CS 240: Writing, Compiling, and Running Java Code Transcript

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

[00:00:00] J**EROD WILKERSON:** Let's learn how to write, compile, and run Java code.

[00:00:04] When you write Java code, all Java code goes inside of a class.

[00:00:08] That's subtly different than C++.

[00:00:10] In C++, you can't write code inside of a class, but in Java you have to write code inside of a class; there's nowhere else to write it.

[00:00:17] You don't have standalone functions like you do in C++.

[00:00:21] In Java, when you write a class, you put that code inside of a .java file.

[00:00:28] You create a file—the filename has to be the same as the class name that you're writing—and that goes in a file with the .java extension.

[00:00:39] There are a few exceptions where you can have more than one class inside of a class file, but for now, in general, we write one class per source file, and as I mentioned, the filename must exactly match the class name.

[00:00:56] Then we compile that code using the Java compiler and we get a .class file.

[00:01:01] The .class file is the file that contains the byte code that I mentioned in an earlier lecture and that .class file will also have the same name.

[00:01:10] If we have a class called MyClass, we will put it in a file called MyClass.java, and when we compile it, we'll get a class called MyClass.class.

[00:01:21] That’s what will be invoked by the JVM.

[00:01:25] We also have main methods in Java like you do in C++.

[00:01:30] In order to make a class executable, you need to create a main method for it.

[00:01:35] The main method looks mostly the same as it does in C++.

[00:01:39] Here I have two examples of a main method.

[00:01:41] If we look at the top one, it’s declared as public static void.

[00:01:45] Main methods in Java have to be public, static, and void, and they take one parameter which is a String array, which by convention is normally called args.

[00:01:54] Now if you compare that to what you know about C++, you will notice that there’s a difference.

[00:01:59] In C++, we have two parameters.

[00:02:01] We have one that we usually call argv, which has the values, and one called argc, which is the size of the array.

[00:02:08] The reason we have two parameters in C++ is that there’s no way in C++ to find out how big an array is.

[00:02:18] When you allocate it, you have to keep track of how big it was when you allocated it; there’s no way to find out later how big it was.

[00:02:24] That’s not true in Java.

[00:02:25] In Java, we can just call .length on an array and it will return the size of the array.

[00:02:32] Since we can always find the size of the array, there’s no reason to include the size in the main method.

[00:02:41] That second version of the main method is just a shorthand.

[00:02:47] The compiler, when it sees code with a data type followed by three dots, it just takes that as a variable arguments specification.

[00:02:58] That means we could have some number of arguments that we would specify within that main method as a comma-separated list and the compiler will just take those arguments and compile them into a String array that looks like the first example.

[00:03:14] That’s where your code goes.

[00:03:17] Now you don’t really worry about files if you’re using an IDE.

[00:03:20] You can write Java from the command line, you can do it with just a standard text editor and compile it from the command line if you want to.

[00:03:26] It’s not what I would recommend.

[00:03:28] I would recommend using an IDE.

[00:03:30] If you do that, it will manage the files for you, but it will be doing what I just described in the background.

[00:03:36] It’ll be putting your code in .java file and compiling it to a .class file.

[00:03:44] Here’s the basic syntax for creating a simple Java class.

The following lines of code are the basic syntax for a Java class:

```
public class SimpleJavaClass {

        public static void main(String [] args) {

                System.out.println(“Hello BYU!”);
```

```
    }
}
```

End of code.

[00:03:49] Usually, you use the keyword “public”, use the keyword “class”, and then you give it a name.

[00:03:54] And as I said before, that name has to match the file that you put it in.

[00:03:59] You can see we have curly braces and in this example, I have a main method that just prints out the words “Hello BYU”.

[00:04:08] Output is a little bit different in Java than C++.

[00:04:11] The way we do output is you call System.out, and then we have either “println” or “print” or a few other methods that we can call to do output.

[00:04:20] I’m going to switch over to my IDE now and show you a few examples of Java code.

