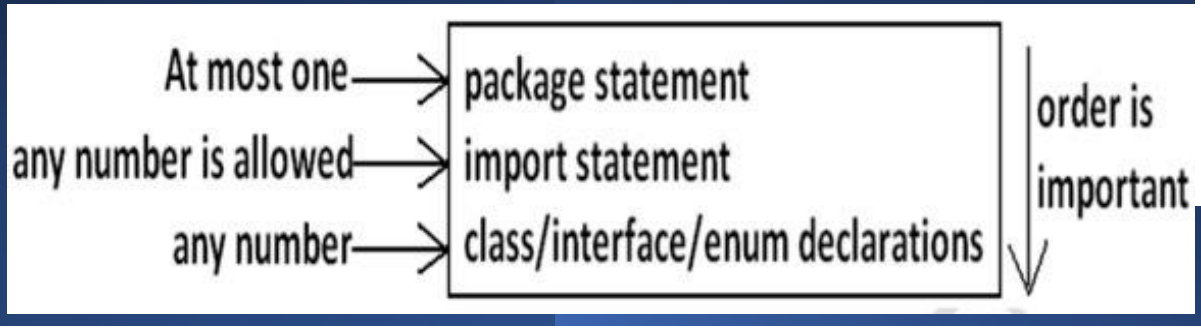
The main objectives of packages are:

- To resolve name confects.
- To improve modularity of the application.
- To provide security.
- There is one universally accepted naming conversion for packages that is to use internet domain name in reverse.




```
package JavaExample.com.durgajobs.itjobs;

public class Test {

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

Import statement:



- In Java, the **import** statement is used to bring certain classes or the entire packages, into visibility. As soon as imported, a class can be referred to directly by using only its name.
- whenever we are using import statement it is not require to use fully qualified names, we can use short names directly.
- This approach decreases length of the code and improves readability.
- "Import statement is totally compile time concept" if more no of imports are there then more will be the compile time but there is "no change in execution time".

```
import java.util.ArrayList;

class Import{
    public static void main(String args[]){
        ArrayList l1=new ArrayList();
        java.util.ArrayList l2 =new java.util.ArrayList(); // Fully qualified name
    }
}
```

- **Types of Import Statements:**

1) Explicit class import
2) Implicit class import.

Explicit class import:

Example: Import java.util.ArrayList;

✓ This type of import is **highly recommende**d to use because it improves readability of the code.

Implicit class import:

Example: import java.util.*;

✓ It is never recommended to use because it reduces readability of the code.

Static Import statement: 1.5v

- ✓ **Static import:** This concept introduced in 1.5 versions.
- ✓ we can access static members by using class name but whenever we are using static import it is not require to use class name we can access directly.

```
public class WithoutStaticImport {

    public static void main(String args[]){
        System.out.println(Math.sqrt(4));
        System.out.println(Math.max(10,20));
        System.out.println(Math.random());
    }
}
```

```
import static java.lang.Math.sqrt;
import static java.lang.Math.*;

public class WithStaticImport {

    public static void main(String args[]){
        System.out.println(sqrt(4));
        System.out.println(max(10,20));
        System.out.println(random());
    }
}
```

Class Modifiers :

Whenever we are writing our own classes compulsory, we have to provide some information about our class to the jvm.

Like

- Whether this class can be accessible from anywhere or not.
- Whether child class creation is possible or not.
- Whether object creation is possible or not etc.

The only applicable modifiers for Top Level classes are:

1. Public
2. Default
3. Final
4. Abstract
5. Strictfp

If we are using any other modifier we will get compile time error.

Public Modifier:

- If a class declared as public then we can access that class from anywhere.
- Within the package or outside the package.

- If class Test is not public then while compiling

Test1 class we will get compile time error saying pack1.Test is not public in pack1;

cannot be accessed from outside package.

```
package JavaExample.PublicClassModifier.Pack1;

public class Test {

    public void methodOne(){
        System.out.println("test class methodone is executed");
    }
}

package JavaExample.PublicClassModifier.Pack2;

import JavaExample.PublicClassModifier.Pack1.Test;

public class Test1 {
    public static void main(String args[]) {
        Test t = new Test();
        t.methodOne();
    }
}
```

Default Modifier:

- If a class declared as the default, then we can access that class only within the current package hence default access is also known as "**package level access**".
- It provides more accessibility than private. But it is more restrictive than protected, and public.

```
package JavaExample.DefaultClassModifier.Pack1;

class Test {
    public void methodOne() {
        System.out.println("test class method one is executed");
    }
}

package JavaExample.DefaultClassModifier.Pack1;

class Test1 {
        public static void main(String args[]){
            Test t=new Test();
            t.methodOne();
        }
}
```

Final Modifier:

- Final is the modifier applicable for classes, methods and variables.

Final Methods:

- Whatever the methods parent has by default available to the child.
- If the child is not allowed to override any method, that method we have to declare with final in parent class.

That is final methods cannot be overridden.

```
class Bike{
    final void run(){System.out.println("running");}
}
class Honda extends Bike {
    void run() {
        System.out.println("running safely with 100kmph");
    }
}
class FinalMethod {
    public static void main(String args[]) {
        Honda honda = new Honda();
        honda.run();
    }
}
// Output:Compile Time Error
```

Final Modifier:

Final Class :-

- If a class declared as the final, then we can't create the child class that is ***inheritance concept is not applicable*** for final classes.

```
final class Bike{
     void run(){System.out.println("running");}
}
class Honda extends Bike {
    void run() {
        System.out.println("running safely with 100kmph");
    }
}
class FinalMethod {
     public static void main(String args[]) {
         Honda honda = new Honda();
         honda.run();
     }
}
// Output:Compile Time Error
```

Final Variable :-

- If you make any variable as final, **you cannot change the value of final variable(It will be constant).**

```
public class FinalVariable {
    final int speedlimit=90;//final variable
    void run(){
        speedlimit=400;
    }
    public static void main(String args[]){
        FinalVariable obj=new  FinalVariable();
        obj.run();
    }
}
// Output:Compile Time Error
```

Abstract Modifier:

Abstract is the modifier applicable only for **methods and classes** but not for variables.

Abstract Methods:

public abstract void methodOne(); ⟶ valid

public abstract void methodOne(){} ⟶ invalid

- Even though we don't have implementation still we can declare a method with abstract modifier.
- **That is abstract methods have only declaration but not implementation.**
- Hence abstract method declaration should compulsorily end with semicolon";"

- Child classes are responsible to provide implementation for parent class abstract methods.

Abstract Modifier:

Abstract Class : -

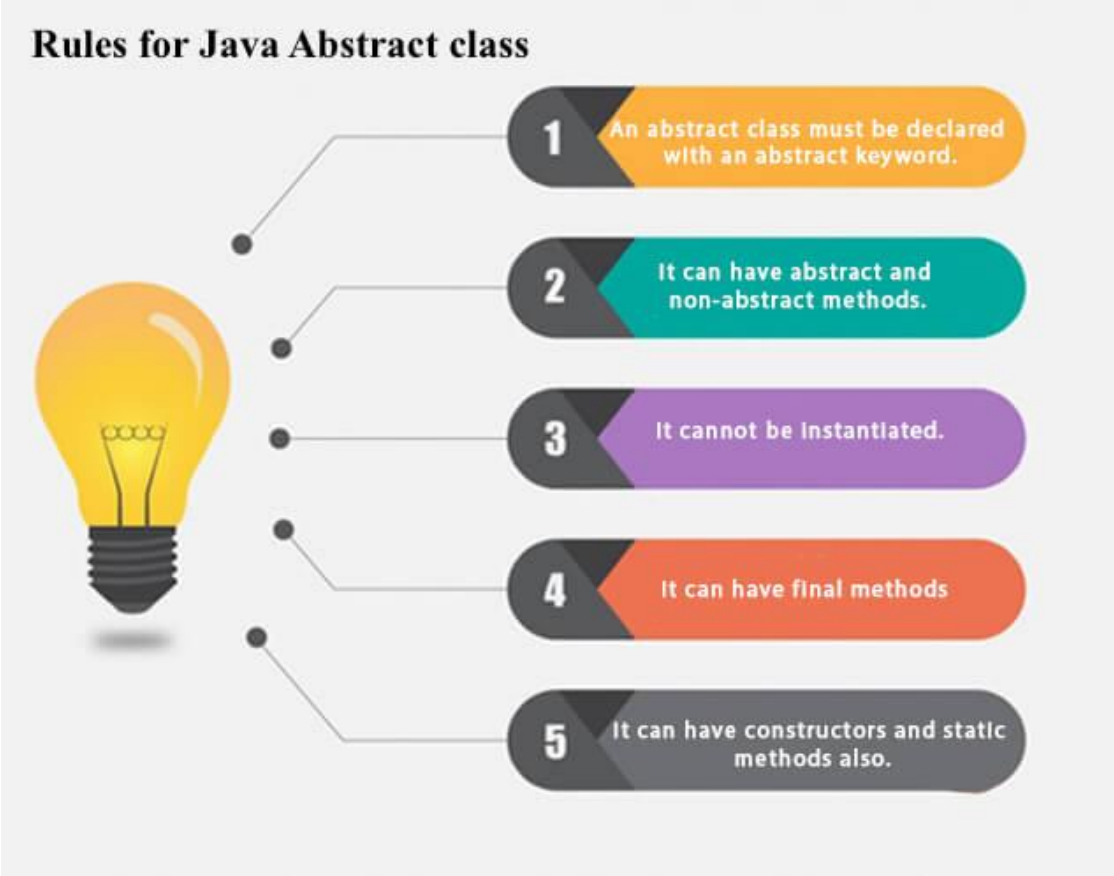
A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

- Child classes are responsible to provide implementation for parent class abstract methods.

