OOPs Concept

List of OOPs concept

- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Inheritance

When one object acquires properties of other objects. It is used to add additional functionalities to existing class and for reusability of code. In Java we achieve this using extends (one class to another) and implements (one interface to another) key word. One of the main reason of using is method overriding so that run time polymorphism can be achieved.

1. A class cannot extend itself, will be a compile time error.
2. Only methods and variables of parent class can be inherited to child class (depends on access modifiers also, will be discussed later). SIB(static initialization block), IIB (instance initialization block)and Constructors cannot be inherited but they are executed while creating object of child class.

```
class A
{
		int i;

		static
		{
				System.out.println("Class A SIB");
		}

		{
				System.out.println("Class A IIB");
		}

		A()
		{
				System.out.println("Class A Constructor");
		}
}

class B extends A
{
		int j;
}

public class Testing {

		public static void main(String[] args) {
				// TODO Auto-generated method stub
				B b=new B();


		}
```

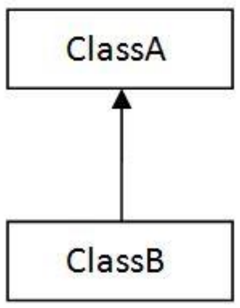
Output will be :

Class A SIB, Class A IIB, Class A Constructor

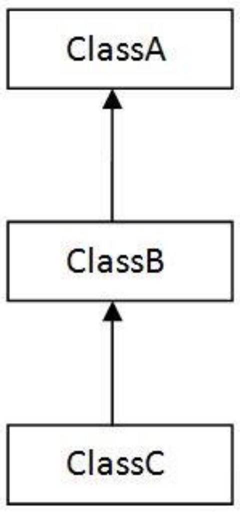
- ***3.*** Static members of super class are inheriting to sub class as static members and non-static members are inheriting as non-static members only.
- ***4.*** In Java every class we create extends to java.lang.Object super class.

Types of Inheritance in Java

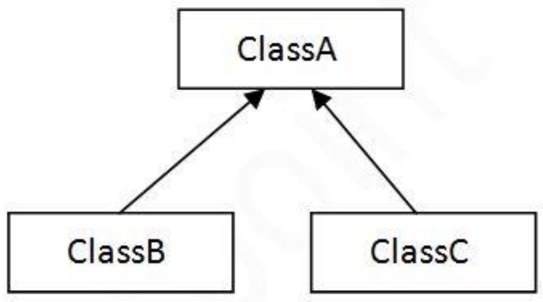
1. Single Inheritance: one class is extended by one class only.
2. Multilevel Inheritance: one class is extended by others class and that is extend by other class thus forming chain of inheritance.
3. Hierarchical Inheritance: one class is extended by many classes.
4. Hybrid Inheritance: combination of above
5. Multiple Inheritance: one class extends multiple classes. It is not supported in Java due to ambiguity. We can achieve this using concept of Interface.



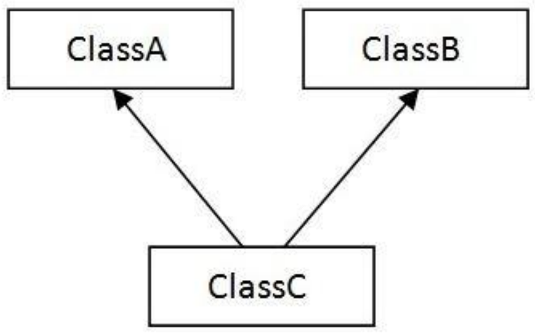
1) Single



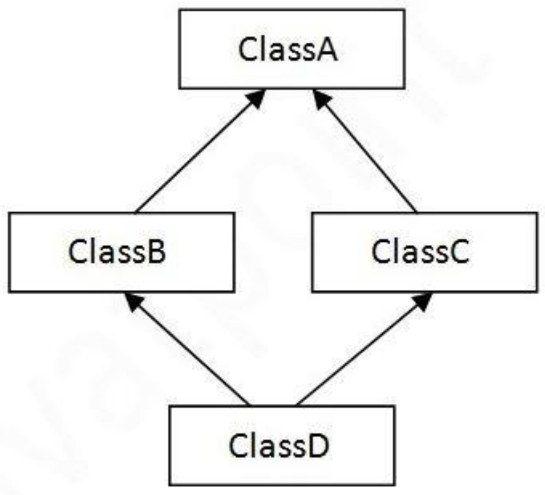
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

Effect of Access Modifiers on Inheritance

Sl No.	Access Modifier	Accessed	Used
01	Public	All subclasses inside and outside of package	All subclasses inside and outside of package
02	Protected	All subclasses inside and outside of package	***Protected members cannot be used outside the package***
03	Default	Only in current package	Only in current package
04	Private	No	No

Top Interview questions on Inheritance

1. What happens if super class and sub class having same field name?
   Ans : Super class field will be hidden in subclass. We can access this super class hidden field in subclass using super keyword.
2. You know that all classes in java are inherited from java.lang.Object class. Are interfaces also inherited from Object class.?
   Ans : No, only classes in java are inherited from Object class. Interfaces in java are not inherited from Object class. But, classes which implement interfaces are inherited from Object class.

