


# Intro to Java

not intro to computer science



# Outline of this lecture

- Introduction to me
  - About the course / Resources for course
  - Intro to Intro to Java
  - Hortsman Chapter 1 / 2
  - About homework
  - Homework
- 

# ME!

- At heart, I'm a software engineer
  - which fits nicely with this course; it's not theoretical it's really an engineering course
- Studied at University of Chicago
- Been coding Java professionally for 11 years
  - worked at Union Pacific Railroad / McGraw Hill / HBO
- CTO / Cofounder of Dash - <https://dash.by>



# About CS9053 - Spring '16

- Not intro to computer science / programming
- Introduction to Java - practical & pragmatic

## *To Be Successful...*

- Write your **own** code
- Attend lectures
- Program the concepts / don't just know them

# Resources for CS9053 - Spring '16

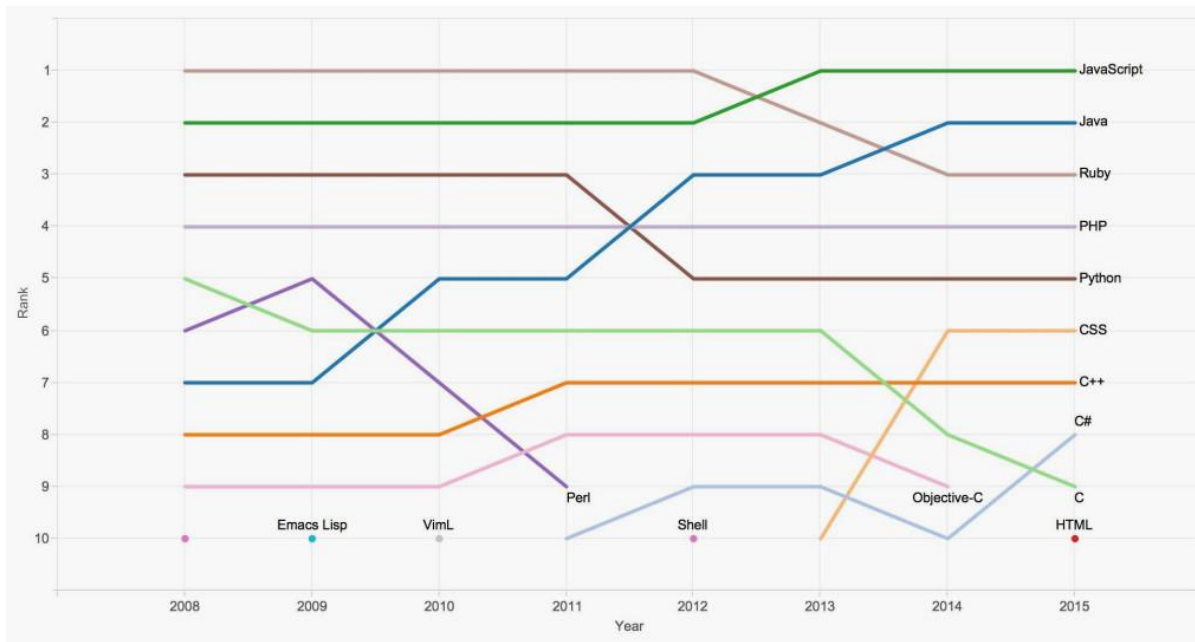
Github! <https://github.com/NYU-CS9053/Spring-2016>

- must become a member to access
  - supplemental material
  - all lectures posted after presentation
  - all homeworks posted/submitted

# WAT?!? You're using Java?!?

Why not Python/Ruby/Javascript/C++/...

- Java's virtual machine (JVM) is **fast**
- Java is ubiquitous (libraries/JSR/support)
- Java as a language/ecosystem is mature
- Java plays nicely with others



# WAT?!? You're using Ruby/Javascript

```
failbow1:~(master!) $ jsc  
> [] + []
```

Every language has its thorns...