```
abstract class vehicle
{
public abstract int getNoOfWheels();
}
```
vehicle.java

```
class Bus extends vehicle
{
public int getNoOfWheels(){
return 7;
}
}
```
Bus.java

```
class Auto extends vehicle
{
public int getNoOfWheels(){
return 3;
}
}
```
Auto.java



Member modifiers:

Public members:

- If a member declared as the public then we can access that member from anywhere "but the corresponding class must be visible" hence before checking member visibility we have to check class visibility.

Default member:

- If a member declared as the default then we can access that member **only within the current package** hence default member is also known **as package level access.**

Member modifiers:

Private members :

- If a member declared as the private, then we can access that member only with in the current class.
- Private methods are not visible in child classes *whereas abstract methods should be visible in child classes to provide implementation hence private, abstract combination is illegal for methods.*

Protected members:

- If a member declared as the protected, then we can access that member within the **current package anywhere but outside package only in child classes.**
- We can access protected members within the current package anywhere either by child reference or by parent reference
- But from outside package we can access protected members only in child classes and should be by child reference only that is we can't use parent reference to call protected members from outside package.

Compression of private, default, protected and public:

visibility	private	default	protected	public
1)With in the same class	✓	✓	✓	✓
2)From child class of same package	✗	✓	✓	✓
3)From non-child class of same package	✗	✓	✓	✓
4)From child class of outside package	✗	✗	✓ but we should use child reference only	✓
5)From non-child class of outside package	✗	✗	✗	✓

Static modifier:

- Static is the modifier applicable for methods, variables and blocks.
- We can't declare a class with static but inner classes can be declaring as the static.
- In the case of instance variables for every object a separate copy will be created but in the case of static variables a single copy will be created at class level and shared by all objects of that class..

Native modifier:

- Native is a modifier applicable only for methods but not for variables and classes.
- The methods which are implemented in non-java are called native methods or foreign methods.

The main objectives of native keyword are:

- To improve performance of the system.
- To use already existing legacy non-java code.
- To achieve machine level communication(memory level - address)
- Pseudo code to use native keyword in java.

Synchronized modifier:

- Synchronized is the modifier applicable for methods and blocks but not for variables and classes.
- If a method or block declared with synchronized keyword, then at a time only one thread is allow to execute that method or block on the given object.
- The main advantage of synchronized keyword is **we can resolve data inconsistency problems.** But the main disadvantage is it increases waiting time of the threads and effects performance of the system.

Hence if there is no specific requirement never recommended to use synchronized keyword. For syncronized methods compulsory implementation should be available , but for abstract methods implementation won't be available , Hence abstract - syncronized combination is illegal for methods.

Interface in Java

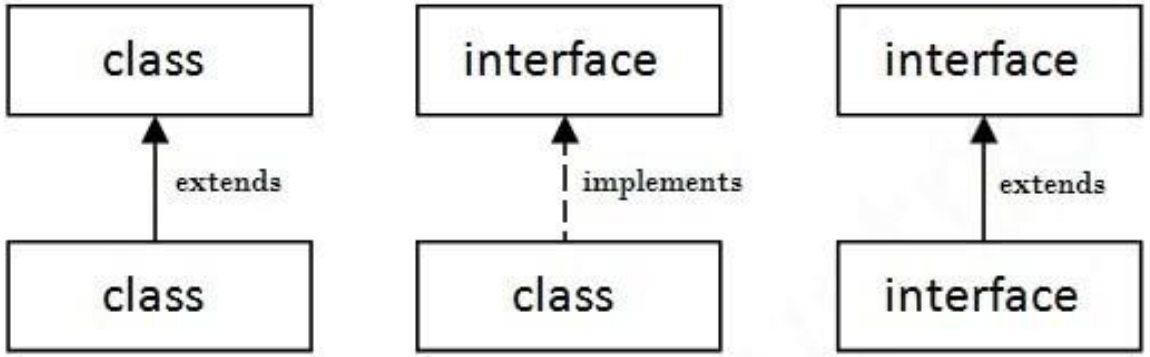
- An **interface in Java** is a blueprint of a class. It has abstract methods.
- It is used to achieve abstraction and multiple inheritance in Java.
- An interface is declared by using the interface keyword.
- It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.

Syntax:

```
interface <interface_name>{

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

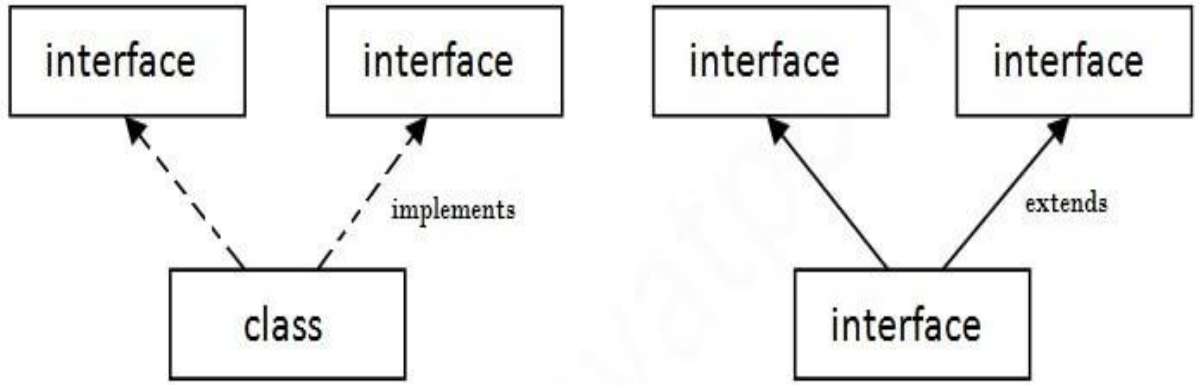
The relationship between classes and interface

```
package JavaExample.Interface;

public interface printable {
    void print();
}
class A6 implements printable{
    public void print(){System.out.println("Hello");}

    public static void main(String args[]){
        A6 obj = new A6();
        obj.print();
    }
}
```



