

Notes

Look together at Piazza. LMK if you didn't get an invite—that means I don't have your email in my database

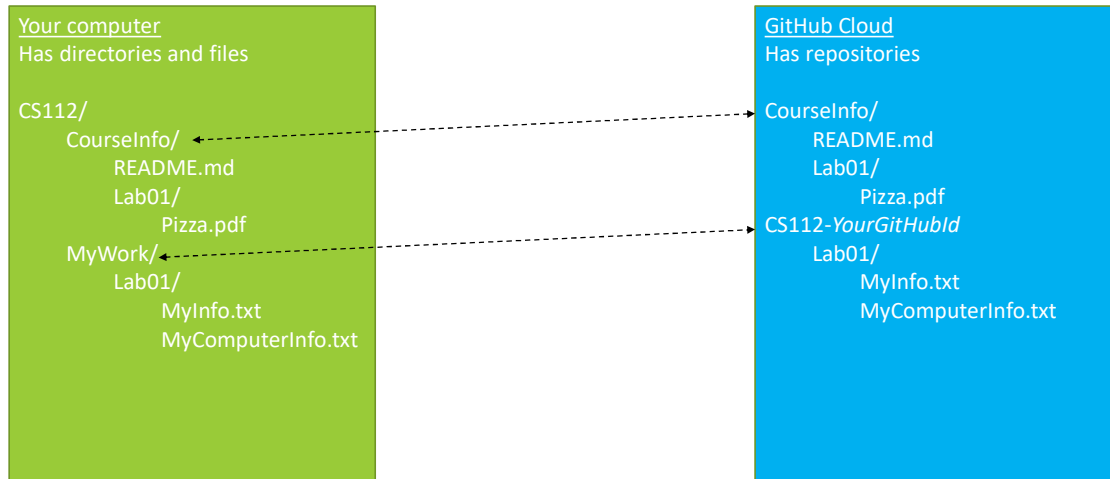
Look together at CourseInfo README, Panopto video player

A few people have not sent me their GitHub IDs yet. Please do so ASAP and LMK if you need help

"Flipped classroom?"

- Let's try "flipping" next lecture
- I will post video by Thursday morning...to GitHub CourseInfo/README
- Watch the video before class! Short quiz.

GitHub and git review



Your computer and GitHub do not stay in sync automatically.

You use "git" commands to "pull" files from GitHub to your computer.

You use "git add", "git commit", and "git push" to "push" files from your computer to the GitHub cloud.

REPOSITORIES contain current and past versions of files, along with commit comments.

Details matter to computers!

If I ask for a directory to be called "Lab01"...

- lab01 is not the same
- Lab1 is not the same

Why does it matter?

- A. Because Paul is annoying
- B. Because Paul's automated grading software only knows to look for "Lab01"
- C. Because if your SW must work with other people's software (or with existing libraries), it is important that your SW **match exactly** with the **specification** of the other software or libraries
- D. All of the above

Also, your repo must be named 'CS112-YourGitId', not just 'YourGitId'. See me to fix.

CS112 –
Java
Programming

Spring 2024

Copyright 2023 Paul Haskell. All rights reserved.

Java Beginnings

A brief history of computer languages

Assembly language = machine language but with names (ADD, JUMP) instead of numbers

C (1972) eliminates biggest annoyances of Assembly (function calling, +-* / operators, automatically manage sizes of variables, etc)

C++ (1985) eliminates some of C's biggest problems (name "collisions" in big projects, different permissions for SW designers vs users, object oriented)

Java (1996) eliminates some of C++'s biggest challenges (memory management, platform dependence) but introduces a few new ones

Python (~2000) eliminates some of Java's difficulties but has its own limitations

How many computer languages are there? ([LINK](#))

Java may be the most popular today

From last time: Machine Language = numbers in memory that the CPU interprets as ADD, MULT, LOAD, etc instructions

of languages estimated to be between 300 and 9000

At my previous work, C++ and Java mostly. Some C#

How computer languages work

“Interpreter” or “Compiler” software translates human-friendly “source code” into “machine code”. Python can be interpreted or compiled

```
% python
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [AMD64] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("I am interpreted")
I am interpreted
>>>
```

```
% cat hello.py
## hello.py – a basic Python file
Message = "I am compiled"
print(Message)

% python hello.py
I am compiled

%
```

EXPLAIN DIFFERENCE: Interpreter = LINE BY LINE execution. Compiler = convert ENTIRE FILE OF SOFTWARE into machine language file, which can be run directly.

Java is compiled...

