Lab 5: Project 2 (getting started)

Pre-lab

- Get the project 2 starter files using `git pull skeleton master`
- Watch the lab 5 video.

Introduction

In this lab, you will get started on project 2. Project 2 is a solo project — no partners. Your work must all be your own. It will be long, arduous, and at times frustrating. However, we hope that you will find it a rewarding experience by the time you are done.

More than any other assignment or in the course, it is extremely important that you begin early. A significant delay in starting this project will leave you most likely unable to finish.

The project 2 spec will be released in beta form 2/17 (Wednesday), and should be effectively complete 2/19 (Friday). However, it will be possible to start the project even before its final release.

Project 2

In this project, you will build a text editor from scratch. It will support a variety of non-trivial features (scroll bars, arrow keys, cursor display, word wrap, saving to files), but you will not be building any of those features in lab today.

In this lab, you'll take a simple example `SingleLetterDisplaySimple.java` and use it to build a simple barely-usable text editor with the following features:

- The text appears starting at the top left corner.
- The delete key works properly.
- The enter key works properly.

HelloWorlding

In this project, you will work with a massive library with a very complex API called JavaFX. It will be your greatest ally and most hated foe. While capable of saving you tons of work, understanding of its intricacies will only come through substantial amounts of reading and experimentation. You are likely to go down many blind alleys on this project, and that's fine and fully intentional.

When working with such libraries, there is a practice I call "Hello Worlding." Recall that the very first thing we ever did in Java in this course was write a program that says "Hello World".

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

Here, the goal was to give us a foundation upon which all else could be built. At the time, there were many pieces that we did not fully understand, and we deferred a full understanding until we were ready. Much like learning sailing, martial arts, swimming, mathematics, or Dance Dance Revolution, we start with some basic actions that we understand in isolation from all else. We might learn a basic throw in Judo, not imagining ourselves in a match, but focused entirely on the action of the throw. Only later will we start trying to use a specific throw in a specific circumstance. This is the nature of abstraction.

Programming is no different. When we are learning new things, we will need to keep our learning environment as simple as possible. Just as you probably should not try to learn a new swimming kick in 10 foot waves, you should learn about the features of a new library using the simplest test programs possible.

However, we musn't go too far: one must still be in the water to truly learn to swim. Practicing the arm movements for the butterfly stroke on dry land may lead to little understanding at all. Finding the right balance between a "too-simple" and "too-complex" experimentation environment is an art that we hope you'll learn during this project.

I call this art HelloWorlding. HelloWorlding is the practice of writing the simplest reasonable module that shows that a new action is even possible.

For example, the SingleLetterDisplaySimple module described in the next section of this lab is about the simplest possible example that shows all of the strange features of JavaFX that we'll need to use to complete this lab.

Likewise, the bare-bones Editor that you'll be building in this lab is itself a HelloWorld — a stepping stone to the glorious editor you'll build by week 8. This is learning to swim in calm seas.

Examples

SingleLetterDisplaySimple

Kay Ousterhout (lead developer on this project) has created a number of example files for you to use to learn the basic actions that you'll need to succeed in using JavaFX.

Try compiling and running `SingleLetterDisplaySimple`. Things to try out:

- Typing letters.

- Using the shift key to type capital letters.
- Using the up and down keys to change the font size.

If you get "package does not exist" messages, this means you installed the OpenJDK (an open source alternative to Oracle's implementation of Java). You will need to install JavaFX.

Once you've had a chance to play around with the demo, it's time to move on to building our editor.

Creating a Crude Editor

Head to your `proj2/editor` folder. In this folder, there should be a file called `Editor.java`. This is the file that we'll edit today. We'll be adding three new features to our editor:

- Task 1: The ability to show all characters typed so far (instead of just the most recent).
- Task 2: Display of text in the top left corner of the window (instead of in the middle).
- Task 3: The ability to delete characters using the backspace key.

This lab is graded on effort and you'll receive full credit even if you don't complete all 3 of these features by Friday. However, we strongly recommend that you do so if at all possible.

