**[Top 15 Oops Concepts Interview Questions](http://www.instanceofjava.com/2015/03/oops-concepts-interview-questions.html)**

.What are the oops concepts in java?

* Oop is an approach that provides a way of modularizing a program by creating partitioned memory area for both data and methods that can be used as template for creating copies of such modules on demand.
* [The four oops concepts are](http://www.instanceofjava.com/2014/09/oops-oops-concepts-are-rules-which-are.html)
* [Encapsulation](http://www.instanceofjava.com/2015/03/program-on-encapsulation.html)
* [Polymorphism](http://www.instanceofjava.com/2015/03/polymorphism-definition-program.html)
* [Inheritance](http://www.instanceofjava.com/2015/03/inheritance-definition-and-programs.html)
* [Abstraction](http://www.instanceofjava.com/2016/02/abstract-method-class-example-programs.html)

2. What is encapsulation?

* The process of binding the data with related methods known as encapsulation.
* Class is the base for encapsulation.
* [Check here for more points on Encapsulation](http://www.instanceofjava.com/2014/09/encapsulation-need-for-encapsulation.html)

3.What is class ?

* A class is a specification or blue print or template of an object.
* Class is a logical construct , an object has physical reality.
* Class is a structure.
* Class is a user defined data type in java
* Class will acts as base for encapsulation.
* Class contains variables and methods.

1. package com.instanceofjava;
3. class Demo{
5. int a,b;
6. void show(){
7. }
9. }

4. What is an object?

* Object is instance of class.
* Object is dynamic memory allocation of class.
* Object is an encapsulated form of all non static variables and non static methods of a particular class.
* The process of creating objects out of class is known as instantiation.

1. package com.instanceofjava;
3. class Test{
5. int a,b;
6. void print(){
7. System.out.println("a="+a);
8. System.out.println("b="+b);
9. }
11. public static void main(String [] args){
13. Test obj= new Test();
14. obj.a=10;
15. obj.b=20;
16. obj.print();
17. }
18. }

Output:

1. a=10
2. b=20

5. What are the Object Characteristics?

* The three key characteristics of Object are
* State
* Behavior
* Identity

State:

* Instance variables value is called object state.
* An object state will be changed if instance variables value is changed.

Behavior:

* Behavior of an object is defined by instance methods.
* Behavior of an object is depends on the messages passed to it.
* So an object behavior depends on the instance methods.

Identity:

* Identity is the hashcode of an object, it is a 32 bit integer number created randomly and assigned to an object by default by JVM.
* Developer can also generate hashcode of an object based on the state of that object by overriding hashcode() method of java.lang.Object class.
* Then if state is changed , automatically hashcode will be changed.

6.What is Inheritance?

* As the name suggests , inheritance means to take something that already made.
* One of the most important feature of Object oriented Programming. It is the concept that is used for re usability purpose.
* Getting the properties from one class object to another class object.

7. How inheritance implemented in java?

* Inheritance can be implemented in JAVA using below two keywords.  
  1.extends  
  2.implements
* extends is used for developing inheritance between two classes or two interfaces, and implements keyword is used to develop inheritance between interface and class.

1. package com.instanceofjava;
2. class A{
4. }

1. package com.instanceofjava;
2. class B extends A{
4. }

8. What are the types of inheritances?

* There are two types of inheritance  
  1.Multilevel Inheritance  
  2.Multiple Inheritance

Multilevel Inheritance:

* Getting the properties from one class object to another class object level wise with some priority is known as multilevel inheritance.

1. package com.instanceofjava;
3. class A{
5. }
7. class B extends A{
9. }
11. class C extends B{
13. }

Multiple Inheritance:

* The concept of getting multiple class objects to single class object is known as multiple inheritance. multiple inheritance is not supported by java
* [Check here for Why java does not supports multiple inheritance](http://www.instanceofjava.com/2014/12/why-java-does-not-supports-multiple.html)

9. What is polymorphism?

* Defining multiple methods with same name,

**Static polymorphism:**

* Defining multiple methods with same name with different parameters**.**
* Is also known as method overloading.

1. package com.instanceofjava;
2. class Demo{
4. void add(){
5. }
7. void add(int a, int b){
8. }
10. void add(float a, float b){
12. }
13. public static void main(String [] args){
14. Demo obj= new Demo();
16. obj.add();
17. obj.add(1,2);
18. obj.add(1.2f,1.4f);
19. }
20. }

**Dynamic Polymorphism:**

* Defining multiple methods with same signature in super class and sub class.
* The sub most object method will be executed always.
* [what is the difference between method overloading and method overriding?](http://www.instanceofjava.com/2014/12/what-is-difference-between-method.html)

 10. Similarities and differences between this and super keywords?

**this:**

* This is a keyword used to store current object reference.
* It must be used explicitly if non -static variable and local variables name is same.
* System.out.print(this); works fine

**super:**

* Super is a keyword used to store super class non -static members reference in sub class object.
* used to separate super class and sub class members if both have same name.
* System.out.println(super); compilation Error

[**Top 10 Interview Programs and questions on method overriding in java**](http://www.instanceofjava.com/2016/02/method-overriding-in-java-definition.html)

Posted by: InstanceOfJava Posted date: **Feb 17, 2016** / comment : 0

**1.What is method overriding in java?**

* Defining multiple methods with same name and same signature in super class and sub class known as method overriding.
* Method overriding is type of polymorphism in java which is one of the main object oriented feature.
* Redefined the super class method in sub class is known as method overriding.
* method overriding allows sub class to provide specific implementation that is already defined in super class.
* Sub class functionality replaces the super class method functionality (implementation).

