CSCI 1102 Computer Science 2

Spring 2018

Problem Set 1 : Getting Started

8 Points

Due Tuesday January 23, 2018 Midnight

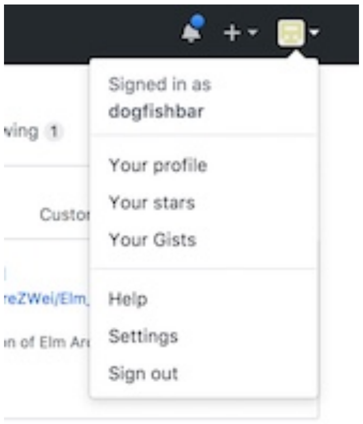
> This first problem set will be delivered to you twice: first as a PDF attachment to an email and then again after you've joined the class GitHub organization BC-CSCI1102 it will reappear as the README.md (markdown) file in your git repository for problem set 1.
>
> You can and should complete parts 0, 1 and 2 of this problem set based on the PDF form of this writeup. To complete part 3, you'll need the git repository. You'll be receiving an email with a link allowing you to clone that repository.
>
> All subsequent problem sets will be delivered to you just once as emails with links for cloning a git repository.

In this course, we're going to develop a lot of software in Java. The purpose of this problem set is to get your system set up and to get your feet wet with Java and with the Unix command line. There are 4 parts. You should complete part 0 by midnight tonight; you should complete parts 1 and 2 by midnight Thursday January 18 and you should complete part 3 by midnight Tuesday January 23.

Part 0: Join

These three tasks should be taken care of by midnight Tuesday January 16.

1. If you haven't already done it, accept the email that you received from Piazza to join the class Piazza forum; if you've lost or didn't receive that email, let us know;
2. If you haven't already done it, take a short survey;
3. Join the course GitHub organization BC-CSCI1102. In order to join this organization, you'll need a GitHub account. Sign up if you don't already have one, it's free. You can make any GitHub ID that you'd like (and that you won't mind sending to prospective employers or admissions committees down the road ...) but for the purposes of this course, you are **required** to set the **Name** field of your GitHub account to your full name as specified in the BC system. E.g., mine is "Robert Muller". Note no middle initial is required. The **Name** field is found from **Settings**



(You won't receive much credit for this first problem set unless you set that correctly.) Once you've joined GitHub, email your GitHub ID to me. You'll subsequently receive an invitation to join the BC-CSCI1102 GitHub organization. Accept that invitation.

Part 1: Development Setup

The development setup has 2 parts: A. installing a Java compiler & runtime system and B. installing the course library software and code editor.

A. Installing the Java Compiler & Runtime System

We're going to be using Oracle's implementation of Java integrated with a library developed by the authors of our Algorithms textbook (the authors Robert Sedgewick & Kevin Wayne, heretofore SW). Some common names and symbols:

- Java SE — Java Standard Edition (produced and maintained by Oracle);
- JVM — Java Virtual Machine;
- JDK — Java Development Kit;
- JRE — Java Runtime Environment; the JVM + class libraries, etc.;
- IDE — Integrated Development Environment (a fancy editor);
- algs4 — the symbol used in naming assets related to the 4th edition of our Algorithms textbook.

Although Oracle has released Java SE 9.0.1, **for this course, we are going to use the previous version: Java SE 8u151. This version of the JDK can be found on the Java SE Downloads page. NOTE: YOU'LL NEED TO SCROLL DOWN TO FIND THE JDK DOWNLOAD BUTTON FOR SE 8u151**.

JDK DOWNLOAD

Downloading the JDK also downloads the JRE so you don't need to download the JRE. Follow the installation instructions and you should be all set. If something goes wrong, contact a course staffer.

