1. Java OOPs Concepts

OOPs: Object Oriented Programming System

Object-Oriented Programming is a methodology to design a program using classes and objects.

OOPs concepts:

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

Object

Any entity that has state and behavior is known as an object.

For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class.

2. Class

A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.

A class in java can contain:

- **data member // variable (int, float, char, double, long, short) int a=10**
- **method // function**
- **constructor**
- **class and interface**

Syntax to declare a class:

```
class<class_name>{
data member; // variable
method;  // function
}
```

Syntax for creating an object for class:

```
classname objname=new classname();
```

Simple Example of Object and Class

```
class Student1{
        int id; //data member (also instance variable)
         String name; //data member(also instance variable)

public static void main(String args[]){
        Student1 s1=new Student1(); //creating an object of Student
        System.out.println(s1.id);
        System.out.println(s1.name);
         }
}
```

Object

Object is an instance of a class.

Class is a template or blueprint from which objects are created.

Method in Java

In java, a method is like function i.e. used to expose behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

new keyword

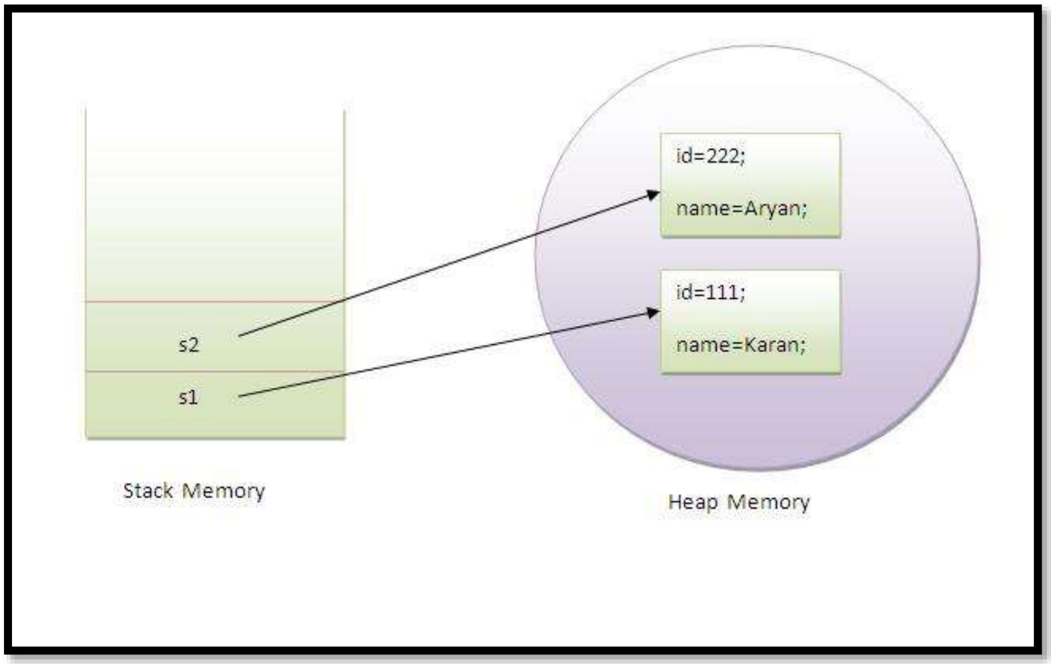
The new keyword is used to allocate memory at runtime.

```
class Student3{

      int insertRecord(int r, int n){  //method1 definition
            int sum=r+n;
            return sum;
       }

public static void main(String args[]){
      Student3 s1=new Student3();  //S1
      int x=s1.insertRecord(111,222); // calling
      System.out.println(x);
 }
}

class Student4{

      void insertRecord(int r, int n){  //method1 definition
            int sum=r+n;
            System.out.println(sum);
       }

public static void main(String args[]){
      Student2 s1=new Student2();  //S1
      s1.insertRecord(111,222); // calling

 }
}
```

Example of Object and class that maintains the records of students

```
class Student2{

int rollno;

 String name;


void insertRecord(int r, String n){  //method

rollno=r;
name=n;
 }

void displayInformation(){System.out.println(rollno+" "+name);}//method

public static void main(String args[]){
  Student2 s1=new Student2();
  Student2 s2=new Student2();
s1.insertRecord(111,"Karan");
s2.insertRecord(222,"Aryan");

s1.displayInformation();
s2.displayInformation();

 }
}
```

```
111 Karan
222 Aryan
```



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

Ex2:

```
class Rectangle{
int length;
int width;

void insert(int l,int w){
length=l;
width=w;
 }

Void calculateArea(){
System.out.println(length*width);
}
public static void main(String args[]){
  Rectangle r1=new Rectangle();
  Rectangle r2=new Rectangle();

r1.insert(11,5); // calling method insert
r2.insert(3,15);

r1.calculateArea();
r2.calculateArea();
}
}
```

```
Output:55
 45
```

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation.

