Difference between Method Overloading and Overriding in Java?

Method Overloading vs Method Overriding

Though the name of the method remains the same in the case of both method overloading and overriding, the main difference comes from the fact that method overloading is resolved during compile time, while method overriding is resolved at runtime. Also, rules of overriding or overloading a method are different in Java. For example, a private, static and final method cannot be overridden in Java but you can still overload them. For overriding both name and signature of the method must remain the same, but for the overloading method, the signature must be different. Last but not least difference between them is that call to overloaded methods are resolved using static binding while the call to an overridden method is resolved using dynamic binding in Java.

By the way, Method overloading and method overriding in Java are two important concepts in Java which allows Java programmer to declare a method with the same name but different

behavior. Method overloading and method overriding are based on Polymorphism in Java.

In the case of method overloading, a method with the same name co-exists in the same class but must have a different method signature, while in the case of method overriding, a method with the same name is declared in the derived class or sub class. Method overloading is resolved using static binding in Java at compile time while method overriding is resolved using dynamic binding in Java at runtime.

In short, when you overload a method in Java its method signature got changed while in the case of the overriding method signature remains the same but a method can only be overridden in sub class. Since Java supports Polymorphism and resolves objects at run-time it is capable to call an overridden method in Java.

By the way difference between method overloading and overriding is also one of the popular Java design questions and appears in almost all levels of Java interviews.

What are method overloading and overriding in Java?

In this Java tutorial, we will see how Java allows you to create two methods of the same name by using method overloading and method overriding. We will also touch base on how methods are bonded or called by Compiler and Java Virtual Machine and finally we will answer popular interview questions difference between method overloading and method overriding in Java.

This article is in my series of Java articles which discusses interviews like the difference between Synchronized Collection and Concurrent Collection or How to Stop Thread in Java.

Please let me know if you have some other interview questions and you are looking answer or reason for that and here in Javarevisited we will try to find and discuss those interview questions.

How to Overload a Method in Java

If you have two methods with the same name in one Java class with a different method signature then it's called the overloaded method in Java. Generally, the overloaded method in Java has a different set of arguments to perform something based on a different number of inputs.

You can also overload constructor in Java, which we will see in the following example of method overloading in Java. The binding of the overloading method occurs during compile-time and overloaded calls resolved using static binding. To overload, a Java method just changes its signature.

Just remember in order to change the signature you either need to change the number of arguments, type of argument, or order of argument in Java if they are of different types. Since the return type is not part of the method signature simply changing the return type will result in a duplicate method and you will get a compile-time error in Java.

In our example of the Loan and PersonalLoan class, createLoan method is overloaded. Since you have two `crateLoan()` method with one takes one argument lender while the other take two arguments both `lender` and `interestRate`. Remember you can overload static methods in Java, you can also overload a private and final method in Java but you can not override them.

How to Override a Method in Java

In order to override a Java method, you need to create a child class that extends parent. The overridden method in Java also shares the same name as the original method in Java but can only be overridden in sub-class. The original method has to be defined inside interface or base class, which can be abstract as well.

When you override a method in Java its signature remains exactly the same including return type. JVM resolves the correct overridden method based upon an object at run-time by using dynamic binding in Java.

For example in our case when we call `personalLoan.toString()` method even though `personalLoan` object is of type Loan actual method called would be from `PersonalLoan` class because object referenced by `personalLoan` variable is of type `PersonalLoan()`.

This is a very useful technique to modify the behavior of a function in Java based on different implementations. `equals()`, `hashcode()` and `compareTo()` methods are classic example of overridden methods in Java.

Another important point is that you can not override the static method in Java because they are associated with Class rather than object and resolved and bonded during compile-time and that's the reason you cannot override the main method in Java.

Similar to static, private and final methods are also not overridden in Java.

By the way, as part of overriding best practice, always use @Override annotation, while overriding method from an abstract class or interface.

Rules of Method Overriding in Java

Following are rules of method overriding in java that must be followed while overriding any method. As stated earlier `private`, `static` and `final` method can not be overridden in Java.

1. Method signature must be the same including return type, number of method parameters, type of parameters, and order of parameters.