B. Installing the algs4 Course Library Software

One of the great things about the Algorithms book is the extensive software library developed by the authors. The library is housed in the Java archive `algs4.jar`. The java compiler (i.e., `javac`) and the JVM (i.e., `java`) both need to know how to find non-standard libraries in order translate and run your code. Java uses a special environment variable called `CLASSPATH` to guide the search for such libraries so this variable needs to be set correctly in order to compile and run our code. The `CLASSPATH` variable also needs to be set properly in the editor/IDE.

The simplest option for setting all of this up is to use the automated installation script developed by SW:

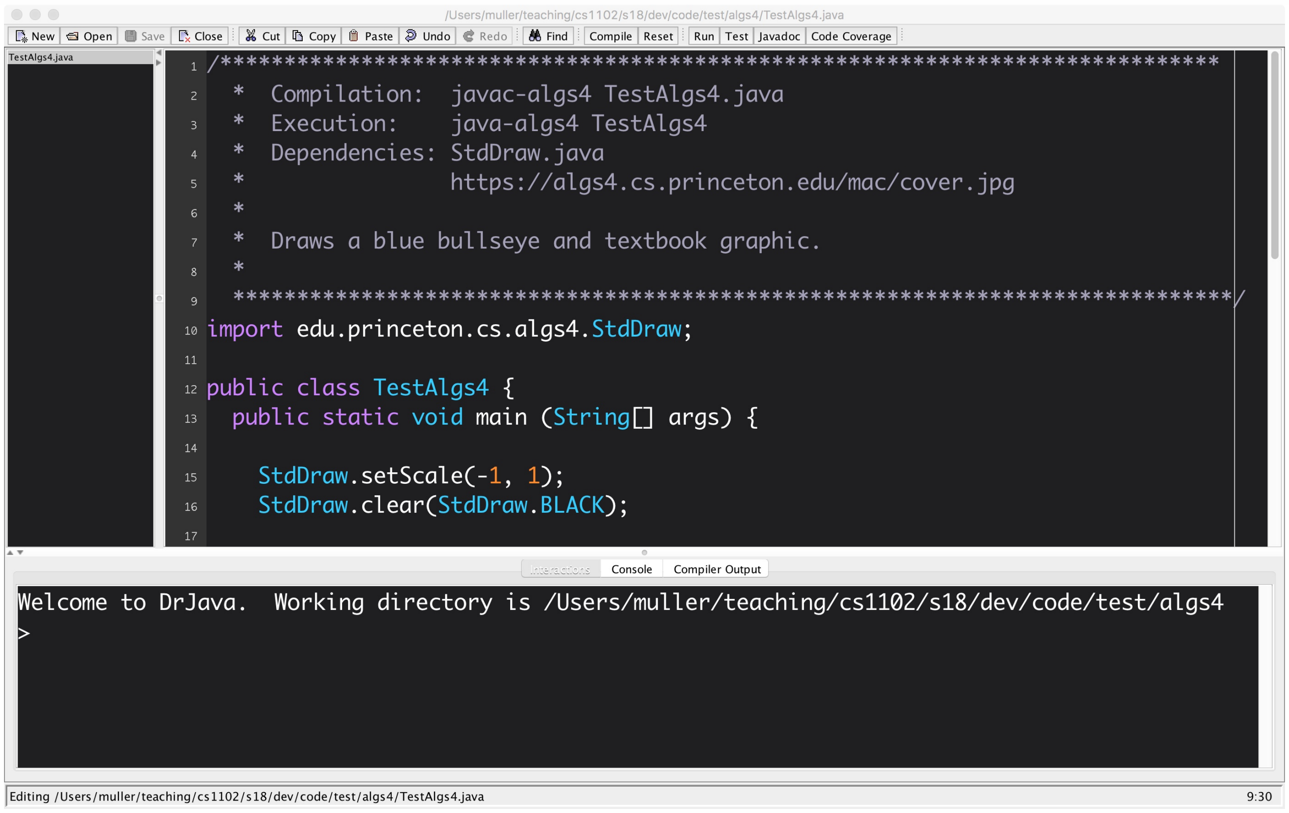
- MacOS
- Windows

Following the instructions on these pages and executing the downloaded installer script will download the `algs4.jar` library for you (along with other tools), it will set your `CLASSPATH` variable appropriately **and** it will install and configure a simple pedagogical IDE for Java called Dr. Java. Dr. Java certainly isn't the only Java IDE out there, there are many others:

