# CS 240: Static Variables and Methods Transcript

*This video shows a split screen of Professor Wilkerson on the right and a PowerPoint screen on the left. Any text displayed or action performed that is not verbalized will be included in italics as visual descriptions.*

[00:00:00] **JEROD WILKERSON:** To fully understand classes and objects in Java, you need to understand the keyword “static” and you need to know about static variables and methods.

[00:00:09] We’ll start with variables.

[00:00:11] Most variables are what we call instance variables.

[00:00:15] With an instance variable, each object when you create the object, it gets a copy of all the instance variables.

[00:00:23] That means we have one copy of each instance variable for every object that we have.

[00:00:29] An example of that would be a date object.

[00:00:32] If I have two date objects, I would expect them to be able to represent different dates.

[00:00:37] The variables that represent the date are not shared across the objects; they’re separate.

[00:00:44] Each object will have its own copy of them.

[00:00:47] With static variables, I do only have one copy of the variable.

[00:00:52] Normally what we want is instance variables when we’re doing object-oriented programming, but static variables, unlike instance variables, are not associated with an object; they are associated with the class.

[00:01:03] We use these in special cases where we really need to have a shared variable.

[00:01:07] If we think about our date class example again, if the variables that represent the date were static, there’d only be one copy of them, which would mean that all date objects would represent the same date.

[00:01:20] Of course, that’s not what we would want.

[00:01:22] Those variables should not be static, but in some cases, we do want a static variable.

[00:01:26] For example, if we wanted to have a counter that would count the number of objects we’ve created from a class, we wouldn’t want that to start over or be kept by each object.

[00:01:38] That would be a case where we would have a static variable.

[00:01:41] Static variables. Think of them as being associated with a class.

[00:01:46] The way I access them is I access them with the class name instead of a reference.

[00:01:51] If I have a class called MyClass and it has a static variable, I would reference it by calling MyClass.name of the variable instead of using a reference.

[00:02:01] I can also have static methods.

[00:02:05] With methods, an instance method is a method that’s associated with a specific instance of an object.

[00:02:12] A static method is not associated with an object; it’s associated with a class.

[00:02:16] Again, the way I would invoke an instance method is by using my reference to my object to invoke the method.

[00:02:23] The way I would invoke a static method is by using the class name in place of a reference.

[00:02:28] Most of your methods should be instance methods, and so they should be invoked on a particular object.

[00:02:35] Most of the time, when you think about methods, if we think about a Person class.

[00:02:39] If I have a Person class, it will have instance variables representing the Person, and then when I call the methods, most of those methods would be something I'm trying to do with that particular Person.

[00:02:50] Those should be instance methods.

[00:02:53] But I can have static methods and those are not associated with an instance, and often I don't have a reference.

[00:03:01] If I have a static method, there may not be any objects, and so there might not be any references referring to any objects.

[00:03:07] The way I will typically invoke a static method is by class name, again.

[00:03:11] I would use the class name in front of the dot and then call a method that way.

[00:03:18] A few things to keep in mind about static methods.

[00:03:23] First of all, they're not the norm, so most of your methods should be instance methods.

[00:03:27] Really, if you make every method static, you're basically throwing away object-oriented programming.

[00:03:34] It’s like programming in C, if you make everything static, and we throw away a lot of benefits when we do that.

[00:03:40] In general, by default, make methods instance methods, not static, and then only make them static if you have a special reason to do that.

[00:03:50] You’ll notice though that main methods are static.

[00:03:54] The reason for that is that’s the starting point for a program.

[00:03:59] We need to be able to invoke it before we’ve had any opportunity to create any objects.

[00:04:04] There is no object to invoke it on, so it makes sense that a main method would be static.

[00:04:09] If I do that though, if I have a static method, then I am not able to directly access any instance variables or methods from that static method.

[00:04:19] If you look right here, my main method is static, and I have an instance variable called myInstancevVariable, and I’m trying to set it equal to some number.

*The following lines of code is the above example:*

```
public class static StaticExampleWrong1 {

    private int myInstanceVariable;

    public static void main(String [] args) {

        myInstanceVariable = 10;

        myInstanceMethod();

    }
    public void myInstanceMethod() {

    }

}
```

*End of code.*

[00:04:28] That is illegal and will not compile.

[00:04:30] The reason for that is because an instance variable is associated with an object and I have no object, so it actually doesn’t even make sense for me to try to set that variable from within a static method.

[00:04:42] The same thing is true of calling an instance method from a static method, it doesn’t make sense to try to do that.

[00:04:50] By calling an instance method, I’m saying I want to invoke the method on some particular object.

[00:04:57] From within a static method, there is no object, so it doesn’t make sense to call that.

[00:05:02] How can we fix that? There are good ways and bad ways to fix that.

[00:05:07] The easy thing to do would be to declare everything static.

[00:05:11] I could make the instance variable static, and here’s what that would look like.

*The following lines of code is the above example:*

```
public class static StaticExampleWrong1 {

    private static int myInstanceVariable;

    public static void main(String [] args) {
        myInstanceVariable = 10;

        myInstanceMethod();

    }

    public **static** void myInstanceMethod() {

    }

}
```

*End of code.*

[00:05:14] I can make the instance variable static, I can make my instance methods static, and then this will compile no problem and everything will work.

[00:05:21] The problem with that is, now I've thrown away object-oriented programming, and I might as well just program in C, and I don't have the benefits of object-oriented programming.

[00:05:30] I don't have the benefits of inheritance, polymorphism, which you'll learn about later, and so I'm losing a lot by doing that.

[00:05:39] There's a better way, and that better way is to create an instance inside of my main method.

*The following lines of code is the above example:*

```
public class static StaticExampleWrong3 {

    private int myInstanceVariable;

    public static void main(String [] args) {

        StaticExample3 instance = new StaticExample();
        instance.myInstanceVariable = 10;

        instance.myInstanceMethod();

    }

    public void myInstanceMethod() {

    }

}
```

*End of code.*

[00:05:45] You’ll notice that the first thing I do in this example, the first line of code inside of the main method is I create an example of the class that contains the main method.

[00:05:54] That may make it seem a little strange to you at first when you see a method inside of a class creating an instance of that class, but it’s just perfectly normal to do an object-oriented programming, so you just have to get used to understanding that that’s not unusual; it’s a normal thing to do.

[00:06:10] What that does is it takes me out of the static world.

[00:06:13] It takes me out of that static C style of programming, and it gives me an object to be able to call methods on and access instance variables of.

[00:06:23] Here I’ve created StaticExample3, and I created a reference named instance that refers to it.

[00:06:30] Now when I say instance.MyInstanceVariable and set that equal to 10, I am setting the variable of a particular instance equal to 10.

[00:06:39] That makes sense because that variable is declared here as an instance variable.

[00:06:42] That’s perfectly okay.

[00:06:44] Then when I call my instance method using that same reference, I’m calling that method on an actual object. Again, that’s legal.

[00:06:53] That is the right way to go from the static world to the object-oriented world.

[00:06:58] Create an object first and then use the reference to that object to access instance variables and call static methods.