2. The overriding method can not throw a higher Exception than the original or overridden method. This means if the original method throws `IOException` then the overriding method can not throw super class of IOException like Exception but it can throw any sub-class of IOException or simply does not throw an Exception. This rule only applies to check Exception in Java, overridden method is free to throw any unchecked Exception.

3. The Overriding method can not reduce the accessibility of overridden method, which means if the original or overridden method is public then the overriding method can not make it protected.

Difference between Method Overloading vs Overriding in Java

Overloading vs Overriding in Java is one of the popular java interview questions at many companies and asked at different levels of programmers. Here are some important differences between overloading and overriding in Java. Though It's more important is to understand how to use both overloading and overriding, these difference are good from the interview perspective and gives some basic idea as well:

1) First and most important difference between method overloading and overriding is that In the case of method overloading in Java, the signature of the method changes while in the case of method overriding it remains the same.

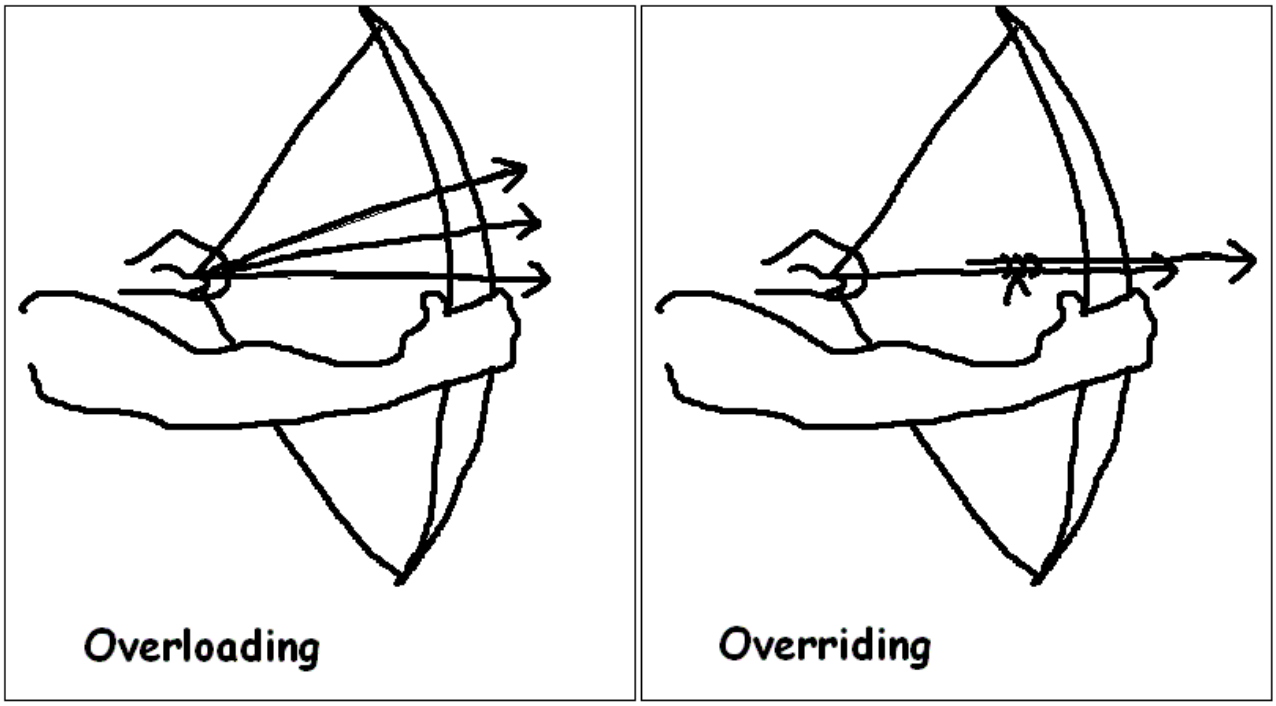
2) Second major difference between method overloading vs overriding in Java is that You can overload the method in one class but overriding can only be done on the subclass.

3) You can not override `static`, `final`, and `private` methods in Java but you can overload static, final, or private methods in Java.

4) Overloaded method in Java is bonded by static binding and overridden methods are subject to the dynamic binding.

5) Private and final method can also be not overridden in Java.

