Guidelines for Final Project

Deadlines: Initial Design **(3%):** Due 4/12, Final Project **(7%):** Due 4/26
This is a group project (maximum 4 member teams)

Objective:

Design, develop and implement a working software system in Java. All code must be maintained on GitHub (See instructions on Canvas)

Problem Statement – Movie Theatre

Congrats! You're hired to create a system for a movie theatre. This system will be used by customers to purchase tickets, view movies, and learn about movies. This System will also be used by theatre managers to add new movies and edit the content of movies.

The program will have three perspectives:

- **Customer's perspective**: The customer will start on a screen that prompts them to choose one of five options:
    - **Purchase a ticket**
        - Switch the user to the payment perspective
    - **View movies**
        - Display all of the movies and show times for the current day
        - Display all of the movies that are being shown over the next week
    - **View tickets**
        - Display the user's ticket collection
    - **Enter Manager Mode**
        - If this is selected, the user must be prompted to enter a password.
        - If they get the password right, change to the Manager's perspective
        - Otherwise, stay on the Customer's perspective
        - If they get the password wrong 3 times, terminate the program
    - **Close Application**
        - Terminates the program

- **Payment perspective:** This is the screen for handling payment
    - Where they will be prompted to enter the name of the movie they wish to buy and which type of ticket they would like to buy (child, adult, senior). Then give them the option to add another ticket to the order or proceed to payment.
    - Display the full price of their order (sum of all ticket prices)
    - They will then select a payment type
        - Cash
        - Credit
    - If cash is selected, add the tickets to their tickets collection
    - If Credit is selected, ask them to enter their card information and add the tickets to their ticket collection
    - After payment is made, show the user their receipt and return to the customer's perspective

- **Manager's perspective**: The Manager will start on a screen that prompts them to choose one of four options:
    - **Add movie**
        - Prompt the manager to enter the title, description, ticket price, runtime, rating, and showtimes of the movie
    - **Edit movie**
        - Prompt the manager to enter the title of a movie
            - If the move is not found, ask them to try again
        - Then prompt the manager to edit the title, description, ticket price, runtime, rating, or showtimes of the movie
    - **Enter Customer Mode**
        - No verification needed for this, just switch to customer mode
    - **Close Application**
        - Terminate the program

NOTES:
Showtimes include the dates and times of the movie. So maybe they are inputted in the format…
01/23/2019 [12:00, 14:00, 16:00, 18:00], 01/25/2019 [11:30, 13:30, 15:30], …

There are six types of tickets: Child Matinee, Child Primetime, Adult Matinee, Adult Primetime, Senior Matinee, Senior Primetime

Each movie has one ticket price variable and the price of a ticket type can be calculated from this variable

- Child matinee = (3/4) * ticket_price * (1/2)
- Child primetime = (3/4) * ticket_price
- Adult matinee = ticket_price * (1/2)
- Adult primetime = ticket_price
- Senior matinee = (3/4) * ticket_price * (1/2)
- Senior matinee = (3/4) * ticket_price

You must look at the actual real-life time to determine if you should charge the customer for matinee or primetime.

Special Deals:

- On Wednesday, senior matinee tickets are free
- On Saturdays, kids matinee tickets are buy one get one free
- Everyday, there is a family deal where child tickets (both matinee and primetime) are half off if they are purchase with two adult tickets

If someone tries to purchase a child's ticket for an R-rated movie, you must reject their order

System Features:

The software you are designing is intended to function as if it were actually going to be used by a movie theatre, thus the user experience you create must be realistic. Here are some required functionalities:

- Users should be able to purchase tickets and view all the tickets they have purchased
- In the payment perspective, the user should be able to specify which movie they want to buy tickets for, which type of ticket they want, how many tickets they want, and how they will pay for their ticket. After their purchase is confirmed, they should be shown a receipt that lists each ticket they purchased along with its price and the total price of the order
- Users should be able to view all current and upcoming movies
- The user should be able to switch to manager mode if and only if they can enter the correct password (you can choose the password)
    - If they enter the incorrect password 3 times, terminate the program
- In the manager's perspective, the user should be able to add movies to the schedule and they should be able to edit the title, description, ticket price, runtime, rating, and showtimes of any movie on the schedule
- From the manager's perspective, the user should be able to switch to the customer perspective
- The system should be user friendly, so there should never be any confusion about what a certain action will do. This means you will probably have to print instructions to the user anytime they are expected to give user input
- The system must automatically apply any of the special deals detailed in the "NOTES" section. You are also encouraged and expected to incorporate some of your own special deals. In the case that multiple deals are applicable, you must only apply the deal that gives the customer the best price.
- Make your system user friendly by providing sufficient guidelines and help to use the system (For example: if your system is expecting any input in a specific format, be sure to specify that in the instructions as well as the Testing part of your report)
- **Note:** Your Library system must have **state persistence** by saving the necessary data to files. State persistence allows your system to outlive the process that created it. For example, if you add a few books and/or users (students) to the system when you initially run it and then close the execution before running the system again, the system must be able to remember the previous books and/or users and its state. This will allow you to use the previously entered information without having to create student accounts and table information, etc. again and again. You can complete the entire project by storing data in files or any other persistent storage medium.