Multiple inheritance in Java by interface

- If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

```java
interface Printable {
    void print();
}

interface Showable {
    void show();
}

class Test implements Printable, Showable {

    public void print() {
        System.out.println("Hello");
    }

    public void show() {
        System.out.println("Welcome");
    }

    public static void main(String args[]) {
        Test obj = new Test();
        obj.print();
        obj.show();
    }
}
```

Java 8 interface

- **Java 8 Default Method in Interface :**

```
interface Drawable {
    void draw();

    default void msg() {
        System.out.println("default method");
    }
}
class Rectangle implements Drawable {
    public void draw() {
        System.out.println("drawing rectangle");
    }
}
class DefaultInterfaceMethod {
    public static void main(String args[]) {
        Drawable d = new Rectangle();
        d.draw();
        d.msg();
    }
}
```

- **Java 8 Static Method in Interface :**

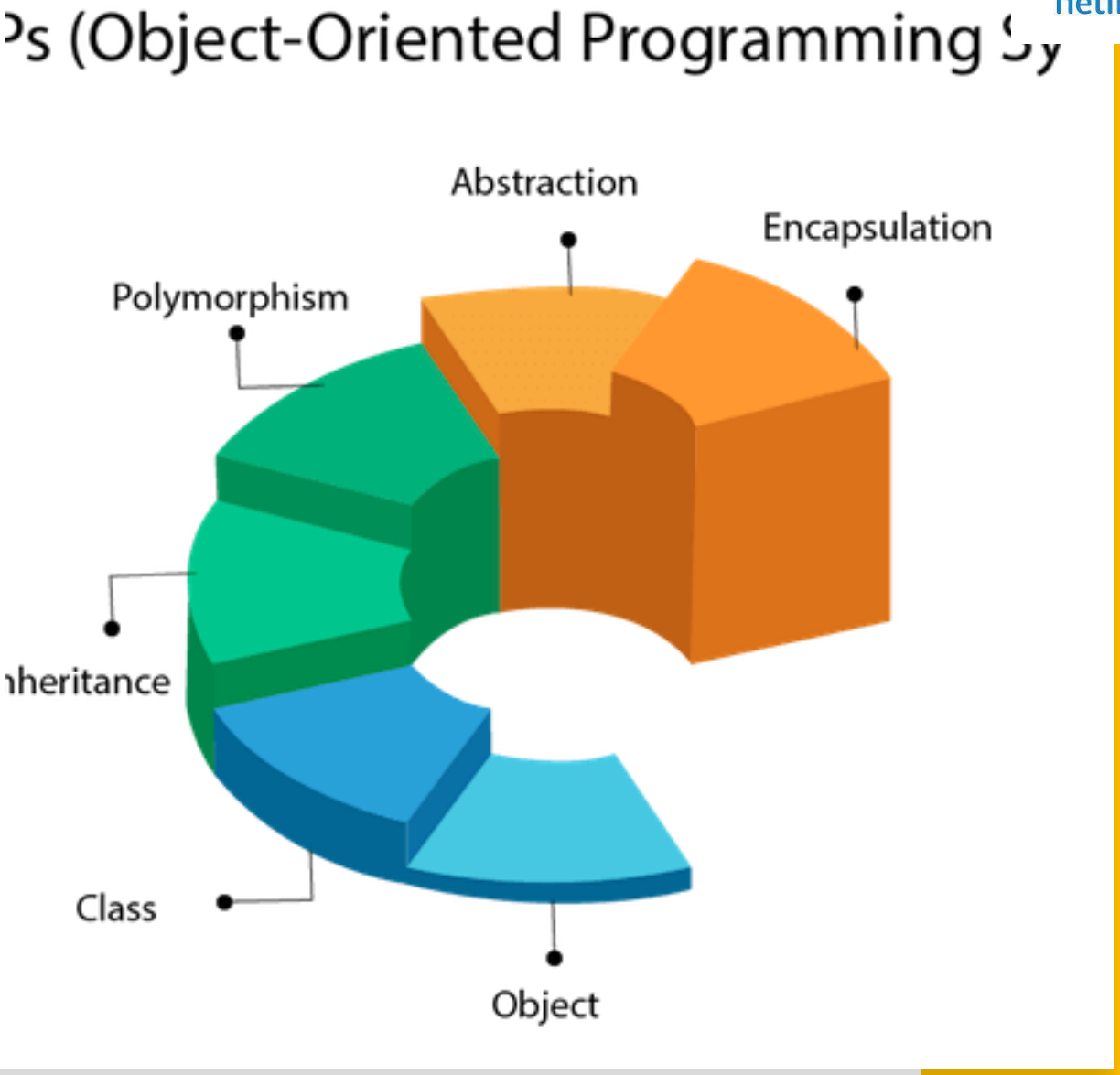
```
interface Drawable {
    void draw();

    static int cube(int x) {
        return x * x * x;
    }
}

class Rectangle implements Drawable {
    public void draw() {
        System.out.println("drawing rectangle");
    }
}

class TestInterfaceStatic {
    public static void main(String args[]) {
        Drawable d = new Rectangle();
        d.draw();
        System.out.println(Drawable.cube(3));
    }
}
```

OOP's Concepts

- **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.

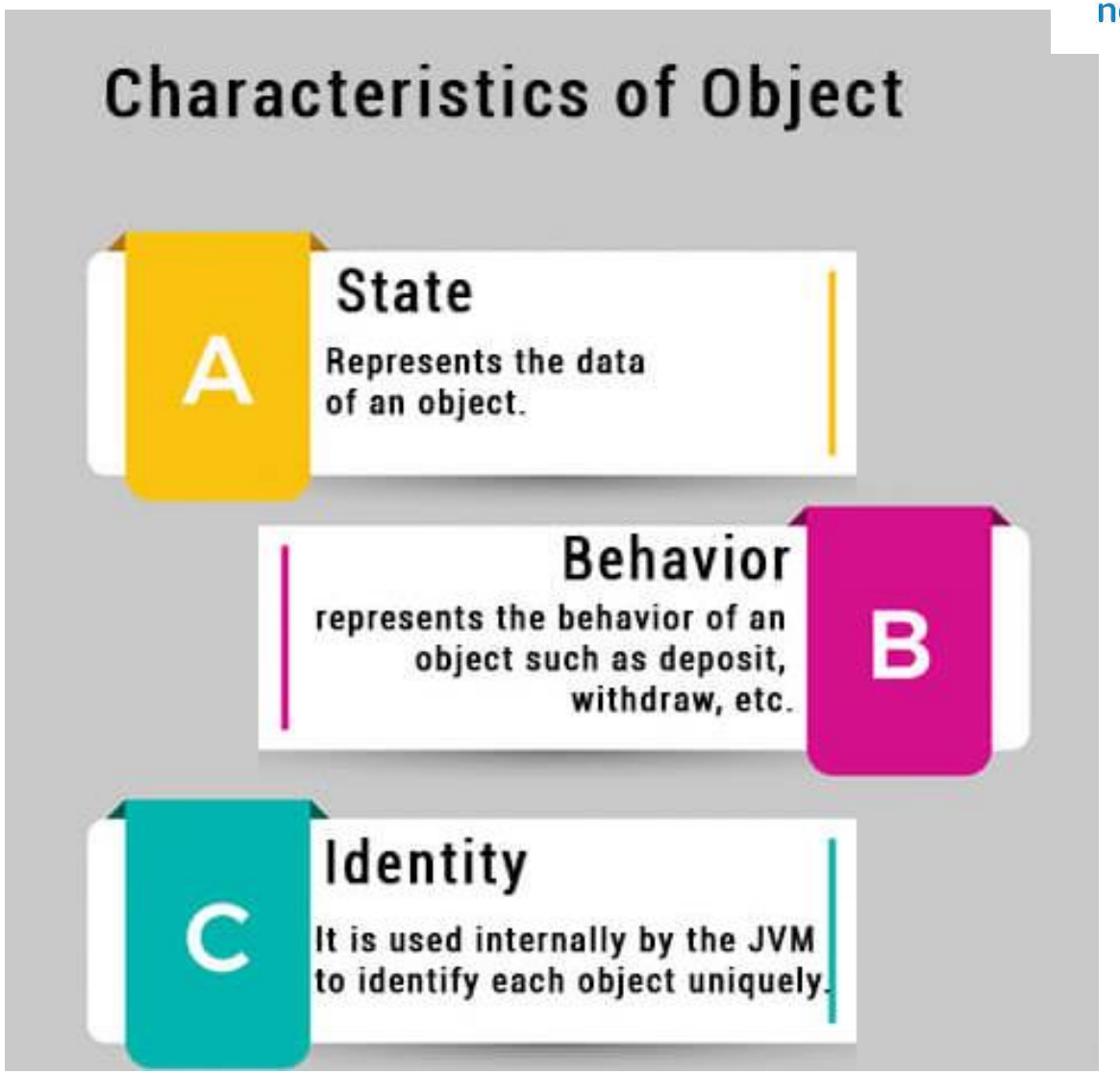
It simplifies software development and maintenance by providing some concepts:

- ✓ Object
- ✓ **Class**
- ✓ Inheritance
- ✓ Polymorphism
- ✓ Abstraction
- ✓ Encapsulation



Object In JAVA

- Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.
- An Object can be defined as an instance of a class.
- Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

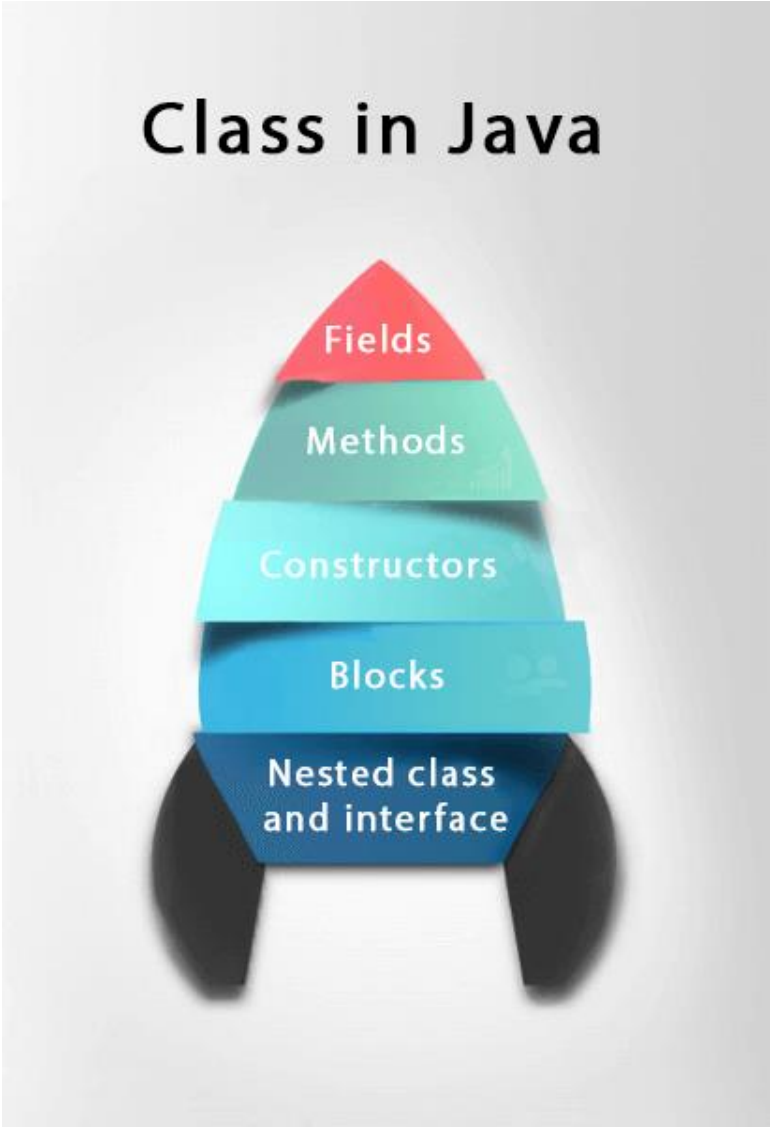


Class In JAVA

- *Collection of / Group of objects* is called class. It is a logical entity.
- A class can also be defined as a blueprint from which you can create an individual object. **Class doesn't consume any space.**

A class in Java can contain:

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



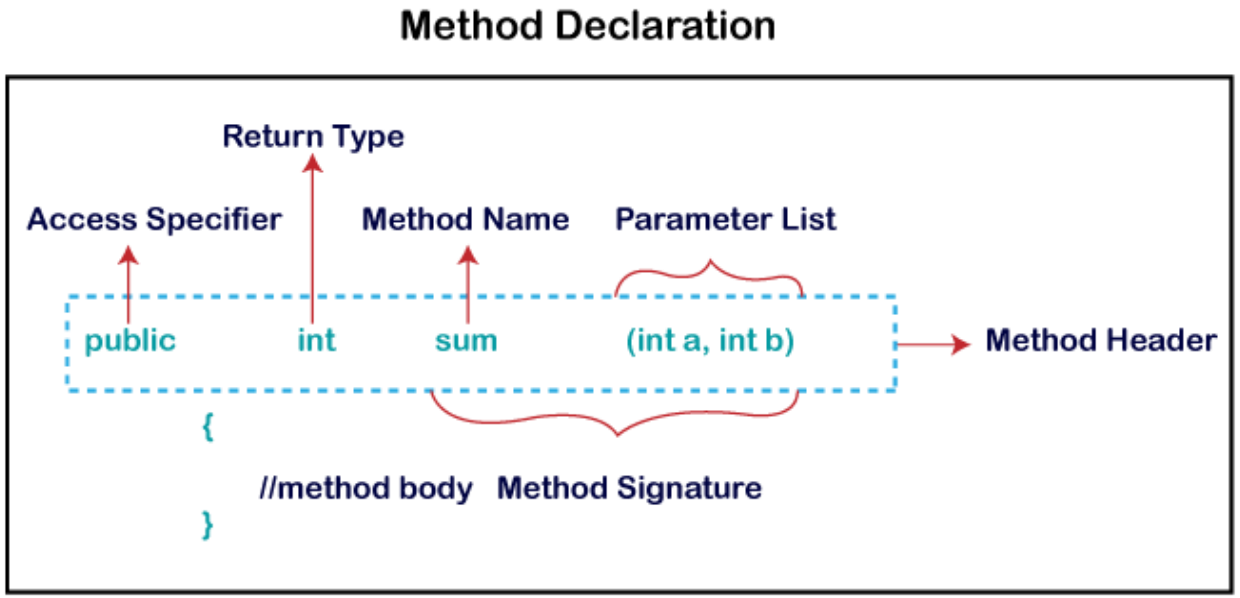
Methods In JAVA

- In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

Methods Declaration



Methods In JAVA

Naming a Method

1. **Single-word method name:** sum(), area()
2. **Multi-word method name:** areaOfCircle(), stringComparision()

Types Of Methods In JAVA

1. Predefined Method
2. User-defined Method

Instance Method

- The method of the class is known as an **instance method**.
- It is a **non-static** method defined in the class. Before calling or invoking the instance method, **it is necessary to create an object of its class.**

Static Method

- A method that has static keyword is known as static method.
- We can also create a static method by using the keyword **static** before the method name.
- The best example of a static method is the **main()** method.