**2. Can we override private methods in java?**

* No. It’s not possible to override private methods because private methods in super class will not be inherited to sub class.
* [Read Can a private method in super class be overridden in Sub class?](http://www.instanceofjava.com/2015/07/can-we-override-private-method-in-java.html)

**3. Can we override static methods of super class in sub class?**

* NO. It’s not possible to override static methods because static means class level so static methods not involve in inheritance.
* [Can we override static methods in java](http://www.instanceofjava.com/2015/06/can-we-override-static-methods-in-java.html)

**4. Can we change the return type of overridden method in sub class?**

* No. Return type must be same in super class and sub class.

1. package MethodOverridingExamplePrograms;
2. public class Super{
4. void add(){
5. System.out.println("Super class add method");
6. }
7. }
8. package MethodOverridingInterviewPrograms;
9. public class Sub extends Super{
11. int add(){    //Compiler Error: The return type is incompatible with Super.add()
13. System.out.println("Sub class add method");
14. return 0;
15. }
16. }

**5. Can we change accessibility modifier in sub class overridden method?**

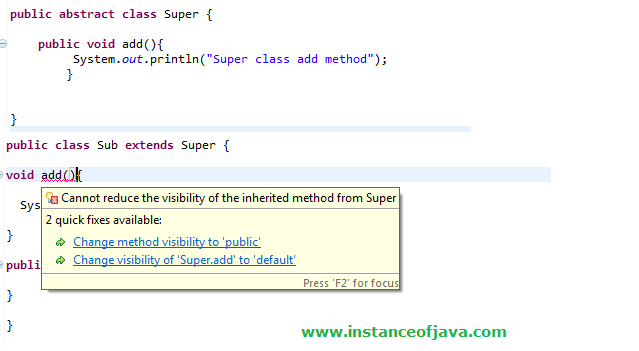
* Yes we can change accessibility modifier in sub class overridden method but should increase the accessibility if we decrease compiler will throw an error message.

| **Super class method** | **Subclass method** |
| --- | --- |
| protected | protected, public |
| public | public |
| default | default , public |

**6.What happens if we try to decrease accessibility from super class to sub class?**

* Compile time error will come.

1. package MethodOverridingExamplePrograms;
2. public class Super{
4. public void add(){
5. System.out.println("Super class add method");
6. }
7. }
8. package MethodOverridingInterviewPrograms;
9. public class Sub extends Super{
11. void add(){ //Compiler Error: Cannot reduce the visibility of the inherited method
12. from Super
14. System.out.println("Sub class add method");
15. }
16. }

[](https://4.bp.blogspot.com/-1-mreDnk1RA/VsQcJuh1KVI/AAAAAAAAAk4/K0U7cHNpx3g/s1600/method+overriding.png)

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

* Yes if super class method throws unchecked exceptions.
* No if super class method throws checked exceptions.

**8. What are the rules we need to follow in overriding if super class method throws exception?**

* If sub class throws checked exception super class should throw same or super class exception of this.
* If super class method throws checked or unchecked exceptions it’s not mandatory to put throws in sub class overridden method.
* If super class method throws exceptions in sub class if you want to mention throws then use same class or its sub class exception.

**9.What happens if we not follow these rules if super class method throws some exception.**

* Compile time error will come.

1. package MethodOverridingExamplePrograms;
2. public class Super{
4. public void add(){
5. System.out.println("Super class add method");
6. }
7. }
8. package MethodOverridingInterviewPrograms;
9. public class Sub extends Super{
11. void add() throws Exception{ //Compiler Error: Exception Exception is not compatible with
12. throws clause in Super.add()
13. System.out.println("Sub class add method");
14. }
15. }
16. package MethodOverridingExamplePrograms;
17. public class Super{
19. public void add(){
20. System.out.println("Super class add method");
21. }
22. }
23. package MethodOverridingInterviewPrograms;
24. public class Sub extends Super{
26. void add() throws NullPointerException{ // this method throws unchecked exception so no
27. isuues
28. System.out.println("Sub class add method");
29. }
30. }

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

* No. As mentioned above already
* If super class method throws exceptions in sub class if you want to mention throws then use same class or its sub class exception.
* So we cannot change from unchecked to checked

[**Top 15 Java abstract class and abstract method interview questions and programs**](http://www.instanceofjava.com/2016/02/abstract-method-class-example-programs.html)

Posted by: InstanceOfJava Posted date: **Feb 16, 2016** / comment : 0

**1. What is abstract class in java?**

* Hiding the implementation and showing the function definition to the user.
* Abstract class contains abstract methods and concrete methods(normal methods)

**2. How can we define an abstract class?**

* Using abstract keyword we can define abstract class.
* Check below code for abstract class example program.

