Assignment 4: Byte Me!

1 Important Instructions

1. This assignment is a take-home lab assignment. No extensions whatsoever will be provided. Any submission after the deadline will not be evaluated.
2. Only submit one zip file. Naming Convention to be followed **Rollnum-AP-A4.zip**.
3. **Usage of ChatGPT, Claude, Gemini, and other LLMs** will be treated as **plagiarism**.
4. **Copying code, whether in full or partial**, from any other course student, or from online sources **without inclusion of due credit** to the source in your README file, will be treated as **plagiarism**.
5. If you see any ambiguity or inconsistency in a question, please seek clarification from the TAs by **commenting under the GC post** of the Project Deadline only. All doubts will be resolved in the comments, so keep an eye out on the comments for any clarifications the TAs/TF might offer.
6. Doubts on the day of submission will not be entertained. With doubts coming, some parts of the assignment may get updated. So, make sure you regularly follow the GC Post and the comments section.

2 Introduction

You are tasked with enhancing the "Byte Me!" food ordering system developed in Assignment 3 by incorporating new features. These updates should introduce a graphical user interface (GUI), efficient I/O stream handling, and JUnit testing. You are not required to modify existing functionalities but to expand upon them using these advanced concepts. Ensure seamless integration of these additions with the current system while applying the object-oriented programming principles covered in class.

3 Feature Requirements

3.1 GUI Enhancements (15 marks)

- Implement GUI screens that allow:
  - To browse the canteen menu, including item details (name, price, and availability), through an easy-to-navigate interface.
  - To view pending orders, displaying details such as order number, items ordered, and status (e.g., preparing, out for delivery).
  - The GUI will only DISPLAY the menu and pending orders and will not allow updates to their status. Updates and all other operations shall only be done through CLI. There should be at least two pages in the GUI: one for the menu and another for pending orders. Additional pages are optional.
  - First, the method for the CLI shall be run, followed by the method for the GUI during the demo. There will not be parallel execution. Data exchange between the CLI and GUI should be managed through I/O Stream Management to ensure that updates performed via the CLI are reflected in the GUI. The GUI does not modify data but serves solely as a display interface.
- **More details:**
  - Use a GUI library of your choice. You may use:
    - **AWT/Swing**: Create interfaces using JPanel, JFrame, JTable, etc., for simpler and straightforward GUIs.
    - **JavaFX**: You can use components like TableView, ListView, and Scene Builder for a modern look and feel.
  - Include interactive elements such as:
    - Button for navigation (e.g., to switch from menu page to orders page).
    - Tables or lists to display menu items and orders.
    - ~~Options for admins to update order quantity and status.~~
  - Ensure the design is user-friendly and intuitive.

3.2 I/O Stream Management (20 marks)

- Implement any two out of the three file handling features to manage user and order data:
  - **Save order histories for (every) user:** Use file handling to store the order history of (each) user ensuring that the order details (such as food items, quantity, and price) are saved.
  - **Manage users by retrieving their data or registering new users:** Implement functionality to read and write user data from a file. When users log in, their data should be retrieved and when new users register, their details should be appended to the file. The system should handle both existing and new users efficiently.
  - **Temporary Cart Storage:** Use a file to store cart data during a session. The file should be updated in real-time as the user updates the cart.

3.3 JUnit Testing (15 marks)

- Implement JUnit tests to verify the system's behavior in two out of the three specified scenarios:
  - **Ordering out-of-stock items:** Test the system's ability to prevent customers from ordering items that are out of stock. The test should simulate placing an order for an unavailable item and verify that the appropriate error message is displayed and the order is not processed.
  - **Invalid login attempts:** Test the login functionality by simulating invalid login attempts such as entering incorrect usernames or passwords. The test should ensure that the system responds with the correct error message and does not grant access to the user.
  - **Cart Operations**: Test the functionality of updating the cart by simulating the following actions:
    - Adding an item to the cart and verifying the total price is updated accurately.
    - Modifying the quantity of an existing item in the cart and confirming that the total price is recalculated correctly.
    - Attempting to set an item quantity to a negative value and verifying that the system prevents this action.

4 Marking Scheme

- **GUI Implementation:** 15 marks
- **I/O Stream File Management:** 20 marks
- **JUnit Testing:** 15 marks