**Q1) What is Inheritance?**

* Inheritance is the process by which objects of one class acquire the properties of objects of another class.
* A class that is inherited is called a superclass.
* The class that does the inheriting is called a subclass.
* Inheritance is done by using the keyword extends.
* The two most common reasons to use inheritance are:
  + To promote code reuse
  + To use polymorphism

**Q2) What is Polymorphism?**

* Polymorphism is briefly described as "**one interface, many implementations**."
* Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form.

**Q3) How does Java implement polymorphism?**

(Inheritance, Overloading and Overriding are used to achieve Polymorphism in java).  
Polymorphism manifests itself in Java in the form of multiple methods having the same name.

* In some cases, multiple methods have the same name, but different formal argument lists (overloaded methods).
* In other cases, multiple methods have the same name, same return type, and same formal argument list (overridden methods).

**Q4) Explain the different forms of Polymorphism.**

There are two types of polymorphism

* **Compile time polymorphism (static polymorphism or early binding)**. Compile time polymorphism is method **overloading**.
* **Runtime time polymorphism (dynamic polymorphism or late binding)** is done using inheritance and interface by means of method **overriding**  
  **Note**: *From a practical programming viewpoint, polymorphism manifests itself in three distinct forms in Java:*
* *Method overloading*
* *Method overriding through inheritance*
* *Method overriding through the Java interface*

**Q5) What is runtime polymorphism or dynamic method dispatch?**

In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called **is based on the object being referred to** by the reference variable.

**Q6) What is Dynamic Binding?**

Binding refers to the process of connecting method call to the method body. Dynamic binding (also known as late binding) means that the code associated with a given procedure call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance.

**Q7) What is method overloading and overriding?**

**Overloading:**

* In java we can declare two or more methods with the same name in the class provided their parameter declarations vary.
* The java compiler observes the signature of the methods and is able to differentiate between method signatures.
* The difference can be in the
  + In the number of parameters
  + Sequence of the order of parameters
  + Data types of parameters
* The difference in method signatures helps the java compiler to bind the method call to the exact method.

**Overriding:**

* Method overriding occurs when sub class declares a method that has the same type arguments as a method declared by one of its superclass.
* In method of overriding the java compiler does not decide which method is called by the user, since it has to wait till the object of the subclass is created.
* JVM calls the method depending on the reference type of the object which is used to call the method.

**Rules for Overriding:**

* + 1. The method signature should be same
    2. The return type must be same (from java 5 co-variant return types are allowed)
    3. The overriding method cannot have a more restrictive access modifier than the method being overridden. only default 🡪 protected 🡪 public is allowed
    4. You cannot override a method marked final
    5. You cannot override a method marked static

**Q8) What are the differences between method overloading and method overriding?**

|  |  |  |
| --- | --- | --- |
|  | **Overloaded Method** | **Overridden Method** |
| **Arguments** | Must change | Must not change |
| **Return type** | Can change | Can’t change except for covariant returns |
| **Exceptions** | Can change | Can reduce or eliminate. Must not throw new or broader checked exceptions |
| **Access** | Can change | Must not make more restrictive (can be less restrictive) |
| **Invocation** | Reference type determines which overloaded version is selected. Happens at compile time. | Object type determines which method is selected. Happens at runtime. |
|  | Adds functionality | Changed functionality |

**Q9) Can overloaded methods be override too?**

Yes, derived classes still can override the overloaded methods. Polymorphism can still happen. Compiler will not binding the method calls since it is overloaded, because it might be overridden now or in the future.

**Q10) Is it possible to override the main method?**

**NO**, because main is a static method. A static method can't be overridden in Java.

**Q11) How to invoke a superclass version of an Overridden method?**

To invoke a superclass method that has been overridden in a subclass, you must either call the method directly through a superclass instance, or use the super prefix in the subclass itself. From the point of the view of the subclass, the super prefix provides an explicit reference to the superclass' implementation of the method.