1. package Abstraction;
2. public **abstract**class AbstractDemo {
3. //
4. //
5. }

3. How to declare an abstract method?

* By Using abstract keyword in the method signature we can declare abstract method.

1. package Abstraction;
2. public **abstract**class AbstractDemo {
3. //
4. **abstract**void show();
5. }

 4. Can we define abstract class without abstract method?

* Yes we can define abstract class without abstract methods.
* It is not mandatory to create at-least one abstract method in abstract class.

1. package Abstraction;
2. public **abstract**class AbstractDemo {
3. //
5. }

 Read more at [Can you define an abstract class without any abstract methods? if yes what is the use of it?](http://www.instanceofjava.com/2014/12/can-you-define-abstract-class-without.html)

**5.Can we create object for abstract class?**

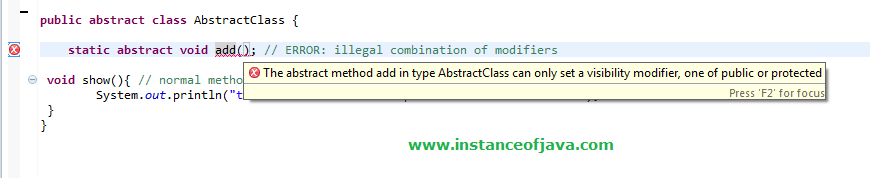
* Abstract class cannot be instantiated directly.
* Means we cannot create object for abstract class directly.
* Through sub class abstract class members will get memory. Whenever we create sub class object of abstract class abstract class object will be created. I.e. abstract class members will get memory at that time.

1. package Abstraction;
2. public abstract class AbstractClass {
4. abstract void add(); // abstract method
6. void show(){ // normal method
7. System.out.println("this is concrete method present in abstract class");
8. }
10. public static void main(String[] args){
12. AbstractClass obj= new AbstractClass ();
13. // ERROR: Cannot instantiate the type AbstractClass
15. }
16. }

**6. Is it possible to declare abstract method as static?**

* No. it’s not possible to declare abstract method with static keyword.
* If we declare abstract method with static compiler will throw an error.

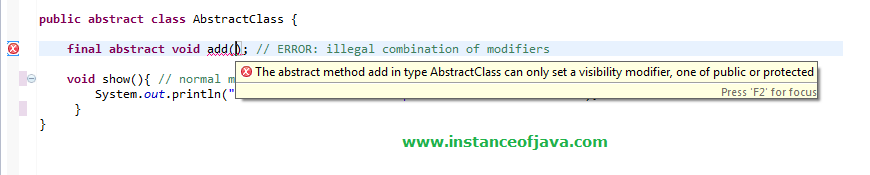
1. package Abstraction;
2. public abstract class AbstractClass {
4. static abstract void add(); // ERROR: illegal combination of modifiers
6. void show(){ // normal method
7. System.out.println("this is concrete method present in abstract class");
8. }
9. }

[](https://4.bp.blogspot.com/-W6mVFe1szLs/VsaY-6AvYjI/AAAAAAAAAlI/EnxKN0DuT08/s1600/abstract+static+errr.png)

**7. Can we declare abstract method as final?**

* No. it’s not possible to declare abstract method with final keyword.
* If we declare abstract method with final compiler will throw an error.
* Because abstract methods should be override by its sub classes.

1. package Abstraction;
2. public abstract class AbstractClassExample {
4. final abstract void add(); // ERROR: illegal combination of modifiers
6. void show(){ // normal method
7. System.out.println("this is concrete method present in abstract class");
8. }
9. }

[](https://4.bp.blogspot.com/-I8nublVRFBM/VsaaitJ4eYI/AAAAAAAAAlU/VUNvXbYnmFw/s1600/abstract+final.png)

**8. Is it possible to declare abstract method as private?**

* No. its not possible to declare abstract method with private .
* If we declare abstract method with private compiler will throw an error.
* Because abstract methods should be override by its sub classes.

1. package Abstraction;
2. public abstract class AbstractClassExample {
4. private abstract void add(); // ERROR: illegal combination of modifiers
6. void show(){ // normal method
7. System.out.println("this is concrete method present in abstract class");
8. }
9. }

**9. Is it possible to declare abstract method as public ?**

* Yes we can declare abstract methods as public.

