Large Assignment #1
Due: Friday, February 28, 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.

Project Overview
For this project, you are required to work with a partner to implement software for managing a music library. There are three main components to the code, and there are several other non-code requirements as well. Each part has specific instructions and requirements (including varying policies on use of AI, so read and follow all the instructions carefully).

Part 0. Email the following people with the names of the two people who will be working together on this by Friday, February 14, 2025 at 11:59 PM. We need this information in order to set up the groups on D2L, so it is part of your grade!
Email all three of these addresses:
lotz@cs.arizona.edu,
mramoshernandez@arizona.edu
nhealeystewart@arizona.edu

Part 1. `MusicStore.java`

Note: You must code this part yourself without any help from AI or any other unauthorized resources. In other words, the regular academic integrity policy applies for this part of the assignment.

You are allowed to use any library APIs available through Java, but keep in mind that if you are using anything unusually advanced, you may be asked to explain your code.

This part of the code is separate from the main application, but we need to provide a pseudo-database of music to work with. You should implement this in a class called

`MusicStore.java`. The main idea here is to store the music, so that the main application (the user library) can interact with it in an efficient way. This means that it is a great opportunity to use some of the data structures that are provided through Java library code.

You are provided with some text files that include album information. A sample file is given below with some comments to explain the general format that all these files follow. Note that all the albums are named with the same format: `<album title>_<artist>.txt`
For example, the album shown below is in the file named `Old Ideas_Leonard Cohen.txt` .

```
Old Ideas,Leonard Cohen,Singer/Songwriter,2012
Going Home
Amen
Show Me the Place
Darkness
Anyhow
Crazy to Love You
Come Healing
Banjo
Lullaby
Different Sides
```

The first line of the file is the heading, which is in the following format:

```
Album Title,Artist,Genre,Year
```

The rest of the file is a list of the songs on the album. Note that albums have a specific order, so you should make sure to store these in the order that they are listed.

In addition to the files containing individual albums, there is also another file called `albums.txt`, which contains a list of all the album titles and artists in the following format:

```
<Album Title>,<Artist>
```

Your code will need to read each item from the `albums` file, construct each album's file name, and then read in the album information.

To do this, I recommend you check out the following APIs:

- `java.io.BufferedReader` – can help with reading in from a file
- `java.lang.String` – has some very useful methods for splitting Strings

Note: You should read the rest of this document before you start working on Part 1 as your design will depend on the music store's client, which is the LibraryModel described in Part 3.

Part 2. MVC

Before getting into the details of the user library we need to briefly explain what MVC is. MVC stands for Model-View-Controller, which is often considered a design pattern, but may be more appropriately considered a higher-level concept of an overall architecture. Many applications use versions of MVC because it clearly separates the model, the view, and the controller.

For this particular assignment, we will not be using the Controller part because with a text-based UI, it doesn't really do much. We will revisit the idea later when we start talking about GUI's and event-driven programming. For now, we will focus on the Model (M) and the View (V).

Briefly, the Model is where the data is stored and managed. So in this library application, it will store the user's library data and control how the data is accessed. **It does not communicate directly with the user, so there should be no user input and no printing to output anywhere in the model. All code that interacts directly with the user through user input and printing to output should be restricted to the View.**

One way to think about this is to keep in mind that all Model-related code is in the *backend* and all View-related code is in the *frontend.*

The View is the part that the user interacts with. It has two main jobs in this application:

- prompt the user for commands, get those commands, and communicate those commands/requests to the model
- receive the requested data from the model and display it to the user–in this case through a text-based user interface

You are required to implement your project with a clear separation between the model and the view.

Part 3. The Model

You should think of the Model as not just a single class but a collection of classes. However, you should also specifically have a `LibraryModel.java` class that keeps track of the user's library and interacts with other classes including the View and the MusicStore. **But you can (and should) create other classes to model some of the objects that will be used, such as Song, Album, and PlayList.**

Note: You must code this part yourself without any help from AI or any other unauthorized resources. In other words, the regular academic integrity policy applies for this part of the assignment.

You are allowed to use any library APIs available through Java, but keep in mind that if you are using anything unusually advanced, you may be asked to explain your code.

You are expected to use good design practices in the model, and you need to be able to describe and justify the design according to what is covered in class.

Part 4. The View
As indicated in Part 2, the view is simply the user interface, and its only purpose is to interact with the user and communicate with the model. It should not be storing or manipulating any of the data. It simply gets user requests and gets information from the model based on those requests.

