Lab 3

Objectives:

- JavaFX
- ArrayList
- File I/O
- UML Diagrams

Task: We are making an application to store movie cast info! Starting with Disney movies in 2019. We can enhance our app in a future version.

Need organized info of movie casts. You are starting with Disney movies released in 2019. The app will use Java objects and show cast information with a simple interface. Follow the detailed description to obtain all the requirements of the application we need.

Getting Started

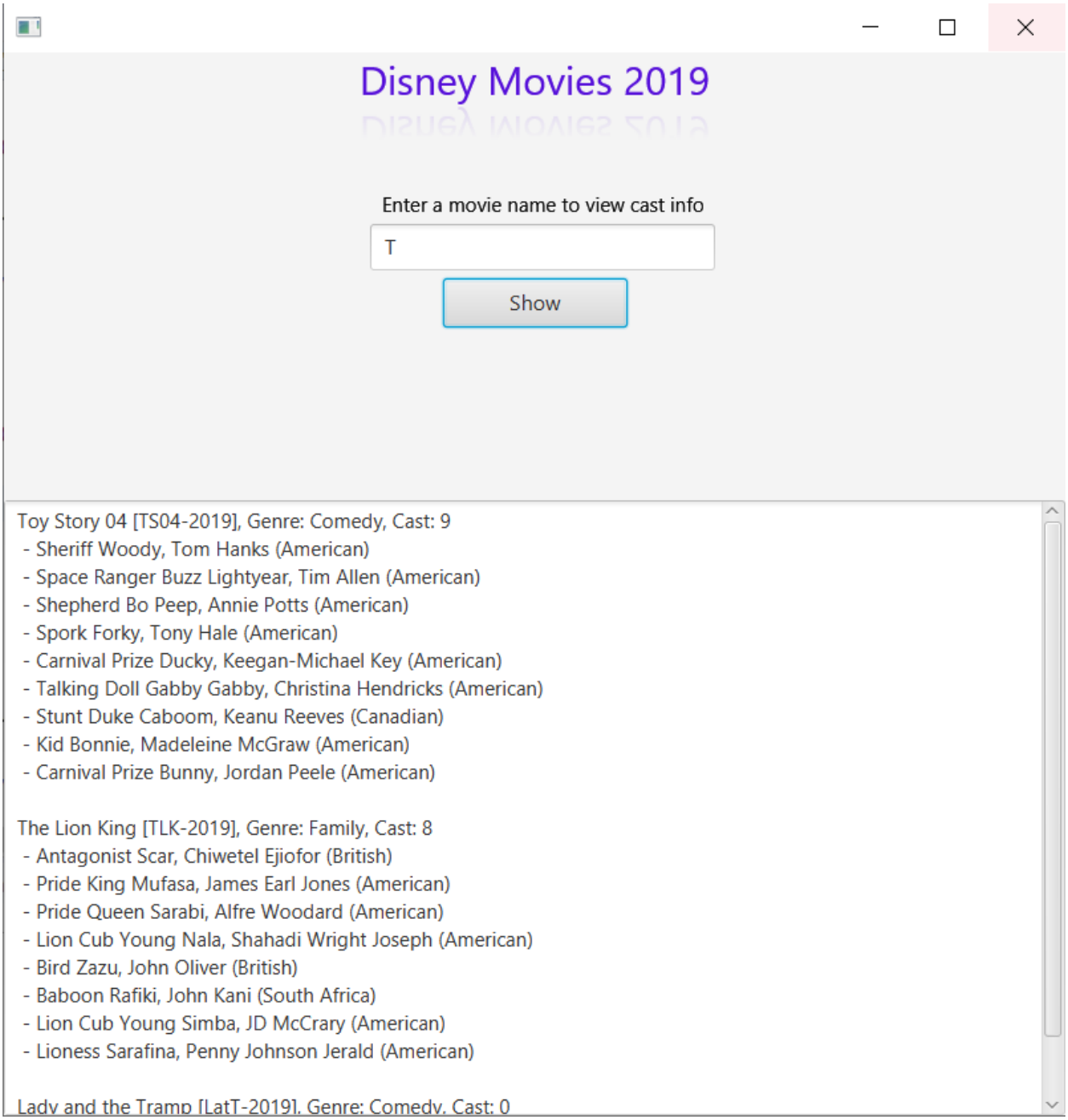
To begin this lab, create a new **JavaFX project** named **abc123-lab03**, and create the following:

- **Main.java** - in the application package
- **MainController.java** - in the application.controller package
- **DisneyMovies.java** - in the application.model package
- **Movies.java** - in the application.model package
- **Cast.java** - in the application.model package
- **Main.fxml** - in the application.view package

This lab is your first hands-on experience with JavaFX on your own. It will follow the MVC design pattern (as all of our JavaFX apps this semester). There will be 3 model classes, 1 view, and 1 controller.