1. package Abstraction;
2. public abstract class AbstractClassExample {
4. public abstract void add();
6. void show(){ // normal method
7. System.out.println("this is concrete method present in abstract class");
8. }
9. }

**10. Is it possible to declare abstract method with default?**

* Yes we can declare abstract methods with default.

1. package Abstraction;
2. public abstract class AbstractClassExample {
4. abstract void add();
6. void show(){ // normal method
7. System.out.println("this is concrete method present in abstract class");
8. }
9. }

**11. Is it possible to declare abstract method with protected modifier?**

* Yes we can declare abstract methods as protected.

1. package Abstraction;
2. public abstract class AbstractClassExample {
4. protected abstract void add();
6. void show(){ // normal method
7. System.out.println("this is concrete method present in abstract class");
8. }
9. }

**12.What are the valid and invalid keywords or modifier with abstract class?**

* public,  protected and default are valid.
* static,  final and private are invalid.

**13.Can abstract method declaration include throws clause?**

* Yes. We can define an abstract class with throws clause.

1. package Abstraction;
2. public abstract class AbstractClassExample {
4. abstract void add() throws Exception;
6. void show(){ // normal method
7. System.out.println("this is concrete method present in abstract class");
8. }
9. }

**14.What happens if sub class not overriding abstract methods?**

* If sub class which is extending abstract class not overriding abstract method compiler will throw an error.

1. package Abstraction;
2. public abstract class AbstractClass {
4. abstract void add() throws Exception;
6. }
7. package Abstraction;
8. public class Sample extends AbstractClass {
10. //Compiler Error: The type Sub must implement the inherited abstract method Super.add()
11. }

**15. Can we escape of overriding abstract class in sub class which is extending abstract class?**

* Yes we can escape from overriding abstract method from super abstract class by making our class again as abstract.
* The class which is extending our class will get responsibility of overriding all abstract methods in our class and in super class.

1. package Abstraction;
2. public abstract class AbstractClass {
4. abstract void add() throws Exception;
6. }
7. package Abstraction;
8. public abstract class Sample extends AbstractClass {

11. }
12. package Abstraction;
13. public class Example extends Sample{
15. //Compiler Error: The type Sub must implement the inherited abstract method Super.add()
16. }

[**Top 10 interview questions on java interfaces**](http://www.instanceofjava.com/2016/03/java-interface-interview-questions.html)

Posted by: InstanceOfJava Posted date: **Mar 1, 2016** / comment : 2

**1.What is an interface in Java.**

* Before Java 8 interfaces are pure abstract classes which allow us to define public static final variables and public abstract methods(by default).
* In java 8 introduced static methods and default methods in interfaces.
* [**Java 8 Interface Static and Default Methods**](http://www.instanceofjava.com/2015/02/java-8-interface-static-default-methods.html)
* We can develop interfaces by using "interface" keyword.
* A class will implement all the methods in an interface.

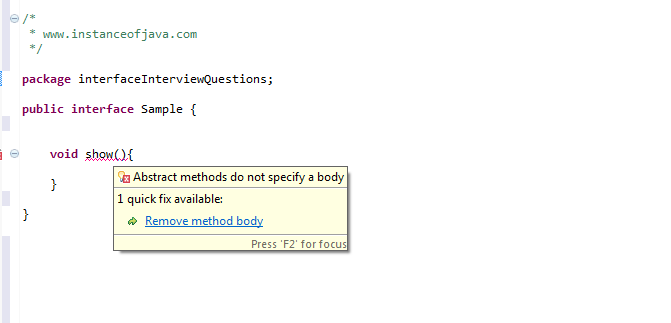
1. package com.instanceofjava;
2. interface JavaInterface{
4. int x=12;
5. void show();
6. }
7. package com.instanceofjava;
8. interface A implements JavaInterface {
10. void show(){
11. // code
12. }
13. }

1. package com.instanceofjava;
2. interface Java8Interface{
4. int x=12;
5. void show();
7. **default**void display(){
9. System.out.println("default method of interface");
11. }
13. **Static**void print(String str){
15. System.out.println("Static method of interface:"+str);
17. }

20. }

**2. What will happen if we define a concrete method in an interface?**

* By default interface methods are abstract.
* If we declare any concrete method in an interface compile time error will come.
* Error: Abstract methods do not specify a body

[](https://4.bp.blogspot.com/-WlTDjb2dQVg/VtWXfP3-RDI/AAAAAAAAAng/t1gYP8Ll6ak/s1600/interface+concrete+method.png)

**3.Can we create non static variables in an interface?**

* No. We cannot create non static variables in an interface.
* If we try to create non static variables compile time error comes.
* By default members will be treated as public static final variables so it expects some value to be initialized.

1. package com.instanceofjava;
2. interface JavaInterface{
4. int x, y; // compile time error
5. }

**4.What will happen if we not initialize variables in an interface.**

* Compile time error will come because by default members will be treated as public static final variables so it expects some value to be initialized.