Creating multiple objects by one type only

```
class Rectangle2{
int length;
int width;

void insert(intl,int w){
length=l;
width=w;
```

```
 }

voidcalculateArea(){System.out.println(length*width);}

public static void main(String args[]){
  Rectangle2 r1=new Rectangle2(),r2=new Rectangle2();//creating two objects

r1.insert(11,5);
r2.insert(3,15);

r1.calculateArea();
r2.calculateArea();
}
}
```

```
Output:55
 45
```

3. Method Overloading in Java

If a class have multiple methods by same name but different parameters/arguments, it is known as **Method Overloading**.

Advantage of method overloading?

Method overloading **increases the readability of the program**.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments

2. By changing the data type

1 Example of Method Overloading by changing the no. of arguments

```
class OverloadingEx1{

void sum(int a,int b){System.out.println(a+b);}

void sum(int a,int b,int c){System.out.println(a+b+c);}


public static void main(String args[]){

  OverloadingEx1 obj=new OverloadingEx1 ();

obj.sum(10,10,10);

obj.sum(20,20);


  }

}
```

```
Output:30
 40
```

2)Example of Method Overloading by changing data type of argument

```
class OverloadingEx2{

void sum(int a,int b){System.out.println(a+b);}

void sum(double a,double b){System.out.println(a+b);}


public static void main(String args[]){

  OverloadingEx2 obj=new OverloadingEx2();

obj.sum(10.5,10.5);
obj.sum(20,20);


 }

}
```

```
Output:21.0
40
```

4. Constructor in Java

Constructor in java is a *special type of method* that is used to initialize the object.

Java constructor is *invoked at the time of object creation*.

Rules for creating java constructor

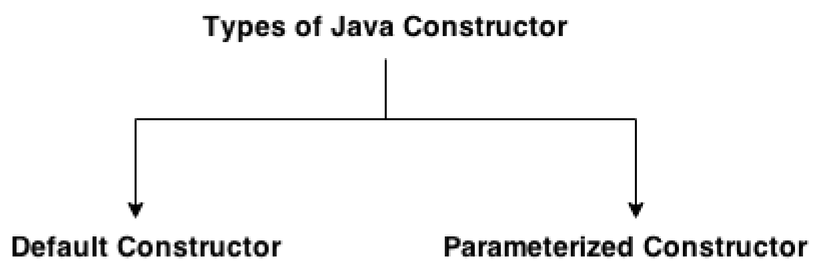
There are basically two rules defined for the constructor.

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

Types of java constructors

There are two types of constructors:

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



Java Default Constructor

A constructor that have no parameter is known as default constructor.

Syntax of default constructor:

```
<class_name>(){}
```

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1{
Bike1(){System.out.println("Bike is created");}
public static void main(String args[]){
Bike1 b=new Bike1();
}


}
```

What is the purpose of default constructor?

Default constructor provides the default values to the object like 0, null etc. depending on the type.

Example of default constructor that displays the default values

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

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

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

Output:

```
0 null
0 null
```

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

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

  ParamCons (int i,String n){
  id = i;
  name = n;
  }
  void display(){System.out.println(id+" "+name);}
```

```
    public static void main(String args[]){
    ParamCons s1 = new ParamCons (111,"Karan");
    ParamCons s2 = new ParamCons (222,"Aryan");
    s1.display();
    s2.display();
   }
}
```

Output:

```
111 Karan
  222 Aryan
```

Constructor Overloading in Java

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example of Constructor Overloading

```
class Student5{
    int id;
    String name;
    int age;
    Student5(int i,String n){
    id = i;
    name = n;
    }
    Student5(int i,String n,int a){
    id = i;
    name = n;
    age=a;
    }
    void display(){System.out.println(id+" "+name+" "+age);}

    public static void main(String args[]){
    Student5 s1 = new Student5(111,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

Output:

```
111 Karan 0
222 Aryan 25
```

Difference between constructor and method in java

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
Constructor name must be same as the class name.	Method name is not same as class name.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.

5 Java static keyword

The **static keyword** in java is used for memory management. We can apply java static keyword with variables, methods and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)

1) Java static variable

If you declare any variable as static, it is known static variable.

- The static variable can be used to refer the common property of all objects
- The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable

It makes your program **memory efficient** (i.e it saves memory).

Understanding problem without static variable

```
class Student{
    int rollno;
    String name;
   String college="ITS";
}
`
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created. All student have its unique rollno and name so instance data member is good. Here, college refers to the common property of all objects. If we make it static, this field will get memory only once.