By the way, you might have heard about "a picture is worth more than a thousand words" and this is made true by the following image. By looking at the pic you can clearly understand the difference between method overloading and overriding in Java.



Handling Exception while overloading and overriding method in Java

While overriding a method it can only throw checked exception declared by overridden method or any subclass of it, which means if the overridden method throws IOExcpetion then the overriding method can throw sub classes

of `IOExcpetion` like `FileNotFoundException` but not wider exception like Exception or Throwable.

This restriction is only for checked Exception for RuntimeException you can throw any RuntimeException. The overloaded method in Java doesn't have such restrictions and you are free to modify the throws clause as per your need.

Method Overloading and Overriding Example in Java

Here is an example of both method overloading and method overriding in Java. In order to explain the concept, we have created two classes `Loan` and `PersonalLoan`. createLoan() method is overloaded as it has different versions with a different signature, while `toString()` method which is original declared in Object class is overridden in both `Loan` and `PersonalLoan` class.

```
public class OverloadingOverridingTest {

    public static void main(String[] args) {

        // Example of method overloading in Java
        Loan cheapLoan = Loan.createLoan("HSBC");
        Loan veryCheapLoan = Loan.createLoan("Citibank", 8.5);

        // Example of method overriding in Java
        Loan personalLoan = new PersonalLoan();
        personalLoan.toString();
    }

}

public class Loan {
    private double interestRate;
    private String customer;
    private String lender;

    public static Loan createLoan(String lender) {
        Loan loan = new Loan();
        loan.lender = lender;
        return loan;
    }

    public static Loan createLoan(String lender, double interestRate) {
        Loan loan = new Loan();
        loan.lender = lender;
        loan.interestRate = interestRate;
        return loan;
    }
```

```
    @Override
    public String toString() {
        return "This is Loan by Citibank";
    }

}

public class PersonalLoan extends Loan {

    @Override
    public String toString() {
        return "This is Personal Loan by Citibank";
    }
}
```

Things to Remember

1) In the case of method overloading method signature gets changed while in case of overriding signature remains the same.

2) Return type is not part of the method signature in Java.

3) Overloaded method can be subject to compile-time binding but the overridden method can only be bind at run-time.

4) Both overloaded and overridden method has the same name in Java.

5) Static method can not be overridden in Java.

6) Since the private method is also not visible outside of class, it can not be overridden and method binding happens during compile time.

7) From Java 5 onwards you can use annotation in Java to declare overridden method just like we did with `@override`. `@override` annotation allows compiler, IDE like NetBeans and Eclipse to cross-verify or check if this method is really overridden super class method or not.

Covariant Method Overriding in Java

One of my reader Rajeev makes an interesting comment about one change related to return type of overriding method from Java 5 onwards, which enable to use subtype of the return type of overridden method.

This is really useful when the original method returns a general type like `java.lang.Object`. If you are overriding the clone() method in Java then you can use this feature to return the actual type, instead of returning `java.lang.Object` and can save caller from type-casting cloned object. Here is the actual comment from Rajeev:

Hi Javin, I visit your blog regularly and I found that you missed *covariant return* which is added in Java 5 in the case of method overriding. When a subclass wants to change the method implementation of an inherited method (an override), the subclass must define a method that matches the inherited version exactly.

Or, as of Java 5, you're allowed to change the return type in the overriding method as long as the new return type is a subtype of the declared return type of the overridden (super class) method. Let's look at a covariant return in action:

```
class Alpha {
    Alpha doStuff(char c) {
        return new Alpha();
    }
}

class Beta extends Alpha {
    Beta doStuff(char c) { // legal override in Java 1.5
        return new Beta();
    }
}
```

You can see that the `Beta` class which is overriding the `doStuff()` method from the `Alpha` class is returning the `Beta` type and not the `Alpha` type. This will remove type casting on the client side. See here to learn more about covariant method overriding in Java.

AS I said one a good example of this is overriding the clone method and using return type as Actual type instead of `java.lang.Object`, which is suggested by Joshua Bloch in Effective Java as well. This is in fact one of the Java best practices while implementing the clone method in Java. By the way don't forget to follow these Java overloading best practices, when doing it in your project.