**Java Method Overriding**

In the last tutorial, we learned about inheritance. Inheritance is an OOP property that allows us to derive a new class (subclass) from an existing class (superclass). The subclass inherits the attributes and methods of the superclass.

Now, if the same method is defined in both the superclass and the subclass, then the method of the subclass class overrides the method of the superclass. This is known as method overriding.

**Example 1: Method Overriding**

class Animal {

public void displayInfo() {

System.out.println("I am an animal.");

}

}

class Dog extends Animal {

@Override

public void displayInfo() {

System.out.println("I am a dog.");

}

}

class Main {

public static void main(String[] args) {

Dog d1 = new Dog();

d1.displayInfo();

}

}

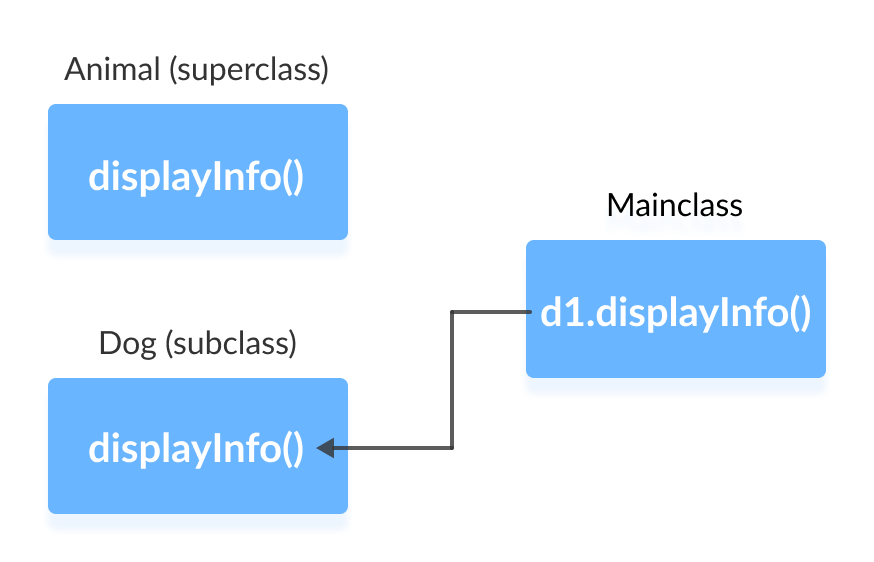
[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

I am a dog.

In the above program, the displayInfo() method is present in both the Animal superclass and the Dog subclass.

When we call displayInfo() using the d1 object (object of the subclass), the method inside the subclass Dog is called. The displayInfo() method of the subclass overrides the same method of the superclass.



Notice the use of the @Override annotation in our example. In Java, annotations are the metadata that we used to provide information to the compiler. Here, the @Override annotation specifies the compiler that the method after this annotation overrides the method of the superclass.

It is not mandatory to use @Override. However, when we use this, the method should follow all the rules of overriding. Otherwise, the compiler will generate an error.

**Java Overriding Rules**

* Both the superclass and the subclass must have the same method name, the same return type and the same parameter list.
* We cannot override the method declared as final and static.
* We should always override abstract methods of the superclass (will be discussed in later tutorials).

**super Keyword in Java Overriding**

A common question that arises while performing overriding in Java is:

**Can we access the method of the superclass after overriding?**

Well, the answer is **Yes**. To access the method of the superclass from the subclass, we use the super keyword.

**Example 2: Use of super Keyword**

class Animal {

public void displayInfo() {

System.out.println("I am an animal.");

}

}

class Dog extends Animal {

public void displayInfo() {

super.displayInfo();

System.out.println("I am a dog.");

}

}

class Main {

public static void main(String[] args) {

Dog d1 = new Dog();

d1.displayInfo();

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

I am an animal.

I am a dog.

In the above example, the subclass Dog overrides the method displayInfo() of the superclass Animal.

When we call the method displayInfo() using the d1 object of the Dog subclass, the method inside the Dog subclass is called; the method inside the superclass is not called.

Inside displayInfo() of the Dog subclass, we have used super.displayInfo() to call displayInfo() of the superclass.

It is important to note that constructors in Java are not inherited. Hence, there is no such thing as constructor overriding in Java.

However, we can call the constructor of the superclass from its subclasses. For that, we use super(). To learn more, visit [Java super keyword](https://www.programiz.com/java-programming/super-keyword).

**Access Specifiers in Method Overriding**

The same method declared in the superclass and its subclasses can have different access specifiers. However, there is a restriction.

We can only use those access specifiers in subclasses that provide larger access than the access specifier of the superclass. For example,

Suppose, a method myClass() in the superclass is declared protected. Then, the same method myClass() in the subclass can be either public or protected, but not private.

**Example 3: Access Specifier in Overriding**

class Animal {

protected void displayInfo() {

System.out.println("I am an animal.");

}

}

class Dog extends Animal {

public void displayInfo() {

System.out.println("I am a dog.");

}

}

class Main {

public static void main(String[] args) {

Dog d1 = new Dog();

d1.displayInfo();

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

I am a dog.

In the above example, the subclass Dog overrides the method displayInfo() of the superclass Animal.

Whenever we call displayInfo() using the d1 (object of the subclass), the method inside the subclass is called.

Notice that, the displayInfo() is declared protected in the Animal superclass. The same method has the public access specifier in the Dog subclass. This is possible because the public provides larger access than the protected.

**Overriding Abstract Methods**

In Java, abstract classes are created to be the superclass of other classes. And, if a class contains an abstract method, it is mandatory to override it.

We will learn more about abstract classes and overriding of abstract methods in later tutorials.