Example of static variable

```
//Program of static variable

class StaticVar{
int rollno;
   String name;
static String college ="ITS";

StaticVar(int r,String n){
rollno = r;
name = n;
   }
void display (){System.out.println(rollno+" "+name+" "+college);}

public static void main(String args[]){
 StaticVar s1 = new StaticVar(111,"Karan");
 StaticVar s2 = new StaticVar(222,"Aryan");

s1.display();
s2.display();
 }
}
```

```
Output:111 Karan ITS
  222 Aryan ITS
```

2) Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance/object of a class.
- static method can access static data member and can change the value of it.

Example of static method

```
//Program of changing the common property of all objects(static field).


class StaticMethod{

int rollno;

   String name;

static String college = "ITS";


static void change(){

college = "BBDIT";

   }


StaticMethod(int r, String n){

rollno = r;

name = n;

   }


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


public static void main(String args[]){

StaticMethod.change();


  StaticMethod s1 = new StaticMethod (111,"Karan");

  StaticMethod s2 = new StaticMethod (222,"Aryan");

  StaticMethod s3 = new StaticMethod (333,"Sonoo");
s1.display();
s2.display();
s3.display();
   }
}
```

```
Output:111 Karan BBDIT
       222 Aryan BBDIT
```

333 Sonoo BBDIT

Restrictions for static method

There are two main restrictions for the static method. They are:

1. The static method can not use non static data member or call non-static method directly.

```
class NonStaticMember{
 int a=40;//non static

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

Output:Compile Time Error

why java main method is static?

Because object is not required to call static method

If it was non-static method, jvm create object first then call main() method that will lead the problem of extra memory allocation.

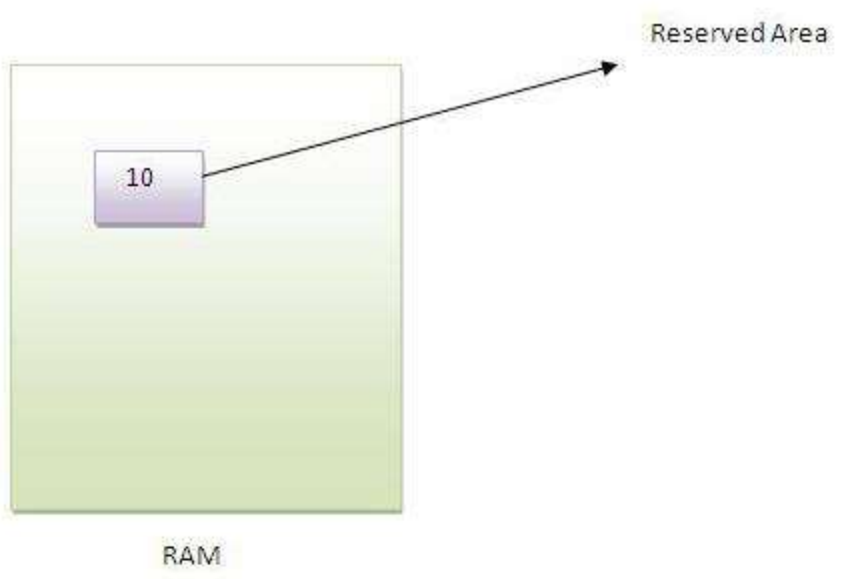
6. Variable and Datatype in Java

1. Variable
2. Types of Variable
3. Data Types in Java

Variable is a name of memory location.

Variable

Variable is name of reserved area allocated in memory.

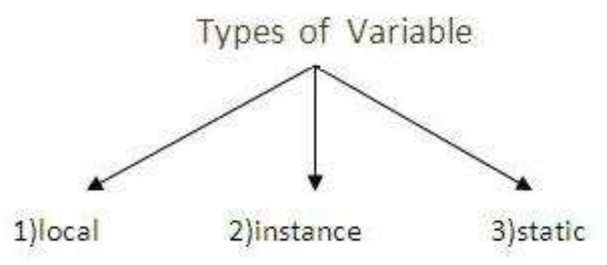


