CS 4063/5063
Homework: Prototype C
Due Tuesday 2022.03.08 at 11:00pm.

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

In this assignment you will continue to develop a movie collection application in JavaFX. You will learn how to implement menus that support file open and save capabilities. You will also learn how to implement an accordion that supports interactive faceted filtering of movies in the table that you implemented in PrototypeB. More generally, you will learn how to utilize *observable collections* and *properties* to develop the kinds of highly interactive, interconnected user interfaces that are now common in productivity, analytics, and visualization applications.

Preparing for Implementation

In the PrototypeB assignment, you added a `CollectionPane` for browsing the movies in a collection, with a `TableView` that shows and allows editing of key movie attributes. Although the `EditorPane` was linked to the `TableView` to show the currently selected movie, this link was shallow. Each pane showed and supported editing on independent copies of the collection.

In this assignment, the goal is for all panes to show a single, common set of movies and for the entire UI to stay in sync as the user browses, filters, and edits movie attributes. The first step is to centralize the management of data objects (movies, genres, ratings) inside the `Model` class. The second step is to store the objects in *observable* collections to facilitate change tracking. To make things easier, we have done both of those steps for you already. Study the details in the `Model`'s `addObservables()` method, and read about JavaFX *properties* and *collections* here:

https://docs.oracle.com/javase/8/javafx/properties-binding-tutorial/binding.htm
https://docs.oracle.com/javase/8/javafx/collections-tutorial/collections.htm

Prototyping Your Refined Design

The third step is to add methods for accessing properties in the `Movie` class. This step enables tracking of changes (including via user editing) to movie objects throughout the UI. Next, you will integrate your `MenuBar` design from DesignC into the `View` class, with `MenuItem`s to allow the user to open and save their movie collections as CSV files. You will also implement the design of your `Accordion` from DesignC and apply filtering to movies in the `TableView` using the interactive parameters entered in the widgets in your `Accordion`. Then, with the improvements to the `Model` and `Movie` classes in place, you will extend various methods in the `EditorPane` and `CollectionPane` classes to add the listening & updating needed to wire it all up.

Implementing Your Prototype

In the DesignC assignment, you continued to develop the Collection and Editor wireframes. In this assignment, you will implement the **menubar** and the **accordion** area as a prototype. The prototype will be horizontal, yet highly interactive, and with vertical features to open and save files. Start by putting a copy of your `DesignC.bmpr` file in the `Results` directory. *(We need your design file for comparison with your prototype UI. You can create a design file to include now even if you didn't finish the DesignC assignment.)*

To make implementing easier, you'll add code to designated places. In the `Build/ou-cs-hci` source tree, go into the `edu.ou.cs.hci.assignment.prototypec` package and modify the `Movie.java`, `View.java`, `pane/EditorPane.java`, `pane/CollectionPane.java` files.

The `EditorPane` and `CollectionPane` classes start from my "solution" to the PrototypeB assignment. (Look for "OPTIONAL" in `View.java` for how to view it.) You are welcome to use my designs, but I encourage you to use your own if they work well for you. **If you decide to use your own**, carefully replace the widget and layout code in my panes with the code from yours. Be careful to preserve new code that deals with movie data, which now comes from `Model`. You will also need to modify your code to use the new observables (and keys) provided by `Model`.

The places to add code are as follows:

#0 (in `Movie`) — Add property access methods for all attributes.

#1 (in `Movie`) — Add code to convert attribute values into strings for saving.

#2a-c (in `View`) — Implement menus, with listeners for the file open and save options.

#3a-b (in `View`) — Add code to support file choosing for the file open and save options.

#4a-b (in `EditorPane`) — Add listeners for tracking changes to movie properties.

#5 (in `EditorPane`) — Add code to update widgets when movie properties change.

#6a-d (in `CollectionPane`) — Add accordion widgets and listeners for their properties.

#7a-b (in `CollectionPane`) — Add listeners for tracking changes to movie properties.

#8 (in `CollectionPane`) — Add code to filter the table on accordion widget properties.

#9a-d (in `CollectionPane`) — Add code to update the UI when movie properties change.

I recommend following the order above. Except for TODOs #2–#3, closely examine the nearby code provided for the *title* and *image* attributes. You can usually copy and modify that code to implement the other attributes. Keep readability in mind and document your code helpfully.

You shouldn't need to change any of the other classes. Compiling the build will create a script called `prototypec` (in `build/install/base/bin`) for running your program. *As you test, keep in mind that movie attributes should sync everywhere in the UI, but filtering parameters will not. Views apply independent filters; a table will not show the selected movie if it is filtered out.*

Evaluating Your Prototype

Recruit two people as participants for a quick evaluation of your prototype. Ask them to practice the task from DesignC several times. For the Open and Save steps, tell them to simply cancel in the file chooser dialogs. Once they indicate confidence in performing the task, ask them to perform it three more times. Use a stopwatch and record how long it takes them to perform the task each time. Compare their average time to the Fitts' Law result you calculated in DesignC.

Identify a way to significantly improve the efficiency of the task in the prototype UI. Then identify an alternative task (one likely to be performed as frequently) that would be slowed down by your proposed improvement. Write up a brief description (< 300 words) of your evaluation process, comparison, proposed improvement, and drawbacks for the alternative task. Put your writeup in the `Results` directory as `evaluation.txt`. *(If you didn't complete a Fitts' Law analysis in DesignC, you can do so now by adding a Comment to your `DesignC.bmpr` file in `Results`.)*

Turning It In

Turn in a complete, cleaned, renamed, zipped **COPY** of your `PrototypeC` directory:

- Put a copy of your `DesignC.bmpr` file in the `Results` directory.
- Take a screenshot of a window showing your *Collection* tab in an informative graphical state.
- Put the screenshot in the `Results` directory as `collection.png` or `collection.jpg`.
- Do the same for your *Editor* tab, as `editor.png` or `editor.jpg`.
- Go into the `ou-cs-hci` directory.
  - Make sure it contains all of the modifications and additions that you wish to submit.
  - Run `gradlew clean` to reduce the size of your build.
  - If you're using an IDE, remove any IDE-specific files (such as the Eclipse `bin` directory).
- Append your 4x4 to the `PrototypeC` directory; mine would be `PrototypeC-weav8417`.
- Zip your entire renamed `PrototypeC` directory (including `About`, `Build`, and `Results`).
- Submit your zip file to the Homework - Prototype C 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 menu and accordion design appear as components in your prototype; how well the prototype reflects the design's layout and style; whether browsing, filtering, and editing interactions have the expected effects; and how clearly your code is organized and documented. The maximum score is 20 out of 20.