

Inside the Java Compiler and Virtual Machine

Nosheen Zaza
nosheenzaza@hotmail.com

And you are....?

- I am a former TA at this department.
- & Before that, a student here as well.
- Now a master student at Uppsala University in Sweden.
- There I work with the programming languages group, and we do many sorts of interesting things.
- Happily married to this subject's lecturer :)

You all (hopefully) have done these.....

- Wrote some Java code using some text editor, IDE...
- From some menu or toolbar, clicked on (Compile).
- Fixed compile time errors (in most cases).
- Then from some other, or the same menu or toolbar, clicked on (Run) or something similar
- Your program, worked, crashed or whatever.
- But, **do you really know what happens under the hood?**
- During this lecture, I will try to show you what really happens when you compile and run Java

When you click (Compile)...

- Your text editor or IDE runs the Java compiler, it gives it all the names of files you are compiling as parameters.
- You can do that on your own as well.
- But where is the compiler? And How does your editing tools knows where it is?
- Can I have more than one Java compiler on my computer?
- In what programming language is the Java compiler written?
- Can I look at the source code of the compiler?

So the Java compiler is...


- A Java program itself!
- It takes Java source code, it translates it into Java byte code...
- Which is written to one (or more) Java class files.
- Ever wondered what is inside a Java class file?

Inside a Java class file

- I bet many of you had opened Java class files before (mistake, out of curiosity), most likely using your text editor.
- Any success?
- I would say what most of you saw, was something like this...

◆◆◆◆^@^@^@2^@^]
^@^F^@^O ^@^P^@^Q^H^@^R
^@^S^@^T^G^@^U^G^@^V^A^@^F<init>^A^@^C()V
^A^@^DCode^A^@^OLineNumberTable^A^@^Dmain
^A^@^V([Ljava/lang/String;)V^A^@
SourceFile^A^@
Test.java^L^@^G^@^H^G^@^W^L^@^X^@^Y^A^@^
VHello compiler course!
^G^@^Z^L^@^[^@^\^A^@^DTest^A^@^Pjava/lang/Ob
ject^A^@^Pjava/lang/System^A^@^Co\$
^@^@^@^F^@^A^@^@^@^A^@◆^@^K^@^L^@^A
^@ ^@^@^@% ^@^B^@^A^@^@^@
◆^@^B^R^C◆^@^D◆^@^@^@^A^@
^@^@^@
^@^B^@^@^@^C^@^H^@^D^@^A^@^M^@^

But why?

- Since Java class files are stored in binary format (sequences of zeros and ones), not text.
 - Another why?
 - To make IO faster!
 - For real applications (and even the tiniest ones), thousands of classes are compiled and loaded (I will show you in a while).
 - (Good you too do this, when you want to read and write files as fast as possible)
- So if class files are sequences of digits, why all I see is  `^@....?`
- Since your text editor interprets these digits the way it likes! It does not know the right way to interpret them.

Then how do I know what they mean?

- Java class file format has it all

<http://docs.oracle.com/javase/specs/jvms/se5.0/html/Overview.doc.html#32310>

- OK, I know, a bit hard to understand.
- So first I will show you what is inside a class file, from a tool you all have and can read it almost decently.
- Then I will show you another tool that does this task a lot better.

javap

- Comes with JDK
- Disassembles class files (yes, if someone has the class file, he can get the original source code from it very easily, and yes there are ways around this).
- Also interprets the class file, as it is.
- Type `javap --help`
- Useful options: `-c`, `-verbose`, `-s`

There are other options

- Depends on what you want.
- I like and use ClassEditor.
- Since it better reflects the structure of a class file.
- You can edit the class file directly!

Nice, but how is any of this useful?

- Some interesting applications:
 - Say you want to know the time spent in each method in your application.
 - You want to know the sizes of all objects created during the running of your application.
 - Your application contains a 1000+ source files.
 - Manually??
 - Build a tool that analyses the source code?
- You can of course transform the source file.
 - However, it is closer to natural language than the bytecode, and more flexible
 - Harder to reason about.
 - More cases to handle.

Tools to help you

- Many good byte code transformation libraries.
- I use ASM.
- Some other commonly used ones:
 - Javaassist: Claimed to be simpler to use than ASM.
 - Both source and byte code modification.
 - Thus, you do not need to know the specification of the byte code.
 - Soot: Mainly used for optimization.

That's all I wanted to mention about compilation

- Questions?
- Anything else you want to know?

Running Java Applications

- Your application cannot run directly on your machine, on its own.
- The bytecode is not machine code, your machine cannot understand and execute it.
- In your application, you do not deal with memory directly (allocation, freeing...), but this does not mean that this should not be done.
- Your application runs with the help of a virtual machine.
- This machine interprets the byte code, converts it to machine code suitable for your machine, and runs it.
- It also does memory management for you.

Why this indirection?

- I would say this is the one of the greatest ideas to make programming a lot easier and cleaner.
- If you have ever done any C or C++ programming, you would see why.
- Memory management is the most annoying thing on the planet.
- You really need to carefully consider the architecture and the operating systems that your application is expected to run on.
 - Why do many of C/C++ libraries have their own types?

Why this indirection? (2)

- With a virtual machine, you only need to be compatible with one architecture (the one of the virtual machine).
- Memory management is handled by the virtual machine.
- Thus, the burdens of considering many architectures, managing memory and other things are not yours anymore, but the virtual machine coders'.
- Different virtual machines for different operating systems/architectures.

But still remember

- All this is nice and promising, but not 100% realistic.
- To be a good programmer, it is always a good idea to learn about different architectures, where your application is expected to run.
 - It is probably not the greatest idea to create the exact same application to run on a Desktop and a mobile phone.
 - You may want to make use of a certain architecture's features, the virtual machine can make this somehow harder.
- Always use the right tool for the right purpose!

Executing the bytecode

- The Java virtual machine is stack-based.
- `java Test.java`
- To get some interesting information:
 - `java -verbose Test`
 - `java -Xprof Test`
- To get even more interesting information:
 - JVMTI

Questions?