- IntelliJ IDEA — industrial-strength/baroque IDE but popular with many BC/CS juniors and seniors;
- Eclipse — likewise;
- vscode — a relatively new state-of-the-art IDE from Microsoft, designed for JavaScript development but adapting to other languages; looks nice!
- atom — from GitHub; great for JavaScript (and Reason/OCaml) but the Java support is still rough around the edges;
- emacs — retro
- vim — very retro.

Feel free to use whatever IDE that you'd like. But 1. you can definitely do everything that needs to be done in this course with SW's basic pain-free Dr. Java setup and 2. if you choose to use another IDE you're off on your very own adventure.

You'll find the Dr. Java icon on your Desktop, fire it up. You should see something like


/Users/muller/teaching/cs1102/s18/dev/code/test/algs4/TestAlgs4.java
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test Javadoc Code Coverage
TestAlgs4.java
Interactions Console Compiler Output
Editing /Users/muller/teaching/cs1102/s18/dev/code/test/algs4/TestAlgs4.java
9:30

```
/******************************************************************************
 *  Compilation:  javac-algs4 TestAlgs4.java
 *  Execution:    java-algs4 TestAlgs4
 *  Dependencies: StdDraw.java
 *                https://algs4.cs.princeton.edu/mac/cover.jpg
 *
 *  Draws a blue bullseye and textbook graphic.
 *
 ******************************************************************************/
import edu.princeton.cs.algs4.StdDraw;

public class TestAlgs4 {
  public static void main (String[] args) {

    StdDraw.setScale(-1, 1);
    StdDraw.clear(StdDraw.BLACK);

```

```
Welcome to DrJava.  Working directory is /Users/muller/teaching/cs1102/s18/dev/code/test/algs4
>
```

This code can be compiled and executed on the JVM by clicking the `Run` button at the top.

Part 2: Unix & git

In this course, we'll be using the Unix command shell to manage files. Among other things, we'll be using the Unix `git` command together with the GitHub website to distribute and collect course materials. The first order of business is to get access to Unix and then learn how to use it.

- **MacOS** If you're using a Mac you're in luck — MacOS is a version of Unix. You'll want to open the Finder, then select Applications -> Utilities (scroll down). Then scroll down to the `Terminal` app. You'll want to drag that icon to your Dock because you'll be using the Terminal app a lot from here on out.

  The algs4 install script set the `CLASSPATH` variable within Dr. Java but it did not set it in your Unix environment. This setting can be taken care of each time you fire up the Terminal app by defining it in your `.profile` file. Making sure to use the single-quote marks, in a Unix command shell type exactly:

  ```
  > cd
  > echo 'export CLASSPATH=$CLASSPATH:/usr/local/algs4/algs4.jar' >> .profile
  > source .profile
  ```

- **Windows 10** If you're using Windows 10, you're in reasonable shape — Windows 10 supports the Unix Bash command shell. Follow the installation instructions here. See a course staffer if you have a problem.

- **Windows earlier version** If you're running an earlier version of Windows, we're going to have to improvise. The easiest solution would be either 1. to upgrade to Windows 10 and follow the instructions above, or 2. borrow a Mac for the term (a reach, I know). If you cannot upgrade to Windows 10 for some reason or borrow a Mac, let us know. We'll make a virtual Linux machine available to you.

Once you have a Unix shell open, learn the basics of the Unix command shell by following one or both of the following tutorials.

