

Programming Assignment 2

Due: 11:59pm, Saturday, January 16

Overview

The goals of this section are to:

1. Write your first Java program
2. Use a simple graphics API
3. Allow room for a creative extension to the assignment

Setup

After HW1, you should know how to connect to the ieng6 servers, move and edit .java files, and compile and run your programs. This assignment requires one starter file and an external library. Set up your directory via:

```
$ mkdir HW2
$ cd HW2
$ cp /home/linux/ieng6/cs11wb/public/HW2/* .
```

Turtles (40 pts)

By the end of this assignment you will have created a program that can draw an image similar to:



However, instead of **CS11WBXX**, be sure to use your own login id!

To get started, you will be editing the file `CS11TurtleGraphics.java`. This program uses some concepts that we have not touched on yet, but that's ok! Your goal is to read through the code, make small changes, and compile and run the program to see if it's working the way you expect.

The code makes use of classes from the provided `turtleClasses` library, namely `World` and `Turtle`. `World` is the drawing canvas and `Turtle` is the "pen" that will do the drawing. You will only need to use methods of `SimpleTurtle` to do all of your drawing. See the documentation for this class [here](#). Note: the methods `turnLeft()` and `turnRight()` do not make the turtle face in the left or right direction. Rather, they rotate the turtle 90 degrees counterclockwise or clockwise. Another method that you may find useful to draw line segments at an angle (such as at the top of the '1') is `turn()`, which turns the turtle by a specified number of degrees. There is no specific, required style to draw your letters. As long as it can be clearly understood what each character actually is and the width and height requirements are observed.

Use the code given for 'C' as a guide for drawing the other letters, and start by completing the code for `drawS()`. From there, add one letter at a time. Do not change the size of the 1080x480 pixels for the `World`. Each character should be drawn 40 pixels wide and 80 pixels high. There should be 40 pixels of space between each character and between each line of text.

Grading

We will be grading this assignment based on your programs ability to match the expected output. We'll deduct points if a letter is off or missing, if there is an issue with spacing, or if there's anything on the output that wasn't asked for. If you want to get creative, read the next section!

Programmer of the week (optional)

If you'd like to compete in the programmer of the week challenge, you must complete the above part of the assignment as described. Once you get that to work, create another class, `POWeek.java`, and again, by using the turtle API, make an image or animation. If you've done programming before and are aware of loops, conditions, threads, etc. feel free to use them. Your only restriction is that your animation has to be based off the turtle API.

Your tutor will present your animation at our weekly meeting and we'll vote for the top 5 submissions from each class and show those to the students to vote for the programmer of the week. The winner will get an award, 1% extra credit added to their final grade, and be famous for a week.

Style Requirements (10 pts)

You will be graded for the style of programming on this assignment. A few key requirements for style are given below and at <https://google.github.io/styleguide/javaguide.html>. These

guidelines for style will have to be followed for all the remaining assignments. I've underlined the comments that students have lost the most points on in the past. Read them carefully.

- Use reasonable comments to make your code clear and readable.
- Use comments for all classes and methods.
 - The comments should describe the purpose of your program and methods. See the starter file for examples on how to comment a method.
- Use reasonable variable names that are meaningful.
- Use static final constants to make your code as general as possible. No hardcoding constant values inline.
- Judicious use of blank spaces around logical chunks of code makes your code much easier to read and debug.
- Keep all lines less than 80 characters. Make sure each level of indentation lines up evenly. The default settings in vim and emacs files that I provided in HW1 will help on this. Anytime you see your code change to a gray color you've exceed 80 characters and should break your code into 2 lines.
- Every time you open a new block of code (use a '{'), indent farther by 2 spaces. Go back to the previous level of indenting when you close the block (use a '}'). Do not use tabs. The default setting files for vim and emacs that I gave you in HW1 will automatically change your tabs to spaces, so this should be taken care of for you, but double check that it's working.
- Always recompile and run your program right before turning it in, just in case you commented out some code by mistake.

Turnin Instructions

Remember the deadline to turn in your assignment is Saturday, January 16 by 11:59pm.

When you are ready to turn in your program in, type in the following command and answer the prompted questions:

```
$ cd ~/
$ bundleP2
Good; all required files are present:

    HW2

Do you want to go ahead and turnin these files? [y/n]y
OK.  Proceeding.

Performing turnin of approx. 6144 bytes (+/- 10%)
Copying to /home/linux/ieng6/cs11wb/turnin.dest/cs11wb.P2
...
Done.
Total bytes written: 31744
Please check to be sure that's reasonable.
Turnin successful.
```

bundleP2 will look for the directory HW2 and will compress the files into a single turnin submission. You can turnin your program multiple times. The bundle program will ask you if you want to overwrite a previously-turned in homework. ONLY THE LAST SUBMISSION IS USED!