Java for CS106A Programmers

Those of you who are coming to CS108 without any Java Programming past what you learned in CS106A have a few things you'll need to be aware of. Java in CS106A was based on the ACM teaching libraries rather than the professional programming libraries we'll be using for CS108. In this handout we go over two important differences between writing code with the teaching libraries and writing professional Java code.

Main vs. Run

In CS106A, each program contained a main class which was subclassed from either the ACM Library's `ConsoleProgram` class or `GraphicsProgram` class. Your primary class included a method called `run` which was executed using the special CS106A version of Eclipse. This `run` method is added by the ACM teaching libraries and is not the standard way to execute a program in Java.

Professional Java uses a method similar to that used by the C++ and C languages from CS106B and CS107. As you'll recall these languages have a function called `main`. When you start up your C or C++ program the code in the `main` function is executed. Here's the canonical Hello World example in C:

```
#include <stdio.h>

int main() {
   printf("hello world!\n");
}
```

In C or C++ when we tell a program to run the computer looks for the function named `main` and executes it. However, Java does not have functions, it only has methods. [1] The closest it comes to having a function is a class method. Therefore that's exactly what Java uses to start a program.

Here's what our Hello World program looks like in Java:

---

[1] Methods are different from functions because methods operate in the context of a specific object. In contrast functions are not associated with specific objects or classes. A *class method* (also known as a static method) is a special method which is associated with a class, but not an object based on that class. Because it is not associated with an actual object, it works very similarly to a function.

```
class HelloWorld {

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

A few things to notice:

- This is not a self contained function, functions don't exist in Java, so this code must be a method contained within a class.
- The method is defined as `static`, which means that it is a class method, no actual `HelloWorld` instance will need to be created in order to execute the method.
- The method has an `args` parameter. This may be used to provide input information if the program is executed from a command prompt, rather than using a Graphical User Interface (GUI). We'll take a look at it much later in the quarter.

When creating a `public static void main` in your own program make sure that your signature matches our example above exactly, including the keywords `public`, `static`, and `void` and the `args` parameter listed (even if you don't plan to use the parameter).

You can place this `public static void main` in any of the classes in your program (in fact you can place a different `public static void main` method in all of your classes if desired). When actually executing the program, you'll select one of the classes which has a `public static void main` and use that to execute the program.

To execute a Java program in Eclipse, right-click on any class which has a `public static void main` method in it, then choose Run As > Java Application or Debug As > Java Application depending on whether or not you want to use the debugger.

Printing

Using the ACM teaching libraries you could call the method print and as long as you called it in a class based on either the `ConsoleProgram` class or the `GraphicsProgram` class your data was printed to the console. As you will no longer be using `ConsoleProgram` and `GraphicsProgram` you'll need to use a different method to print.

There are two `PrintStream` objects which are automatically setup for you to print with. You can access these using either:

```
System.out
```

Or

```
System.err
```

Notice that `System` is capitalized as these are static variables (aka class variables) found on the `System` class itself.[2]

For our purposes `System.out` and `System.err` can be used interchangeably. By default both these `PrintStreams` will output to Eclipse's Console which is found on one of the tabs in the bottom Eclipse panel. However, you can also reassign these to new values, for example, if you were writing a program for actual end-users you might want to have the `System.out` print regular output messages, but reassign `System.err` to actually output to a separate error message file that technicians could take a look at if the end-users ran into problems.

These `PrintStream` objects have a print method which works exactly the same as the one you used in CS106A. So you can do something like:

```
System.out.print("Hello World!");
```

They also have much fancier features available. Check the online documentation on `PrintStream` for more information.

---

[2] In Java class names are capitalized, while variable names use lower-case letters. In this case `System` is capitalized which indicates that it a class not a variable. This naming scheme is a convention and is not enforced by the language, but it is followed by virtually all Java programmers, so while you could define a class with a lower-case name or a variable with an upper-case name, you would confuse anyone reading your code.