// From subclass

super.overriddenMethod();

**Q12) What is super?**

super is a keyword which is used to access the method or member variables from the superclass. If a method hides one of the member variables in its superclass, the method can refer to the hidden variable through the use of the super keyword. In the same way, if a method overrides one of the methods in its superclass, the method can invoke the overridden method through the use of the super keyword.   
**Note**:

* *You can only go back one level.*
* *In the constructor, if you use super(), it must be the very first code, and you cannot access any*this.xxx*variables or methods to compute its parameters.*

**Q13) How do you prevent a method from being overridden?**

To prevent a specific method from being overridden in a subclass, use the final modifier on the method declaration, which means "this is the final implementation of this method", the end of its inheritance hierarchy.

public **final** void exampleMethod() {  
        // Method statements  
       }

**Q14) Why multiple inheritance is not supported through class in Java, but it can be possible through interface ?.  
Multiple inheritance is not supported by class because of ambiguity. In case of interface there is no ambiguity because implementation to method(s) is provided by the implementing class**

**Q15) Explain types of object referencing in JAVA?**

**Normal Reference: When an object is created as below:**

**SubClass obj = new SubClass();**

**Here the reference and the object both are of the type SubClass.**

**Advantage : By using subclass reference, we will have access to both parts(methods and variables) of the object defined by the superclass or subclass.**

**Disadvantage : We can use subclass reference to hold only for that particular subclass objects only.**

**Polymorphic Reference: When an object is created as below**

**SuperClass obj = new SubClass();**

**Here the reference is of SuperClass and the object is of SubClass.**

**Advantage : We can use superclass reference to hold any subclass object derived from it.**

**Disadvantage : By using superclass reference, we will have access only to those parts(methods and variables) of the object defined by the superclass, i.e. we cannot call a method which is only defined in sub-class and not in super-class**

**Q16) What is Multiple Inheritance?**

**When we try to inherit a class from one or more classes. Multiple inheritance is not supported in JAVA, however it can be achieved through Interfaces.**

**Multiple inheritance can lead to the Diamond problem where the JAVA compiler is not able to decide which version of the method it needs to call and there is ambiguity among two same methods present in different super classes.**

**GrandParent**

**/ \**

**/ \**

**Parent1 Parent2 Diamond problem**

**\ /**

**\ /**

**Test**

**1) What do you mean by inheritance.?**

Inheritance is one of the key features of object oriented programming. Through inheritance, a class (Sub Class) can inherit properties of another class (Super Class). Sub class can have it’s own properties along with the inherited properties from it’s super class.

**2) What are the types of inheritance.?**

There are 5 types of inheritance.

1). Single Inheritance : One class is extended by only one class.

2). 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.

3). Hierarchical Inheritance : One class is extended by many classes.

4).Hybrid Inheritance : It is a combination of above types of inheritance.

5). Multiple Inheritance : One class extends more than one classes. (Java does not support multiple inheritance.)

**3) Can a class extend more than one classes or does java support multiple inheritance? If not, why?**

No, a class in java can not extend more than one classes or java does not support multiple inheritance. To avoid ambiguity, complexity and confusion, java does not supports multiple inheritance. 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | 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.  } |

**4) How do you implement multiple inheritance in java?**

Through interfaces, we can implement multiple inheritance in java. As classes in java can not extend more than one classes, but a class can implement more than one interfaces.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | interface A  {    }    interface B  {    }    class C implements A, B  {      //Class implementing two interfaces.  } |

**5) You know that all classes in java are inherited from java.lang.Object class. Are interfaces also inherited from Object class.?**

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.

**6) 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.

**7) Can a class extend itself.?**

No, A class can not extend itself.

**8) Are constructors and initializers also inherited to sub classes.?**

No, Constructors and initializers(Static initializers and instance initializers) are not inherited to sub classes. But, they are executed while instantiating a sub class.