Source code files have suffix “.java”

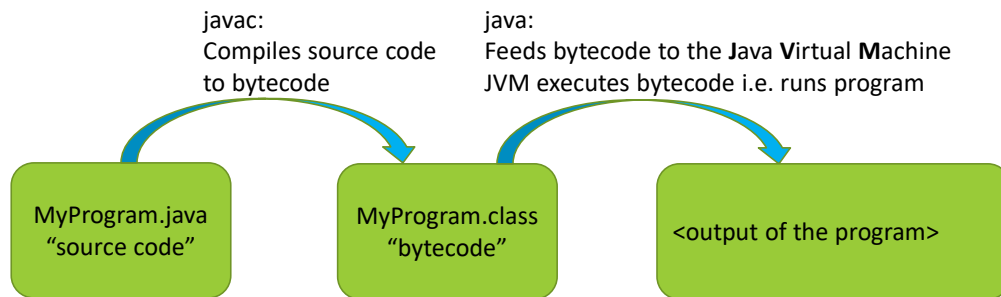
Compiler is called “javac”

But Java does not compile to machine code! It compiles to a format called “bytecode” that must be executed by another program called “java”.

```
% javac MyProgram.java  
  
% java MyProgram  
I ran my first Java program!  
  
%
```

Show example in command window!

Java is compiled and then interpreted



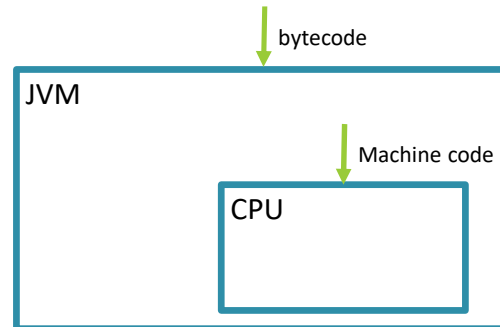
What if we just run `%java MyProgram.java` ?

Annoying—sort of works, but has bugs. Please don't do it! Wrong answers in today's homework!

What's a Virtual Machine?

The Java Virtual Machine looks like a CPU that has bytecode as its machine language

- JVM is actually software—but "acts like" a CPU



Why the JVM? *Device Independence*

Java designers wanted every Java program to run the same way on any type of computer.

For any bytecode program, the JVM, customized to each of dozens of computer types, gives same output results and same “look”.

Bytecode files—generated by `javac`—have filename extension “.class”

“.class” file are fed to the JVM via the “java” program.

When you run a program, you omit the “.class” filename extension:

- `javac UpdateInventory.java`
- `java UpdateInventory`

Java Syntax – Quick Review

1. All code must be inside a class.

- `class <className>` is the syntax to define a new class, e.g.

```
class Automobile {  
    ...  
}
```

2. A class contains *methods* and *variables*

And the definition of the class is inside a matching pair of curly braces
- Parentheses, brackets, curly braces

Java Syntax – Quick Review

3. A method has a *return-type*, *name*, *input arguments*, and a *body of code*

```
class Automobile {  
    double calculateValue(int age, long mileage) { ... }  
}
```

Java Syntax – Quick Review

Every program must have a method called `main()`

```
class FirstProgram {  
    public static void main(String[] args) { int variable = 1; }  
}
```

- When “java” runs any program, it runs the `main()` function first.
- `public/private/protected`
- `static...`
- `void...`

Talk about static later...

Java Variables

Every variable in Java has a:

- type
- name
- value

As we use a variable, its value can change but not its name or type.

Examples:

```
int numberOfSodasInACase = 24; // semicolon at end of statement
numberOfSodasInACase = 32;
float numberOfSodasInACase = 32.5;
float NumberOfSodasInACase = 32.5;
```

Which line is no good? "float numberOfSodasInACase...". That variable already exists (and has type int)

What is difference from Python? In Python we can change the type of a variable

Java Variable Names

A legal variable name in Java:

- May include letters, numerical digits, the underscore '_', and/or the '\$' symbol
- Cannot start with a numerical digit
- May not be one of Java's built-in "reserved" keywords e.g. "class", "static", "public", "int"
- Letters are case-sensitive

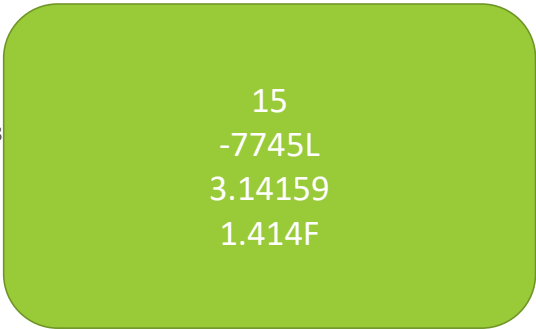
Common style:

- Start with a letter
- Never use the '\$' character
- Use "camel case" for variable names e.g. `allWordsAfterFirstAreCapitalized`
- Use all-caps for constants e.g. `double POUNDS_PER_KILO = 2.2046;`

Java Variable Types

Java has 8 built-in basic types:

- boolean – only values are `true` and `false`
- byte – an 8-bit signed integer
- short – a 16-bit signed integer
- int – a 32-bit signed integer
- long – a 64-bit signed integer
- float – a 32-bit floating point number
- double – a 64-bit floating point number
- char – a text character...



