**Java OOPs Concepts**

In this page, we will learn about basics of OOPs. Object Oriented Programming is a paradigm that provides many concepts such as **inheritance**, **data binding**, **polymorphism** etc.

**Simula** is considered as the first object-oriented programming language. The programming paradigm where everything is represented as an object, is known as truly object-oriented programming language.

**Smalltalk** is considered as the first truly object-oriented programming language.

**OOPs (Object Oriented Programming System)**

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some 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. It is a logical entity.

**Inheritance**

**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.



**Polymorphism**

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convense the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

**Abstraction**

**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.



**Encapsulation**

**Binding (or wrapping) code and data together into a single unit is known as encapsulation**. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

**Advantage of OOPs over Procedure-oriented programming language**

|  |
| --- |
| 1)OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows. |
| 2)OOPs provides data hiding whereas in Procedure-oriented prgramming language a global data can be accessed from anywhere. |
| 3)OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language. |

|  |  |
| --- | --- |
| Global Data | Object Data |

**Method Overloading in Java**

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

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs. So, we perform method overloading to figure out the program quickly.

**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 |

**In java, Methood Overloading is not possible by changing the return type of the method.**

1. class Calculation{
2. void sum(int a,int b){System.out.println(a+b);}
3. void sum(int a,int b,int c){System.out.println(a+b+c);}
5. public static void main(String args[]){
6. Calculation obj=new Calculation();
7. obj.sum(10,10,10);
8. obj.sum(20,20);
10. }
11. }

**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*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

**Rules for creating java constructor**

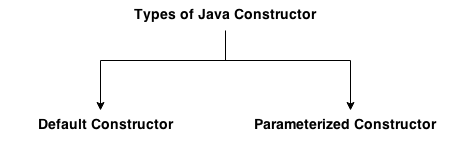
There are basically two rules defined for the constructor.

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. |

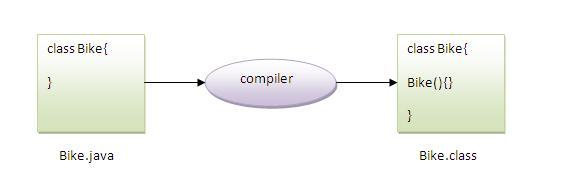
**Syntax of default constructor:**

1. <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. |

1. class Bike1{
2. Bike1(){System.out.println("Bike is created");}
3. public static void main(String args[]){
4. Bike1 b=new Bike1();
5. }
6. }



**Java parameterized constructor**

|  |
| --- |
| A constructor that have parameters is known as parameterized constructor. |

**Why use parameterized constructor?**

|  |
| --- |
| Parameterized constructor is used to provide different values to the distinct objects. |

**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. |

1. class Student4{
2. int id;
3. String name;
5. Student4(int i,String n){
6. id = i;
7. name = n;
8. }
9. void display(){System.out.println(id+" "+name);}
11. public static void main(String args[]){
12. Student4 s1 = new Student4(111,"Karan");
13. Student4 s2 = new Student4(222,"Aryan");
14. s1.display();
15. s2.display();
16. }
17. }

**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**

1. class Student5{
2. int id;
3. String name;
4. int age;
5. Student5(int i,String n){
6. id = i;
7. name = n;
8. }
9. Student5(int i,String n,int a){
10. id = i;
11. name = n;
12. age=a;
13. }
14. void display(){System.out.println(id+" "+name+" "+age);}
16. public static void main(String args[]){
17. Student5 s1 = new Student5(111,"Karan");
18. Student5 s2 = new Student5(222,"Aryan",25);
19. s1.display();
20. s2.display();
21. }
22. }

**Difference between constructor and method in java**

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

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

**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.

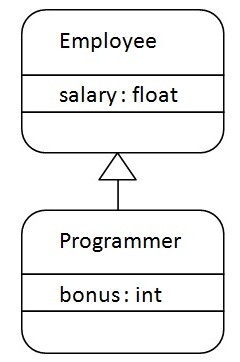
**Syntax of Java Inheritance**

1. class Subclass-name extends Superclass-name
2. {
3. //methods and fields
4. }

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.

