CS61B Lecture 1

Wednesday, January 22nd, 2020

Announcements (oops wrong thread)

- Sign up for a lab and discussion section using the SignUpGenius poll, available from the course website. If you can't find a slot, attend any section you can (although you have second priority for seating).
- Labs start today. In (or preferably before) lab this week, get a CS61B Unix account from https://inst.eecs.berkeley.edu/webacct.
- Because labs will be crowded, you might want to bring your laptop.
- If you plan to work from home, try logging in remotely to one of the instructional servers (`...@X.cs.berkeley.edu`, where X is `ashby`, `derby`, `cedar`, `gilman`, `oxford`, or `solano`).
- We'll be using Piazza for notices, on-line discussions, questions.
- General information about the course is on the home page (grading, lateness, cheating policy, etc.).
- Concurrent enrollment students will be processed later.
- Lectures will be screencast.
- Lectures will be in 150 Wheeler from next Monday.
- There are two required texts for this course, available online. A textbook is also available, Head First Java, available for the first part of the class.

Course Organization

- Labs are important: exercise of programming principles as well as practical dirty details go there. Generally we will give you homework points for doing them.
- Homework is important, but really not graded: use it as you see fit and turn it in! You get points for just putting some reasonable effort into it.
- Individual projects are really important! Expect to learn a lot. Projects are not team efforts (that's for later courses).
- Use of tools is part of the course. Programming takes place in a programming environment:
  - Handles editing, debugging, compilation, archiving versions.
  - Professor Hilfinger's setup: Emacs + gjdb + make + git, (documented in one of the readers and on-line). But we'll look at IntelliJ in lab.
  - Tests are challenging: better to stay on top than to cram.
- Tests, 40%; Projects, 50%; HW, 10%

Programming, not Java

Here, we learn **programming**, not Java. The reason for this is programming principles span across many languages. We look for connections between programming languages (such as Python in 61A and Java here), and not particularly caring about Syntax (e.g. `(+ x y)` and `x + y` is different cosmetically, but fundamentally the same).

Our first Java program

Here is some code that prints "Hello world"! In Python, we write:

```
print("Hello, world")
```

In Java, we write:

```
/** Traditional first program.
 *  @author P. N. Hilfinger */

public class Hello {
    /** Print greeting.
    ARGS is ignored. */

    public static void main(String[] args){
        System.out.println("Hello, world!");
    }
}
```

So what's all this public class void stuff? Well, these are things that were abstracted away in Python, so you didn't have to mess with them. Java lets you modify some details of how your code fundamentally works.

This also gets into a critical difference between Python and Java. Python is **dynamically typed**, which means that the information of what kind of data is carried in the variable, is in the data itself (for example, Python knows 2 is an integer unless it's in quotes).

Java is **statically typed**, which means the program itself must contain information about what's in its variables, which results in some of the extra annotation you see here.

Comments

Java comments can either start with `//` and go to the end of the line (like `#` in Python), or they can extend over any number of lines, bracketed by `/*` and `*/` (like `"""` in Python).

Professor Hilfinger does not use `//` comments except for things meant to be written over, and the style checker will flag any instances of them in your code.

The multiline comments, typically using `/**`, are called **documentation comments** or **doc comments**. They are just comments, but many tools interpret them as providing documentation for the things that follow them. The style checker will look for them in your code.

Classes

```
public class Hello {
    /** Print greeting.
    ARGS is ignored. */
```

The next piece that we encounter is the `class` declaration. We learnt about classes in 61A and Python. In Java, every piece of code you write is in some class, as the creators decided it should be uniform. In this case, we chose a natural name, `Hello`, which corresponds to its filename.

The curly braces enclose the body of the program. Unlike Python, indentation is not important to whether your program works or not (but you should probably still use them to make them look good).

All classes, in turn, belong to some package. The `Hello` class belongs to some package, in this case, the **anonymous package**. The projects you get in this class will be enclosed in a package.

Methods (Functions)

```
    public static void main(String[] args){
```

Functions in Java contain more information than those in Python. They specify the types of values taken in as parameters, and the types of values returned by the function.

The type `void` has no possible values, which means this function returns nothing. Meanwhile, the brackets enclose what is taken in, and in this case it takes in a `(String[])`. The `String` is the same thing as Python strings, while `[]` means "array of". An array is very similar to a Python list, except for one key difference: their size is fixed once created.

Functions named “main” and defined like the example about are special: they are what get called when one runs a Java program (in Python, the main function is essentially anonymous).

Selection

```
        System.out.println("Hello, world!");
```

Much like Python, `X.Y` tells the system to look for the variable `Y` that is in, or applies to the thing computed by `X`. `System.out` tells the computer to find a variable named `out` in the class named `System`, and likewise, `System.out.println` is a method `println` that applies to the object referenced by the value of variable `System.out`.

Access

And finally, the final part of the code has to deal with the `public` we saw earlier. Every declared entity in Java has **access permissions** indicating what pieces of code may mention it.

`public` classes, methods, and variables may be referred to anywhere else in the program. We sometimes refer to them as exported from their class (for methods or variables) or package (for classes).

`static` methods are just like Python functions outside of any class, or a function in a class annotated `@staticmethod`. A static variable is like a Python variable defined outside of any class or a variable selected from a class, as opposed to from a class instance.

Other variables are local variables (in functions) or instance variables (in classes), and these are as in Python.