CS 240: Classes and Objects-Overview Transcript

[00:00:00] **PROFESSOR:** Today we're going to start working on the spelling corrector project. Our goal here is to prepare you to successfully complete that project. It's the first one of the semester and you need to start working on it really soon,so you'll have enough time to finish it.

[00:00:18] The purpose of the spelling corrector project is to help you learn how to write aJava class correctly. The first thing we're going to do is we're going to talk about how to write a Java class correctly. Then after we learn how to do that, then we're going to talk about the spellingcorrector project itself and a lot of the details about how you should go about doing that project.

[00:00:39] Let's start with learning how to write a Java class. I'm going to do that by demonstration. I'm going to open IntelliJ on my computer (*Professor opens IntelliJ program*) andI'm going to **Create a New Project** (*Professor clicks on the **Create New Project** icon*). It's going to be a Java project (*Professor selects **Java***), and I'm just going to useall the default settings (*Professor selects **Next** button*).

[00:01:03] I am going to give it a name that I want it to have…

*Start visual description. Professor selects a dropdown icon next to the **Project location** box. He selects the **01-Classes and Objects** folder, then moves down and selects the **Demo** folder. End visual description.*

[00:01:10] I'm going to select the folder that I'm going to put the project in, and then I'mgoing to click the **Finish** button, which will create the project. When the project comes up, of course, it's empty. (*Professor selects the Demo folder icon*) And so what we're going to do is we're going to drill into the project.

[00:01:28] We're going to go into the source folder, **SRC**, (*Professor selects the SRC folder)* and on the SRC folder what I'm going to do is I'm going to right-click and select **New**, **Java class** *(Professor selects **New,** then moves to the dropdown menu and selects **Java Class**)*. That's how you create a new class, is just right-click on the folder where you want to put it and select **New**, **Java class**, and then I'm going to give the class aname.

[00:01:49] In this case, we're going to create a class named "person" (*Professor types "Person" into the name field)* and so IntelliJ creates an empty person class for us.

Start visual description. Professor places the cursor before the first line of code and bumps it down two lines. He then selects the square bracket directly beneath the first line of code and bumps it down four more lines. He moves his cursor up, placing it two lines beneath the first line of code. He indents his cursor. End visual description.

[00:02:03] What I'm going to do is, I'm going to give this "person" class some properties.I'm going to give... I guess people have names, so we'll give it a "name" property and people have ages, so I'm going to give it an "age" property.

Start visual description. Professor types the following code:private String name; private age. End visual description.

[00:02:18] In this case, let's assume that people may have a year in school if they're a student, so I'm going to create a property which is of type "YearInSchool".

Start visual description. Professor types the following code directly beneath the previous code: Private YearInSchool year;"YearInSchool" is colored red. End visual description.

[00:02:40] Now the challenge with the "YearInSchool" is that that data type doesn't exist.It doesn't know what "YearInSchool" is; that's why it's colored red. What I'm going to do is I'm going to create another Java class. Right-click; **New**, **Java class** (*Professor repeats aforementioned process for creating a new class*), and I'm going to create a new class called "YearInSchool." (*Professor types "YearInSchool" into the name field.)* But in this case, I'm going toselect **Enum** as the class type, E-N-U-M, and that means that this new class will be an enumeration.

Start visual description. Professor puts his cursor before the first line of code and bumps it down two lines. He then selects the square bracket directly beneath the *first line of code and bumps it down three more lines. He moves his cursor up, placing it one line beneath the first line of code. He indents his cursor. End visual description.*

[00:03:04] What this does is it creates a new datatype called “YearInSchool” and what I need to do now is specify the different values that “YearInSchool” can take on. I’m going to say “year” in school can either be “freshman”, “sophomore”, “junior”, “senior,” or “grad” student. (*Professor types each grade in all caps, with commas between each word.*)

[00:03:26] That creates a datatype that can take on only those values. For example, in a program, if I declared a variable of type “YearInSchool”, I could set it to have a value of “YearInSchool.junior”, if I wanted to for example.