Depending on whether you're using the command line or IntelliJ, follow the directions below.

Command Line

Let's start by making sure you can compile and run the skeleton file. If you're using the command line, head to the editor subfolder of your proj2 directory.

```
$ cd proj2
$ cd editor
```

Lab 5 Navigation

```
$ javac Editor.java
```

This should compile your Editor. However, because Editor is part of a package, you'll need to run the code from one folder up, and using the full package name:

```
$ cd ..
```


This should start up your program, which at the moment, doesn't do anything. Press control-C in order to quit.

IntelliJ

Let's make sure we can compile our file in IntelliJ. Open up the Editor.java file that came as part of the proj2 skeleton. Try to compile the file. Compilation should complete successfully. Don't bother running the file at this point, since it won't display anything.

Task 1: Multiple Characters

Your first goal is to make it so that your Editor is just like SingleLetterDisplaySimple, except that it displays all characters typed so far, as shown in the lab5 video. Copy and paste the relevant parts of SingleLetterDisplaySimple demo over to your Editor.java file. Now modify this code in such a way that your editor shows everything that has been typed so far. Don't worry about the delete/backspace key for now.

This will probably take quite a bit of experimentation!

Hint: You'll need to change the argument of setText.

Recommendation: If at any point you need a list-like data structure, use the java.util.LinkedList class instead of one of your Deque classes from project 1.

Task 2: Top Left

Now modify your code so that the text is displayed not in the center of the screen, but in the top left.

Task 3: Backspace

Lab 5 Navigation

Now modify your code so that backspace properly removes the most recently typed character. Once you're done with this, you're done with the lab.

Our First Blind Alley

It turns out that the approach we've adopted so far is

The issue is that there is no easy way that we can support an important later feature of the project: Clicking. This feature is supposed to allow us to click on a certain part of the screen and move the cursor to that postiion. JavaFX can tell us the X and Y coordinates of our mouse click, which seems like it'd be plenty of information to suppose click-to-move-cursor.

However, the problem is that we're deferring all of the work of laying out the text to the setText method. In other words, if our entered text is:

```
I really enjoy

eating delicious
potatoes with

      the sunset laughing at me,
every evening.
```

Our current approach lets the JavaFX figure out how to display all of these letters on the screen. If we clicked on the line right after the word "with", JavaFX would dutifully tell us the X and Y coordinate. However, we'd have no way of figuring out that this is position #48 in the text.

You'll need to come up with an alternate approach. There is a way to do this without using any new JavaFX features. However, it means you're going to need to build a text layout engine. This might be a good next step in the project.

Note: While JavaFX does support allowing you to click on a letter to get the position of this letter, in this example, the place we are clicking is not on a letter, but just on whitespace in the middle of the text.

Running the 61B Style Checker

Remember that all code submitted from this point forward will required to obey the CS61B style guidelines. As noted, you

be a useful reference. Instead, it will be much easier to simply run the style checker. You can do this by running the style61b.py script provided in the lib folder (you may need to pull from skeleton again if you don't see it). For example, on my machine, I can run it as follows.

```
$ python3 /Users/jug/work/bqd/javalib/style61b.py *
```

Try it out on the files in your IntList folder. You should see that there are at least two style errors (the two we put in, plus whatever you may have introduced yourself). You are not required to pass these checks until after the midterm (though from now on, we will be running the style checker for your reference in the autograder, for no credit).

When you pass the style check, the output should look like:

```
Starting audit...
Audit done.
```

Some of these style rules may seem arbitrary (spaces vs. tabs, exact indentation, must end with a newline, etc.). They are! However, it is likely that you'll work with teams in the future that have similarly stringent constraints. Why do they do this? Simply to establish a consistent formatting style among all developers. It is a good idea to learn how to use your tools to conform to formatting styles.

Submission

Submit Editor.java and MagicWord.java. If you're submitting before 10 PM on Wednesday, use the magic word "early".

Lab 5
Navigation