Polymorphism

It means performing a single task in different way. In Java we achieve this using method overloading and method overriding.

Polymorphism has two types

Static Polymorphism/Static Binding/Early Binding:

Compile type polymorphism is called static polymorphism.

Example : Method Overloading, Constructor Overloading and operator Overloading (+ operator is used to add two numbers as well as for concatenation of two strings)

Method Overloading

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

Method overloading *increases the readability of the program*.

Method overloading can be achieved by two ways- by changing no of parameters and data type of argument

Why Method Overloading is not possible by changing the return type of method only?

.

Ans :

In method overloading return type of overloaded method can be changed it will not affect but changing only return type will give ambiguity error.

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur

```
class Adder{
static int add(int a,int b){return a+b;}
static double add(int a,int b){return a+b;}
}
class TestOverloading3{
public static void main(String[] args){
System.out.println(Adder.add(11,11));//ambiguity
}}
```

Which method will be called will be ambiguity. Compile time error.

Note : main method can be overloaded.

In method overloading type promotion happens if no match found.

```
class OverloadingCalculation1{
  void sum(int a,long 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[]){
  OverloadingCalculation1 obj=new OverloadingCalculation1();
  obj.sum(20,20);//now second int literal will be promoted to long
  obj.sum(20,20,20);
```

output : 40,60

Ambiguity while type promotion

```
class OverloadingCalculation3{
  void sum(int a,long b){System.out.println("a method invoked");}
  void sum(long a,int b){System.out.println("b method invoked");}

  public static void main(String args[]){
  OverloadingCalculation3 obj=new OverloadingCalculation3();
  obj.sum(20,20);//now ambiguity
```

Example of Method Overloading with Type Promotion if matching found

```
class OverloadingCalculation2{
  void sum(int a,int b){System.out.println("int arg method invoked");}
  void sum(long a,long b){System.out.println("long arg method invoked");}

  public static void main(String args[]){
  OverloadingCalculation2 obj=new OverloadingCalculation2();
  obj.sum(20,20);//now int arg sum() method gets invoked
  }
```

Method overloading example with different return type

```
public class MethodOverloading
{
    void methodOverloaded()
    {
        //No argument method, return type is void
    }

    int methodOverloaded(int i)
    {
        //Returns int type
        return i;
    }

    int methodOverloaded(double d)
    {
        //Same return type as of above method
        return 0;
    }

    void methodOverloaded(double d)
    {
        //Duplicate method because it has same method signature as of above
method
```

Return type does not affects overloading but Changing only return type and not changing other no arguments and data type of argument gives ambiguity

Note : Overloaded methods may be static or non-static. This also does not effect method overloading.

Note : Access modifiers does not have any effect on method overloading.

Top Interview questions on method overloading

1. **Can overloaded method be overrided?**

Ans : Yes, we can override a method which is overloaded in super class.

2. **Can we declare overloaded methods as final?**

Ans : Yes, we can declare overloaded methods as final.

3. **Can overloaded methods be synchronized?**

Ans : Yes. Overloaded methods can be synchronized.

4. **What is method signature? What are the things it consist of?**

Ans : Method signature is used by the compiler to differentiate the methods. Method signature consist of three things.

a) Method name

b) Number of arguments

c) Types of arguments

Note : It does not consists of return type and thrown exceptions

5. **Is it possible to have two methods in a class with same method signature but different return types?**

Ans : No, compiler will give duplicate method error. Compiler checks only method signature for duplication not the return types. If two methods have same method signature, straight away it gives compile time error.

6. **How do compiler differentiate overloaded methods from duplicate methods?**

Ans : Compiler uses method signature to check whether the method is overloaded or duplicated. Duplicate methods will have same method signatures i.e same name, same number of arguments and same types of arguments. Overloaded methods will also have same name but differ in number of arguments or else types of arguments.

Dynamic Polymorphism/Dynamic Binding/Late Binding

Runtime polymorphism is called Dynamic Polymorphism.