Start visual description. *Professor moves his cursor down three lines from the line where he listed the years of school. He types the following code:YearInSchool year = YearInSchool.JUNIOR. End visual description.*

[00:03:50] “YearInSchool” is a datatype that can take on the values “freshman”, “sophomore”, “junior”, “senior”, or “grad”. And you’ve probably seen this before in C++. This is how they do it in Java.

[00:04:00] Now in C++, an enumeration underneath is really just an integer. In C++ “freshman” might have the value “0”, “sophomore” might have the value“1”, etc. That’s not how it works in Java. In Java, enumerations are actually classes. “Freshman” is an object, “sophomore” is an object, etc.

[00:04:20] In Java when you create an enumeration, really you’ve just created a new class that has a restricted set of values that a variable of that type can take on. That’s going to be really important later to remember that enumerations are objects; they’re not integers.

Start visual description. Professor deletes the following code:YearInSchool year = YearInSchool. JUNIOR. End visual description.

[00:04:34] Having created the “YearInSchool” enumeration, let’s go back to the “person class”. (*Professor selects the “person class” file).* Now the compiler is happy because we created the “YearInSchool” type. Next thing I want to do for these fields that I’ve created is create getters andsetters for those. And I could write those by hand, obviously, but it’s really better to let yourdevelopment environment write your getters and setters for you.

[00:05:00] In IntelliJ, what I can do is I can go up to the **Code** menu and I can ask it to **Generate** some methods for me and it’ll ask me what methods I want to generate. In this case, I want to generate getters and setters, so I select **Getter-and-Setter** from the list and then it asks me, which fields do I want to create getters and setters for? In this case, I want to create getters and setters for all of the fields. I’m going to select all of them (*Professor selects “name:String”, “age:int”, and “year:YearInSchool”*) and just say **Ok**.

Start visual description. The following code appears on screen:public String getname (){return name} public int get Age (){return age}; public void set Age (int age) {this age = age}; public YearInSchool getYear () {return year}; public void set Year (YearInSchool year) {this year = year}; End visual description.

[00:05:29] Just that easily you can see that IntelliJ wrote the getter and setter methods forme. That’s a nice way to save yourself some work. The next thing we’ll do on this class is we’ll add some constructors.

Start visual description. Professor puts his cursor two lines under the following code:Private YearinSchool year;End visual description.

[00:05:48] In this case, we’ll have a constructor that allows the caller to initialize all thefields. We’ll have a parameter for the “name”, we’ll have a parameter for the “age”,and also the “YearInSchool”.

Start visual description. The professor types the following code:public Person (String name, int age, YearInSchool year). End visual description.

[00:06:10] Then inside my constructor, what I'll do is I'll just call the setter methods to initialize those fields with the parameter values.

Start visual description. Inside the curly brackets, the professor types thefollowing code: setName (name);setAge(age); setYear(year). End visual description.

[00:06:26] Now I can create a "Person" object and I can initialize all the fields with that constructor. Let's add another constructor as well. Let's add a second constructor. In this case, we notice that most people are freshman when they start schooland so it's convenient perhaps to have a constructor that just initializes the "YearInSchool" to "freshman" automatically. In this case, we'll create a second constructor that only takes a "name" and an"age", and it automatically initializes the "YearInSchool" to "freshman".

Start visual description. The following code appears two lines under the code previously described:Public Person(String name, int age). End visual description.

[00:06:57] Now to do this, rather than duplicate the code that I had in the first constructor, what I'm going to do is I'm going to have the second constructor actually call the first one. In Java, constructors can call each other and you do it with the "this" keyword. In this case, this constructor's going to call the first constructor, it's going to pass in the "name" and the "age" that were provided, and it's also going to hard-codethe "YearInSchool" to be "freshman".

Start visual description. Inside the curly brackets of the most recently-added code, the following code is typed: This (name, age, YearInSchool. FRESHMAN).End visual description.

[00:07:27] That just shows you that number one, you can have multiple constructors on a class, and number two that constructors can call each other. That's a nice feature. There we have some constructors. Now one thing to notice in Java also is that each member of a class is individuallylabeled as "private" or "public" or "protected".

[00:07:50] This is a little bit different than C++, where in C++ you have blocks of "public"things and blocks of "private" things. In Java, you actually label each member individually as “public”, “private”, or“protected.” That basically shows you how to create a simple class in Java.