CS 240: Java Architecture Transcript

This video shows a split screen of Professor Wilkerson on the right and a PowerPoint screen on the left. Visual descriptions are not needed.

[00:00:00] **JEROD WILKERSON: It** is important for you to understand the architecture of Java, to understand how it works compared to other programming languages.

[00:00:06] As I mentioned in an earlier segment, Java is sometimes referred to as a hybrid language; it's a combination of both compiled and interpreted.

[00:00:14] So we need to understand what it means for a language to be compiled, what it means for it to be interpreted, and then how Java is in between the two.

[00:00:23] With a compiled programming language, you have your source code, and the idea is that you write your source code once and then you run it through different compilers, depending on what platforms you want the code to run on.

[00:00:36] You can see in this visual, we have the source code.

[00:00:40] If we wanted to run it on a Mac, we would compile it with a Mac compiler and that would give us executable code for the Mac, which we could then run.

[00:00:49] If we want to run it on a PC, we'd run it through a PC compiler and that will give us an executable for the PC and we could run the program on the PC.

[00:00:57] The idea is that we shouldn't have to change the source code.

[00:01:00] We should just be able to run through different compilers.

[00:01:02] That was the big deal of C when it first came out, it was supposed to be portable and what I just described was supposed to work really well.

[00:01:09] For the most part it did, but unfortunately, it didn't completely work for a few reasons.

[00:01:13] First of all, I mentioned earlier in a previous segment that data types can be different sizes on different platforms with most languages, and that's true with C and C++.

[00:01:25] If you're writing code that has some expectations about data type sizes, then that code has to be rewritten for different platforms.

[00:01:35] Also, C and C++ don't have built-in support for multi-threaded programming or network programming.

[00:01:42] To do either one of those, you have to use operating system calls, and of course, those operating system calls are different for different operating systems.

[00:01:50] As soon as you use any of those features which are quite commonly used in programs, you have to have specific source code for a specific platform.

[00:01:59] Those are the disadvantages of fully compiled code, but the big advantage is compiled code is really fast.

[00:02:08] We also have interpreted languages, and those are fundamentally different.

[00:02:13] The way interpreted languages work is you have your source code.

[00:02:16] To run it, you just run it on an interpreter for that language for a specific platform.

[00:02:22] That is very portable and it actually works.

[00:02:24] So you write your source code once and you're able to run it on any platform that has an interpreter.

[00:02:30] It's very portable, that's the big advantage.

[00:02:32] But the disadvantage is it's slow compared to compiled code.

[00:02:36] When they developed Java, they wanted both advantages.

[00:02:39] They wanted portable code that was fast.

[00:02:41] The way they did that, is they made Java a hybrid.

[00:02:44] With Java, we still read our source code once and it actually works that you write your source code once.

[00:02:50] There's not really a way to write different source code for different platforms.

[00:02:54] You run it through one compiler and that gives you something called Java byte-code.

[00:03:00] Byte-code looks like fully compiled code for some specific platform, but it's really an imaginary platform.

[00:03:07] Sun created what they call the Java Virtual Machine specification, which is a specification for a hardware machine that at least at the time didn't exist.

[00:03:17] So the compiler would compile code to that specification.

[00:03:21] It turns out that the compiled code looks almost the same on most or all platforms.

[00:03:28] Then you can take that Java byte-code, which can now be ran on an interpreter.

[00:03:34] The interpreter for Java is a JVM that stands for Java virtual machine.

[00:03:39] There's one for basically any platform you could want to run code on.

[00:03:43] That JVM will take the Java byte-code and translate it to be specific for that operating system.

[00:03:49] So it will run on the operating system the JVM was written for.

[00:03:53] Because byte code is almost exactly like the machine code for any particular machine, there isn't a lot of translation that needs to happen, and so it's still very fast.

[00:04:04] We get the benefit of being portable and we get very fast code.

[00:04:10] Just to summarize: compiled code is really fast but not portable, interpreted code is slow but portable. Java seeks to have the best of both: it seeks to be fast and portable.

[00:04:22] Now if that was the end of the story, I would have to tell you that Java is fast, but it's not as fast as C or C++, it's not as fast as fully compiled code, but the fact that Java has a unique architecture provides a way to optimize Java that in some ways goes beyond what you can do with compiled code.

