Java Inheritance Interview Questions

Inheritance
Java Inheritance Interview Questions, Programs and Examples.

Table of Contents:

Top Five Inheritance Questions Asked In Interviews?

Question: How To Describe Inheritance In Interview?

- Inheritance means one object acquires all the properties and behaviors of parent object.
- Inheritance is a compile-time mechanism.
- A super-class can have any number of subclasses. But a subclass can have only one superclass.
- The **extends keyword** indicates that you are making a new class that derives from an existing class.
- The superclass and subclass have **“is-a”** relationship between them.
- Inheritance is used For **Method Overriding** and For **Code Reusability.**

Question: Types of Inheritance in Java?

There are 5 types of Inheritance.

> **Single Inheritance**: One class is extended by only one class.
>
> **Multilevel Inheritance:** One class is extended by a class and that class in turn is extended by another class thus forming a chain of inheritance.
>
> **Hierarchical Inheritance**: One class is extended by many classes.
>
> **Hybrid Inheritance**: It is a combination of above types of inheritance.
>
> **Multiple Inheritance**: One class extends more than one classes. (Java does not support multiple inheritance.)

Question: Can a class extend more than one classes or does java support multiple inheritance? If not, why?

Java does not support multiple inheritance. This feature is avoided intentionally to avoid ambiguity, complexity and confusion.

For example, If Class C extends Class A and Class B which have a method with same name, then Class C will have two methods with same name.

This causes ambiguity and confusion for which method to use. To avoid this, java does not supports multiple inheritance.

```
class A
{
    void methodOne()
    {
        System.out.println("From methodOfClassA");
    }
}

class B
{
    void methodOne()
    {
        System.out.println("From methodOfClassB");
    }
}

class C extends A,B (If it is supported)
{
    //two same methods will be inherited to Class C.

    //This causes ambiguity and confusion.
}
```

Question: Can a Class Extend Itself in Java?

Answer: A class can not extend itself.

Question: What is Method Overriding And Method Hiding in Java?

Method Overriding:

An instance method in a subclass with the same signature (name, number and the type of its parameters) and return type as an instance method in the superclass *overrides* the superclass's method.

The overriding method has the same name, number and type of parameters, and return type as the method that it overrides.

An overriding method can also return a subtype of the type returned by the overridden method. This subtype is called a *covariant return type*.

When overriding a method, you might want to use the `@Override` annotation that instructs the compiler that you intend to override a method in the superclass.

If, for some reason, the compiler detects that the method does not exist in one of the superclasses, then it will generate an error.

Method Hiding:

If a subclass defines a static method with the same signature as a static method in the superclass, then the method in the subclass ***hides*** the one in the superclass.

The distinction between hiding a static method and overriding an instance method has important implications:

- The version of the overridden instance method that gets invoked is the one in the subclass.
- The version of the hidden static method that gets invoked depends on whether it is invoked from the superclass or the subclass.

Consider an example that contains two classes. The first is `Animal`, which contains one instance method and one static method:

```
public class Animal {
    public static void testClassMethod() {
        System.out.println("The static method in Animal");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Animal");
    }
}
```

The second class, a subclass of `Animal`, is called `Cat`:

```
public class Cat extends Animal {
    public static void testClassMethod() {
        System.out.println("The static method in Cat");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Cat");
    }

    public static void main(String[] args) {
        Cat myCat = new Cat();
        Animal myAnimal = myCat;
        Animal.testClassMethod();
        myAnimal.testInstanceMethod();
    }
}
```

The `Cat` class overrides the instance method in `Animal` and hides the static method in `Animal`. The `main` method in this class creates an instance of `Cat` and invokes `testClassMethod()` on the class and `testInstanceMethod()` on the instance.

The output from this program is as follows:

```
The static method in Animal
The instance method in Cat
```

As promised, the version of the hidden static method that gets invoked is the one in the superclass, and the version of the overridden instance method that gets invoked is the one in the subclass.

