# #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:

**super();** or **super(arguments);** calls just the super constructor, if written as the first line of a subclass constructor.

**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 Student class. Override the getFood() method to call the superclass getFood() but add a “No ” in front of it, as in "No Hamburger". Make our student Javier a Vegan and try out your code!

<https://youtu.be/oKZnHNM9Ew4>

## #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.

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.

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.

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.

## #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.

# #Inheritance and super

As we've recently learned, the super keyword is a reference variable that points to the parent class. It's widely used when inheritance comes into the picture, so we're going to need to build some familiarity with it. Of course, our understanding of super is dependent upon our understanding of inheritance, so let's get a little practice with that first.

### #Follow the steps below:

**Create a** Vehicle **Class (in the right package on Eclipse of course)**

Declare String colour and String brand attributes.

Define parameterized constructor for colour and brand that sets the class attributes.

**Create a** Car **class that extends** Vehicle

Define a parameterized constructor for colour, brand, and steeringWheel

Call the super keyword appropriately in your constructor (remember, a child class inherits the attributes of its parent class)

**Create a** Bike **class that extends the** Vehicle **class**

Declare a String bikeHandle

Define a parameterized constructor for colour, brand, and bikeHandle

Call the super keyword appropriately in your constructor (remember, a child class inherits the attributes of its parent class)

Execute the program and make sure it passes through the test case.

## #****Using super Keyword with Functions****

Follow the steps below:

Create a print method in your Vehicle class: public void print() { System.out.println(this.colour + " " + this.brand); }

Define a print() method in the Car class which calls its parent's print() method.

Create a print() method in the Bike class which calls its parent's print method.

Test your code again using the Vehicle's main method.

# #Object Superclass

The **Object** class is the superclass of all other classes in Java and a part of the built-in java.lang package. If a parent class isn’t specified using the **extends** keyword, the class will inherit from the Object class. What does a class inherit from the Object class? There are two main methods that are most used, toString() and equals(Object), from the Object class:

1. String toString()
2. boolean equals(Object otherObjectToCompareTo)

For a quick visual primer on how the Object class is the superclass of all other Java classes, take a look at this video here. Notice how he uses his IDE to see what methods are available–this is a common developer trick that keeps us from having to memorize code. See if you can't take advantage of it yourself:

## #The toString() method

One commonly overridden Object method is toString(), which is often used to print out the attributes of an object. It is a good idea to write your own toString() method in every class you create. When you don't specify what the toString() method should do, it just gives you the name of the object and its place in memory when you run the toString() command on it. This, generally speaking, is not very useful to us. Therefore, it's a good idea to define your own toString() method that gives you the details you actually need. In a subclass, toString() can call the superclass's toString() method using super.toString() and then add on its own attributes.

### #****Coding Exercise****

In the following code, the Person class overrides the Object's toString() method and the Student class overrides the Person's toString() method. Each class adds on its own special attributes.

After trying the code below, add another subclass called APStudent that extends Student with a new attribute called apScore. Then create the parameterized constructor in the APStudent class(use super keyword to call the parent constructor), override its toString() method to call the superclass method and then add on the apScore in the returned string. Create an APStudent object in the main method to test it. **Test it using a value of 90 as the** APStudent**'s** apScore **while keeping other attributes similar to those of the** student **object.**

You might also want to see what the toString() method does when we don't offer the class its own implementation of the method. Try deleting the specified toString methods and see what comes out. Note: if this is too advanced for you at the moment, check the Extra Resources below for a similarly worked problem in YouTube format.

# #Inheritance and Constructors

Remember: classes are abstractions. To get down to business with your classes and really do some work, you're going to need to instantiate some objects. Let's talk about how inheritance and constructors work with subclasses and superclasses, and how we can use them to make things happen in our code.

Subclasses inherit all the private instance variables in a superclass that they extend, but they cannot directly access them since they are private. Constructors, on the other hand, are not inherited at all, although we can use them by making the appropriate calls.

So how do you initialize inherited private variables if you don’t have direct access to them in the subclass? In Java you can put a call to the parent constructor as the first line in a subclass constructor. In Java, the superclass constructor can be called from the first line of a subclass constructor by using the keyword super and passing appropriate parameters, for example super(); or super(variable); as shown in the code below. The actual parameters given to super() are used to initialize the inherited instance variables, for example the name instance variable in a Person superclass.

public class Employee extends Person {

public Employee() {

super(); // calls the Person() constructor

}

public Employee(String theName) {

super(theName); // calls Person(theName) constructor

}

}