Large Assignment #2
Due: Friday, March 21, 2025 by 11:59 PM

Objectives.

- Practice working with a partner to design and implement software.
- Practice using Github to collaborate and keep track of code.
- Utilize data structures and library classes provided through Java.
- Design and implement working software according to good design principles from the course.
- Provide strong evidence that the software works as expected through unit tests and running the software.
- Document software and the design process using tools covered in class, including UML diagrams.
- Optional: Practice using AI software for generating working code. *** (NOTE: This is only allowed in part of the assignment, so read the instructions carefully to make sure you do not violate the academic integrity policy.)
- Understand the purpose of the model (M) and the view (V) in the MVC design pattern.
- Organize code into appropriate packages and create an executable jar file.
- Practice using design patterns, inheritance, and interface types.

Project Overview

For this project, you should continue to work with your partner from LA #1 to implement software for managing a music library. If there are issues with this, you should contact me (Lotz) right away to discuss your options. This project builds on LA #1, so you need to have some code to start with. That should be whatever code you had in LA #1.

Part 1. Additional Functionality

Starting with your code from LA 1 (fixing issues as needed), you need to add the following features to the system, while still applying good design practices as covered in class.

A. **Multiple Users.** Add a feature to simulate multiple users. This means that the system needs to support
    a. Creating user accounts with usernames and passwords.
    b. Saving a user's library data so that it can be loaded when they log in again later.
    c. Security measures – you may need to do a little research here, but there are Java library methods that support password *salting* and *cryptographic hashing*, which you should use when storing the passwords.
    d. Simulated username/password database – you can use a text file or a json file to do this.
    e. When you test this, make sure you show that it works with multiple users and that a user's library data gets loaded when they log in again after saving it.

B. **Tracking Song "Plays".**
    a. Implement functionality to simulate the user playing a song. NOTE: This does not mean you actually have to play any music. But if the user selects a song for

playing, that should be different than searching for a song or adding a song to the library.

b. You need to track how often songs are played and maintain lists of
    i. The 10 most recently played songs (in reverse order – most recently played first)
    ii. The 10 most frequently played songs (in order of number of plays – most frequently played first)
    iii. Those lists should be maintained as automatic playlists that the user can then query, and they should be saved so that the data is loaded when the user logs in again later.

C. **Other Functionality.** *Note that these all refer to the user's library*, not the Music Store, which should be fairly static because it is not specific to users but simulates the whole music database. Implement the following additional functionality
  a. Get a list of songs in the library sorted* (in ascending order) by
    i. song title
    ii. artist
    iii. rating
  b. Remove a song or album from the library.
  c. "shuffle" – This should simulate the shuffle functionality for the list of songs in the library. That means that you should
    i. put the songs in random order (using Collections.shuffle)
    ii. implement the ITERATOR design pattern so that a for-each loop can be used to iterate through the song list
    iii. you should also implement this for playlists
  d. Search for a song and get its album information – it should include the whole album but also let the user know if the album is already in the library or not – note that this should be an additional request after the song is searched for – so when the user searches for a song, it should still produce the same output as before, but now the user should be able to request the album information
  e. When songs are added to the user's library, the album should be added as well but it should only list the songs that have been added, not the whole album.
  f. Search for songs by genre. Note that all the albums provided have a genre listed. You can use those as a guide and add others as you see fit.
  g. Implement the following automatic playlists (meaning that the user doesn't create these, the system provides them automatically):
    i. Favorite Songs (all the songs the user has marked as favorite or rated as 5)
    ii. Any genre for which there are at least 10 songs in the library. For example, if the user's library has 15 ROCK songs, 8 COUNTRY songs, and 20 CLASSICAL songs, there should be playlists for ROCK and CLASSICAL, but not COUNTRY.
    iii. Top Rated (all the songs rated as 4 or 5)

*The user should be able to specify how they want the list sorted when they do the query.

NOTE: The rules about AI use are the same as for LA #1. You can use it for generating code in the View, but all of your backend code needs to be your own.

Part 2. Organization, Documentation, & Testing

- You are expected to apply good design principles that are covered in class, including any guidelines about comments. Otherwise, the general criteria is that you provide enough comments to make your code readable and understandable.
- You also must provide a UML diagram (or multiple diagrams as necessary) which cannot be handwritten for the classes in the LibraryModel. This should show the interaction of the classes in the Model using appropriate connectors as covered in class. There are some free online resources for creating nice UML diagrams. Here are a couple:
    - https://www.lucidchart.com/pages/examples/uml_diagram_tool
    - https://app.diagrams.net/
- You also must include unit tests that provide at least 90% coverage for all the code in the backend – the model and the store.
- Finally, you must provide a video in **.mp4** format that is no longer than 20 minutes and includes all of the items listed in Part 3 below.
- Your code should be well-organized into logical packages.
- You should also create an executable jar file that runs the application as well as compressing all the source, test, and resource files.
- You also need a README that explains how to run your jar file from the terminal.