Abstract Method

- The method that does not has method body is known as abstr
- It always declares in the **abstract class**. It means the class its
- To create an abstract method, we use the keyword **abstract**.

```
abstract class Demo
{
    abstract void display();
    public void square(int a){
        System.out.println("instance method " + (a*a));
    }
    public static void add(int a,int b){
        System.out.println("static method " + (a+b));
    }
}
public class MyClass extends Demo
{
    void display()
    {
        System.out.println("Abstract method ");
    }
    public static void main(String args[])
    {
        Demo obj = new MyClass();
        obj.display();
        obj.square(2);
        add(2,4);
    }
```

Data Hiding :

- ✓ Our internal data should not go out directly that is outside person can't access our internal data directly.
- ✓ By using **private** modifier, we can implement data hiding.
- ✓ Data hiding means hiding the internal data within the class to prevent its direct access from outside the class.

After providing proper username and password only , we can access our Account information.

- The main advantage of data hiding is security.

Note : recommended modifier for data members is private.

```
public class SystemUser {

    private String username;
    private String password;
    private String oldPassword;
    private String firstName;
    private String lastName;
    private String email;
    private String contact;
    private String imageId;
    private Boolean active;

}
```

Abstraction :

✓ Hide internal implementation and just highlight the set of services, is called abstraction.

Ex.

- .exe File
- .apk file

✓ **By using abstract classes and interfaces we can implement abstraction.**

- **The main advantages of Abstraction are:**

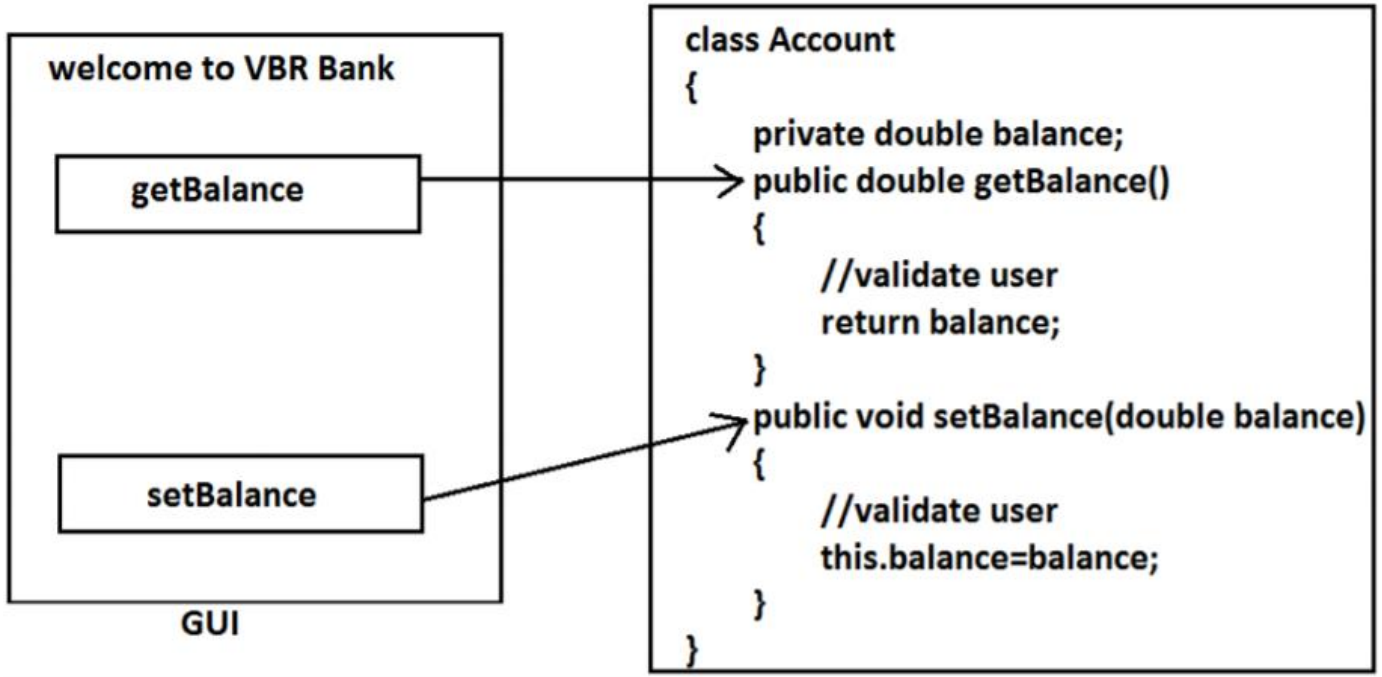
1. We can achieve security as we are not highlighting our internal implementation.(i.e., outside person doesn't aware our internal implementation.)
2. Enhancement will become very easy because without effecting end user we can be able to perform any type of changes in our internal system.
3. It provides more flexibility to the end user to use system very easily.
4. It improves maintainability of the application.

Encapsulation :

- Binding of **data and corresponding methods** into a single unit is called Encapsulation .
- If any java class follows data hiding and abstraction such type of class is said to be encapsulated class.

Encapsulation = Data hiding + Abstraction

- Every data member should be declared as private and for every member we must maintain getter & Setter methods.



Tightly encapsulated class :

- A class is said to be tightly encapsulated if and only if every **variable of that class declared as private** whether the variable has getter and setter methods are not , and whether th ese methods declared as public or not, these checking's are not required to perform.

Note:

- if the parent class is not tightly encapsulated then no child class is tightly encapsulated.

```
class A
{
    private int x=10; (valid)
}
class B extends A
{
    int y=20;(invalid)
}
class C extends A
{
    private int z=30; (valid)
}
```

Inheritance in Java / IS-A relationship

- **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object.

The syntax of Java Inheritance : -

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

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

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

✓ The **extends** keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

Example of Inheritance

Conclusion :

- Whatever the parent has by default available to the child but whatever the child has by default not available to the parent. Hence on the child reference we can call both parent and child class methods. But on the parent reference we can call only methods available in the parent class and we can't call child specific methods.

- Parent class reference can be used to hold child class objects but by using that reference we can call only methods available in the parent class and child-specific child specific methods we can't call.

- Child class reference cannot be used to hold parent class object.

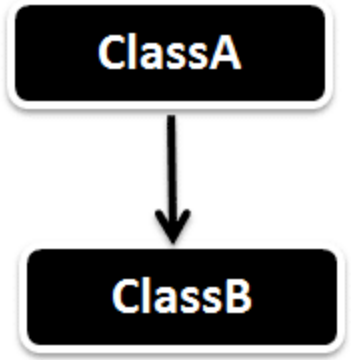
```
class Parent {
        public void methodOne(){ }
}
class Child extends Parent {
        public void methodTwo() { }
}

class Test
{
    public static void main(String[] args)
    {
        Parent p=new Parent();
        p.methodOne();
        p.methodTwo();
        Child c=new Child();
        c.methodOne();
        c.methodTwo();
        Parent p1=new Child();
        p1.methodOne();
        p1.methodTwo();
        Child c1=new Parent();
    }
}
```

C.E: cannot find symbol
symbol : method methodTwo()
location: class Parent

C.E: incompatible types
found : Parent
required: Child

Single Inheritance in Java

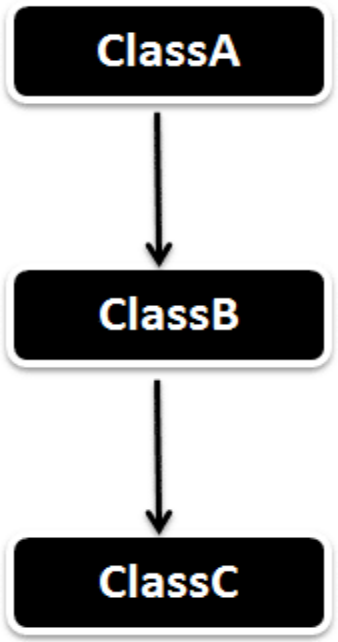
- Single Inheritance is the simple inheritance of all, When a class extends another class(Only one class) then we call it as **Single inheritance.**




```
class ClassA
{
    public void dispA()
    {
        System.out.println("disp() method of ClassA");
    }
}
public class ClassB extends ClassA
{
    public void dispB()
    {
        System.out.println("disp() method of ClassB");
    }
    public static void main(String args[])
    {
        ClassB b = new ClassB();
        b.dispA();
        b.dispB();
    }
}
```