# Java WAT

```
System.out.println((1.21 - 1.11) == 0.1);
```

<https://gist.github.com/blangel/da2921fa73f23824ff7e>



# Java WAT

```
long microsecondsInDay = 24 * 60 * 60 * 1000 * 1000;  
long millisecondsInDay = 24 * 60 * 60 * 1000;  
long millisecondsInSecond = (microsecondsInDay / millisecondsInDay);  
long thirteenSecondsInMs = (millisecondsInSecond * 13);  
System.out.println(thirteenSecondsInMs);
```

<https://gist.github.com/blangel/60de4bc1fcc349ccfc0d>

variant of Puzzle 3 in **Java Puzzlers** by Joshua Bloch & Neal Gafter



**65 WAT**

# Java WAT

```
char x = 'x';  
int i = 0;  
System.out.print(true ? x : 0);  
System.out.print(false ? i : x);
```

<https://gist.github.com/blangel/9f1cc619adb253de628b>

variant of Puzzle 8 in **Java Puzzlers** by Joshua Bloch & Neal Gafter



# WATs in all langs - why learn Java?

- It's ubiquitous - for good reasons
  - learn the language and decide for yourself
- Don't feel intimidated by Java
  - there are many misconceptions
- You may have to...
  - even if not directly programming with it, many libraries are built with it. And many other languages will be run on the JVM (even non-jvm based languages, like Ruby). Good to know about the level of abstraction beneath you.

# Java Terminology

**JVM** - Java virtual machine

**JRE** - Java runtime environment (includes JVM and supporting libraries)

**javac** - java compiler (takes .java and produces .class)

**java** - command to run the JVM

**JDK** - Java development kit (JRE, javac and supporting tools)

**bytecode** - compiled from source by the javac and interpreted by the JVM

**jar file** - a zip file containing .class files (and some metadata)

**classpath** - directories the JVM should search for .class / jar files



# Java Terminology (cont)

**JIT** - just-in-time compiling; turning bytecode into machine code

**JNI** - Java native interface - allows native code and Java to interoperate

**javadoc** - parses source code comments and generates documentation

**J2EE** - Java 2 Enterprise Edition - set of libraries to assist in creating client/server architectures (EJBs/etc) Avoid at all costs

**J2SE** - Java 2 Standard Edition - a.k.a. JDK. Just say JDK

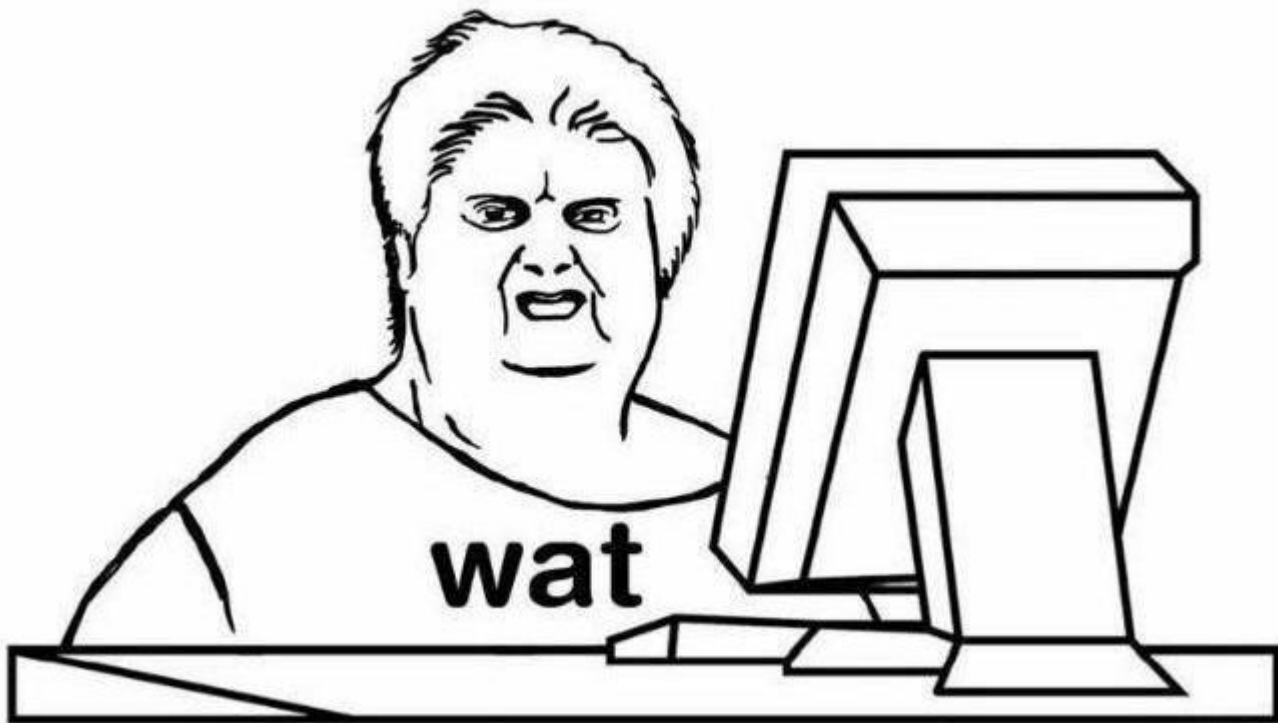
**JCP** - Java community process ([jcp.org](http://jcp.org)) - place to add new features into Java