Tutorials

1. If you are new to Unix, git and GitHub you're going to need to learn how to use these systems. Two tutorials from UC Berkeley:
    1. Unix & GitHub
    2. Unix

Once you've completed the tutorial, use the Unix `cd` and `mkdir` commands to make a course home directory `cs1102` in some convenient place. This is where all of your course materials should be stored.

Part 3: Java Code

By now you should have received an email inviting you to clone a git repository for problem set 1. If you haven't done so already, clone that repo and move the folder housing the repo into your `cs1102` folder. Then do the following 4 exercises. They are contained in folders `one`, `two`, `three` and `four`.

1. (2 Points) Java's `System` package has utilities for input and output. In particular, there's a handy function named `format` in `System.out` supporting *formatted output* (There is a companion `format` function in the `String` class: `String.format`). For example, the call

    ```
    System.out.format("The %s rock!", "Patriots");
    ```

    would print to the console

    ```
    The Patriots rock!
    ```

    The string `"The %s rock!"` is a *format string* and `%s` represents a hole in the string accepting a string as input. Other hole specifiers are `%d`, `%f` and `%c` for integers, reals and character (respectively). For example, the call

    ```
    System.out.format("BEGIN--%d--%c--%f--%s--END", 12, 'A', 3.14, "Mei");
    ```

    prints

```
BEGIN--12--A--3.140000--Mei--END
```

In the `one` folder, modify the file `Hello.java` so that it uses the `System.out.format` procedure to print `Hello World!`. Of course, this can be done with a format string with no holes at all. You are looking to write the body of the function:

```
public class Hello {
  public static void main(String[] args) {
    // YOUR CODE HERE!
  }
}
```

2. (2 Points) Once a Java program `A.java` successfully compiles, the app (i.e., `A.class`) can be run on the JVM from within Dr. Java or from the Unix shell. Java apps can receive *command line arguments* just as they can in Python and OCaml. For example, given the `Hello` code above, the body of the `main` function can access the command line arguments in the array `args`. In

   ```
   > java Hello Boston College
   ```

   the string `Boston` would be in `args[0]` while the string `College` would be the value in `args[1]`.

   In the folder `two` rewrite the `Hello` function so that it accepts a name from the command line. Running it as in

   ```
   > java Hello Hao
   ```

   should print

   ```
   Hello Hao!
   ```

Note that if you want to provide command line arguments from within Dr. Java, you should run the JVM through the `Interactions` tab on the bottom.

3. (2 Points) Java is notorious for it's complicated IO architecture. It's very flexible but it's tough on beginners! Thankfully the author's of our text provide a simpler interface for IO. In this problem you'll use the `readString` function from `StdIn`, i.e., you'll use the `StdIn.readString` function to support greetings for *two* names: one from the command line and the other from user input. All of this takes place in the `Hello` program in the folder `three`. Running that program should go as follows:

```
> java Hello Mary
Type a name: Martha
Hello Mary and Martha!
```

4. (2 Points) Maybe it's before your time, but Jack can be a very bad boy. In the folder `four` there is a file named `Jack.java`. Modify it so that it accepts an integer `n` as a command line argument. Your program should use `n` to determine how many times to print the string

   ```
   All work and no play makes Jack a bad boy
   ```

   For example, the call

   ```
   > java Jack 3
   ```

   would print

   ```
   All work and no play makes Jack a bad boy
   All work and no play makes Jack a bad boy
   All work and no play makes Jack a bad boy
   ```

   For this problem you'll need two things: 1. to convert a string representation of an integer such as `"343"` to an actual `int` `343` and you'll need to call the `System.out.format` function repeatedly. For the former, take a stroll through Java's built-in `Integer` class to see if there are any pre-defined library functions that might convert a string such as `"343"` to an integer `343`. For the latter, maybe google Java for-loops. If you want style points, write it recursively!

Once your code works to your satisfaction, use the `git add`, `git commit -m "message"` and `git push` commands to stage, commit and upload your work to your master repo on GitHub. You're done!