MultiLevel Inheritance in Java

- In **Multilevel Inheritance** a derived class will be **inheriting a parent class** and as well as the derived class **act as the parent class** to other class.




```
class ClassA
{
    public void dispA()
    {
        System.out.println("disp() method of ClassA");
    }
}
class ClassB extends ClassA
{
    public void dispB()
    {
        System.out.println("disp() method of ClassB");
    }
}
public class ClassC extends ClassB
{
    public void dispC()
    {
        System.out.println("disp() method of ClassC");
    }
    public static void main(String args[])
    {
        ClassC c = new ClassC();
        c.dispA();
        c.dispB();
        c.dispC();
    }
```

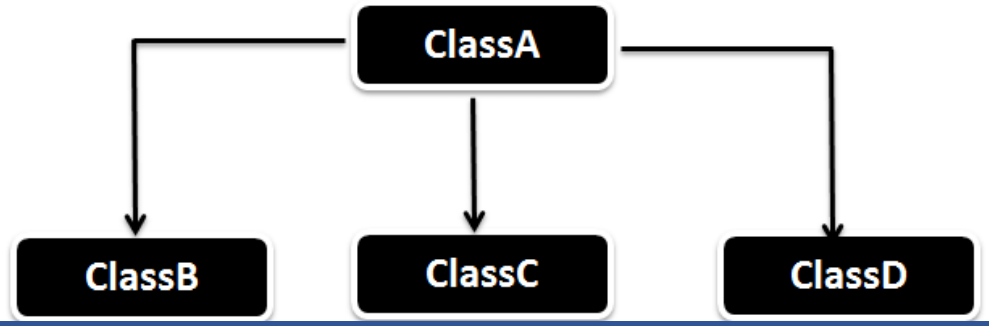
Hieracrchical Inheritance in Java

netlink

- In **Hierarchical inheritance** one parent class will be inherited by **many** sub classes

```
class ClassA
{
    public void dispA()
    {
        System.out.println("disp() method of ClassA");
    }
}
class ClassB extends ClassA
{
    public void dispB()
    {
        System.out.println("disp() method of ClassB");
    }
}
class ClassC extends ClassA
{
    public void dispC()
    {
        System.out.println("disp() method of ClassC");
    }
}
class ClassD extends ClassA
{
    public void dispD()
    {
        System.out.println("disp() method of ClassD");
    }
}
```

```
public class HierarchicalInheritanceTest
{
    public static void main(String args[])
    {
        ClassB b = new ClassB();
        b.dispB();
        b.dispA();

        ClassC c = new ClassC();
        c.dispC();
        c.dispA();

        ClassD d = new ClassD();
        d.dispD();
        d.dispA()
    }
}
```



HAS-A relationship.

- Has-A relationship simply means that an **instance of one class has a reference** to an **instance of another class** or an other instance of the same class

  - 1. HAS-A relationship is also known as **composition (or) aggregation.**
  - 2. There is no specific keyword to implement HAS-A relationship but mostly we can use **new** operator.
  - 3. The main advantage of HAS-A relationship is reusability.

Composition vs Aggregation:

- Has-A relationship simply means that an instance of one class has a reference to an instance of another class or an other instance of the same class

- **Composition:**

- Without existing container object if there is no chance of existing contained objects then the relationship between container object and contained object is called composition which is a strong association.

- **Aggregation :**

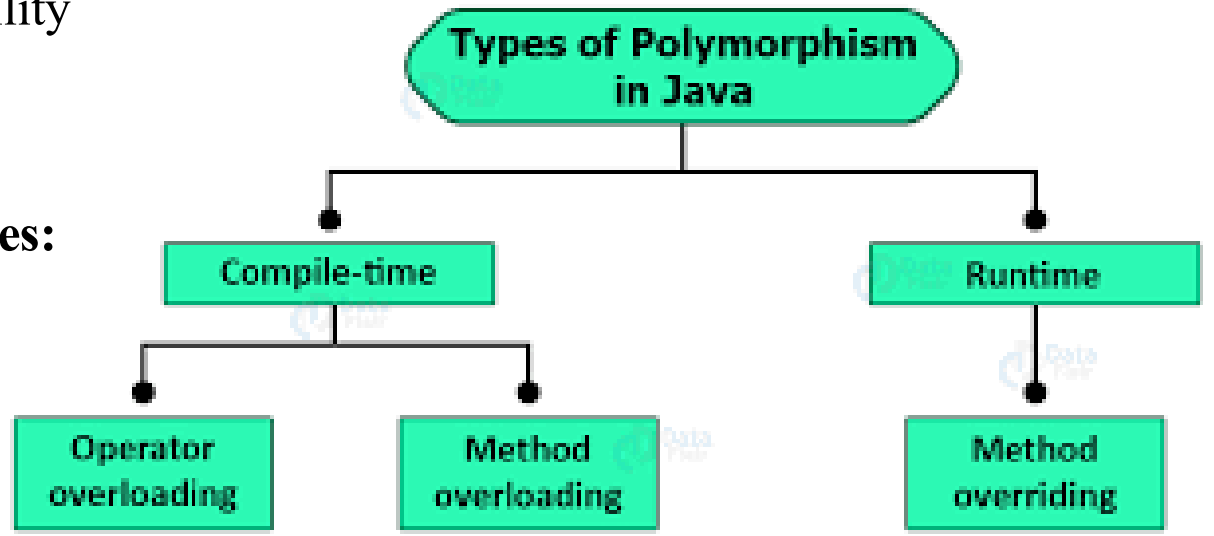
- Without existing container object if there is a chance of existing contained objects such type of relationship is called aggregation. **In aggregation objects have weak association.**

polymorphism in java :

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

In Java polymorphism is mainly divided into two types:

- Compile time Polymorphism
- Runtime Polymorphism



Compile time Polymorphism / OverLoading

- It is also known as **static** polymorphism. This type of polymorphism is achieved by function overloading or operator overloading. But Java doesn't support the Operator Overloading.

Method Overloading:

- When there are multiple functions with same name but different parame then these functions are said to be **overloaded**. Functions can be overloaded by **change in number of arguments** or/and **change in type arguments**.

Note :

In overloading **method resolution is always based on reference type** and runtime object won't play any role in overloading.