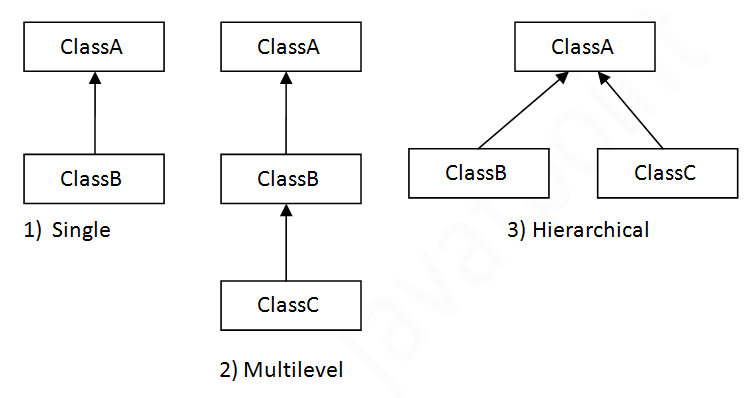
1. class Employee{
2. float salary=40000;
3. }
4. class Programmer extends Employee{
5. int bonus=10000;
6. public static void main(String args[]){
7. Programmer p=new Programmer();
8. System.out.println("Programmer salary is:"+p.salary);
9. System.out.println("Bonus of Programmer is:"+p.bonus);
10. }
11. }

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**

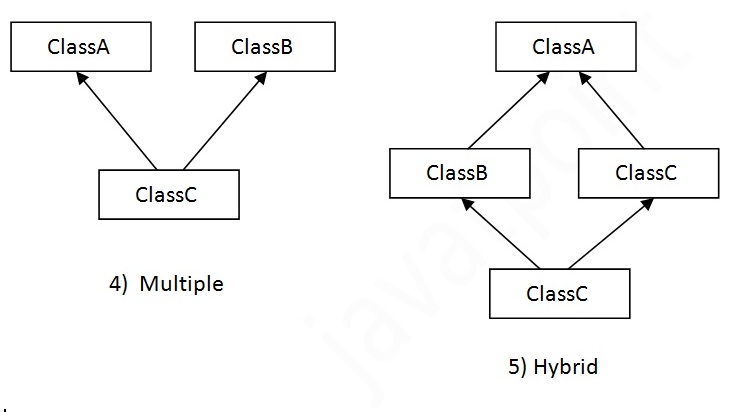
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



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

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



**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.

1. class A{
2. void msg(){System.out.println("Hello");}
3. }
4. class B{
5. void msg(){System.out.println("Welcome");}
6. }
7. class C extends A,B{//suppose if it were
9. Public Static void main(String args[]){
10. C obj=new C();
11. obj.msg();//Now which msg() method would be invoked?
12. }
13. }

**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.

1. class Vehicle{
2. void run(){System.out.println("Vehicle is running");}
3. }
4. class Bike2 extends Vehicle{
5. void run(){System.out.println("Bike is running safely");}
7. public static void main(String args[]){
8. Bike2 obj = new Bike2();
9. obj.run();
10. }

**Abstract class in Java**

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract 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**

1. 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**

1. 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.

1. abstract class Bike{
2. abstract void run();
3. }
5. class Honda4 extends Bike{
6. void run(){System.out.println("running safely..");}
8. public static void main(String args[]){
9. Bike obj = new Honda4();
10. obj.run();
11. }
12. }

**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.

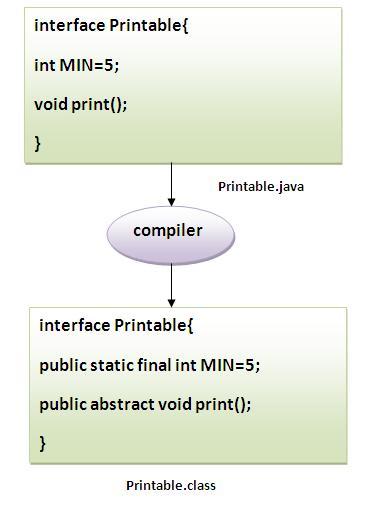
**Why use Java interface?**

There are mainly three 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.
* It can be used to achieve loose coupling.

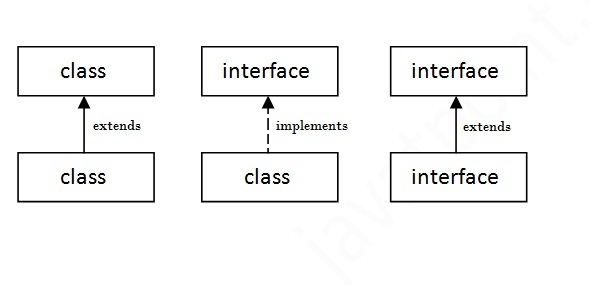
**The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.**

