Transcript: Introducing Classes & Objects

>> Now let's take a look at using Classes and Objects. In particular, we're gonna take a look at reference types and object creation, the String class and its methods, packages and the import declaration. The Random class, which we'll use for generating pseudo-random numbers. The Math class, which has methods for square root, exponentiation, all of the trig functions, and so on.

We'll look at formatting output using NumberFormat and DecimalFormat. We'll finish up with wrapper classes, which are used to create objects of the primitive types. So, let's begin with the review of primitive types. Recall that a variable can be used to store a primitive type, for example, if we declare number to be type int, this declares a, a variable, that can hold a 32 bit integer.

We could then come along and use an assignment statement to set a number equal to 67. And in this case, the variable number now actually holds the value 67. Recall that Java has eight primitive types. Byte, short, int, and long are all integer types. They're 8 bit, 16 bit, 32 bit, and 64 bits, respectively.

And then we've got two floating point types, float and double, double is 64 bits. Of these we'll probably use int and double most of the time. We also have char, which holds a single character. And then Boolean, which can take on the value true or false. All other types in Java are object types or reference types.

So let's look at the, at the basics here of objects. Objects are defined by classes. The type of an object is a class name, rather than one of our primitive types. So if we want to declare a variable that can reference an object, then we declare using the class name.

For example, a string object, we could declare the variable title to be type string. And then objects are created with a new operator and a variable, variable can then be assigned the reference to that object. For example, here we say title = new String and then we send it the string literal using classes.

We could also have declared the variable and, and assigned it all at once. Here in this example we're saying string team equals a new string Red Sox. Now the string is used so often that Java allows a shortcut version of this where we don't use the new operator.

In, in this, example we're saying String location equals "Shelby Center", and this actually creates the String Object and, and then assigns the reference to, to location. So, more about creating objects, again, an object variable is really a reference variable. It doesn't hold the object itself, but rather a memory location where the object is stored.

So, if you think of primitive types as suitcases that actually hold the contents, say an int, or a double. Then you can think of reference variables, as suitcases that are actually called the memory location that contains an address, that points to the actual contents, if you will, or say, a string.

So, graphically it looks like this, here we've got a primitive type, num 1, that actually holds the value 52. The and next we have reference type name 1, this would be a string. And, and it's, it references Steve Jobs, so it was created and, and it's actually pointing at Steve Jobs.

And, and the reason we do this, for instance for a string, we don't know if the string is going to be 10 characters or 1,000characters. And so what happens, the memory address just points to some place that Java found room to store it. So, the declaration itself doesn't create an object, it just says that this particular variable can reference an object of that type.

So in this case here string title can really point to any string, but that's all it can point to. Now, if we haven't created an instance, we can actually set the variable to null, to indicate that no object has been created, but it would allow us to actually check the value.

So we could, if we set title equal to null, then we add an if statement say title == null. And if it does equal null, then we could maybe print out that No title's been set yet. We couldn't call a method on it, we'll see, because we can't call a method on a null object.

Now notice that that null is not the same here as, as setting title to quote, quote. Quote, quote would just be setting it to a, an empty string or a string of length zero. Now the new operator, which is used to create objects, in this case here we've got title, = new String.

And we're gonna set the title to the String literal Using Classes. This string here, no, notice it's, it's the name of a class. Class string, in fact, it was typed the, the type of title is string and then we're using that same name string we call this the constructor.

Every class has a constructor. If, if one's not created for it, it get a default constructor with the same name of the class. And in this case, we're, we're using this, the string construct, constructor to create a new string object based on that value passed in. Creating an object is called instantiation.

It just means it crates an instance of the class, an instance of a class is an object, and, and so on. And we've used this before here, creating a new scanner on System.in and, myScan is just pointing to an instance of the scanner class, or, or scanner class object, if you will.

Now, objects, unlike primitives, can have methods, and this provides functionality. We've used some of these. For example, on the scanner, we've used nextint, and we call that on the scanner object. For instance it's invoked using the dot operator, myScan.nextInt, gave us the, an integer value, if you will.

Well on, for strings, we can do things like title.length. And this returns an integer value that's the length of title, where title is our string. And, in that first statement there, we're actually assigning it to the, variable count, which has been declared an int. So title.length actually returns an int value, and we've assigned it.

In the second statement here, rather than assigning it we're just saying, length is. And then we're, we're actually printing out, we're calling the, the, a length method on title right there in the print statement. And that's gonna return a, a value as well. A method may accept parameters or arguments as input.

And maybe not have a return value. Here is one here myScan.useDelimiter, this would tell the scanner that we're gonna delimiter the values with a comma rather than white space. Or you might have both a return value and parameters, in this case, we're saying title, this was a string, remember.

And we're calling the charAt method on it, and we're saying we want the character at location 2 and title. We're gonna get that character back and then we're gonna assign it to the char variable, single letter. Let's

look at some examples of this using interactions. So in jGRASP, recall down at the bottom you got several tabs there.

One is interactions, if you go to interactions, you can hit return or so to actually start it. Here you can actually declare some variables and, and assign them. So let's create a string title that's we've been playing with here, and let's it equals using classes. If I get it spelled correctly there.

Using classes. Now when I hit enter here on the keyboard, this is gonna create the string object called title. And it's now up on the workbench. This is just another tab, we call that the debug tab. This is the workbench tab and it comes up automatically when we declare this.

And we can see that it's, it's equal to using classes. We can also throw this on the Canvas. Let's just throw this on the Canvas, oops, we missed the Canvas there. If you don't put it on Canvas it opens it individually. I'm gonna actually put it on the canvas there.

With it on the canvas you can save it, so that's kind of one useful thing. Now, when you do open a viewer on this or put it on the canvas, the default view is going to be formatted. So, I'm going to change it back to that. This is what you would normally see, but in this case I want to change the, the viewer.

I wanna change it to the character array so I can actually see the individual characters there. So now down here in in our interactions tab, I'm gonna say, I'm gonna call some methods on title. Let's say title.length, and all method calls have parentheses. You may have an argument in there, or you may not.

But this should return the length of title. And we see here, looking at our, our viewer that it's got characters, and they're indexed 0 to 12, so that's 13. So it has linked 13. So we might might want to see what the upper case version of this looks like, so we might say title.2 two upper case, again, parenthesis.

Now notice I'm just entering an expression so this is just giving the back a value, and there's using classes in all uppercase. Notice it didn't affect string, the string title itself. So title is not changed here. If I actually wanted to change title I could assign it to that.

I could've said title = title.toUpperCase() and made that a statement. And when I do this, the return value that comes back actually gets gets assigned, it's a new string that comes back. You, you saw it right up here using classes. Well this string reference then gets assigned to title, and, and so title changes there.

So let's, let's hit return and now we see up here at the top, we see the letters that changed, the capital U was already there and so on. And so we see that lots of different sting methods we can call. Let's do one more caret. Let's do title.umcharAt, so this is just gonna return a string, rather a char value and let's get the charAt let's get the charAt 4 there.

So, I am looking at, at four, so this is index 4 And that returns the G there it's at index 4. We'll be exploring a lot of these as we go and lot more about objects and methods to come here.