Objectives:

- JavaFX
- Multiple views and user interaction
- ArrayList
- UML Diagrams

Task: Create an app for the Daily Bugle

News of your app development skills has spread across, well, the universe! Owners of the Daily Bugle newspaper have asked for your assistance getting their most critical content in front of their readers - in the form of a desktop app! They have provided data for classifieds advertisements and crossword puzzles. This could be fun!

Getting Started

To begin this lab, create a new **JavaFX project** named according to the lab guidelines, and create the following:

- **Main.java** - in the application package
- **MainController.java** - in the application.controller package
- **ClassifiedsController.java** - in the application.controller package
- **CrosswordController.java** - in the application.controller package
- **Classifieds.java** - in the application.model package
- **Advertisement.java** - in the application.model package
- **Crossword.java** - in the application.model package
- **Main.fxml**
- **Classifieds.fxml**
- **Crossword.fxml**

Note: fxml files will be at the top of the Eclipse project - creating a new FXML document will put them in this default location.
Data files will be stored in a **data** folder at the top of the project (this includes sample data files provided in this lab).

App Design

Your program will show a view similar to the one shown in attached file **AppView1.png** when the app is run. This view displays a logo for the newspaper and two buttons. (*Note that the image* <u>*can*</u> *be "hard-coded" - does not need to be dynamically loaded).*
This view will be the **Main.fxml**

When the user clicks on the Classifieds button, they will be taken to a second view, see attached file **AppView2.png.** This view displays up to 4 “help wanted” advertisements. It also includes a logo for the newspaper and a “Home” button to return users to the previous screen.
This view will be **Classifieds.fxml**

From the main view, if the user clicks on the Crossword button, they will be taken to a third view, see attached file **AppView3.png.** This view displays the newspaper logo and “Home” button again, in addition to a crossword puzzle game. Clues to solve the puzzle are on the left, boxes to input letters are arranged on the center/right, and two buttons on the bottom of the view help users work on the puzzle.
This view will be **Crossword.fxml**

If the user is completing the crossword puzzle and clicks on the “Check” button, the incorrect letters entered should be made red in color. An example is shown in **AppView3-example1.png.**
If the user clicks the “Show Answers” button, all letters should be revealed (completing the puzzle). An example is shown in **AppView3-example2.png**.

You may customize your app how ever you choose - this includes images, sizes, fonts, colors, size of the app, configuration. Even make your own newspaper!
Remember to ensure your app works on all display sizes. For this lab, you can do this by making your app **no larger than 800x800**.

The app must have the described GUI components on each view.

Model

Classifieds

The **Classifieds.java** class will represent the classifieds section of this newspaper, in our case containing Help Wanted advertisements. The class Classifieds will have an object method called **loadAds** which takes in a file name and stores the data as **Advertisement** objects. The class **Advertisement.java** will have fields for a title, phone number, a boolean for “full time work” (if this is false, the position is part-time), a date when the ad was posted, and a name for the individual posting the ad. The Classifieds class should maintain an ArrayList of Advertisement objects. We will limit to 4 ads for the sake of this lab. Sample ads have been provided in **ads.csv**.

Crosswords

The **Crossword.java** class will represent the crossword puzzle section of this newspaper. Clues for the example crossword puzzle are provided in **down.csv** and **across.csv** - you may choose to use these or build your own. Note that the clues must be read from files, they cannot be hard-coded onto the interface. A class method called **loadData** will take in 2 file names and read the clues from the files. You may decide how you would like to store the clues and their corresponding answers.

All classes in the model must always have getters and setters for all class variables. Constructors are required for all required variables in a class.

Note that the controllers in your application should ***never*** *read files or update data. Instead, to follow MVC, the controller classes will call upon the model classes to complete these tasks.*

Reminders

Always test your app thoroughly before submitting, to ensure everything is working properly.
You must export the Eclipse project, including all files & dependencies for the project (this includes images, text files, fxml, etc). Submit the zip file on Blackboard.
As always, follow the instructions on the lab guidelines.

Rubric:

- (20pts) Correctness - app functions as described.
- (15pts) MVC - app is implemented as described, adhering to MVC design pattern.
- (10pts) Main view
- (10pts) Classifieds view
- (10pts) Crosswords view
- (20pts) Model - Advertisement and Crossword
- (10pts) UML Diagram (including fxml files)
- (5pts) Comments - Javadoc comments on all classes (does not include fxml)

Submissions which do not compile will receive a maximum of 10 points total.