Summary

The following table summarizes what happens when you define a method with the same signature as a method in a superclass.

	Superclass Instance Method	**Superclass Static Method**
Subclass Instance Method	Overrides	Generates a compile-time error
Subclass Static Method	Generates a compile-time error	Hides

Defining a Method with the Same Signature as a Superclass's Method

Note: In a subclass, you can overload the methods inherited from the superclass.

Such overloaded methods neither hide nor override the superclass instance methods—they are new methods, unique to the subclass.

Question: What is Method Overriding And Method Hiding in Java?

Super class field will be hidden in the sub class. You can access hidden super class field in sub class using super keyword.

Question: A class member declared protected becomes member of subclass of which type?

A class member declared protected becomes private member of subclass.

Question: How do you restrict a member of a class from inheriting to it’s sub classes?

By declaring that member as a private. Because, private members are not inherited to sub classes.

Question: Are constructors and initializers also inherited to sub classes in Java?

No, Constructors and initializers(Static initializers and instance initializers) are not inherited to sub classes.

But, they are executed while instantiating a sub class.

Question: Are static members inherited to sub classes in Java?

Yes, Static members are also inherited to sub classes.

Question: What is Super Keyword In Java?

The super keyword in java is a reference variable that is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

Super keyword has three purposes

- super is used to refer immediate parent class instance variable.
- super() is used to invoke immediate parent class constructor.
- super is used to invoke immediate parent class method.

Question: What are some characteristics of constructors in Super and child classes in Java?

Most important points about constructors are

- Constructors are not inherited.
- If you do not make a constructor, the default empty constructor is automatically created.
- If any constructor does not explicitly call a super or this constructor as its first statement, a call to super() is automatically added.

In Java, constructor of base class with no argument gets automatically called in derived class constructor. For example, output of following program is:

```
Base Class Constructor Called
Derived Class Constructor Called

// filename: Main.java
class Base {
  Base() {
    System.out.println("Base Class Constructor Called ");
  }
}

class Derived extends Base {
  Derived() {
    System.out.println("Derived Class Constructor Called ");
  }
}

public class Main {
  public static void main(String[] args) {
    Derived d = new Derived();
  }
}
```

But, if we want to call parameterized contructor of base class, then we can call it using super().

The point to note is base class constructor call must be the first line in derived class constructor.

For example, in the following program, super(_x) is first line derived class constructor.

```
// filename: Main.java
class Base {
  int x;
  Base(int _x) {
    x = _x;
  }
}

class Derived extends Base {
  int y;
  Derived(int _x, int _y) {
    super(_x);
    y = _y;
  }
  void Display() {
    System.out.println("x = "+x+", y = "+y);
  }
}

public class Main {
  public static void main(String[] args) {
    Derived d = new Derived(10, 20);
    d.Display();
  }
}
```

Output:

```
x = 10, y = 20
```

Question: What is Method Overloading In Java?

Overloading is a process of declaring two methods with same name but different method signature

E.g. System.out which is object of PrintStream class has several println() method to print different data types e.g. byte, short, int, char, float and double.

All of them are called overloaded method. Overloaded method calls are resolved during compile time in Java and they must have different method signatures.

Question: What are rules of overloading a method in Java?

The only rule of method overloading is that method signature of all overloaded method must be different.

Method signature is changed by changing either number of method arguments, or type of method arguments e.g. System.out.println() method is overloaded to accept different primitive types like int, short, byte, float etc.

They all accept just one argument but their type is different.

You can also change method signature by changing order of method argument but that often leads to ambiguous code so better to be avoided.

Question: What is Method Overriding In Java?

Method overriding is another way to define method with same name but different code and it must be in sub class.

Overriding is based upon run-time Polymorphism as method calls are resolved at run-time depending upon actual object.

For example if a variable of type Parent holds an object of Child class then method invoked will be from child class and not parent class, provides its overridden.

