Transcript: Intro Writing Classes

>> In this module, we will take a look at the details of writing classes. In particular, we will look at the anatomy of a class, with respect to its state and behaviors. Constructors, that is how we create an instance of a class. UML diagrams, which show how classes are dependent upon each other.

Encapsulation is a topic we'll take a look at. The anatomy of a method, its parameters, local data, all the details of of actually writing one. Constant fields, public and private. Invoking methods in the same class versus from a, from another class. Building classes incrementally, testing class and interactions, and finally, writing a driver program.

So let's get started here. Thus far, you've written programs that use classes defined in the Java standard class library. You've had a driver program that is a single class with a main method. In general, that's not the way things are gonna work when programs get large. Your driver program is going to actually call methods in other classes.

And, and you'll have other classes that define objects in your program. So looking at object-oriented programing, the idea here is that a class defines a set of objects. And each of these objects hold data and, and the object will have a specified behavior. Each class should be contained in a separate file.

This actually facilitates testing. An object which is created from a class has state and behavior. For example, consider the Scanner class that we've used. We use it to create a Scanner object, that following line of source code there. Scanner input = new Scanner(System.in). Here we declared a reference variable input that can refer to a Scanner object.

And then we create the new Scanner object calling its constructor. And we send it, the variable System.in, which is, of course, the keyboard. So its state includes the target, and, which is System.in. And it also includes, the input data being scanned. That is, the state of the scanner object, itself.

And behavior for the scanner object include reading in of the next int, or next double. Reading next line and so on. Essentially all the methods defined for the scanner class. Let's consider a six-sided die. Its state might include the face value. And it would be in the range 1 to 6.

That is, we roll a die, and the face value is the surface that's, that's up. And and it'll, it can have the values of 1 to 6 there. Behaviors for the die might include roll. And again, this is where we roll and get a random value back between 1 and 6, that represents the face value.

We might have a method that sets the face value to a specific value. And we might have another method, called getFaceValue, which returns the face value that's that's currently up. The way we would use die in a program, we could declare a variable, let's say a DieObj of type die to be a new die.

This is called the die constructor. And then we might say dieObj.roll. And that would actually roll the die and presumably we have a five-sixths chance of getting a new face value. And if we want to get that face value, we might say int rollResult = dieObj.getFaceValue. Now the way we might organize the die class.

Generally in the class, you put the the fields upfront, and here we got a a static variable. Rather it is static, and final. It is a constant, MAX = 6. That's a maximum number of face, of the face value. And then we've got an instance variable, face value called int faceValue.

And this is where the, that value for the faceValue will be, of course. And this is followed by the, constructor, and methods. Here we, we show roll a dot dot down to toString. And we'll take a look at those here directly. So once you get this, you can create multiple die in a program.

A program will not necessarily use all of the aspects of a given class. It might not use the the static constant there. It may not use some of the other methods there. But next let's take a look at at the source code for the die class. And also RollingDice which is the has a main, main method in it.