```
int data=50;//Here data is variable
```

1. Types of Variable

There are three types of variables in java

- local variable
- instance variable
- static variable



Local Variable

A variable that is declared inside the method is called local variable.

Instance Variable

A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static.

Static variable

A variable that is declared as static is called static variable. It cannot be local.

Example to understand the types of variables

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

7. Inheritance in Java

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

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

Why use inheritance in java

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

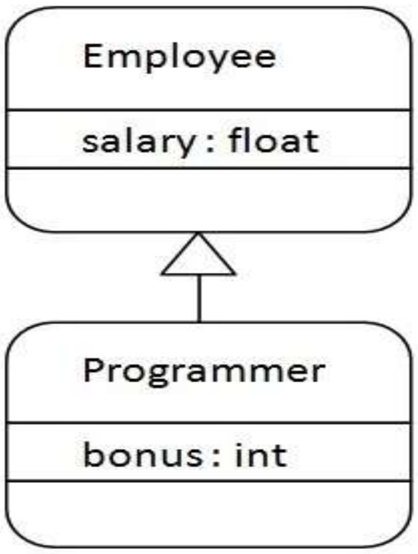
Syntax of Java Inheritance

```
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.

In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

Understanding the simple example of inheritance



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. Relationship between two classes is **Programmer IS-A Employee.**It means that Programmer is a type of Employee.

```
class Employee{

float salary=40000;

}

Public 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);

}

}
```

