CS 240: Java Records Transcript

[00:00:00] A really nice addition to some of the later versions of Java is Java Records and I want to give you a little bit of background information to understand why we need them, why we care about them.

[00:00:10] So first of all, it's very common to have Java classes that really only exist to represent data.

[00:00:16] So you'll have some instance variables, a few methods for accessing those instance variables and a few other things like maybe a um equals method, a hash code method and a two-string method.

[00:00:27] And sometimes that's all we have in our class.

[00:00:30] So a lot of people call those POJOs just plain old Java objects.

[00:00:34] And so we, we write those by creating a class, creating some instance variables.

[00:00:39] And then we can usually get the ID E to generate most of the rest of the code for us.

[00:00:43] We can get the ID E to generate um getters and setters um equals hash code and two string methods.

[00:00:49] But we end up with a lot of code.

[00:00:51] And so um a lot of that code is like I said, it's so common that ID ES can generate it just from having the name of the instance variables.

[00:01:01] Well, if it's so common that ID ES can generate it, maybe you shouldn't be required to write it and that's what records do for you.

[00:01:09] Um Now, before records, there was a, a library called the Lombok library that basically did the same thing and made it so you could um write less code and then Lombok would create um the structure that, that you could kind of derive from the instance variables.

[00:01:24] But now that we have Java Records, we don't really need a library like Lombok.

[00:01:29] So here is an example to show you what I'm talking about.

[00:01:32] So here we have this class, it's called the pet class and notice that it has three instance variables, it has a constructor and then it has get and set methods for in each of the instance variables.

Start visual description. The professor demonstrates the creation of a Java class called pet with three instance variables. The screen shows the class definition, constructor, and getter methods for each instance variable. End visual description.

[00:01:44] Um Or actually I think it just has methods me to put my reading glasses on.

[00:01:49] So I can see it just has um get methods and then it has equals hash code and two string.

[00:01:53] You can get your ID E to generate all of that just from the first four lines um just from having the name of the class in a class file and these three instance variables, you could generate it, but it's actually 52 lines of code.

[00:02:06] So that's a lot of code that you can tell what the code should be just by looking at these four lines.

[00:02:12] So that's what records do for us. They make it so we don't have to write all of this code so we can create a, a record.

[00:02:20] And so to create one, we just substitute the word class for record.

Start visual description. The professor demonstrates how to create a Java record by substituting the word class with record. The screen shows the simplified syntax and the resulting code, which is significantly shorter than the class definition. End visual description.

[00:02:24] You still put it in a dot Java file just like you put a class.

[00:02:28] And when you compile it, it still gets compiled, and it still generates a um dot class file that has the byte code in it.

[00:02:36] But this record is equivalent to all of that code.

[00:02:41] So if I create this record, public record pet, um and then I specify the parameters in a constructor, there's just one constructor in a record and opening and closing parentheses and that's all I need.

[00:02:52] So we just reduced 52 lines of code down to one and this class is basically the same as this class. It has all of the same.

[00:03:02] Um It has the three getters, and it has the equals hash code and two string methods.

[00:03:08] So here are some features of records.

[00:03:10] First of all, they're immutable, which means they can't be changed.

Start visual description. The professor demonstrates the immutability of Java records. The screen shows that once a record is created, its instance variables cannot be changed, and there are no setter methods available. End visual description.

[00:03:13] Once you create a record, you can't change its instance, variables, all of the fields are final and there are no setter methods available for them.

[00:03:21] Um It’s a simplified constructor syntax. So, you don’t have multiple constructors.

[00:03:26] You don’t, you can’t overload the constructors, you just have one that takes all of the parameters.

[00:03:31] So you specify all the parameters when you construct the instance.

[00:03:35] But you still notice we’re constructing the instance just like we would for a class, it has automatic getters.

[00:03:41] So it automatically generates the Getters.

[00:03:43] So even though we don’t see any code, this is the record.

[00:03:46] So this code right here is the full um uh sorry, this is the full record, and this is creating an instance of it.

[00:03:53] Once we create an instance, we can access the name variable by, by just using our reference and calling dot name.

[00:04:02] Now, one thing, I’m not sure why they did this, but they didn’t follow their own convention for those Getters.

Start visual description. The professor demonstrates how to access instance variables in a Java record. The screen shows the use of the dot notation to access the name variable of a record instance. End visual description.

[00:04:07] So the Java convention for get methods has always been um that the Getter is the word get followed by the name of the instance variable with the first letter and upper case.

[00:04:17] They didn’t follow that in records.

[00:04:19] And so if you declare a name variable, then it’s Getter is just name without the word get in front of it.

[00:04:30] Records also automatically generate an equals method.

Start visual description. The professor demonstrates the automatic generation of methods in Java records. The screen shows the equals, hashCode, and toString methods that are automatically generated for a record. End visual description.

[00:04:33] A hash code method and a two-string method.

[00:04:35] Basically the same um versions that your ID E would generate for you if you told it to generate those.

[00:04:42] OK. Um So these are, are kind of simplified classes.

[00:04:46] They, they get used a lot and they're really useful.

[00:04:49] One thing that we sometimes need to get around though is the fact that they're immutable.

[00:04:54] So remember I told you that you can't change your record.

[00:04:57] But what you can do is you can add methods to the record that will generate a new record from an existing one.

[00:05:03] So here we have our, our pet class or our pet record that has an ID E or an ID, a name and a type.

Start visual description. The professor demonstrates how to add methods to a Java record to generate a new record from an existing one. The screen shows a rename method that creates a new pet record with a modified name variable. End visual description.

[00:05:11] And we could just write a rename method that will look like we're renaming the pet.

[00:05:15] But notice what it's actually doing is it's returning a new pet.

[00:05:19] Um We have the same thing with strings. Strings are immutable.

[00:05:22] So you can’t actually change a string. If you have a string, even if you do something like string concatenation, you’re not changing the string, you’re just making new strings.

[00:05:31] And so it’s the same thing here.

[00:05:33] When we rename a pet, we don’t actually rename the existing pet object.

[00:05:37] We create a new one.

[00:05:38] So here we’re passing the, the name variable into this rename method and we’re creating a new pet and notice that for the id and type parameters, we are accessing them.

[00:05:51] We’re accessing the instance variables from the original pet record, but we are setting the name to be the one that’s passed in.

[00:06:00] So that allows me to generate a new object from an existing one.

[00:06:06] And by using that, um by um by creating methods like that, you, you’ll find that records can be really useful.

[00:06:14] You can use them in a lot of situations, and you’ll write a lot less boilerplate code.