CS 3053
Homework: Prototype A
Due Tuesday 2019.02.12 at 8:00pm.

All homework assignments are individual efforts, and must be completed entirely on your own.

In this assignment you will learn how to develop basic Model-View-Controller applications using Java, Swing, and Gradle. Specifically, you will learn how to write code to lay out components in frames, adjust their individual appearances and behaviors, handle different kinds of interaction events, manage a central shared data model, and compile and run applications using Gradle.

Learning about Gradle

At least two of your prototype assignments will involve writing in Java and building and running using Gradle. Visit https://docs.gradle.org/current/userguide/userguide.html to get a sense of what Gradle does and how it works. You **don't** have to install Gradle on your system. You don't even have to do any Gradle scripting. I've done that for you.

In the `PrototypeA` download, go to the `ou-cs-hci` directory in `Build`. Read the `about.txt` file to learn about running Gradle tasks on the command line and the tasks you're most likely to need. Feel free to take a peek at the `build.gradle` script file. The *Alternative Start Scripts* section at the end describes how to create scripts to run additional main() classes. Most likely you won't need to create any, but the option exists. Don't otherwise change `build.gradle`!

Using Gradle with Eclipse

Gradle and Eclipse play well together. First, make sure you're using a version of Eclipse with the BuildShip plugin installed. Most recent Eclipse for Java distributions come with it. Second, run `gradlew eclipse` in the `ou-cs-hci` directory to prepare it for import into Eclipse. Third, import the `ou-cs-hci` directory as a *Gradle / Existing Gradle Project* using the Eclipse import wizard. Eclipse will add a *Gradle Tasks* pane to the editing window for running Gradle commands. (Other IDEs are less well supported, but can still be used to edit code.)

Building and Running Applications

To compile the build, type `gradlew installDist` on the command line. You need to be in the `ou-cs-hci` directory for this to work. Then go into the `build/install/base/bin` directory and run any of the resulting programs by typing its name. Each program has a `.bat` version for running on Windows. (Note that the `gradlew run` shortcut launches only the `base` program.)

The source code packages are organized to hold code for your prototype assignments as well as example applications that we'll see in class. The download for each prototype assignment will provide an updated `ou-cs-hci` directory with all necessary classes and program scripts.

Exploring an Example

The `swingmvc` program displays a gallery of common UI components. Its purpose is to provide examples of how to create, style, layout, and connect components in a simple MVC application. Start by reviewing the slides from class on MVC, common Java components, and `swingmvc`. Then browse the classes in the `edu.ou.cs.hci.application.swingmvc` package, starting with `Application.java`. Run the program, interact with the gallery components, and trace through the code to see how interactions lead to visible updates. Notice how some components are effectively coupled to each other by depending on the same data value in the model.

Bundling assets in Java applications isn't straightforward. The `edu.ou.cs.hci.resources` package is designed to make it easier. Put any files you need in subdirectories of that package. Gradle is configured to copy any non-`.java` files in the source tree into the same places as the compiled `.class` files. This means that your assets will accompany your application and can be easily accessed by your program, regardless of where it is running. The `swingmvc` code has many examples of accessing both image and text files this way.

Implementing your Refined Design

In the DesignA assignment, you created a refined design for your choice of UI. It's finally time to implement your refined design as a horizontal prototype. To make this easier, you'll be modifying a copy of the `swingmvc` program code. The download provides a copy for you to work on. Go into the `edu.ou.cs.hci.assignment.prototypea` package and modify the classes in it. Focus on `Model.java` and `View.java`; you shouldn't need to change the other two classes. Compiling the build will create a script called `prototypea` for running your program.

The goal is to prototype the components, layout, and shallow interactivity of your design. At this stage of the design process, reproduce the *general* style of the design, but don't bother with fine details. Remember, you're prototyping your Mockup design, not the original UI!

Feel free add Java files or even subdirectories for subpackages as you like. You probably won't need to. Organize your code inside the classes as you like, but keep readability in mind. Avoid very long methods, group related methods into sections, and document your code helpfully. Also remove any unused code left over from the `swingmvc` program before you turn it in.

Turning It In

Turn in a complete, cleaned, renamed, zipped copy of your `PrototypeA` directory:

- Take a screenshot of your application window when it's in an interesting graphical state.
- Put the screenshot in the `Results` directory as `snapshot.png` or `snapshot.jpg`.
- Go into the `ou-cs-hci` directory.
  - Make sure it contains all of the code modifications and additions that you wish to submit.
  - Run `gradlew clean` to reduce the size of your build.
  - If you're using Eclipse, run `gradlew cleanEclipse` and delete the `bin` directory.
- Append your 4x4 to the `PrototypeA` directory; mine would be `PrototypeA-weav8417`.
- Zip your entire renamed `PrototypeA` directory.
- Submit your zip file to the Homework - Prototype A assignment in Canvas.

These steps will make your submissions smaller and neater, which speeds up grading a lot.

To score the assignment, we'll be looking at how many elements in your refined design appear as components in your prototype, how well the prototype reflects the design's *overall* layout and *general* style, whether each interaction has a reasonable effect (to modify a value in the model and update the view correspondingly), and how clearly your code is organized and documented. The maximum score is 20 out of 20.