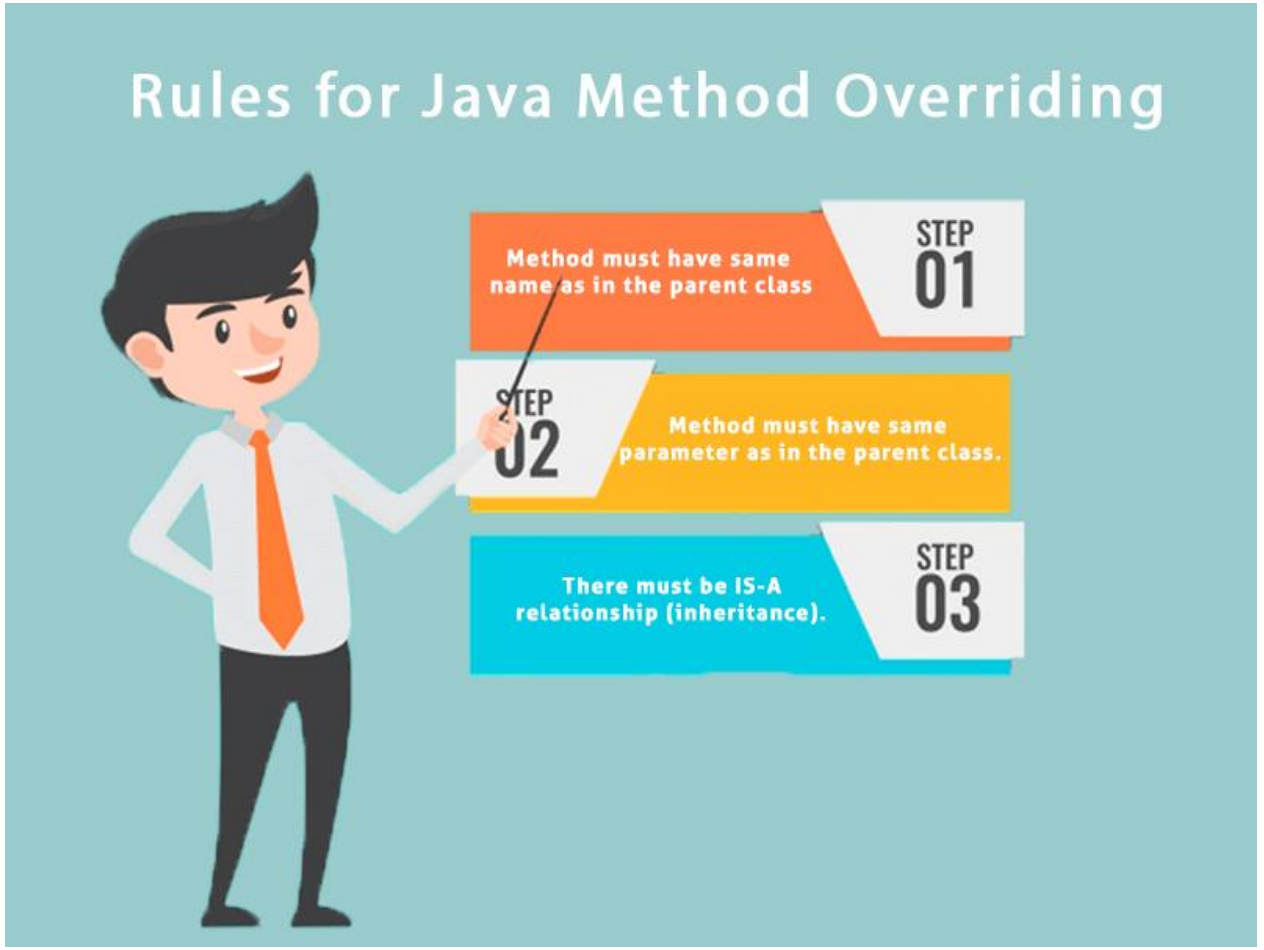
```
class Animal {
}

class Monkey extends Animal {
}

class Case5 {
    public void methodOne(Animal a) {
        System.out.println("Animal version");
    }

    public void methodOne(Monkey m) {
        System.out.println("Monkey version");
    }

    public static void main(String[] args) {
        Case5 t = new Case5();
        Animal a = new Animal();
        t.methodOne(a);//Animal version
        Monkey m = new Monkey();
        t.methodOne(m);//Monkey version
        Animal a1 = new Monkey();
        t.methodOne(a1);//Animal version
    }
```

Runtime polymorphism/ Method OverRiding

- It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

Method overriding

Note: **In overriding runtime object will play the role and** reference type is dummy



netlink

Difference between method overloading and method overriding in java

Property	Overloading	Overriding
1) Method names	Must be same.	Must be same.
2) Argument type	Must be different(at least order)	Must be same including order.
3) Method signature	Must be different.	Must be same.
4)Return types	No restrictions.	Must be same until 1.4v but from 1.5v onwards we can take co-variant return types also.
5) private, static, final methods	Can be overloaded.	Can not be overloaded.
6) Method resolution	Is always takes care by compiler based on referenced type.	Is always takes care by JVM based on runtime object.
7) Throws clause	No restrictions	If child class method throws any checked exception compulsory parent class method should throw the same checked exceptions or its parent but no restrictions for un-checked exceptions.

METHOD HIDING :

- All rules of method hiding are exactly same as **overriding** except the following differences.

- 1. Both Parent and Child class methods should be static.
- 2. Method hiding is also considered as compile time polymorphism (or) static polymorphism (or) early biding.

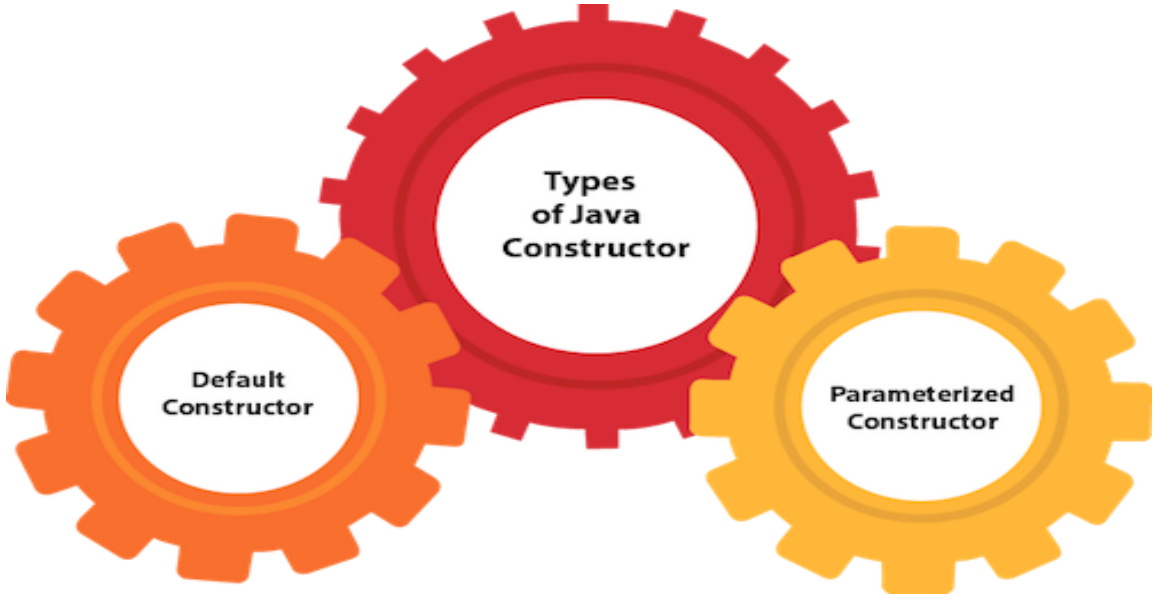
- **Note**: If both Parent and Child class methods are non static then it will become overriding and method resolution is based on runtime object.

```
class Parent {
    public static void methodOne() {
        System.out.println("parent class");
    }
}
class Child extends Parent{
    public static void methodOne(){
        System.out.println("child class");
    }
}
class MethodHiding{
    public static void main(String[] args) {
        Parent p=new Parent();
//          Parent.methodOne();
//          Child.methodOne();
        p.methodOne();//parent class
        Child c=new Child();
        c.methodOne();//child class
        Parent p1=new Child();
        p1.methodOne();//parent class
    }
}
```

Types Of Constructors

here are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



Default Constructor

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

- If there is no constructor in a class, compiler automatically creates a default constructor.

Prototype of default constructor:

1. It is always no argument constructor.
2. The access modifier of the default constructor is same as class modifier. (This rule is applicable only for public and default).
3. Default constructor contains only one line. super(); it is a no argument call to super class constructor.

