Q1) What is Method Overloading in Java ?

Method overloading in Java is an object-oriented programming concept that allows a programmer to declare two methods of the same name but with different method signatures, like change in the argument list or change in the type of argument. Method overloading is a powerful Java programming technique to declare a method that does a similar job but with a different kind of input. One of the most **popular examples of method overloading** is the System.out.println() method whose job is to print data on the console. This method is overloaded to accept all kinds of data types in Java. 

Q2) What is method overriding in OOP or Java ?

Method overriding in Java is a concept based on [polymorphism OOPS concept](http://javarevisited.blogspot.sg/2011/08/what-is-polymorphism-in-java-example.html) which allows the programmer to create two methods with the same name and method signature on the interface and its various implementation and the actual method is called at runtime depending upon the type of an object at runtime. Method overriding allows you to write flexible and extensible code in Java because you can introduce new functionality with the minimal code change. **Method overriding** is different than the [method overloading in Java](http://java67.blogspot.sg/2012/08/what-is-method-overloading-in-java-example.html) which we have discussed in the last article.  
  
  
Q3) What is the method of hiding in Java?

When you declare two static methods with same name and signature in both super class 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.

Q4) Is Java a pure object-oriented language? if not why?

No, based on the fact that in a pure object-oriented language everything is an object and there are many things in [Java](https://javarevisited.blogspot.com/2018/07/10-reasons-to-learn-java-programming.html) that are not objects like primitive data types like boolean, char, short, int, long, float, double, different kinds of arithmetic, logical and bitwise operator like +, -. \*, /, &&, || etc. There are only a few pure OO programming languages are **Smalltalk**and **Eiffel,**If there is more, I may not know but Smalltalk is often touted as the purest form of an object-oriented language.  
  
Q5) What are the rules of method overloading and overriding in Java?

One of the most important rules of method overloading in Java is that the 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, the compiler will throw an error. On the other hand, method overriding has more rules e.g. name and return type must be the same, method signature should also be the 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.  
  
  
  
  
Q6) The difference between method overloading and overriding?

1) First and major difference between Overloading and Overriding is that former occur during [compile time](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html) while later occur during runtime.  
  
2) Second [difference between Overloading and Overriding](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html) is that, you can overload method in same class but you can only override method in sub class.  
  
3) Third difference is that you [can overload static method in Java](http://java67.blogspot.sg/2012/08/can-we-overload-static-method-in-java.html) but you [can not override static method in Java](http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html). In fact when you declare same method in Sub Class it's known as method hiding because it hide super class method instead of overriding it.  
  
4) Overloaded methods are bonded using [static binding](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html) and Type of reference variable is used, while the Overridden method is bonded using [dynamic bondin](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html)g based upon actual Object.  
  
5) [Rules of Overloading and Overriding](http://java67.blogspot.sg/2012/09/what-is-rules-of-overloading-and-overriding-in-java.html) is different in Java. In order to overload a method you need to change its method signature but that is not required for overriding any method in Java.  
  
6) Another difference between method overloading and overriding is that private and final methods can not be overridden but can be overloaded in Java.  
  
7) The overloaded method are fast as compare to the Overridden method in Java.

Q7)Can we overload a static method in Java?

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.

Q8) Can we override the static method in Java?

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 the reference variable of type subclass then new method will be called, but if you use the reference variable of superclass than old method will be called.

Q9) Can we prevent overriding a method without using the final modifier?

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.

Q10) Can we override a private method in Java?

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.

Q11) What is the covariant method overriding in Java?

In the covariant method overriding, the overriding method can return the subclass of the object returned by the original or overridden method. This concept was introduced in Java 1.5 (Tiger) version and it's very helpful in case the original method is returning general type like Object class, because, then by using the covariant method overriding you can return a more suitable object and prevent client-side typecasting. One of the practical use of this concept is when you override the clone() method in Java.

Q12) Can we change the return type of method to subclass while overriding?

Yes, you can, but only from Java 5 onward. This feature is known as covariant method overriding and it was introduced in JDK 5 release. This is immensely helpful if the original method return super-class like clone() method return java.lang.Object. By using this, you can directly return the actual type, preventing client-side type-casting of the result.

Q13) Can we change the argument list of an overriding method?

No, you cannot. The argument list is part of the method signature and both overriding and overridden methods must have the same signature.

Q14) Can we override a method that throws runtime exception without throws clause?

Yes, there is no restriction on unchecked exceptions 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 the original method is throwing IOException than the overriding method cannot throw java.lang.Exception or java.lang.Throwable.

Q15) How do you call a superclass version of an overriding method in a subclass?  
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().

Q16) Can we override a non-static method as static in Java?

Yes, you can override the non-static method in Java, no problem on them but it should not be private or final

Q17) Can we override the final method in Java?

No, you cannot override a final method in Java, the final keyword with the method is to prevent method overriding. You use the final when you don't want subclass changing the logic of your method by overriding it due to security reasons. This is [why the 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 the template design patterns where the template method is made final to prevent overriding.  
Q18) Can we have a non-abstract method inside an interface?

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 methods inside an interface.

Q19) What is the default method of Java 8?

The default method, also known as the extension method is new types of the method which you can add on the interface now. These 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 at Java 8 API documentation you will find several useful default methods on key Java interface like Iterator, Map, etc.

Q20) What is an abstract class in Java?

An abstract class is a class that 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 is 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.

Q21) What is an interface in Java? What is the real user of an interface?

Like an abstract class, the interface is also there to specify the contract of an API. It supports the 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 a more detailed explanation and discussion.

Q22) The difference between Abstract class and interface?

The first and the major difference between an abstract class and an interface is that an abstract class is a class while the interface is an interface, which means by extending the abstract class you can not extend another class because[Java does not support multiple inheritances](http://javarevisited.blogspot.sg/2011/07/why-multiple-inheritances-are-not.html) but you can implement multiple inheritance in Java.

The second difference between an interface and abstract class in Java is that *you can not create a non-abstract method in an interface*, every method in an interface is by default abstract, but you can create a non-abstract method in the abstract class. 