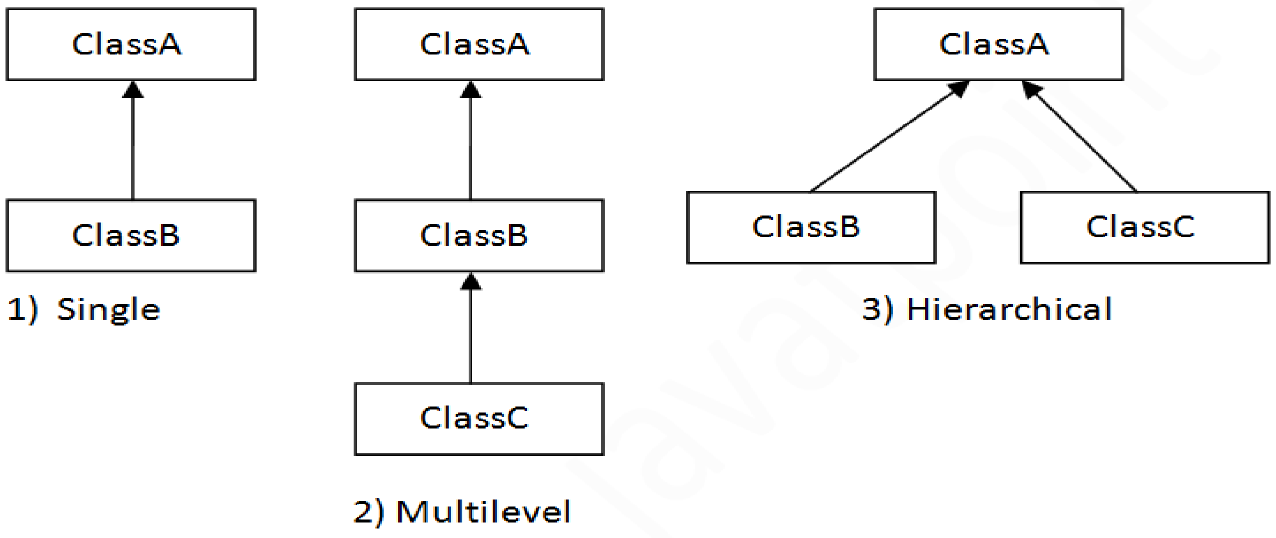
```
Programmer salary is:40000.0
```

Bonus of programmer is:10000

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

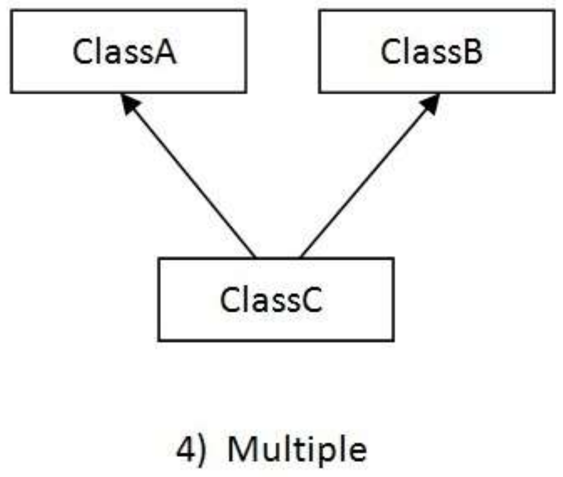
Types of inheritance in java

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

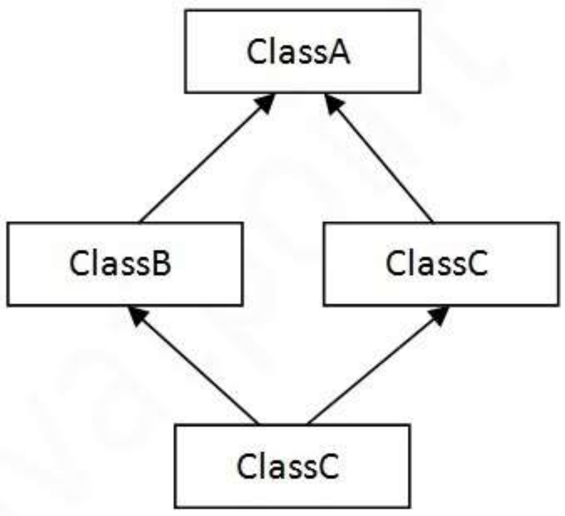


Note: Multiple inheritance is not supported in java through class.

When a class extends multiple classes i.e. known as multiple inheritance. For Example:



4) Multiple



5) Hybrid

.

Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
```

```
public class MultipleInh extends A,B{//suppose if it were

 public static void main(String args[]){
   MultipleInh  obj=new MultipleInh ();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

Compile Time Error

8. 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**.

In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. method must have same name as in the parent class
2. method must have same parameter as in the parent class.
3. must be IS-A relationship (inheritance).

Example of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

```
class Vehicle{
void run(){System.out.println("Vehicle is running");}
}
public class Bike2 extends Vehicle{
void run(){System.out.println("Bike is running safely");}

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

```
Output:Bike is running safely
```

Real example of Java Method Overriding

Consider a scenario, Bank is a class that provides functionality to get rate of interest. But, rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7% and 9% rate of interest.

```
class Bank{
int getRateOfInterest(){return 0;}
}

class SBI extends Bank{
int getRateOfInterest(){return 8;}
}
```

```
class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}

Public class Test2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}
}
```

```
Output:
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Can we override static method?

No, static method cannot be overridden.

Why we cannot override static method?

Because static method is bound with class whereas instance method is bound with object.

Can we override java main method?

No, because main is a static method.

Polymorphism (Over Loading & Over Riding)

Difference between method Overloading and Method Overriding in java

There are three basic differences between the method overloading and method overriding. They are as follows:

Method Overloading	Method Overriding
1) Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2) method overloading is performed within a class.	Method overriding occurs in two classes that have IS-A relationship.
3) In case of method overloading parameter must be different.	In case of method overriding parameter must be same.

9. Final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. The basics of final keyword,

Java Final Keyword
Stop Value Change
Stop Method Overridding
Stop Inheritance
javatpoint.com

1) Java final variable

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

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

```
Output:Compile Time Error
```

2) Java final method

If you make any method as final, you cannot override it.

Example of final method

```
class Bike{
  final void run(){System.out.println("running");}
}

class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}

   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

```
Output:Compile Time Error
```

---

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
final class Bike{}

class Honda1 extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda1 honda= new Honda();
  honda.run();
  }
}
```

Output:Compile Time Error

10. Abstract class in Java

A class that is declared with abstract keyword, is known as abstract class in java.

It can have abstract (method without body).and non-abstract methods (method with body).

Abstraction in Java

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

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstaction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class in Java

A class that is declared as abstract is known as **abstract class**.

It needs to be extended and its method implemented. It cannot be instantiated.

Example abstract class

```
abstract class A{}
```

abstract method

A method that is declared as abstract and does not have implementation is known as abstract method.

Example abstract method

```
abstract void printStatus();//no body and abstract
```

Example of abstract class that has abstract method

In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

```
abstract class Bike{
  abstract void run();


}


class Honda4 extends Bike{
void run(){System.out.println("running safely..");}


public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

```
running safely..
```

Understanding the real scenario of abstract class

In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

```
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 real scenario, object is provided through method e.g. getShape() method
s.draw();
}
}
```

```
drawing circle
```

Another example of abstract class in java

```
abstract class Bank{
abstract int getRateOfInterest();
}
```

```
class SBI extends Bank{
int getRateOfInterest(){return 7;}


}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}


class TestBank{
public static void main(String args[]){
Bank b=new SBI();//if object is PNB, method of PNB will be invoked
int interest=b.getRateOfInterest();
System.out.println("Rate of Interest is: "+interest+" %");
}}
```

```
Rate of Interest is: 7 %
```

Rule: If there is any abstract method in a class, that class must be abstract.