There's no denying that artificial intelligence is part of our lives and part of software development now. But it's just a tool, and it is important for us to learn how to use that tool effectively. To that end, you are allowed (but not required) to use generative AI for this part of the code (and this part ONLY). You must document in the comments whenever you use code that was generated by AI. You will also need to include a description in the video of what the generated code does. Keep in mind that AI is just a tool and if used incorrectly can cause more problems than it solves. It is in your best interest to use it to build small, testable pieces of code rather than one large solution all at once. If something seems wrong, you can also ask it to explain itself, which can help identify the issues.

Part 5. Overall Functionality
The following describes the functionality required for this system.
By interacting with the UI, the user should be able to do all of the following:

search for information from the music store
- for a song by title
- for a song by artist
- for an album by title
- for an album by artist

the expected results of searching
- for a song that is in the database: print the song title, the artist, and the album it's on
- for an album: print the album information and a list of the songs in the appropriate order
- for anything that is not in the database: a message indicating that the item is not there
- for anything that has multiple results: print all the results

search for information from the user library
- should cover all the search cases listed for the music store
- should also be able to search for a playlist by name – the result should print the songs (title and artist)

add something to the library
- add a song to the library (as long as it is in the store)
- add a whole album to the library (as long as it is in the store)

get a list of items from the library
- a list of song titles (any order)

- a list of artists (any order)
- a list of albums (any order)
- a list of playlists (any order)
- a list of “favorite” songs

create a playlist and add/remove songs
- playlists should have a name
- songs should be maintained in the order they are added

mark a song as “favorite”

rate a song
- the ratings are 1 to 5
- songs do not have to be rated so there is no default rating
- songs that are rated as 5 should automatically be set to “favorite”

Additional Requirements
- Even though I’m not so concerned about the code design in the View, you do need to think carefully about designing the UI.
- It needs to be user-friendly (as far as is possible with a text-based system), so make sure you test it thoroughly.
- Here are some examples of bad design that you should avoid:
    - requiring long inputs for commands or long input sequences (i.e. don’t make the user verify that their command was in fact what they wanted)
    - not handling invalid input – (i.e. make sure that if something is typed incorrectly, the user can just retype it; do not make it end the program!)

Part 6. 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 7. 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 class diagram (not hand-written!) for each of the classes in the model. 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 following (a checklist is provided below):
    - Show the unit tests running and the coverage for each class in the backend.
    - Run the code and show all the required functionality.
    - Briefly explain what data structures you used in the music store and library.
    - Briefly explain how you maintained good encapsulation in the model classes.
    - If you use AI in the View, briefly explain the generated code.
- 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.

Part 8. 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 9. Video Checklist

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.

1. Overview of the code & design (~5 minutes)
    a. Describe the data structures you used in the backend (both the MusicStore and the Library).
    b. Describe the design of the backend with respect to the principles discussed in class:
        i. use of visibility modifiers

    ii. classes & their design
    iii. avoidance of antipatterns
    iv. avoidance of problematic escaping references
2. Testing & Running the 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 required functionality*, including special cases. See the list of functionality above and make sure you cover it all. Make sure you include complex enough tests (i.e. don't just add one song to the library or one artist). Include tests where searches fail. Include tests where searches return multiple results (e.g. search for Adele to see multiple albums or search for "Lullaby" to get multiple songs with that title).
3. If you used AI to generate the frontend code (the View), provide a brief explanation of what that code does to show that you understand it. (~2 minutes)*

* I realize that these estimates add up to 22 minutes, but these are only estimates. Also, if you have to include #3, you can go up to 22 minutes without deduction.

Part 10. Grading

Item	**Items to be submitted**	**Criteria**	**Points**
Collaboration	● email partner information ● 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	● An executable jar file that can be extracted to get all your source code, tests, etc., and that can also be run directly from the command line. ● the video* showing that the code works & provides all the required functionality	● full functionality ● code can be run with the jar file ● unit test coverage ● well-structured code and unit tests ● user-friendly UI ● code organization	**20**
Documentation & Design	● UML diagrams ● video* explaining the	● correctness of UML diagrams	**20**

Process	design as specified above	● good design including: ○ use of visibility modifiers ○ class design ○ use of data structures and library code ○ no escaping references to mutable fields ○ avoidance of antipatterns ● 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)	
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 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 diagrams (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!