Lab 3

Important Guidelines

All labs will be submitted on Blackboard, unless otherwise noted in the lab description. Export the project and upload the zip file. In Eclipse, click on the project, select Export, and choose Archive File.

The name of each project must be **abc123-labX**, where abc123 is your UTSA ID and X is the number of the lab. All letters are lowercase.
The name of your submission must be **abc123-labX.zip**. All letters are lowercase.
Any submission not following these requirements may not receive credit.

Reminder: laboratories must be completed individually. Collaboration with anyone violates the policies for academic integrity.
You may discuss algorithms and approaches with others. You may not code together or share a computer, under any circumstance.
If you have questions or concerns about this policy, contact your instructor immediately.

Objectives:

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

Task: The Federation needs you!

The United Federation of Planets needs your help organizing their fleet of interstellar starships. They're requesting software with a simple interface and Java objects to handle starships, which are spacecrafts manned by a crew. With this software, the Federation will be able to accurately track senior staff aboard each starship. Make it so!

Getting Started

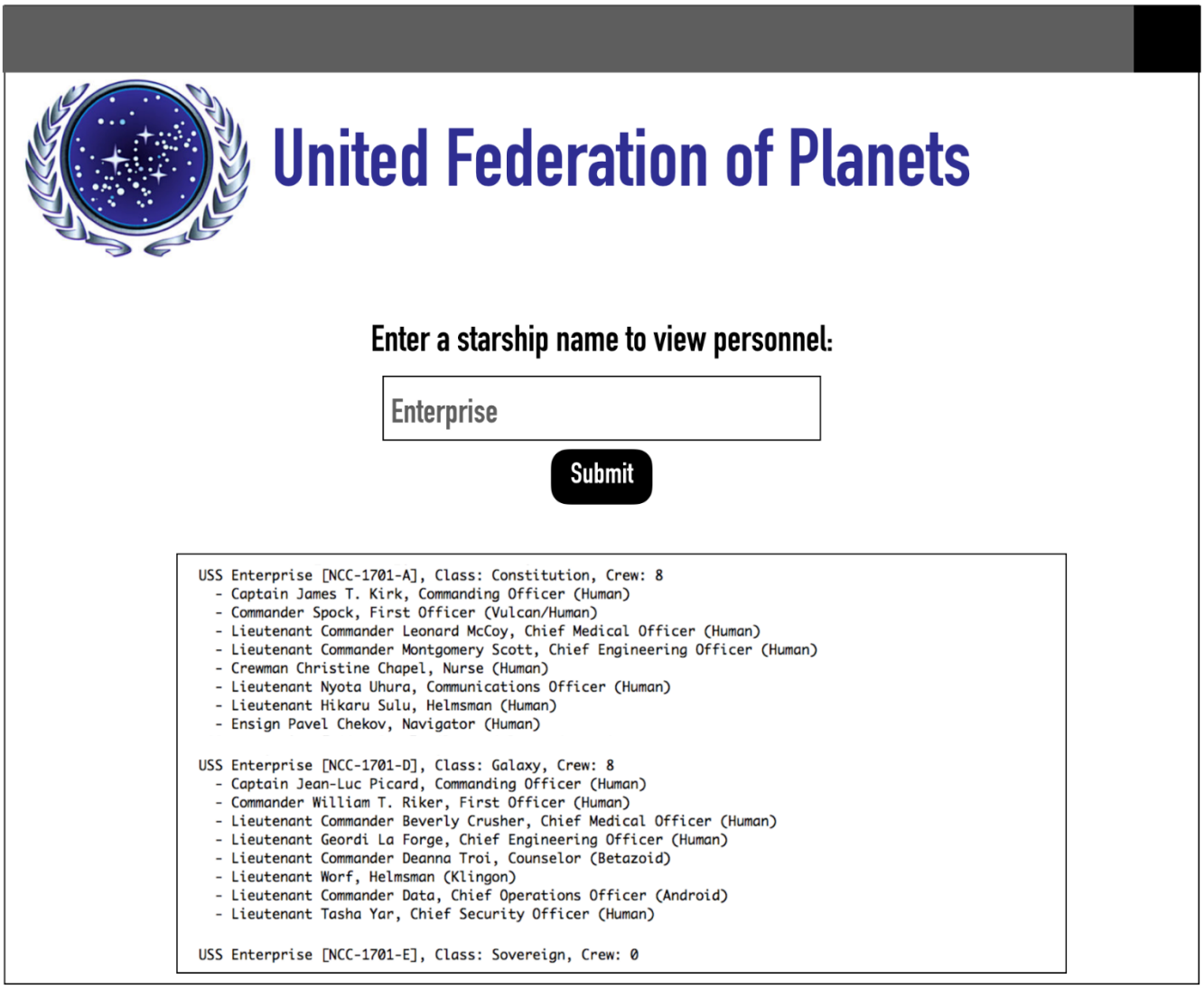
To begin this lab, create a new **JavaFX project** named **abc123-lab4**, and create the following:

- **Main.java** - in the application package
- **MainController.java** - in the application.controller package
- **Fleet.java** - in the application.model package
- **Starship.java** - in the application.model package
- **CrewMember.java** - in the application.model package
- **Main.fxml**

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 similar to 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 submit button, which will populate data into the text area in the lower half of the app.

You may customize your app how ever you choose - 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 (should here as "United Federation of Planets")
- One label above an editable textfield.
- A "submit" button
- A text area to display results.

Making it Work

Main.java will launch the application and shown 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 Fleet class to load the data needed to populate the view. All data for the app has been provided, in csv format, (check data.zip). Unzip this file, and place the folder at the top of your Eclipse project. 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: Fleet, Starship, and CrewMember. For each class, create class variables, constructors, all getters/setters, toString(), and any methods needed to complete the following requirements.

Fleet.java

To be initialized, a Fleet object must have a name. In addition, every Fleet has an ArrayList of Starship objects.
The Fleet will be the access point for the data in the remainder of the app. The controller will call on methods in the Fleet class to access starship names, crew member information, and to load data in the files.

Starship.java

A Starship object must have:

- A name (i.e. USS Enterprise)

- A registry (i.e. NCC-1701-A)
- A class of starship (i.e. Constitution)
- An ArrayList of CrewMember objects

There is no limit on the number of crew members on board a starship. *(They said it's "bigger on the inside", whatever that means..)*

CrewMember.java

A CrewMember object must have:

- A name (i.e. James T. Kirk)
- A position (i.e. Commanding Officer)
- A rank (i.e. Captain)
- A species (i.e. Human)

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 Submit button on the view. The handle method must not read data from the file! Instead, handle calls on a method in the Fleet class to read the data from the files, creating Starships and CrewMembers as needed. This method must be a class method, and will return a Fleet object that has been created using the data in the files.
Within the Fleet class, there will be an object method **getStarshipsByName** which will take in a String name of a starship and return an ArrayList of Starship objects. In the MainController, the handle method has a Fleet object, it should use this object to call getStarshipsByName(..), 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 Starships returned by getStarshipsByName(..) to populate the text area at the bottom of our view.

You may set the name of the Fleet object by default as "United Federation of...", but the remaining data must be read from the given data files.

Testing Your App

Always test your app thoroughly before submitting, to ensure everything is working properly. This is our first JavaFX app, so testing will be a little different.

If you haven't implemented the search yet, you can test your app by having it always display all starships when the Submit 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. 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 ship 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.