In order to override a method, you must follow rules of method overriding which means declaring method with same signature in sub class.

Question: What is Method Hiding In Java?

Static method cannot be overriding in Java because their method calls are resolved at compile time but it didn't prevent you from declaring method with same name in sub class.

In this case we say that method in sub class has hided static method from parent class.

If you have a case where variable of Parent class is pointing to object of Child class then also static method from Parent class is called because overloading is resolved at compile time.

Question: Can you prevent overriding a method without using final modifier?

Yes, there are some funky ways to prevent method overriding in Java. Though final modifier is only for that purpose you can use private keyword to prevent method overriding.

How? If you remember correctly, in order to override a method, the class must be extensible. If you make the constructor of parent class private then its not possible to extend that class because its constructor will not be accessible in sub class.

Which is automatically invoked by sub class constructor, hence its not possible to override any method from that class.

This technique is used in Singleton design pattern, where constructor is purposefully made private and a static getInstance() method is provided to access singleton instance.

Question: Can We Override a Private Method in Java?

No, you cannot override private method in Java. Since private methods are not visible

outside the class, they are not available in sub-class hence they cannot be overridden.

Question: What is co-variant Method Overriding?

One of the rule of method overriding is that return type of overriding method must be same as overridden method but this restriction is relaxed little bit from Java 1.5 and now overridden method can return sub class of return type of original method.

This relaxation is known as co-variant method overriding and it allows you to remove casting at client end.

One of the best example of this comes when you override clone() method. Original Object.clone() method returns Object which needs to cast, but with co-variant method overriding you can directly return relevant type

E.g. Date class returns object of java.util.Date instead of java.lang.Object.

Question: Can we change argument list of overridden method?

No, you cannot change the argument list of overridden method in Java. An overriding method must have same signature as original method.

Only return type can be changed that to only to sub type of return type of original method.

Question: Can we change return type of method in subclass while overriding?

No, you cannot change the return type of method during overriding. It would be violation of rules of overriding.

Though from Java 5 onward you can replace the return type with sub type e.g. if original method has return type as java.lang.Object then you can change return type of overridden method as any type e.g. clone() method.

This is also known as co-variant method overriding in Java.

Question: Can we override a method which throws run-time exception without throws clause?

Yes, you can. There is no restriction on throwing RuntimeException from overriding method.

So if your original method throws NullPointerException than its not necessary to throw NPE from overriding method as well.

Question: How to call super class version of an overriding method in sub class?

You can call it using super keyword. For example if you have a method draw() in both parent and child class.

Then from child class you can invoke parent class method draw() as super.draw(). It's very similar to calling super class constructor from sub class.

Question: What are rules of method overriding in Java?

Some rules of method overriding are following:

Overriding method cannot throw higher exception than overridden one, but that's only true for checked exception.

Overriding method cannot restrict access of overridden method e.g. if original method is public then overriding method must be public.

But it can expand access e.g. if original method is protected than overriding method can be protected or public.

Question: Can we override a non-static method as static in Java?

No, its not possible to define a non-static method of same name as static method in parent class, its compile time error in Java.

Question: Can we override constructor in Java?

No, you cannot override constructor in Java because they are not inherited.

Remember, we are talking about overriding here not overloading, you can overload construct but you cannot override them.

Overriding always happens at child class and since constructor are not inherited and their name is always same as the class name its not possible to override them in Java.

Question: Can we override final method in Java?

No, you cannot override final method in Java. Trying to override final method in subclass will result in compile time error.

Actually making a method final is signal to all developer that this method is not for inheritance and it should be use in its present form.

You generally make a method final due to security reasons.

Question: Can you overload or override main method in Java?