**9) What happens if both, super class and sub class, have a field with same name.?**

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

**10) Are static members inherited to sub classes?**

Yes, Static members are also inherited to sub classes.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | class A  {      static int i = 10;        static void method()      {          System.out.println("Static Method");      }  }    class B extends A  {    }    public class StaticInitializers  {      public static void main(String[] args)      {          B.method();       //Calling inherited static method            System.out.println(B.i);    //printing inherited static field.      }  } |

**1) What is method overloading?**

When a class has more than one method with same name but different parameters, then we call those methods are overloaded. Overloaded methods will have same name but different number of arguments or different types of arguments.

**2) What is method signature? What are the things it consist of?**

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

**3) Can we declare one overloaded method as static and another one as non-static?**

Yes. Overloaded methods can be either static or non static.

**4) How do compiler differentiate overloaded methods from duplicate methods?**

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.

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

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) In “MyClass” , there is a method called “myMethod” with four different overloaded forms. All four different forms have different visibility ( private, protected, public and default). Is “myMethod” properly overloaded?**

Yes. Compiler checks only method signature for overloading of methods not the visibility of methods.

(Click [here](https://javaconceptoftheday.com/method-overloading-in-java/) to read more about method overloading)

**7) Can overloaded methods be synchronized?**

Yes. Overloaded methods can be synchronized.

**8) Can we overload main() method?**

Yes, we can overload main() method. A class can have any number of main() methods but execution starts from **public static void main(String[] args)** only.

**9) Can we declare overloaded methods as final?**

Yes, we can declare overloaded methods as final.

**10) In the below class, is constructor overloaded or is method overloaded?**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public class A  {      public A()      {          //-----> (1)      }        void A()      {          //-----> (2)      }  } |

None of them. It is neither constructor overloaded nor method overloaded. First one is a constructor and second one is a method.

**11) Overloading is the best example of dynamic binding. True or false?**

False. Overloading is the best example for static binding. (Click [here](https://javaconceptoftheday.com/difference-between-static-binding-and-dynamic-binding/) to see what is static binding and what is dynamic binding)

**12) Can overloaded method be overrided?**

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

**1) What is method overriding?**

Modifying a super class method in the sub class is called method overriding. Using method overriding, we can change super class method according to the requirements of sub class.

**2) What are the rules to be followed while overriding a method?**

There are 5 main rules you should kept in mind while overriding a method. They are,

a) Name of the method must be same as that of super class method.

b) Return type of overridden method must be compatible with the method being overridden. i.e if a method has primitive type as it’s return type then it must be overridden with primitive type only and if a method has derived type as it’s return type then it must be overridden with same type or it’s sub class types.

c) You must not reduce the visibility of a method while overriding.

d) You must not change parameter list of a method while overriding.

e) You can not increase the scope of exceptions while overriding a method with throws clause.

(Click [here](https://javaconceptoftheday.com/method-overriding-java/) to see more about rules of method overriding)

**3) Can we override static methods?**

No, Static methods can not be overridden. If we try to override them they will be hidden in the sub class.

**4) What happens if we change the arguments of overriding method?**

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

**5) Can we override protected method of super class as public method in the sub class?**

Yes. You can increase the visibility of overriding methods but can’t reduce it.

**6) Can we change the return type of overriding method from Number type to Integer type?**

Yes. You can change as Integer is a sub class of Number type.

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

Yes, but only with unchecked type of exceptions.

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

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

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

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

(Click [here](https://javaconceptoftheday.com/method-overriding-with-throws-clause/) to see more about method overriding with throws clause)

**10) How do you refer super class version of overridden method in the sub class?**

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

**11) Can we override private methods?**

No question of overriding private methods. They are not at all inherited to sub class.

**12) Can we remove throws clause of a method while overriding it?**

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

**13) Is it possible to override non-static methods as static?**

No. You can’t override non-static methods as static.

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

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

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

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