```
class Bike12{
abstract void run();
}
```

```
compile time error
```

Rule: If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

11. Interface in Java

An **interface in java** is a blueprint of a class. It has static constants and abstract methods only.

The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.

Java Interface also **represents IS-A relationship**.

It cannot be instantiated just like abstract class.

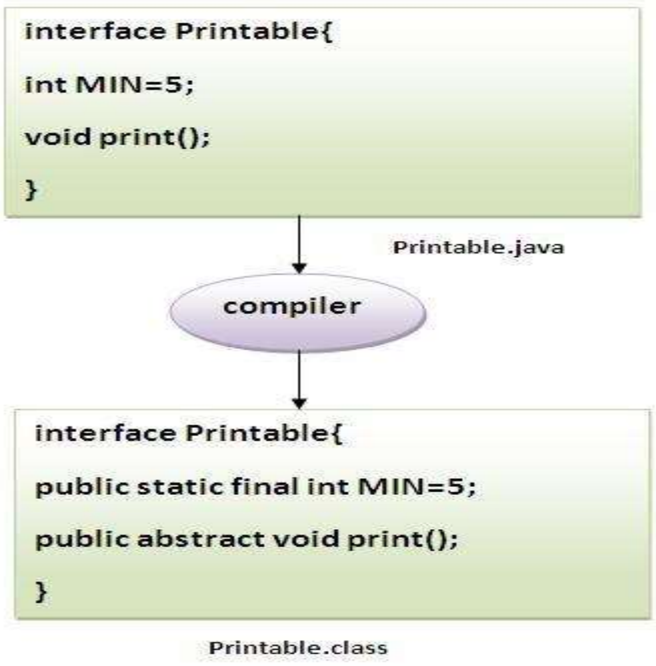
Why use Java interface?

There are mainly below reasons to use interface. They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.

The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

In other words, Interface fields are public, static and final by default, and methods are public and abstract.

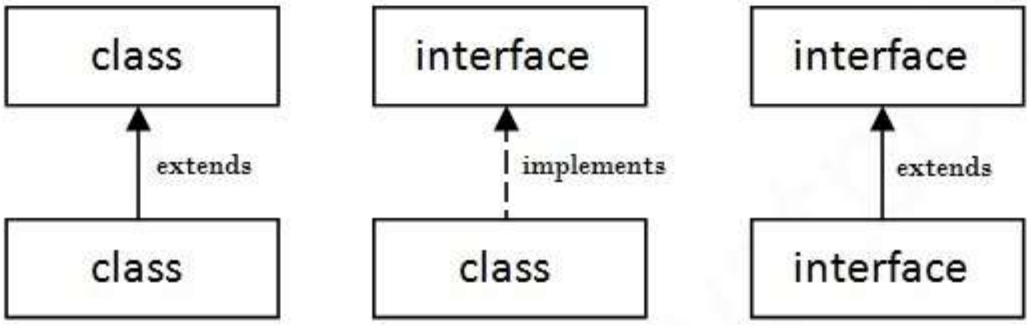


```
interface Printable{
int MIN=5;
void print();
}
```

```
interface Printable{
public static final int MIN=5;
public abstract void print();
}
```

Understanding relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface but a **class implements an interface**.



Simple example of Java interface

In this example, Printable interface have only one method, its implementation is provided in the A class.

```
interface printable{
void print();
}

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

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

```
Output:Hello
```

Interface inheritance

A class implements interface but one interface extends another interface.

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class Testinterface2 implements Showable{

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

public static void main(String args[]){
Testinterface2 obj = new Testinterface2();
obj.print();
obj.show();
 }
}
```

```
o/p:        Hello
            Welcome
```

Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

Abstract class	**Interface**
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can have static methods, main method and constructor**.	Interface **can't have static methods, main method or constructor**.
5) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
6) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

7) **Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

12. Java naming conventions

Java **naming convention** is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

But, it is not forced to follow. So, it is known as convention not rule.

All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

Advantage of naming conventions in java

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that **less time** is spent to figure out what the code does.

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.