1. package com.instanceofjava;
2. interface JavaInterface{
4. int x, y; // compile time error: The blank final field y may not have been initialized
5. }

**5. Can we declare interface members as private or protected?**

* No.

1. package com.instanceofjava;
2. interface JavaInterface{
4. private int x; // compile time error: Illegal modifier for the interface field Sample.x; only
5. public, static & final are permitted
6. protected int a; // compile time error: Illegal modifier for the interface field Sample.a; only
7. public, static & final are permitted
8. }

**7. When we need to use extends and implements?**

* A class will implements an interface.
* A class will extend another class.
* An interface extends another interface.

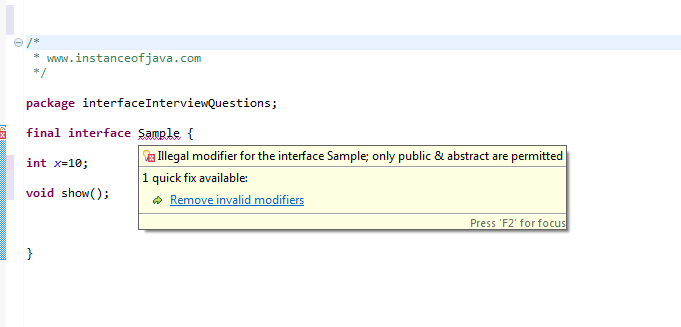
**6.Can we create object for an interface?**

* NO. We cannot create object for interface.
* We can create a variable from an interface

1. package com.instanceofjava;
2. interface JavaInterface{
4. void show();
5. }
6. package com.instanceofjava;
7. interface A implements JavaInterface {
9. void show(){
10. // code
11. }
12. public static void main(String args[]){
14. JavaInterface obj= new JavaInterface(); // Error: Cannot instantiate the type JavaInterface
16. }
17. }

**7.Can we declare interface as final?**

* No. Compile time error will come.
* Error: Illegal modifier for the interface Sample; only public & abstract are permitted

[](https://3.bp.blogspot.com/-Os3PkMKXd4g/VtWhsViGPQI/AAAAAAAAAnw/vSUj5KzjNO4/s1600/interface+final.png)

**8.Can we declare constructor  inside an interface?**

* No. Interfaces do not allow constructors.
* The variables inside interfaces are static final variables means constants and we can not create object from interface so there is no need of constructor in interface that is the reason interface doesn't allow us to create constructor.

[](https://3.bp.blogspot.com/-7C_DF5VxRAs/Vs7bWEG5dVI/AAAAAAAAAm8/vnVVgGOZu4Y/s1600/interface+constructor.png)

**9.What will happen if we are not implementing all the methods of an interface in class which implements an interface?**

* A class which implements an interface should implement all the methods (abstract) otherwise compiler will throw an error.
* The type Example must implement the inherited abstract method JavaInterface.show()
* If we declare class as abstract no need to implement methods.
* No need of overriding default and static methods.

1. package com.instanceofjava;
2. interface JavaInterface{
4. void show();

* }

1. package com.instanceofjava;
2. interface A implements JavaInterface { // The type Example must implement the inherited
3. abstract method JavaInterface.show()
5. public static void main(String args[]){
7. }
8. }

**10.How can we access same variables defined in two interfaces implemented by a class?**

* By Using corresponding interface.variable\_name we can access variables of corresponding interfaces.

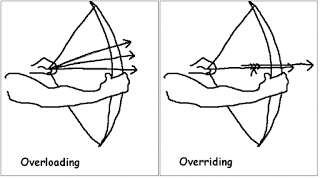
**11.If  Same method is defined in two interfaces can we override this method in class implementing these interfaces.**

* Yes implementing the method once is enough in class.
* A class cannot implement two interfaces that have methods with same name but different return type.

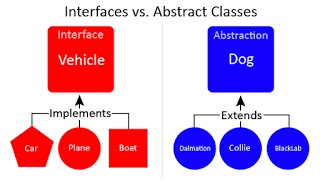
## OOPS Concept Interview Questions in Java

**What is method overloading in OOP or Java?**([answer](http://java67.blogspot.sg/2012/08/what-is-method-overloading-in-java-example.html))  
It's one of the oldest OOPS concept questions, I have seen it 10 years ago and still sees it now. When we have multiple methods with the same name but different functionality then it's called method overloading. For example. System.out.println() is overloaded as we have 6 or 7 println() method each accepting a different type of parameter.

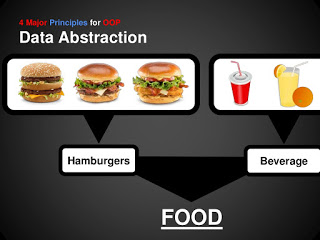
**What is method overriding in OOP or Java?** ([answer](http://java67.blogspot.sg/2012/08/what-is-method-overriding-in-java-example-tutorial.html))  
It's one of the magic of object oriented programming where the method is chose based upon an object at runtime. In order for method overriding, we need Inheritance and Polymorphism, as we need a method with the same signature in both superclass and subclass. A call to such method is resolved at runtime depending upon the actual object and not the type o variable. See the answer for more detailed discussion.  
  