[00:04:58] In Java, we have two things that can make Java actually faster than fully compiled C or C++ code.

[00:05:04] One is JIT compilation and one is the hotspot virtual machine.

[00:05:08] Those two things work together.

[00:05:10] When the JVM, the interpreter, when it interprets code, it actually compiles it as it goes.

[00:05:17] Now that code is compiled, and so it runs as compiled code once it interprets it the first time.

[00:05:23] That would still make Java a little bit slower than fully compiled code, expect we have something called the hotspot virtual machine, which allows us to optimize Java code.

[00:05:34] What the hotspot virtual machine will do is it actually will dynamically recompile code while it's running as it determines that it can make it run faster in a different way.

[00:05:44] If we think about what a compiler has to do when it compiles code, it compiles code at one point in time, and then sometime later, the code runs.

[00:05:56] The compiler doesn't have all the information that we would like it to have to be able to make the best decisions possible and how to compile the code.

[00:06:04] There are different trade-offs that have to be made by a compiler.

[00:06:06] For example, if you have one method, method A calling method B, there are a few different ways that could be compiled.

[00:06:14] We could make it so method A and method B are two different pieces of code.

[00:06:19] In the compiled byte code, method A goes to wherever the method B is compiled and executes it.

[00:06:27] Or we could do something called in-lining where we could take that method B and actually just put the code in method A in the compiled version as if it wasn't a separate method.

[00:06:37] That's usually the fastest way to compile the code, but the downside is if method B is called from lots of different places, we'll have to inline it to lots of different places and that'll make the executable bigger.

[00:06:48] As the executable gets bigger, the code is slower.

[00:06:52] There are trade-offs and that's just one example.

[00:06:54] There actually are many trade-offs that a compiler has to make.

[00:06:57] Compilers are really good, especially C and C++ compilers because they've been around for a long time.

[00:07:03] They make good decisions about that and they create really fast code, but they are making trade-offs.

[00:07:12] With the hotspot virtual machine, and all Java virtual machines are hotspot virtual machines now, so with these hotspot virtual machines, they keep statistics while the code is running.

[00:07:24] If the JVM is ever able to determine that the way that it compiled the code is not optimal, it will recompile it on the fly while the code is running.

[00:07:32] Because of that, there actually are benchmarks where Java runs faster than C and C++ code.

[00:07:39] Java is very fast.

[00:07:40] I'm not going to say that it's always faster than C and C++.

[00:07:43] In fact, probably in most benchmarks, it's not, but it is very fast.

[00:07:47] One of the reasons that it's not quite as fast as C and C++ in most benchmarks is that Java compilers are doing more and Java code is doing more than C++.

[00:07:58] For example, in C++, there are no checks for boundaries of arrays.

[00:08:05] When you're accessing an array, there's no code that's checking to make sure you don't go beyond the bounds of an array.

[00:08:10] That's really dangerous.

[00:08:11] In Java, the compiler will build in checks for that to make sure that you don't access memory outside of an array if you're accessing an array, but that takes some CPU time.

[00:08:23] Because of things like that, Java is safer, usually a little bit slower, but it's still a very fast language compared to other languages.

[00:08:33] There other ways that the hotspot virtual machine can make Java run really fast, and one of them, I mentioned the garbage collection.

[00:08:42] Garbage collection usually makes code runs slower than languages that don't have it, but Java uses what's called a generational garbage collector.

[00:08:51] What we found over time is that most memory that gets used in a program is either used for a really brief time and then not used again, or it's used for the entire length of the program.

[00:09:04] By knowing that, they're able to take memory that's allocated for a Java program and allocate it into different places.

[00:09:12] For example, if memory has been used for awhile and it's still being used in the program, it goes into a part of memory that's not checked very often by the garbage collector.

[00:09:23] The garbage collector checks newer memory and frees it up more often, and that allows it to run faster.

[00:09:30] Even garbage collection does not have the same performance penalty than it might have in other languages.

[00:09:36] That's an overview of the architecture of Java and why we call it a hybrid compiled and interpreted language and what makes it pretty unique compared to other languages.