Part 3. The Video

The following is a checklist for what needs to be included in the video with some recommended time estimates for each one. You should include these items *in this order*, and the video needs to be clear and organized, which probably means you will need to edit it. Please note that if you are marked down for not including something in the video and you believe you did include it, you will be required to provide a timestamp for where in the video the item is when you request a regrade. The TAs will NOT watch the entire video again to hunt for the thing you think they missed. Note that these things should cover what is NEW in LA #2. It should not be a repeat of what you should have covered in LA #1.

1. Overview of the code & design (~ 5 minutes).
    a. Describe data structures you used in the Library, specifically for implementing the new functionality.
    b. Describe the design of the Library with respect to the things discussed in class including
        i. Encapsulation
        ii. Input Validation
        iii. Avoidance of Anti-patterns
        iv. Use of Design Patterns
        v. Use of Composition, Inheritance, and/or Interface Types

        vi. How you implemented the security features for saving the password information.

2. Testing & Running Code (~15 minutes).
    a. Show and run the unit tests, making sure to show that they all pass and that each of the classes in the backend has at least 90% coverage.
    b. Run the code and show all the new functionality. Make sure you use test cases that are complex enough and show cases where something searched for does not exist. Here are some examples (these are NOT exhaustive):
        i. Make sure you show that multiple users are supported and that the data for each is saved after the log out and is loaded again when they log back in.
        ii. Make sure you show what happens if someone puts in the wrong username or password.
        iii. Make sure you show the existence (and correctness) of the automatic playlists, as well as the non-existence of playlists that should not exist (i.e. for genres that do not have enough songs).

Part 4. Collaboration Requirements

- You are required to work on Github, and we will be looking at your commit history to see how well each of you are contributing. Each of you is also required to submit (individually) a collaboration report, which is detailed below.
- Although this is not exactly the same as working on a software development team in the real world, you should still be able to apply some of the principles and practices of Agile development. You will be asked about this in the Collaboration report and you should be able to describe at least three principles/practices that you utilized and that we discussed in class.
- **You are NOT allowed to split up the work so that one person is primarily working on the backend and the other is working on the frontend. Every member of the team must be involved in every part of the project and every part of the code.**

Part 5. Collaboration Report

Each of you should submit a PDF individually for this part, answering the following questions:

1. Did you face any particular challenges in the collaboration aspect of this project? What were they and how did you handle them?
2. What are some things you plan to do differently on future collaborative projects?
3. Do you believe both you and your partner deserve the same grade? Explain your answer.
4. Give a general breakdown of how the work was allocated between the two of you.
5. What Agile principles and practices did you utilize during this project? You should mention at least three things we discussed in class for full credit.

Part 6. Grading

Item	**Items to be submitted**	**Criteria**	**Points**
Collaboration	● share the Github repo with your grader so they can see your code and your commit histories ● Collaboration Report (submit this individually!)	● commit history ● collaboration reports ● work distribution ● repo shared with grader	**10**
Working Software	● the video* showing that the code works & provides all the required functionality	● full functionality ● unit test coverage ● well-structured code and unit tests ● user-friendly UI ● code organization	**20**
Documentation & Design Process	● UML diagrams ● video* explaining the design as specified above	● correctness of UML diagram(s) ● good design including ● readable code ● documented use of any AI for the View ● clear explanation of what any AI-generated code does ● clear separation of the front and backend code (model & view) ● README with instructions for running the jar from terminal	**20**
Video	This is one piece of what is graded in the previous categories, so it isn't a separate category, but I'm listing it here to note that it DOES affect your grade, and we will deduct points if it does not meet the criteria listed here.	● organized ● all team members are involved ● includes all the required parts ● thorough ● does not exceed 20 min ● correct format (.mp4) ● appropriate**	

*This should be one video.
**You are not required to show yourself in the video, but if you do, please make sure you are fully clothed.

Note:
There are some things that will earn you an automatic 0 on this assignment. They include:

- **not following instructions – specifically for submission; this includes, but is not limited to not submitting everything that is required, including the video and all of the source code**
- **submitting code that does not compile and run – if it doesn't compile/run, we can't test it!**

See the syllabus for policies regarding resubmissions and regrade requests.

Submission Procedure.

- Put the following items into a zipped folder and submit to D2L – one submission per group.
    - all the source & test code & other required resources, organized into folders and packages
    - the executable jar file
    - the required UML diagram (in a single PDF)
    - the video (as an .mp4)
- Submit the collaboration report individually on Gradescope as a single PDF. Keep it brief – no more than 1 page!