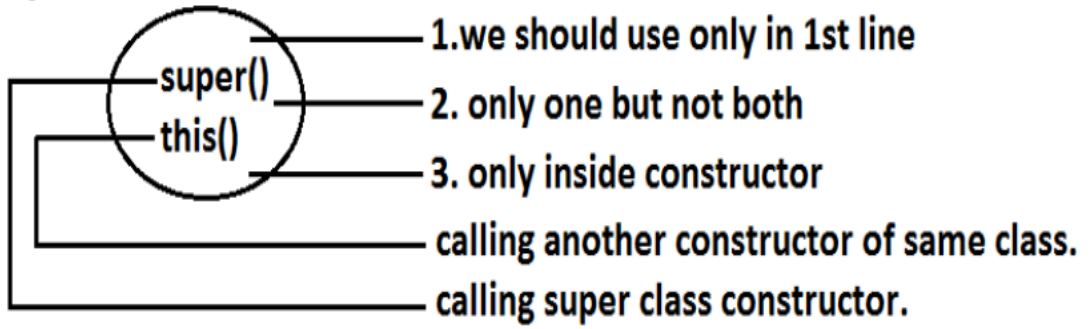
super() vs this():

- The 1st line inside every constructor should be either super() or this() if we are not writing anything compiler will always generate super().

super(), this()	super, this
These are constructors calls.	These are keywords
We can use these to invoke super class & current constructors directly	We can use refers parent class and current class instance members.
We should use only inside constructors as first line, if we are using outside of constructor we will get compile time error.	We can use anywhere (i.e., instance area) except static area , other wise we will get compile time error .

```
class Case1
{
    Case1()
    {
        // super();
        System.out.println("constructor");
    }
}
```

Diagram:



Overloaded constructors :

- A class can contain more than one constructor and all these constructors having the same name but different arguments and hence these constructors are considered as overloaded constructors.

- Parent class constructor by default won't available to the Child. **Hence Inheritance concept is not applicable for constructors and hence overriding concept also not applicable to the constructors. But constructors can be overloaded.**
- We can take constructor in any java class including abstract class also but we can't take constructor inside interface

Constructor Chaining :

- Constructor chaining is the process of calling one constructor from another constructor with respect to current object.

Constructor chaining can be done in two ways:

- **Within same class**: It can be done using **this()** keyword for constructors in same class
- **From base class:** by using **super()** keyword to call constructor from the base class.

```
class Chaining
{
    Chaining()
    {
        this(5);
        System.out.println("The Default constructor");
    }
    Chaining(int x)
    {
        this(5, 15);
        System.out.println(x);
    }
    Chaining(int x, int y)
    {
        System.out.println(x * y);
    }

    public static void main(String args[])
    {
        new Chaining(8, 10);
    }
}
```

Parameterized Constructor

- A constructor that has parameters is known as parameterized constructor.
- If we want to initialize fields of the class with your own values, then use a parameterized constructor.

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

    Student(String name, int id) {
        this.name = name;
        this.id = id;
    }

    public void display() {
        System.out.println("GeekName :" + name +
                " and GeekId :" + id);
    }
}

class Parameterized {
    public static void main(String[] args) {
        Student student = new Student("adam", 1);
        Student student2 = new Student("john", 2);
        student.display();
        student2.display();
    }
```

Constructors Example :-

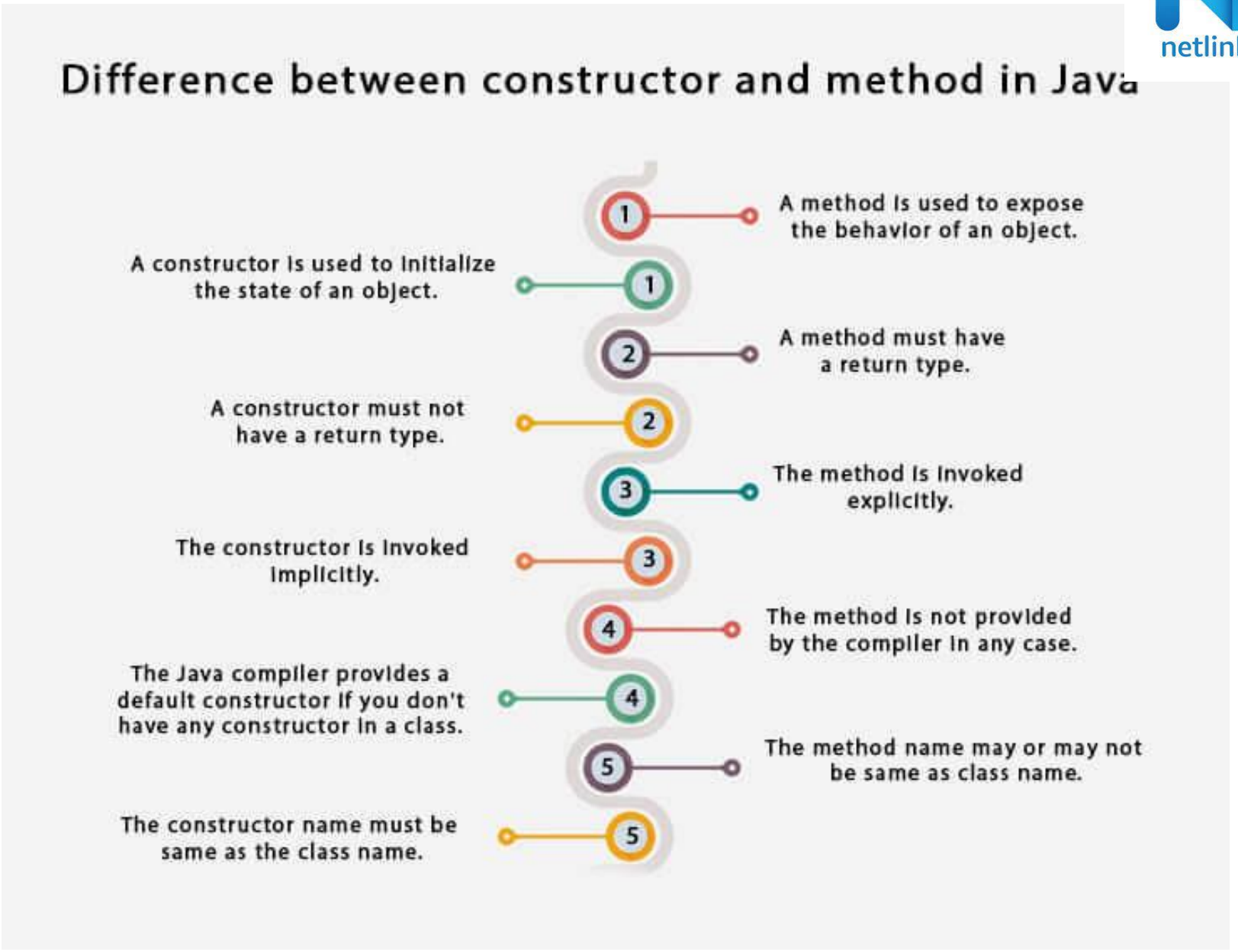
Default Example :

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

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

public class Main {

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

Parameterized Example :

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

    Student(int i, String n) {
        id = i;
        name = n;
    }
    void display() {
        System.out.println(id + " " + name);
    }
}
public class Main {
        public static void main(String args[]) {
            Student s1 = new Student(111, "Karan");
            Student s2 = new Student(222, "Aryan");
            s1.display();
            s2.display();
        }
}
```

Difference between constructor and method in Java



Constructors

- . Object creation is not enough compulsory we should perform initialization then only the object is in a position to provide the response properly.
- Whenever we are creating an object some piece of the code will be executed automatically to perform initialization of an object this piece of the code is nothing but constructor.
- Hence the main objective of constructor is to perform initialization of an object.

There are two rules defined for the constructor.

- Constructor name must be the same as its class name
- A Constructor must have no explicit return type.

- The only applicable modifiers for the constructors are public, default, private, protected.