  
**What is method hiding in Java?**(answer)  
When you declare two static methods with same name and signature in both superclass and subclass then they hide each other i.e. a call to the method in the subclass will call the static method declared in that class and a call to the same method is superclass is resolved to the static method declared in the super class.  
  
  
**Is Java a pure object oriented language? if not why?** ([answer](http://java67.blogspot.com/2014/03/is-java-pure-object-oriented-programming-language.html))  
Java is not a pure object-oriented programming language e.g. there are many things you can do without objects e.g. static methods. Also, primitive variables are not objects in Java. See the answer for a more detailed explanation.  
  
  
**What are rules of method overloading and overriding in Java?**([answer](http://java67.blogspot.sg/2012/09/what-is-rules-of-overloading-and-overriding-in-java.html))  
One of the most important rule of method overloading in Java is that method signature should be different i.e. either the number of arguments or the type of arguments. Simply changing the return type of two methods will not result in overloading, instead compiler will throw an error. On the other hand, method overriding has more rules e.g. name and return type must be same, method signature should also be same, the overloaded method cannot throw a higher exception etc. See the answer for a full list of rules related to method overloading and overriding in Java.  
  
  
**The difference between method overloading and overriding?** ([answer](http://java67.blogspot.sg/2012/09/difference-between-overloading-vs-overriding-in-java.html))  
Several differences but the most important one is that method overloading is resolved at compile time and method overriding is resolved at runtime. The compiler only used the class information for method overloading, but it needs to know object to resolved overridden method calls. This diagram explains the difference quite well, though:

[](http://3.bp.blogspot.com/-OCgsVIfnteA/Vmg9Ep7CmII/AAAAAAAAESY/Wk9ERXhb_Tg/s1600/Difference+between+method+overloading+and+overriding+in+Java.gif)

**Can we overload a static method in Java?** ([answer](http://java67.blogspot.sg/2012/08/can-we-overload-static-method-in-java.html))  
Yes, you can overload a static method in Java. You can declare as many static methods of the same name as you wish provided all of them have different method signatures. See the answer for more detailed explanation and code example.  
  
  
**Can we override static method in Java?**([answer](http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html))  
No, you cannot override a static method because it's not bounded to an object. Instead, static methods belong to a class and resolved at compile time using the type of reference variable. But, Yes, you can declare the same static method in a subclass, that will result in method hiding i.e. if you use reference variable of type subclass then new method will be called, but if you use reference variable of superclass than old method will be called.  
  
  
**Can we prevent overriding a method without using the final modifier?** (answer)  
Yes, you can prevent the method overriding in Java without using the final modifier. In fact, there are several ways to accomplish it e.g. you can mark the method private or static, those cannot be overridden.  
  
  
**Can we override a private method in Java?**([answer](http://java67.blogspot.sg/2013/08/can-we-override-private-method-in-java-inner-class.html))  
No, you cannot. Since the private method is only accessible and visible inside the class they are declared, it's not possible to override them in subclasses. Though, you can override them inside the inner class as they are accessible there.  
  
  
**What is covariant method overriding in Java?**([answer](http://javarevisited.blogspot.com/2014/03/covariant-method-overriding-of-java-5.html))  
In covariant method overriding, the overriding method can return the subclass of the object returned by original or overridden method. This concept was introduced in Java 1.5 (Tiger) version and it's very helpful in case original method is returning general type like Object class, because, then by using covariant method overriding you can return more suitable object and prevent client side type casting. One of the practical use of this concept is in when you override the clone() method in Java.  
  
  
**Can we change the return type of method to subclass while overriding?**(answer)  
Yes, you can, but only from Java 5 onward. This feature is known as co-variant method overriding and it was introduced in JDK 5 release. This is immensely helpful if original method return super-class e.g. clone() method return java.lang.Object. By using this, you can directly return the actual type, preventing client-side type casting of the result.  
  
  
**Can we change the argument list of an overriding method?**([answer](http://javarevisited.blogspot.com/2011/08/what-is-polymorphism-in-java-example.html))  
No, you cannot. The argument list is part of the method signature and both overriding and overridden method must have the same signature.  
  
  
**Can we override a method which throws runtime exception without throws clause?** ([answer](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html))  
Yes, there is no restriction on unchecked exception while overriding. On the other hand, in the case of checked exception, an overriding exception cannot throw a checked exception which comes higher in type hierarchy e.g. if original method is throwing IOException than overriding method cannot throw java.lang.Exception or java.lang.Throwable.  
  
  
**How do you call superclass version of an overriding method in sub class? (**answer**)**  
You can call a superclass version of an overriding method in the subclass by using super keyword. For example to call the toString() method from java.lang.Object class you can call super.toString().  
  
