AUBURN UNIVERSITY

Transcript: Syntax and Semantics

>> Now let's consider more details regarding syntax and semantics. By syntax we mean grammar. In other words the rules of how the vocabulary in a language can be used to compose legal structures. In the context of programs, the language syntax describes how we form legal statements and other constructs in the language.

Semantics means meaning. That is, what a given legal structure in the language means. And again, in the context of programs, the language semantics describes what's gonna happen when a legal statement in the language is executed. In natural languages, some things can be syntactically correct but have no meaning.

Blue ideas sleep furiously. It probably means something to someone, but I'm not sure what that means. Or these statements can be syntactically correct and have many meanings. Time flies like an arrow. The house flies like a saucer. We might think of a house flying like a flying saucer.

Or we might be on a picnic, and we might think of those pesky little critters that, like a saucer with some crumbs in it. Did you ever see a home run? Well, the good news in programming languages, we don't have these situations. There is no ambiguity. So, once we've got a program that's syntactically correct and compiles, it's gonna have the same meaning every time it's run, assuming the same input and any other state there is the same.

Now let’s talk about program errors. There are three categories that you want to be aware of. We have compile-time errors, and what that means is compilation didn't complete. These are syntax errors, and it may be some static semantic errors as well. When you write your program, if you leave the semicolon off of the end of the statement and compile it, it won't compile.

That's an example of a compile-time error. And when this happens, the no bytecode is produced, that is you don't get the dot class file. Logical errors, sometimes called logic errors, this occurs when the program actually compiled okay and even executes to completion. In other words, you run the program and it runs just fine, halted it normally, ran all the way to the end.

However, the results weren't quite right. So, there wasn't a problem with the compiling and it, and it ran okay, it's just that the results weren't what we expected. And then there are run-time errors. And these mean that the program ran and while it was running, it halted abnormally.

In other words, it didn't make it all the way to the end. We have names like this deep-end, crash, blow up, and so on. Maybe illegal operations or exceptions. If you wanna create one of these exceptions in your program, you could, at some point, say, in the middle of your program, divide an integer by zero.

So, you could just add to your print statement that you wanna print ten divided by zero. And when you run that program, it's gonna actually run down to there. When it gets to that statement, it'll actually blow up. Finding errors, we usually do this by testing, as in we identify that there is an error.

In other words, we observe that the output was not correct. That's how we find them. And then we remove them by debugging. So, that's, that's a different exercise. And sometimes, in industry, one group does the testing and finds the errors, and then a whole other group actually does the repair.

Actually debugs the program, and finds the fix. Let's look at a bit of an overview of programming language to see how Java fits in. Just remember a programming language is just an artificial language created by humans to express programs. And then these programs that are expressed in this artificial language are translated into a machine executable form, in our case a class file.

And we have several categories. First, machine languages. You can think of these as very close to the machine, probably binary. And then we had the languages -- assembly languages. And we added a few mnemonics there, but still pretty detailed and low-level. And then, third we have high level languages.

The first high level languages were Fortran and COBOL. And then came Java, C++, Python and many other languages, of course. But I guess Java, C++, and Python are popular today, we shall say. And then a given high level language statement probably represents ten or so assembly language statements.

And even more in machine language. Now, languages in the different categories, say assembly versus a high-level language, are obviously gonna be very different. But even the languages in the same category can vary widely. But the good news is after you learn a language, you can usually move to a high-level language to another high level language.

And, and, and make some sense out of it. Here's some examples. This is Java. This is our War Eagle program where we're printing out War Eagle. Here's essentially the same program in Ada and you can see the program is all different syntactically different reserved words and so on.

No braces for example, but it essentially prints out War Eagle. Here's the same program in C. Again we're printing out War Eagle. And here it is in Perl. So, you see there's a lot of similarity here. And, and the good news is after you learn a language really well and possibly a second language really well, it's pretty easy to go from one language to another.