**JSR** - Java specification request - request to JCP to add a new feature

# IDE - Integrated Development Environment

- Do not use (yet) in this class
  - Like giving someone a power drill who's never used a screwdriver
- Extremely useful in increasing productivity
  - Can even aid in learning provided you have the basics (so not just yet)
- Increased productivity:
  - Small feedback loop
  - Realtime compiling / static code analysis
  - Automatic imports / code styling / etc

# No IDE?!?



# No IDE -> what then?

You'll need an editor.

- Emacs
- Vi/Vim
- Sublime

*These are not (programming) editors*

- Notepad
- TextEdit

# No IDE -> what then? (cont)

javac - to compile. Learn its arguments, you will be tested on them.

```
blangel@lenoir$ javac
Usage: javac <options> <source files>
where possible options include:
-g                        Generate all debugging info
-g:none                  Generate no debugging info
-g:{lines,vars,source}  Generate only some debugging info
-nowarn                  Generate no warnings
-verbose                 Output messages about what the compiler is doing
-deprecation             Output source locations where deprecated APIs are used
-classpath <path>        Specify where to find user class files and annotation processors
-cp <path>               Specify where to find user class files and annotation processors
-sourcepath <path>        Specify where to find input source files
-bootclasspath <path>     Override location of bootstrap class files
-extdirs <dirs>           Override location of installed extensions
-endorseddirs <dirs>      Override location of endorsed standards path
-processor:{none,only}    Control whether annotation processing and/or compilation is done.
-processor <class1>[,<class2>,<class3>...] Names of the annotation processors to run; bypasses default discovery process
-processorpath <path>     Specify where to find annotation processors
-d <directory>            Specify where to place generated class files
-s <directory>            Specify where to place generated source files
-implicit:{none,class}   Specify whether or not to generate class files for implicitly referenced files
-encoding <encoding>      Specify character encoding used by source files
-source <release>         Provide source compatibility with specified release
-target <release>         Generate class files for specific VM version
-version                 Version information
-help                    Print a synopsis of standard options
-Akey[=value]            Options to pass to annotation processors
-X                        Print a synopsis of nonstandard options
-J<flag>                 Pass <flag> directly to the runtime system
```

# No IDE -> what then? (cont)

**java** - to run. Learn its arguments, you will be tested on them (at least classpath, system arguments and program arguments).

```
blangel@lenoir$ java
Usage: java [-options] class [args...]
           (to execute a class)
 or   java [-options] -jar jarfile [args...]
           (to execute a jar file)
where options include:
    -d32          use a 32-bit data model if available
    -d64          use a 64-bit data model if available
    -server       to select the "server" VM
                  The default VM is server,
                  because you are running on a server-class machine.
```

```
-cp <class search path of directories and zip/jar files>
-classpath <class search path of directories and zip/jar files>
           A : separated list of directories, JAR archives,
           and ZIP archives to search for class files.
```

```
-D<name>=<value>
           set a system property
```

```
-verbose:[class|gc|jni]
           enable verbose output
```

```
-version       print product version and exit
```

```
-version:<value>
           require the specified version to run
```

```
-showversion  print product version and continue
```

# Enough Slides - Let's Code!



# Homework for CS9053

- Fork <https://github.com/NYU-CS9053/Spring-2016>
- Commit locally (important to do before datetime due)
- Push (after datetime due)



# Read Chapter 3

All sections will be covered in next lecture

# Homework - Week 1

<https://github.com/NYU-CS9053/Spring-2016/homework/week1>