  
**Can we override a non-static method as static in Java?**(answer)  
Yes, you can override the non-static method in Java, no problem on them but it should not be private or final :)  
  
  
**Can we override the final method in Java?** ([answer](http://javarevisited.blogspot.com/2013/12/when-to-make-method-final-in-java.html))  
No, you cannot override a final method in Java, final keyword with the method is to prevent method overriding. You use final when you don't want subclass changing the logic of your method by overriding it due to security reason. This is [why String class is final in Java](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html). This concept is also used in template design pattern where template method is made final to prevent overriding.  
  
  
**Can we have a non-abstract method inside interface?** (answer)  
From Java 8 onward you can have a non-abstract method inside interface, prior to that it was not allowed as all method was implicitly public abstract. From JDK 8, you can add static and default method inside an interface.  
  
  
**What is the default method of Java 8?** ([answer](http://javarevisited.blogspot.com/2014/07/default-defender-or-extension-method-of-Java8-example-tutorial.html))  
Default method, also known as extension method is new types of the method which you can add on the interface now. This method has implementation and intended to be used by default. By using this method, JDK 8 managed to provide common functionality related to [lambda expression](http://javarevisited.blogspot.com/2014/02/10-example-of-lambda-expressions-in-java8.html) and [stream API](http://javarevisited.blogspot.com/2014/03/2-examples-of-streams-with-Java8-collections.html)without breaking all the clients which implement their interfaces. If you look Java 8 API documentation you will find several useful default methods on key Java interface like Iterator, Map etc.  
  
  
**What is an abstract class in Java? (**[answer](http://java67.blogspot.sg/2014/06/why-abstract-class-is-important-in-java.html)**)**  
An abstract class is a class which is incomplete. You cannot create an instance of an abstract class in Java. They are provided to define default behavior and ensured that client of that class should adore to those contract which are defined inside the abstract class. In order to use it, you must extend and implement their abstract methods. BTW, in Java a class can be abstract without specifying any abstract method.  
  
  
**What is an interface in Java? What is the real user of an interface?**([answer](http://java67.blogspot.sg/2014/02/what-is-actual-use-of-interface-in-java.html))  
Like an abstract class, the interface is also there to specify the contract of an API. It supports OOP abstraction concept as it defines only abstract behavior. It will tell that your program will give output but how is left to implementors. The real use of the interface to define types to leverage Polymorphism. See the answer for more detailed explanation and discussion.  
  
  
**The difference between Abstract class and interface?** ([answer](http://java67.blogspot.sg/2012/09/what-is-difference-between-interface-abstract-class-java.html))  
In Java, the key difference is that abstract class can contain non-abstract method but the interface cannot, but from Java 8 onward interface can also contain static and default methods which are non-abstract. See the answer for more detailed discussion as I have described a lot of points there.

[](http://3.bp.blogspot.com/-iY5H3NVfu14/Vmg99BkXpPI/AAAAAAAAES0/hKRMe87q1P4/s1600/abstract+class+vs+interface+in+Java.png)

**Can we make a class abstract without an abstract method?** ([answer](http://javarevisited.blogspot.com/2013/04/10-abstract-class-and-interface-interview-question-java-answers.html))  
Yes, just add abstract keyword on the class definition and your class will become abstract.  
  
  
**Can we make a class both final and abstract at the same time?** ([answer](http://javarevisited.blogspot.com/2011/12/final-variable-method-class-java.html))  
No, you cannot apply both final and abstract keyword at the class same time because they are exactly opposite of each other. A final class in Java cannot be extended and you cannot use an abstract class without extending and making it a concrete class. As per Java specification, the compiler will throw an error if you try to make a class abstract and final at the same time.  
  
  
**Can we overload or override the main method in Java?** ([answer](http://java67.blogspot.com/2015/06/can-you-overload-or-override-main-in-java.html))  
No, since main() is a static method, you can only overload it, you cannot override it because the static method is resolved at compile time without needing object information hence we cannot override the main method in Java.  
  
  
**What is the difference between Polymorphism, Overloading, and Overriding?** ([answer](http://java67.blogspot.sg/2012/10/difference-between-polymorphism-overloading-overriding-java.html))  
This is slight tricky OOP concept question because Polymorphism is the real concept behind on both Overloading and Overriding. Overloading is compile time Polymorphism and Overriding are runtime Polymorphism.  
  
  
**Can an interface extend more than one interface in Java?**  
Yes, an interface can extend more than one interface in Java, it's perfectly valid.  
  
  
**Can a class extend more than one class in Java?**  
No, a class can only extend another class because Java doesn't support multiple inheritances but yes, it can implement multiple interfaces.  
  
