# Assignment 2

## Overview

The goal of assignment 2 is to create an interactive application in JavaFX that will allow users to access information from an API. The subject of the application should be ***anything you are interested in***.

Your program must be built using IntelliJ and stored in a PRIVATE GitHub repository. ***This is an individual assignment.***

What is an API? An API is an Application Programming Interface. It is used to transfer information between 2 systems. Often the file is transferred in JSON format. We will learn how to use GSON (Google's JSON utility) to convert JSON data into objects during our classes.

When the application is launched, it should show a usable, professional looking JavaFX application. There should be clear instructions on the screen for the user to know how to interact with the application. My advice – keep it simple!

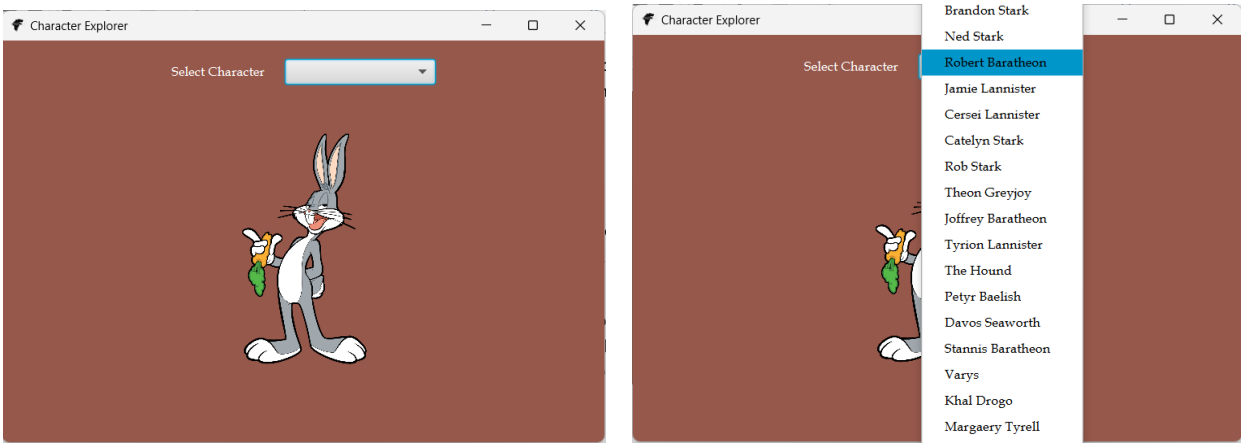
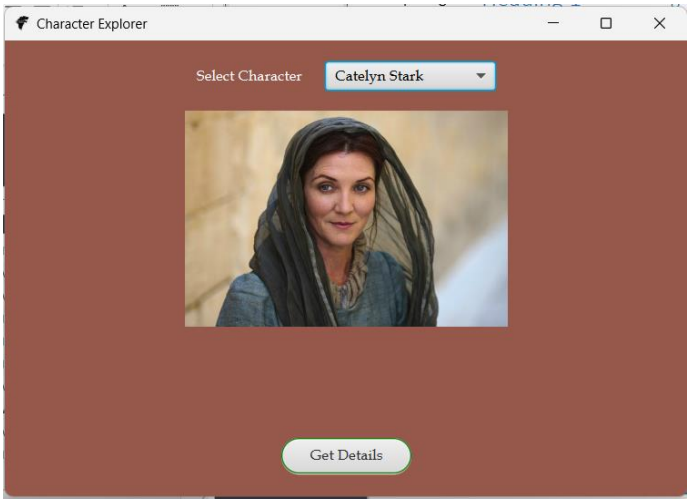



*Figure 1 - Initial launch of project showing an API call and parsing of a JSON file.*

The application must support reading from an API. How you show the information is up to you. It could be in a chart or a list of some variety. However, if you choose to represent the data from the API/JSON file, the user has to be able to interact with it.

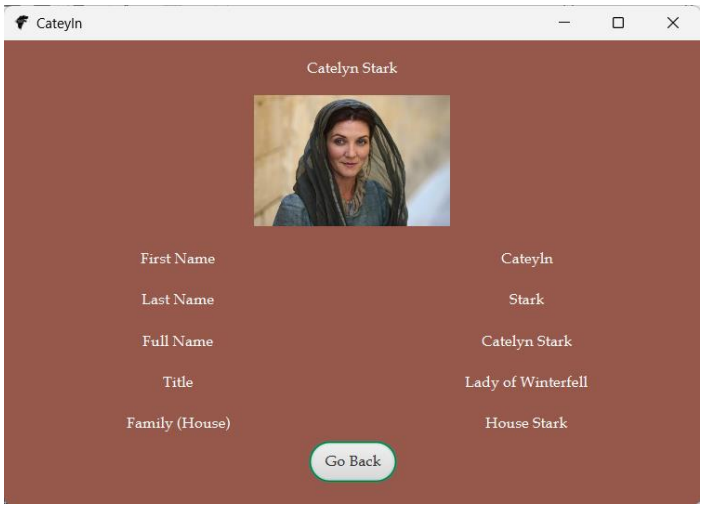
The API you use is up to you. There are many free ones, but be careful in choosing them. Some API's only allow for a maximum # of accesses per day. Others are "freemium" which typically work fine as class examples that are free, but you would need to pay if volumes go over a certain threshold. Use google and search for whatever subject interests you and "Free API".




1. User selected a character from the list.
2. API is called and returns a JSON file.
3. The JSON file is parsed and displayed as Character objects in the ImageView

*Figure 2-Application triggers API call and processes a JSON file.*

After a user clicks on one of the characters in Figure 2.