15
-7745L
3.14159
1.414F

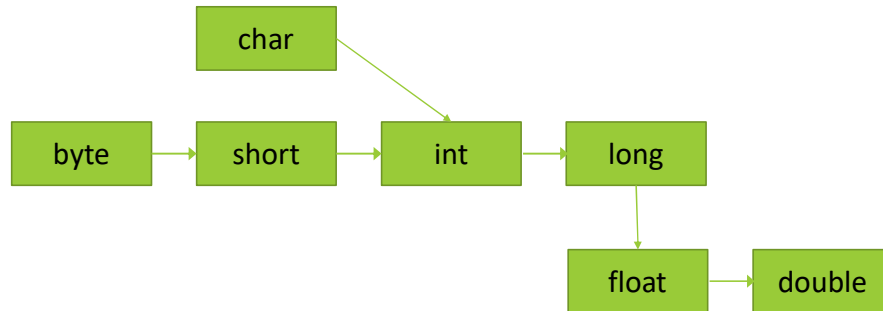
Why would anyone ever use anything other than the biggest data type? Speed: of networking, of processing

- Paul's son and his 30 GB data files: too slow to transfer
- Paul's work and processing 32 bytes in a single instruction: faster to process smaller data types
- What are types of constants in the window?

Conversions between different types

Java allows **automatic** conversion from "smaller" data types to "larger" e.g.

```
short theShort = -28000;  
float theFloat = theShort;
```



Any "multi-hop" conversions are automatic also, e.g. byte to double

Conversions between different types

When converting from "larger" to "smaller" there is a risk of losing data, e.g:

```
float x = 2.5F;  
int y = x; // lose the ".5" part
```

Java requires you to use a "cast" to make large-to-small conversions:
put the name of the desired type, in parentheses, before the value you want to convert,
e.g:

```
float x = 2.5F;  
int y = (int) x; // "(int)" is the cast
```

Conversions between different types

Automatic conversions also will happen if an expression mixes different data types:

```
float x = 0.0, y = 2.0;  
int z = 1;  
x = z/y; // what is value
```

How about in this case?

```
int a = 1, b = 2;  
float c = a/b; // what is value?  
  
float d = (float) a / b; // what is value?
```

Do this in Eclipse!!!

X = 0.5

C = 0: integer division happens first, get result of 0, assign 0 to 'c'

D = 0.5: cast before division, before assignment

Terminology...

Variable **declaration** – declare it exists

```
char myMiddleInitial;
```

Variable **initialization** – assign its initial value

```
myMiddleInitial = '\u039b'; // it might be, if I were Greek
```

Variable **assignment** – assign a new value

```
myMiddleInitial = '\u039e'; // if I were ancient Roman
```

We may initialize at the same time as we declare, but it is not required. We must initialize before we use a variable's value, e.g. to print it or assign it to another variable.

WHY MIGHT WE like TYPED VARIABLES, UNLIKE IN PYTHON?

- Compiler catches errors. We don't have to find them thru testing

"L-values" and "R-values"

```
short v1 = 10;  
short v2 = v1;  
v1 = 20;  
System.out.println(v2);
```

What gets printed?

How about

```
20 = v1;
```

Use a variable on the left? It is the "reference to the variable"

On the right? It is the value of the variable

DRAW DIAGRAMS! THIS IS IMPORTANT!

Lab 02

Let's do some Java

Lab02

Goals:

- Install and set up software that you will use to:
 - Write Java programs: text editor
 - Compile and execute Java programs: **java** and **javac**
- Start to become familiar with these programs.
- Edit, compile, run, and submit several Java programs.

Later we will use graphical program to edit compile, run and debug Java programs.
For now, use javac and java
Watch out for bug just running "java Program.java"!!

Java Installation

There are several Java "packages" available. Most common are:

- **Java Runtime Environment:** for Java *users*. Includes "java" program, JVM, other files such as font files
- **Java Development Kit:** for Java *developers*. Includes JRE, plus "javac", documentation tools, debugging tools, etc. We want this one

Use "git pull" in the CourseInfo directory. Look for the **Lab02.pdf** file, read it, and get started!