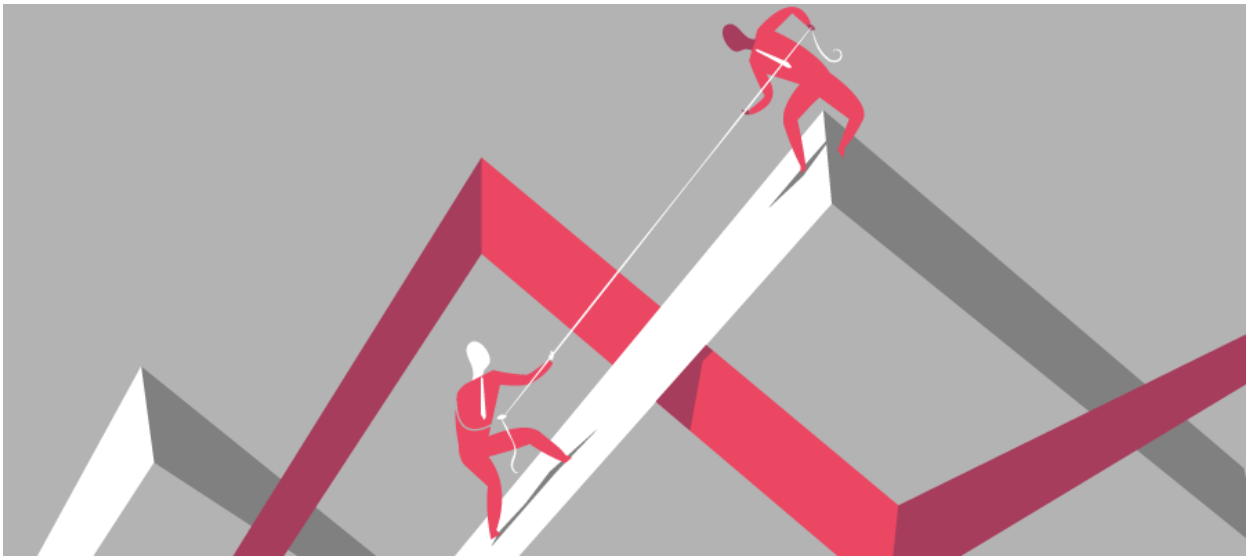
Since main() is a static method in Java, it follows rules associated with static method, which means you can overload main method but you cannot override it.

By the way, even if you overload a main method, JVM will always call the standard public static void main(String args[]) method to start your program.

If you want to call your overloaded method you need to do it explicitly in your code.

Question: Access Modifiers And Inheritance?

Comparison With Other OOP Concepts



Question: What is the difference between Composition and Inheritance?

One of the good question to check the candidate's object-oriented programming skills. There are several differences between Composition and Inheritance in Java, some of them are following:

1. The Composition is more flexible because you can change the implementation at runtime by calling setXXX() method, but Inheritance cannot be changed i.e. you cannot ask a class to implement another class at runtime.
2. Composition builds HAS-A relationship while Inheritance builds IS-A relationship e.g. A Room HAS A Fan, but Mango IS-A Fruit.
3. The parent-child relationship is best represented using Inheritance but If you just want to use the services of another class use Composition. For more differences see 5 reasons to favor composition over Inheritance.

Question: What is the difference between Polymorphism and Inheritance?

Both Polymorphism and Inheritance goes hand on hand, they help each other to achieve their goal.

Polymorphism allows flexibility, you can choose which code to run at runtime by overriding.

Question: What is the difference between Inheritance and Abstraction in Java?

Abstraction is an object oriented concept which is used to simply things by abstracting details.

It helps in the designing system. On the other hand, Inheritance allows code reuse. You can reuse the functionality you have already coded by using Inheritance

Question: What is the difference between Inheritance and Encapsulation?

Inheritance is an object oriented concept which creates a parent-child relationship. It is one of the ways to reuse the code written for parent class but it also forms the basis of Polymorphism.

On the other hand, Encapsulation is an object oriented concept which is used to hide the internal details of a class e.g. HashMap encapsulate how to store elements and how to calculate hash values.

Question: Difference between method overloading and overriding?

Fundamental difference between overloading and overriding is that former took place during compile time while later took place during run-time. Due to this reason.

