# Java and C (part II)

**CSE 351 Spring 2020**

**Instructor:** Ruth Anderson  
**Teaching Assistants:**  
Alex Olshanskyy, Connie Wang, Eddy (Tianyi) Zhou, Jonathan Chen, Millicent Li  
Callum Walker, Diya Joy, Eric Fan, Joseph Schafer, Porter Jones  
Chin Yeoh, Edan Sneh, Jeffery Tian, Melissa Birchfield, Rehaan Bhimani
Administrivia

- Lab 5 (on Mem Alloc) – **NOW due Mon (6/08)**
- Unit Summary #4 – due Wed (6/10)
- hw23 on Java and C – **NOW due Thurs (6/11)**
- Course evaluations now open
  - Please fill these out!
  - Separate ones for Lecture and Section
- **You must log on with your @uw google account to access!!**
  - Google doc for 11:30 Lecture: [https://tinyurl.com/351-06-03A](https://tinyurl.com/351-06-03A)
  - Google doc for 2:30 Lecture: [https://tinyurl.com/351-06-03B](https://tinyurl.com/351-06-03B)
Polling Question [Java II]

What would you expect to be the order of contents in an instance of the Car class?

Vote at [http://pollev.com/rea](http://pollev.com/rea)

A. header, Vehicle vtable ptr, passengers, Car vtable ptr, wheels

B. Vehicle vtable ptr, passengers, wheels

C. header, Vehicle vtable ptr, Car vtable ptr, passengers, wheels

D. [header, Car vtable ptr, passengers, wheels]

E. We’re lost...
Roadmap

C:
```c
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:
```java
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg =
    c.getMPG();
```

Assembly language:
```
get_mpg:
pushq %rbp
movq %rsp, %rbp
...
popq %rbp
ret
```

Machine code:
```
0111010000011000
1000110100000100000000101000100111000010110000011111101000011111
```

Computer system:

Memory & data
Integers & floats
x86 assembly
Procedures & stacks
Executables
Arrays & structs
Memory & caches
Processes
Virtual memory
Memory allocation

Java vs. C
Implementing Programming Languages

- Many choices in how to implement programming models
- We’ve talked about compilation, can also interpret
- Interpreting languages has a long history
  - Lisp, an early programming language, was interpreted
- Interpreters are still in common use:
  - Python, Javascript, Ruby, Matlab, PHP, Perl, …
An Interpreter is a Program

- Execute (something close to) the *source code* directly
- Simpler/no compiler – less translation
- More transparent to debug – less translation
- Easier to run on different architectures – runs in a simulated environment that exists only inside the *interpreter* process
  - Just port the interpreter (program), not the program-being-interpreted
- Slower and harder to optimize