Example : Method Overriding

Method Overriding

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**. It is used to provide specific implementation or add extra feature as required by subclass.

```
package insurance;

public class GeneralInsurance {

		public int calculateInsurancePremium(int age, int planId)
		{
			return 0;
		}
		public int calculateInsurancePremium(int age, int planId, double subsidyPercentage)
		{
			return 0;
		}

}

class Cigna extends GeneralInsurance{

		public int calculateInsurancePremium(int age, int planId)
		{
			return 1000;
		}

		public int calculateInsurancePremium(int age, int planId, double subsidyPercentage)
		{
			return 800;
		}

		class Kaiser extends GeneralInsurance
		{
			public int calculateInsurancePremium(int age, int planId)
			{
				return 1200;
			}

			public int calculateInsurancePremium(int age, int planId, double subsidyPercentage)
			{
				return 700;
			}

		}


}

class Insurance{
		public static void main(String args[]) {
			GeneralInsurance gi=new Cigna();
			System.out.println(gi.calculateInsurancePremium(12, 2));--o/p 1000
			System.out.println(gi.calculateInsurancePremium(19, 78, 34)); o/p- 800
			GeneralInsurance gi1=new Kaiser();
			System.out.println(gi1.calculateInsurancePremium(56, 12)); o/p-1200
			System.out.println(gi1.calculateInsurancePremium(23, 45, 34)); o/p-700
```

In above example calculateInsurancePremium() method is overloaded in GeneralInsurance class and both are overridden in Cigna and Kaiser class. Which method will be called is decided at runtime.

Why we go for method overriding?

1. To achieve specific implementation of a method in subclass
2. Code reusability with adding some extra feature
3. To achieve dynamic polymorphism

Rules for method overriding

- method must have same name as in the parent class
- method must have same parameter as in the parent class.
- must be IS-A relationship (inheritance).
- You can keep same visibility or increase the visibility of overridden method but you can't reduce the visibility of overridden methods in the subclass.
- Return type of overridden method must be same in case of primitive while in case of Object return type It may be same or subclass. In case of subclass, is known as covariant return type. It was introduced as new feature in Java 5.
- Static methods cannot be overridden as we override super class methods in sub class but static method is class property so no question for overriding.
- Main method cannot be overridden as it is static.

ExceptionHandling with MethodOverriding

If the superclass method declares an exception	If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.
If the superclass method does not declare an exception	subclass overridden method cannot declare the checked exception but can declare unchecked exception.

Top Interview questions on method overriding

1. **What happens if we change the arguments of overriding method?**

Ans : If we change the arguments of overriding method, then that method will be treated as overloaded not overridden.

2. **Can we change the return type of overriding method from Number type to Integer type?**

Ans : Yes. You can change as Integer is a sub class of Number type. (covariant return type)

3. **Can we override a super class method without throws clause as a method with throws clause in the sub class?**

Ans : Yes, but only with unchecked type of exceptions.

4. **Can we change an exception of a method with throws clause from SQLException to NumberFormatException while overriding it?(VVI)**

Ans : Yes. Overridden method may throw SQLException or it's sub class exception or any unchecked type of exceptions.

5. **Can we change an exception of a method with throws clause from unchecked to checked while overriding it?**

Ans : No. We can't change an exception of a method with throws clause from unchecked to checked.

6. **How do you refer super class version of overridden method in the sub class?**

Ans : Using super keyword, we can refer super class version of overridden method in the sub class.

7. **Can we override private methods?**

Ans :No

8. **Can we remove throws clause of a method while overriding it?**

Ans : Yes. You can remove throws clause of a method while overriding it.

9. **Is it possible to override non-static methods as static?**

Ans : No. You can't override non-static methods as static.

10. **Can we change an exception of a method with throws clause from checked to unchecked while overriding it?**

Ans : Yes. We can change an exception from checked to unchecked but reverse is not possible.

11. **Can we change the number of exceptions thrown by a method with throws clause while overriding it?**

Ans : Yes, we can change. But, exceptions must be compatible with throws clause in the super class method.

Abstraction

It means hiding internal details and showing functionality only. In Java we achieve this using concept of Abstract class (Partially) and Interface (100%)

Abstract Class

Abstraction in java is used to define only ideas in one class so that the idea can be implemented by its sub classes according to their requirements. Abstract keyword is used make a class abstract.