Its only possible to overload virtual methods in Java. You cannot overload methods which are resolved during compile time e.g. private, static and final method cannot be overridden in Java.

Also, rules of method overloading and overriding are different, for example in order to overload a method its method signature must be different but for overriding method it must be same.

Question: Super() vs This() in Java

Super()

- Super keyword is used to call constructor in the super class.
- Super always refers to the parent of the current class
- Super allows you to access public/protected method/attributes of parent class. You cannot see the parent's private method/attributes.
- Super allows access to constructors from within the class' constructors only.

this()

- this refers to a reference of the current class.
- this allows access methods/attributes of the current class (including its own private methods/attributes).
- this is used to access the methods and fields of the current object. For this reason, it has no meaning in static methods, for example. this keyword use to call constructor in the same class (other overloaded constructor)

Code Review And Inheritance Programs

Question: Following code is showing compile time error. Can you identify the error?

```
class X
{
    //Class X Members
}

class Y
{
    //Class Y Members
}

class Z extends X, Y
{
    //Class Z Members
}
```

Answer:

In Java, a class can not extend more than one class. Class Z is extending two classes – Class X and Class Y. It is a compile time error in java.

Question: What will be the output of following program?

```
class A
{
    int i = 10;
}

class B extends A
{
    int i = 20;
}

public class MainClass
{
    public static void main(String[] args)
    {
        A a = new B();

        System.out.println(a.i);
    }
}
```

Answer:

```
10
```

Question: What will be the output of the below program?

```
class A
{
    {
        System.out.println(1);
    }
}

class B extends A
{
    {
        System.out.println(2);
    }
}

class C extends B
{
    {
        System.out.println(3);
    }
}

public class MainClass
{
    public static void main(String[] args)
    {
        C c = new C();
    }
}
```

Answer:

```
1 2 3
```

Question: How to identify super and invocation hierarchy ?

```
class A
{
    String s = "Class A";
}

class B extends A
{
    String s = "Class B";

    {
        System.out.println(super.s);
    }
}

class C extends B
{
    String s = "Class C";

    {
        System.out.println(super.s);
    }
}

public class MainClass
{
    public static void main(String[] args)
    {
        C c = new C();

        System.out.println(c.s);
    }
}
```

Answer:

```
Class A
Class B
Class C
```

Question: What is the invocation flow in following example ?

```
class A
{
    static
    {
        System.out.println("THIRD");
    }
}

class B extends A
{
    static
    {
        System.out.println("SECOND");
    }
}

class C extends B
{
    static
    {
        System.out.println("FIRST");
    }
}

public class MainClass
{
    public static void main(String[] args)
    {
        C c = new C();
    }
}
```

Answer:

`THIRD SECOND FIRST`

Question: Where super() or this() should be located?

```
public class A
{
    public A()
    {
        System.out.println(1);

        super();

        System.out.println(2);
```

Answer: Constructor calling statements (super() or this()), if written, must be the first statements in the constructor.

Question: Can you identify the member visibilty error in this code ?

```
class X
{
    private int m = 48;
}

class Y extends X
{
    void methodOfY()
    {
        System.out.println(m);
    }
}
```

Answer:

```
Because, private field ‘m’ is not visible to class Y.
```

Keys to interview success

You must understand that knowing the OOP concepts is the key to successful interview.

You will be asked questions about inheritance, abstraction, and interfaces. No technical interview is complete without questions on these topics. And the best way to prepare for them is to understand the core concepts.

They are also important in interview assignments or are often asked in the telephonic interviews.

So make sure you have a sound understanding of the concepts presented above.

About The Author

References

- https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html
- http://www.javaworld.com/article/2076814/core-java/inheritance-versus-composition--which-one should-you-choose-.html
- http://stackoverflow.com/questions/1644317/java-constructor-inheritance
- https://docs.oracle.com/javase/tutorial/java/IandI/super.html