1. User clicked on a character from the list.
2. The reference was sent to API which returns more details about the character in a JSON file
3. The new JSON file is parsed and displayed on the screen
4. There is a “back” button so the user does not hit a dead end.

*Figure 3-When user selected a character, it shows more data and poster art.*

## What to Do:

1. Register your unique project idea/API by going to Java - Assignment 2. You can see what other students are planning to build. No duplicate projects will be allowed, so do not invest significant energy in your project until you receive approval from me.
2. When the application launches, it should display a scene with the ability for a user to trigger an API call or have the application read from a JSON file. If the opening scene is a chart or other graphic derived from a JSON file, it can be pre-loaded.
3. There should be a utility to either change scenes (such as a button) or to select a different chart.
4. All scenes should be styled using CSS. Do not use the default grey background for everything. Add some color, round some corners, add images to buttons, change some fonts… have fun with it!!
5. The icon on the stage should be something unique.

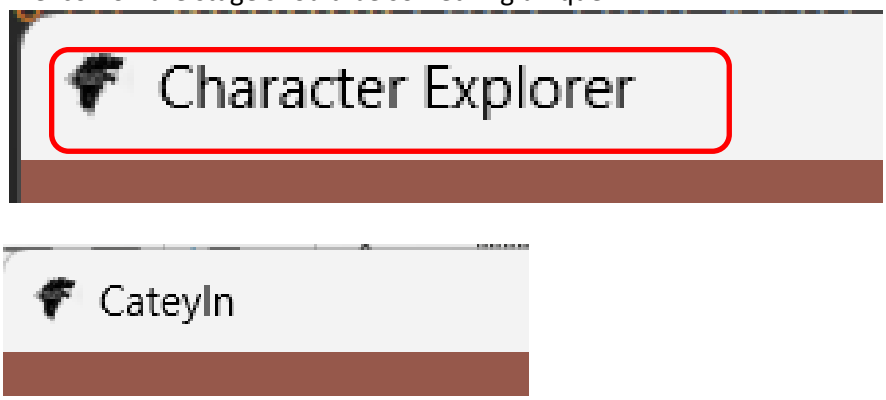



*Figure 4 - Image showing changed icon.*

## Grading

All of your marks will be based on the rubric defined below.

| Criteria | Level 0 | Level 1 | Level 2 | Level 3 |
|---|---|---|---|---|
| **Code Style** | The code does not follow typical Java programming style. I.e. a capital letter to start all class names, lower case letters start variable and method names | The code is indented and has proper upper case/lower case conventions | All of level 1, plus each method has a Javadoc style of comment prior to the method describing what it does | Level 2 plus different directories/packages are used for the model and controller files. All of the views & css files are in a resources area and ALL required libraries are configured to automatically load with maven. |
| **User Experience** | When the program launches, an exception is thrown | The program launches and there is a clear way to interact with the | Level 1 plus a professional look and feel is apparent. | Level 2 plus there are no “dead ends”. In other words, a user cannot |

| Criteria | Level 0 | Level 1 | Level 2 | Level 3 |
|---|---|---|---|---|
| | or the scene loaded is inoperable | program. The overall look and feel does not have a professional look to it. i.e. objects are not aligned or extreme colour choices are made. | | navigate somewhere in the and get stuck (i.e. not able to return to previous steps)<br>Note: this is only achievable if you have at least 2 scenes. |
| **The default scene has the ability to display data derived from a JSON file** | The first scene does not display JSON data. | The first scene has a Chart or some other object populated with data, but it is not populated from a JSON file. | Level 1 plus it is populated based off a JSON file. | Level 2 plus any labels / legend are easy to read/visible. It is obvious on how to use the information. |
| **Change to different scene** | There is no utility to change from the first scene | There is a functional object (i.e. button or radio button) that will change the scene to show a new chart, table, graphic, etc… | Level 1 plus the new scene is populated with information from a JSON file. | |
| **CSS styling** | There is no CSS stylesheet and/or it is not connected to the view object. | The CSS is connected and styles up to 3 elements. | Level 1, plus it styles up to 6 elements. | Level 2, plus it styles more than 6 elements. |
| **Change the icon and title** | The default icon is showing | There is a new icon and a title. | The new icon and title is somewhat related to the content. This is true for all scenes. | |
| **The Model class(es) is well structured** | The model class(es) is either not present or is not used. | The model class has poor naming conventions and/or poor instance variable data types. All instance variables must be private. | Level 1 plus all data types are logical. All method and variable names are logical and follow camelCase style.<br><br>The constructor uses the set methods. | |
| **API / JSON** | The JSON file is NOT included in the GitHub repo in the required directory OR the API call does not return a JSON file | The JSON file is provided instead of a functioning API call | A functioning API call is used to receive JSON data and populate the scene(s) | |

| Criteria | Level 0 | Level 1 | Level 2 | Level 3 |
|---|---|---|---|---|
| **Parsing the JSON file** | Code does not exist to parse the JSON file or it triggers an exception. | The JSON file is parsed and objects are created. The objects are lost (only the last object is kept) | Level 1 plus all objects are kept in an array (or other collection) | Level 2 plus the data returned from the JSON file is used to populate at least 2 scenes. |
| **Submission** | A link to your private GitHub account was not submitted. | A private Github link was sent. PriyanshT is listed as a collaborator AND all project files including the build info are present for Intellij. | Level 1 plus there are at least 2 commits per week and the source URL for the data series is included. | Level 2 plus there are a total of over 6 commits with meaningful changes. In other words, do not submit a series of commits at the last minute with updates to comments. I want to see you working on your project over time. |