You have tremendous amount of freedom in designing this system. You are welcome to add any additional functionality to your library system. Just imagine that you are an actual student/librarian at a school/city.

Note: The design of a GUI is optional. You can also develop a command-line based system.

System Design & Development:

The final project includes two major components: Project Design and Implementation.

Design (3%): The design component will consist of the software design for the project and will include the list of all classes (member variables and functions) to be used, their relationships & collaboration, as well as databases/files. The initial design must precede any implementation. You will start by analyzing the requirements of the project and by identifying the classes required along with attributes and functionalities. A doc/docx document listing the design along with UML diagrams (Class diagrams, activity diagrams, etc.) must be submitted by April 12th (1159pm) on Canvas. Please include the following:

- Skeleton Code with all classes (with member attributes and member functions)
- What are class hierarchies and relationships? (in your report)
- All other data structures or files to be used (in your report)
- Peer evaluations (Groups **and** individual) are REQUIRED
- Also setup a private repository for your source code on github and add all instructors as collaborators (See instructions on Canvas).

It would be beneficial for you to analyze your system thoroughly and provide detailed UML diagrams. Here are some of the questions, answering those will help you put on the good path:

- How many classes will you have and how will they interact?
- How will you store student/librarian ID, passwords, menu options?
- How will you represent the book selection options and the menu?
- How will you deal with adding multiple students to a library system?
- How will the borrowing system be used to handle a transaction?

Implementation (7%): Once you complete the initial design, setup a *private* github repository and start implementing the student, librarian, and library functionalities separately. Test these separately with individual drivers. Be sure to add the appropriate user-friendly menu options that would allow the user to select various actions easily. Ensure that the menu options only work when correct input is provided by the user. Test each option individually to ensure that all available functionality is properly implemented. All sourcecode must be shared amongst team members and instructors via github. Now imagine yourself as the user of the system (as Student, Librarian) and observe the behavior of the system (and see how it changes). Reflect on the efficiency of your choices. Review your initial design and make the necessary changes to remove any unnecessary attributes and functions. You may have to add some more functionality that you might have missed in your initial design. Implement the updated design and report the updates to the initial design along with a discussion regarding the efficiency of your choices. A doc/docx document listing the final design including UML diagrams must be submitted along with working Source Code and video demo by April 26th (1159pm). See the **checklist** given below.

Testing: Develop and build unit tests for each operator. Create different drivers for testing each component of the system as a Student, Librarian separately and then integrate the entire system and test with a separate main driver. Please ensure that the complete system is properly tested before submission.

Deliverables: The following items must be submitted:

- **All source and header files** related to the system implementation (on Canvas as well as github).
- **All drivers** that are part of the testing as Librarian, and Student separated in different folders
- A **README** file with any information regarding compilation and testing I need to know in order to successfully compile and run your system. Include **any other files** needed to compile or test
- **Source code must be properly organized, readable, and must use proper best coding practices.**
- **The report file** with the final design with discussion, and details of testing activities. Feel free to show additional testing you performed in the report
- Video Demo (10 minutes long max) showing all features of your working system
- Peer evaluations (Groups **and** individual) are REQUIRED

Project Checklist:

You can use the following checklist to ensure that you have submitted everything required for the project:

- Did you submit evidence of successful compilation or testing (screen captures)?
- Did you submit the instructions required to compile your code?
- Did you add all instructors as collaborators to your private github repository?
- Did you submit all files including any new header files used for compilation on Canvas (compressed as one file)?
- Did you implement user's features: password, unique ID, Order Entry, etc.?
- Did you submit a report with your designs (and UML diagrams), evidence of successful testing of all features?