super Keyword

Sometimes you want a subclass to do more than what a superclass's method is doing. Maybe you still want to execute the superclass method, but you also want to override the method to do something else. But, since you have overridden the parent method, how can you still call that original version if you need to? Enter the `super` keyword: you can use `super.method()` to force the *parent's* method to be called.

We've used `super()` before to call the superclass' constructor, so let's clarify. There are two uses of the keyword super:

1. **super();** or **super(arguments);** calls just the super constructor, if written as the first line of a subclass constructor.
2. **super.method();** calls a superclass' method (not its constructors).

The keyword **super** is very useful as it allows us to execute the superclass's method as part of our subclass method's code. Developers typically call the superclass method first and then add onto it in the subclass, but there are times when you may have some preliminary code (e.g., code to set up some variable in a method) before calling the superclass method.

#Coding Exercise

In the example below, the Student class overrides the `getFood()` method of the `Person` class, and it uses `super.getFood()` to call the `Person` class's `getFood()` method before adding on to it. Here, a `Person` class by default has a "Hamburger" as its food, while a `Student` is associated with "Hamburger and Taco" as its food. Before you attempt the task below, read through the code we've provided and make sure you understand it. Spend some time with the `Student.java` file before moving on.

Your task: Add another subclass called `Vegan` that inherits from the `Person` class. Override the `getFood()` method to call the superclass `getFood()` but add a "No " in front of it, as in "No Hamburger". Make a new `Person` that's a `Vegan` and try out your code!

Run

Vegan.java Student.java Person.java

```
public class Vegan extends Person {
    public Vegan(String name) {
        super(name);
    }
    public String getFood() {
        return "No Hamburger";
    }
}
public class Student extends Person {

    public Student(String name) {
        super(name);
    }

    public String getFood() {
        String output = super.getFood();
        return output + " and Taco";
    }
}
 public class Person {

      private String name;

      public Person(String name) {
         this.name = name;
      }

      public String getFood() {
         return "Hamburger";
      }

      public static void main(String[] args) {
         Person p = new Student("Javier");
         System.out.println(p.getFood());

         p = new Vegan("Vegan");
         System.out.println(p.name);
         System.out.println(p.getFood());
      }
}
```

How does this work? Remember that an object always keeps a reference to the class that created it and always looks for a method during execution, starting with the class that created it. If it finds the method in the class that created it, it will execute

that method. If it doesn't find it in the class that created it, **it will look at the parent of that class**. It will keep looking up the ancestor chain (from parent, to parent of parent, to parent of parent of parent, etc.) until it finds the method. This chain will continue all the way up to the built-in `Object` class if it has to. The method has to be there somewhere, or else the code would have thrown an error during compilation.

When the student `getFood()` method is executed it will start executing the `getFood` method in `Student`. When it gets to `super.getFood()` it will execute the `getFood` method in `Person`. This method will return the string `"Hamburger"`. Then, execution will continue in the `getFood` method of `Student` and return the string `"Hamburger and Taco"` for our non-vegan students.

#Check your understanding

Given the following class declarations, and assuming that the following declaration appears in a client program: `Base b = new Derived();`, what is the result of the call `b.methodOne();`?

```
public class Base {
public void methodOne() {
System.out.print('A');
methodTwo();
}

public void methodTwo() {
System.out.print('B');
}
}

public class Derived extends Base {
public void methodOne() {
super.methodOne();
System.out.print('C');
}

public void methodTwo() {
super.methodTwo();
System.out.print('D');
}
}
```

You can step through this example using the Java Visualizer by clicking on the following link: Super Example.

The `toString()` method is a commonly overridden method. A subclass can override the superclass `toString()` method and call `super.toString()` before adding on its own instance variables.

```
// overriden toString() in subclass
public String toString()
{
return super.toString() + "\n" + subclassInstanceVariables;
}
```

#Another Perspective

Before we jump into a programming challenge, here's another perspective (and a bit more explanation) on how to use the super keyword. Notice how he compares it to the `this` keyword we've already been using:

Embed anything (PDFs, Google Maps, Spotify, etc...)

#Programming Challenge : Customer Info

The `Customer` class below keeps track of the names and addresses of customers. It has a `toString()` method that prints out the `name` and `address` of the object.

1. Create a subclass called `OnlineCustomer` that inherits from the `Customer` class and adds a new instance variable for the `email` address of an online customer.
2. Override the `toString()` method in the `OnlineCustomer` class to call the super class `toString()` method and then add on the `email` address. See the example above for help.
3. Test the class by uncommenting the `OnlineCustomer` objects in the `main` method.

Complete the `OnlineCustomer` class below which inherits from `Customer` and adds an `email` address and overrides the `toString()` method.

2

Run

Customer.java OnlineCustomer.java

```
public class Customer {
    private String name;
    private String address;

    public Customer(String n, String a) {
       name = n;
       address = a;
    }

    public String toString() {
       return "Name: " + name + "\nAddress: " + address;
    }

    public static void main(String[] args) {
       Customer c = new Customer("Fran Santiago", "123 Main St., Anytown, USA"
);
       System.out.println(c);
       OnlineCustomer c2 = new OnlineCustomer("Jasper Smith", "456 High St., A
nytown, USA", "jsmith456@gmail.com");
       System.out.println(c2);
    }
 }
 public class OnlineCustomer extends Customer {

    private String email;

    public OnlineCustomer(String n, String a, String e) {
        super(n, a);
        email = e;
    }

    @Override
    public String toString() {
        return super.toString() + "\nEmail: " + email;
    }
}
```

#Summary

- The keyword `super` can be used to call a superclass's constructors and methods.
- The superclass's method can be called in a subclass by using the keyword `super.methodName()` and passing appropriate parameters.

#Extra Resources

For yet another perspective on working with inheritance and the super keyword, feel free to check out these YouTube videos when you have time:

Java Tutorial | Inheritance

Java Tutorial | Super Method

Back
Unsubmit
Next