Syntax : abstract class Bank

```
abstract class Bank
{
    abstract void calculateInterest();  // It is just an idea
}

class SBI extends Bank
{
    void calculateInterest()
    {
        System.out.println("4%");
        //Implementation of the idea according to requirements of sub
of sub class
    }
}
```

Important points regarding abstract class :

- ✓ Abstract class can't be instantiated even though class contains only concrete methods. An abstract class can contain both abstract as well as concrete methods. Even final methods also.
- ✓ Any class extending an abstract class must implement all abstract methods. If it does not implement, it must be declared as abstract.
- ✓ It is not compulsory that abstract class must have abstract methods. It may or may not have abstract methods. But the class which has at least one abstract method must be declared as abstract.
- ✓ we can create objects to sub classes of abstract class, provided they must implement abstract methods.
- ✓ Inside abstract class, we can keep any number of constructors. If you are not keeping any constructors, then compiler will keep default constructor.

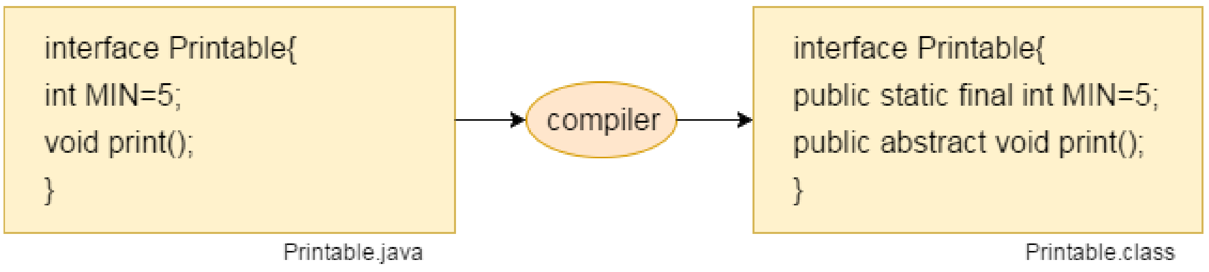
- ✓ Abstract methods can’t be private. Because, abstract methods must be implemented somehow in the sub classes. If you declare them as private, then you can’t use them outside the class.
- ✓ Abstract Methods can’t be static.
- ✓ Abstract class can provide implementation of interface.
- ✓ final and abstract are totally opposite in nature. A final class or method can not be modified further where as abstract class or method must be modified further. “final” keyword is used to denote that a class or method does not need further improvements. “abstract” keyword is used to denote that a class or method needs further improvements.

Interface

Interface in Java is mechanism to achieve 100% abstraction. It contains only abstract methods.

It represents a IS-A relationship.

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



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

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

Important Points :

1. An interface extends another Interface called Interface Inheritance.
2. An interface that have no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.
3. Marker interfaces give instructions to JVM that classes implementing them will have special behavior and must be handled with care.
4. By default, Every field of an interface is public, static and final (we will discuss about final keyword Later). You can’t use any other modifiers other than these three for a field of an interface.
5. Like classes, for every interface .class file will be generated after compilation.

6. While implementing any interface methods inside a class, that method must be declared as public. Because, according to method overriding rule, you can't reduce visibility of super class method. By default, every member of an interface is public and while implementing you should not reduce this visibility.
7. As we all know that, any class in java can not extend more than one class. But class can implement more than one interfaces. This is how **multiple inheritance** is implemented in java.
8. SIB – Static Initialization Block and IIB – Instance Initialization Block are not allowed in interfaces.

Java Interview Questions On Interfaces :

1) Can interfaces have constructors, SIB and IIB?

No. Interfaces can't have constructors, SIB and IIB. They show 100% abstractness.

2) Can we re-assign a value to a field of interfaces?

No. The fields of interfaces are static and final by default. They are just like constants. You can't change their value once they got.

3) Can we declare an Interface with "abstract" keyword?

Yes, we can declare an interface with "abstract" keyword. But, there is no need to write like that. All interfaces in java are abstract by default.

4) For every Interface in java, .class file will be generated after compilation. True or false?

True. .class file will be generated for every interface after compilation.

5) Can we override an interface method with visibility other than public?

No. While overriding any interface methods, we should use public only. Because, all interface methods are public by default and you should not reduce the visibility while overriding them.

6) Can interfaces become local members of the methods?

No. You can't define interfaces as local members of methods like local inner classes. They can be a part of top level class or interface.

7) Can an interface extend a class?

No, a class can not become super interface to any interface. Super interface must be an interface. That means, interfaces don't extend classes but can extend other interfaces.

8) Like classes, does interfaces also extend Object class by default?

No. Interfaces don't extend Object class.

9) Can interfaces have static methods?

No. Interfaces can't have static methods.

10) Can an interface have a class or another interface as it's members?

Yes. Interfaces can have classes or interfaces as their members.

Now Here Nested Interface and is interface extend object class javaconceptoftheday.

Encapsulation

It means binding data and code together in a single unit. In Java bean class is example of it for which all variables are private.