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



**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**.



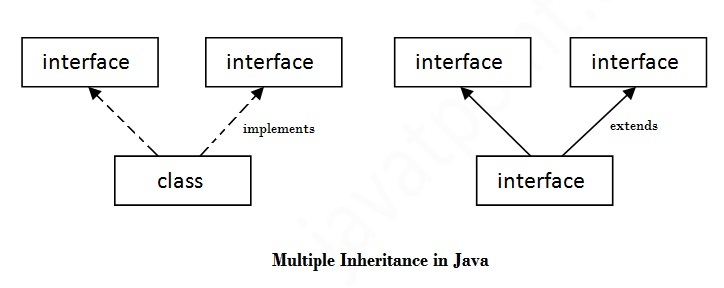
**Simple example of Java interface**

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

1. interface printable{
2. void print();
3. }
5. class A6 implements printable{
6. public void print(){System.out.println("Hello");}
8. public static void main(String args[]){
9. A6 obj = new A6();
10. obj.print();
11. }
12. }

**Multiple inheritance in Java by interface**

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



1. interface Printable{
2. void print();
3. }
5. interface Showable{
6. void show();
7. }
9. class A7 implements Printable,Showable{
11. public void print(){System.out.println("Hello");}
12. public void show(){System.out.println("Welcome");}
14. public static void main(String args[]){
15. A7 obj = new A7();
16. obj.print();
17. obj.show();
18. }
19. }

# 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(); } |

**Java Package**

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

Package in java can be categorized in two form, built-in package and user-defined package.

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

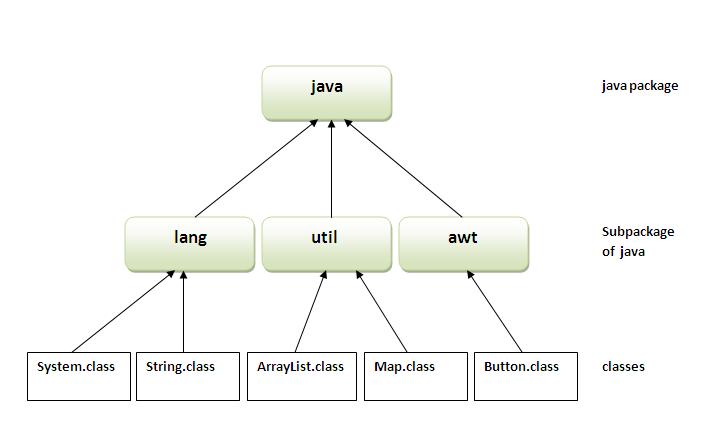
Here, we will have the detailed learning of creating and using user-defined packages.

**Advantage of Java Package**

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



**Simple example of java package**

The **package keyword** is used to create a package in java.

1. //save as Simple.java
2. package mypack;
3. public class Simple{
4. public static void main(String args[]){
5. System.out.println("Welcome to package");
6. }
7. }

**How to compile java package**

If you are not using any IDE, you need to follow the **syntax** given below:

1. javac -d directory javafilename

For **example**

1. javac -d . Simple.java

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

**How to run java package program**

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

|  |
| --- |
| **To Compile:** javac -d . Simple.java |
| **To Run:** java mypack.Simple |

Output:Welcome to package

|  |
| --- |
| The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder. |

**How to access package from another package?**

|  |
| --- |
| There are three ways to access the package from outside the package.   1. import package.\*; 2. import package.classname; 3. fully qualified name. |

**1) Using packagename.\***

|  |
| --- |
| If you use package.\* then all the classes and interfaces of this package will be accessible but not subpackages. |

|  |
| --- |
| The import keyword is used to make the classes and interface of another package accessible to the current package. |

**Example of package that import the packagename.\***

1. //save by A.java
3. package pack;
4. public class A{
5. public void msg(){System.out.println("Hello");}
6. }
7. //save by B.java
9. package mypack;
10. import pack.\*;
12. class B{
13. public static void main(String args[]){
14. A obj = new A();
15. obj.msg();
16. }
17. }

**2) Using packagename.classname**

If you import package.classname then only declared class of this package will be accessible.

**Example of package by import package.classname**

1. //save by A.java
3. package pack;
4. public class A{
5. public void msg(){System.out.println("Hello");}
6. }
7. //save by B.java
9. package mypack;
10. import pack.A;
12. class B{
13. public static void main(String args[]){
14. A obj = new A();
15. obj.msg();
16. }
17. }

**3) Using fully qualified name**

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

**Example of package by import fully qualified name**

1. //save by A.java
3. package pack;
4. public class A{
5. public void msg(){System.out.println("Hello");}
6. }
7. //save by B.java
9. package mypack;
10. class B{
11. public static void main(String args[]){
12. pack.A obj = new pack.A();//using fully qualified name
13. obj.msg();
14. }
15. }

Output:Hello