App Design

Your program will show a view like the one shown below when the app is run:



This view will be the **Main.fxml**.

When the user first loads the app, both the text field and the text area will be blank. The user will enter a name into the text field and click the show button, which will populate data into the text area in the lower part of the app.

You may customize your app how ever you like - this includes images, fonts, colors, size of the app, configuration.
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 following GUI components:

- A label for the app (as "Disney Movies 2019")
- One label above an editable text field.
- A "show" button
- A text area to display results.

Making it Work

Main.java will launch the application and show the **Main.fxml**.
MainController.java will need to implement the EventHandler interface, and handle any events that occur when the user interacts with **Main.fxml**.

When the app launches, the MainController will call on the DisneyMovies class to load the data needed to populate the view. All data for the app has been provided, in csv format. Unzip this file and place the folder at the top of your Eclipse project. You can copy the files directly to your project as well.
Before moving forward, take a closer look at the data in these files. You can open them as spreadsheets, or you can view the comma-delimited format by right-clicking on them and choosing "open with > text editor".

The controller will need class variables for most of the GUI components on the view.

The Model

The model of your app will consist of 3 classes: DisneyMovies, Movies, and Cast. For each class, create class variables, constructors, all getters/setters, toString(), and any methods needed to complete the following requirements.

DisneyMovies.java

To be initialized, a DisneyMovies object must have a name. In addition, every DisneyMovies has an ArrayList of Movies objects.
The DisneyMovies will be the access point for the data in the remainder of the app.

The controller will call on methods in the DisneyMovies class to access Movies names, cast information, and to load data in the files.

Movies.java

A Movies object must have:

- A name (i.e. The Lion King)
- Unique ID (i.e. TLK-2019)
- A genre of Movies (i.e. Comedy)
- An ArrayList of Cast objects

There is no limit on the number of cast members in Movies

Cast.java

A Cast object must have:

- A character name (i.e. Woody)
- Corresponding actor's name (i.e. Tom Hanks)
- Role or occupation of the character in the movie (i.e. Sheriff)
- Primary nationality of the actor (i.e. American)

The remaining design of the model of this app is up to you - you may add other model classes if you find it necessary.

App Functionality

Now that we have organized the data of our app (the model), we can focus on what the app does with the data. There will be two concepts to focus on:

1. Loading and searching the data.
This part will be a responsibility of the model and will be called by the controller. The controller will need a method called **handle**, which must be connected to the Show button on the view. The handle method must not read data directly from the file! Instead, handle calls on a method in the DisneyMovies class to read the data from the files, creating Movies and Cast as needed. This method must be a class method, and will return a DisneyMovies object that has been created using the data in the files. Within the DisneyMovies class, there will be an object method **getMoviesByName** which will take in a String name of a Movies and return an ArrayList of Movies objects. In the MainController, the handle method has a

DisneyMovies object, it should use this object to call getMoviesByName(..), passing as a parameter the text entered by the user.

2. Displaying the data on the view.
This part will be implemented in the controller, setting values to the view. Use the ArrayList of Movies returned by getMoviesByName(..) to populate the text area at the bottom of our view.

You may set the name of the DisneyMovies object by default as "Disney Movies 2019", but the remaining data must be read from the given data files.

Testing Your App

Test your app thoroughly before submitting, to ensure everything is working properly. If you haven't implemented the search yet, you can test your app by having it always display all Movies when the Show button is clicked. Before moving beyond this step, ensure that your output in the app matches the format of the example given above.

Next, try searching as in the example above. Ensure all the data is loading correctly and that your data matches that given in the example image above.

Don't forget to try searching for a name that is not in our data files. Your app should not crash or show an exception in the console under this type of condition. Instead, show a message in the text area indicating no movie of that name could be found.

Submission: You must export the Eclipse project, including all files & dependencies for the project (this includes images, text files, fxml, etc).

Rubric:

- (40pts) Correctness - app functions as described.
- (20pts) MVC - app is implemented as described, adhering to MVC design pattern.
- (20pts) Main.fxml - has all required GUI components.
- (10pts) UML Diagram (includes fxml) - place the UML diagram at the top of (within) your Eclipse project, prior to exporting.
- (10pts) Comments - Javadoc comments on all classes (does not include fxml)

Submissions which do not compile will receive a maximum of 10 points total.
Submissions not using JavaFX will receive no credit.