  
**What is the difference between abstraction and polymorphism in Java?** ([answer](http://java67.blogspot.sg/2015/05/difference-between-abstraction-and.html))  
Abstraction generalizes the concept and Polymorphism allow you to use different implementation without changing your code. This diagram explains the abstraction quite well, though:

[](http://2.bp.blogspot.com/-hBvfoo4bOVY/Vmg9ZhHoQyI/AAAAAAAAESk/FUcbr0i1gDo/s1600/OOP+Abstraction.jpg)

## Object Oriented design principle and pattern Interview Questions

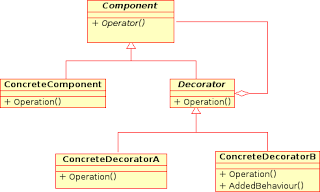
Now let's see some OOPS concept questions based upon the SOLID design principles and GOF design patterns which take advantage of OOPS concept discussed here.

**What problem is solved by Strategy pattern in Java?** ([answer](http://java67.blogspot.com/2014/12/strategy-pattern-in-java-with-sample.html))

Strategy pattern allows you to introduce new algorithm or new strategy without changing the code which uses that algorithm. For example, the Collections.sort() method which sorts the list of the object uses Strategy pattern to compare object. Since every object uses different comparison strategy you can compare various object differently without changing sort method.

**Which OOP concept Decorator design Pattern is based upon?**([answer](http://java67.blogspot.com/2013/07/decorator-design-pattern-in-java-real-life-example-tutorial.html))

Decorator pattern takes advantage of Composition to provide new features without modifying the original class. A very good to-the-point question for the telephonic round. This is quite clear from UML diagram of Decorator pattern, as you can see the Component is associated with Decorator.

[](http://2.bp.blogspot.com/-jnzC4Kx48Oc/Vmg-K8LVjqI/AAAAAAAAES8/sYAEghzm688/s1600/Decorator+Design+Pattern+in+Java+UML.png)

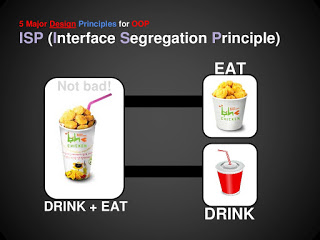
**When to use Singleton design pattern in Java?** ([answer](http://java67.blogspot.com/2012/08/what-is-singleton-pattern-in-java.html))  
When you need just one instance of a class and wants that to be globally available then you can use [Singleton pattern](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html). It's not free of cost though because it increase coupling between classes and make them hard to test. This is one of the oldest design pattern questions from Java interviews. Please see the answer for more detailed discussion.

**What is the difference between State and Strategy Pattern?**([answer](http://javarevisited.blogspot.com/2014/04/difference-between-state-and-strategy-design-pattern-java.html))  
Though the structure or class diagram of State and Strategy pattern is same, their intent is completely different. State pattern is used to do something specific depending upon state while [Strategy](http://java67.blogspot.com/2014/12/strategy-pattern-in-java-with-sample.html) allows you to switch between algorithms without changing the code which uses it.

**What is the difference between Association, Aggregation, and Composition in OOP?** ([answer](http://javarevisited.blogspot.com/2014/02/ifference-between-association-vs-composition-vs-aggregation.html))  
When an object is related to another object it called association. It has two forms, aggregation, and composition. The former is the loose form of association where the related object can survive individual while later is a stronger form of association where a related object cannot survive individually. For example, the city is an aggregation of people but is the composition of body parts.

**What is the difference between Decorator, Proxy and Adapter pattern in Java?**([answer](http://javarevisited.blogspot.com/2015/01/adapter-vs-decorator-vs-facade-vs-proxy-pattern-java.html))  
Again they look similar because their structure or class diagram is very similar but their intent is quite different. Decorator adds additional functionality without touching the class, Proxy provides access control and Adapter is used to make two incompatible interfaces work together.

**What is the 5 objects oriented design principle from SOLID?**([answer](http://javarevisited.blogspot.com/2012/03/10-object-oriented-design-principles.html))  
SOLID is the term given by Uncle Bob in his classic book, the [Clean Code](http://www.amazon.com/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882?tag=javamysqlanta-20), one of the must-read books for programmers. In SOLID each character stands for one design principle:  
S for Single Responsibility Principle  
O for Open closed design principle  
L for Liskov substitution principle  
I for Interface segregation principle  
D for Dependency inversion principle

[](http://2.bp.blogspot.com/-WwI87RZsjQs/Vmg9bXRGYYI/AAAAAAAAESs/NSOrLHWL670/s1600/OOP+-+Interface+Segregation+Principle.jpg)

**What is the difference between Composition and Inheritance in OOP?**([answer](http://javarevisited.blogspot.sg/2015/06/difference-between-inheritance-and-Composition-in-Java-OOP.html))

This is another great OOPS concept question because it test what matters, both of them are very important from a class design perspective. Though, both Composition and Inheritance allows you to reuse code, former is more flexible than later. Composition allows the class to get an additional feature at runtime, but Inheritance is static. You can not change the feature at runtime by substitution new implementation. See the answer for more detailed discussion