[00:04:33] Here we can see in my editor the same class that we were looking at in the slide.

[00:04:38] Because it’s a main method, we can execute it from IntelliJ.

[00:04:42] There are actually a few different ways we can execute a class in IntelliJ, but one way is to just push the little green Run icon next to the main method.

[00:04:52] If I push that, I can select Run, and then the Java program runs.

[00:04:56] There’s also a Run icon by the class name.

[00:05:00] On any class that has a main method, it will have Run icon there that you can use to run it.

[00:05:06] The definition of a program in Java is a class that has a main method in it, but most classes don’t have main methods; most classes are just intended to be building blocks for other classes.

[00:05:18] Here we have an example of a class that you couldn’t run.

[00:05:23] It’s a Point class that just represents a point on some coordinate space.

[00:05:28] It has variables for x and y, it has some constructors that you’ll learn more about later for creating instances of the point class, and then it has some simple methods for getting and setting the values.

[00:05:41] It has a toString method, equals method, and hashCode method, all of which you’ll learn about later.

[00:05:47] This is just a simple class that we can use and we can use it with other classes.

[00:05:53] Now we have a Rectangle class and it represents a rectangle by specifying two points.

[00:06:01] There are different ways that you can represent rectangles, but one of them is to specify two points.

[00:06:06] You could specify a top-left coordinate and a bottom-right coordinate.

[00:06:10] This Rectangle uses the Point class that I just showed you to specify a rectangle.

[00:06:16] Here we see that one class can use another class.

[00:06:19] Again, it just has some constructors and simple methods for getting the points and setting the points for the rectangle.

[00:06:28] Neither of these classes have a main method, so neither of them could be executed.

[00:06:35] Now we have this PointAndRectangleUser class.

[00:06:39] It does have a main method, and it is using both the Point class and the Rectangle class to create points and rectangles and print them out.

[00:06:48] If we run that, you can see that it's printing out some information about the top-left, bottom-right point, and the rectangle.

[00:06:56] One of the things to notice about this is I did not go through any linking step to make this happen.

[00:07:01] I just created the classes, IntelliJ compiled them all for me while I was creating them, and then when I invoked the main method of PointAndRectangle, that got loaded into memory by the JVM, and the JVM saw that this class depended on or made use of the Point class and the Rectangle class.

[00:07:20] That's the point where it loaded those two classes in, dynamically linked them, and made them available.

[00:07:26] That shows how to run classes and how you write your code.

[00:07:32] In order to create a class, you typically will have an src directory in IntelliJ.

[00:07:37] Just right-click *(on the src directory in the left sidebar)*, go to New, Java Class, and you can create a class there. Give it a name.

[00:07:48] That creates a class, and now you can type your code there.

[00:07:52] This code is going in a class called MyClass.java.

[00:07:56] You can see that here, you don't really have to manage the files if you're using IntelliJ, but IntelliJ is managing files in the background for you.

[00:08:03] Another way to run a class is if you select a class as executable, there’s a Run icon right here *(in the menu bar towards the right)*, and you can also right-click *(on the class code)* and find Run.

[00:08:13] So lots of different ways to run a class from IntelliJ.

[00:08:18] These are just some pictures of what I just showed you. *(Wilkerson returns to PowerPoint.)*

[00:08:22] One of the things that you can do is you can recompile all of your code in a project.

[00:08:27] You do that by selecting Build *(in the top menu bar)*, Rebuild Project.

[00:08:30] You almost never need to do that because the compilation happens for you dynamically while you’re typing the code, but every once in a while, it’s possible for something to get out of sync between the source code that you’ve written and the code that IntelliJ has compiled.

[00:08:44] I haven’t seen that happen for a long time, but if it does happen, you just go to Build, Rebuild, and it will recompile all of your code.

[00:08:51] Already talked about how to run.

[00:08:53] That’s what you need to know about how to create, compile, and run Java programs using a tool like IntelliJ.