```
Your source code

Interpreter binary

Interpreter implementation

Python
```
Interpreter vs. Compiler

- An aspect of a language implementation
  - A language can have multiple implementations
  - Some might be compilers and other interpreters

- “Compiled languages” vs. “Interpreted languages” a misuse of terminology
  - But very common to hear this
  - And has *some* validation in the real world (e.g. JavaScript vs. C)

- Also, as about to see, modern language implementations are often a mix of the two. E.g.:
  - Compiling to a bytecode language, then interpreting
  - Doing just-in-time compilation of parts to assembly for performance
“The JVM”

Java programs are usually run by a

Java virtual machine (JVM)

- JVMs interpret an intermediate language called Java bytecode
- Many JVMs compile bytecode to native machine code
  - Just-in-time (JIT) compilation
- Java is sometimes compiled ahead of time (AOT) like C

Note: The JVM is different than the CSE VM running on VMware. Yet another use of the word “virtual”!
Compiling and Running Java

1. Save your Java code in a .java file

2. To run the Java compiler:
   - **javac** Foo.java
   - The Java compiler converts Java into *Java bytecodes*
     - Stored in a .class file

3. To execute the program stored in the bytecodes, Java bytecodes can be interpreted by a program (an interpreter)
   - For Java, this interpreter is called the Java Virtual Machine (the JVM)
   - To run the virtual machine:
     - **java** Foo
     - This Loads the contents of Foo.class and interprets the bytecodes
Virtual Machine Model

High-Level Language Program (e.g. Java, C)

Bytecode compiler (e.g. javac Foo.java)

Virtual Machine Language (e.g. Java bytecodes)

Virtual machine (interpreter) (e.g. java Foo)

Native Machine Language (e.g. x86, ARM, MIPS)

Compile time

Run time

Ahead-of-time compiler
gcc

e.g. C

JIT compiler
Java Bytecode

- Like assembly code for JVM, but works on all JVMs
  - Hardware-independent!
- Typed (unlike x86 assembly)
- Strong JVM protections
JVM Operand Stack

\[ c = a + b \]

- \( i \) = integer
- \( a \) = reference
- \( b \) = for byte
- \( c \) = for char
- \( d \) = for double

Bytecode:

- `iload 1` // push 1st argument from table onto stack
- `iload 2` // push 2nd argument from table onto stack
- `iadd` // pop top 2 elements from stack, add together, and push result back onto stack
- `istore 3` // pop result and put it into third slot in table

No registers or stack locations! All operations use operand stack

Compiled to (IA32) x86:

- `mov 8(%ebp), %eax`
- `mov 12(%ebp), %edx`
- `add %edx, %eax`
- `mov %eax, -8(%ebp)`
A Simple Java Method

Method java.lang.String getEmployeeName()

0 \text{aload 0} \quad \text{// "this" object is stored at 0 in the var table}

1 \text{getfield} \#5 <Field java.lang.String name>
\quad \text{// getfield instruction has a 3-byte encoding}
\quad \text{// Pop an element from top of stack, retrieve its}
\quad \text{// specified instance field and push it onto stack}
\quad \text{// "name" field is the fifth field of the object}

4 \text{areturn} \quad \text{// Returns object at top of stack}

Byte number: 0 \quad 1 \quad 4
\begin{array}{cccc}
\text{aload_0} & \text{getfield} & 00 & 05 \\
\end{array} \quad \text{areturn}

As stored in the .class file: 2AB4 00 05 B0

\text{http://en.wikipedia.org/wiki/Java_bytecode_instruction_listings}
Class File Format

- Every class in Java source code is compiled to its own class file
- 10 sections in the Java class file structure:
  - **Magic number**: 0xCAFEBABE (legible hex from James Gosling – Java’s inventor)
  - **Version of class file format**: The minor and major versions of the class file
  - **Constant pool**: Set of constant values for the class
  - **Access flags**: For example whether the class is abstract, static, final, etc.
  - **This class**: The name of the current class
  - **Super class**: The name of the super class
  - **Interfaces**: Any interfaces in the class
  - **Fields**: Any fields in the class
  - **Methods**: Any methods in the class
  - **Attributes**: Any attributes of the class (for example, name of source file, etc.)
- A .jar file collects together all of the class files needed for the program, plus any additional resources (e.g. images)
Disassembled Java Bytecode

Compiled from Employee.java

```java
class Employee extends java.lang.Object {
    public Employee(java.lang.String,int);
    public java.lang.String getEmployeeName();
    public int getEmployeeNumber();
}
```

Method Employee(java.lang.String,int)
0  aload_0
1  invokespecial #3 <Method java.lang.Object>()
4  aload_0
5  aload_1
6  putfield #5 <Field java.lang.String name>
9  aload_0
10  iload_2
11  putfield #4 <Field int idNumber>
14  aload_0
15  aload_1
16  iload_2
17  invokespecial #6 <Method void storeData(java.lang.String, int)>
20  return

Method java.lang.String getEmployeeName()
0  aload_0
1  getfield #5 <Field java.lang.String name>
4  areturn

Method int getEmployeeNumber()
0  aload_0
1  getfield #4 <Field int idNumber>
4  ireturn

Method void storeData(java.lang.String, int)
...

Other languages for JVMs

- JVMs run on so many computers that compilers have been built to translate many other languages to Java bytecode:
  - **AspectJ**, an aspect-oriented extension of Java
  - **ColdFusion**, a scripting language compiled to Java
  - **Clojure**, a functional Lisp dialect
  - **Groovy**, a scripting language
  - **JavaFX Script**, a scripting language for web apps
  - **JRuby**, an implementation of Ruby
  - **Jython**, an implementation of Python
  - **Rhino**, an implementation of JavaScript
  - **Scala**, an object-oriented and functional programming language
  - And many others, even including C!

- Originally, JVMs were designed and built for Java (still the major use) but JVMs are also viewed as a safe, GC’ed platform
Microsoft’s C# and .NET Framework

- C# has similar motivations as Java
  - Virtual machine is called the **Common Language Runtime**
  - **Common Intermediate Language** is the bytecode for C# and other languages in the .NET framework
We made it! 😊😎😂

C:

```c
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```java
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg = c.getMPG();
```

Assembly language:

```assembly
get_mpg:
    pushq  %rbp
    movq   %rsp, %rbp
    ...
    popq   %rbp
    ret
```

Machine code:

```
0111010000011000 1000110100000100000000101000100111000010110000011111101000011111
```

Computer system:

Windows 10

OS:

Memory & data
Integers & floats
x86 assembly

Procedures & stacks
Executables

Arrays & structs
Memory & caches
Processes

Virtual memory
Memory allocation
Java vs. C