LAB 211 Assignment

Type:	**Long Assignment**
Code:	**J1.L.P0025**
LOC:	**450**
Slot(s):	**N/A**

Title

Laptop RAM Management.

Background

A computer shop specializes in managing laptop RAM. The shop organizes RAM modules by type (e.g., LPDDR5, DDR5, LPDDR4, DDR4…), and within each type, RAM modules are further categorized by bus speed (e.g., 5600MHz, 4800MHz…). Finally, each bus speed category is sorted by brand. The shop requires a management system to efficiently handle the inventory of RAM modules.

Program Specifications

Build a management program. With the following basic functions

1. Build Your Data Structure:
   - Create data structures to manage RAM items, including attributes such as

     code, type, bus, brand, quantity, and production_month_year.
2. **Add** an Item:
   - Add a new RAM item to the **collection**.
   - Ensure the **code is unique** and **validate all other fields**.
3. Search SubMenu:
   - **Search** items by **type, bus, brand**.
   - Option to return to the main menu.
4. **Update** Item Information:
   - Update existing **RAM** item information **by code**.
   - **Validate** and apply changes to the **specified fields**.
5. **Delete** Item:
   - **Mark** a RAM item as **inactive (active = false)** upon **confirmation**.
   - Ensure **inactive items are not displayed** in lists.
6. Show All Items:
   - **Display all** active RAM items, **sorted by type, bus, and brand**.
7. Store Data to Files:
   - **Save** the list of RAM items **to a file and load data** at program startup.

8. Quit Menu:
   - Provide an option to **safely exit the program**.

Each menu choice should invoke **an appropriate function** to perform the selected menu item. Your program **must display the menu after each task** and **wait for the user to select another option** until the user chooses to quit the program.

Features:

This system contains the following functions:

1. Build Your Data Structure (80 LOC):

- Create data structures to manage RAM items, including attributes such as:
  - **code:** Unique identifier for each RAM item (format: RAMx_y, where x indicates the type and y is a numerical order).
  - **type:** Type of RAM (e.g., LPDDR5, DDR5, LPDDR4, DDR4...).
  - **bus:** Bus speed of the RAM (e.g., 5600MHz, 4800MHz…).
  - **brand:** Brand of the RAM module.
  - **quantity:** Number of RAM modules in stock.
  - **production_month_year:** The production date of the RAM module.
  - **active:** Boolean flag indicating whether the RAM module is active or inactive.
- Implement ***object-oriented principles, including abstraction, encapsulation, polymorphism, and inheritance where applicable***.
- **Define classes** such as RAMItem and RAMManagementSystem.

2. Add an Item (70 LOC):

- **Create a submenu** for adding a new RAM item to the collection.
- Ensure that the **code** is unique and validate all other fields (type, bus, brand, quantity).
- Set the **active** field to true by default.
- After adding, prompt the user to continue adding more modules or return to the main menu.

3. Search SubMenu (60 LOC):

- Implement search functionality for the following:
  - **Search by Type (20 LOC):** Display a list of RAM modules matching the specified type, along with **their code, type, production_month_year, and quantity**.
  - **Search by Bus (20 LOC):** Display a list of RAM modules matching the specified bus speed, along with their code, **bus**, production_month_year, and quantity.
  - **Search by Brand (20 LOC):** Display a list of RAM modules matching the specified brand, along with their code, **brand**, production_month_year, and quantity.
- Provide an option to return to the main menu after performing a search.

4. Update Item Information (50 LOC):

- Allow the user to update existing RAM item information by specifying the **code**.
- Validate the existence of the code and ensure that all updated information **(type, bus, brand, quantity)** is valid.
- If no new information is provided, retain the current data.

5. Delete Item (50 LOC):

- Allow the user to delete a RAM item **by marking it as inactive (active = false).**
- Display a **confirmation message** before proceeding with the deletion.
- Ensure that **inactive items are not displayed in the lists** of RAM modules.

6. Show All Items (30 LOC):

- Display all active RAM items in the collection.
- Sort the items by **type**, **bus**, and **brand** for easy viewing.
- Ensure only items where active = true are shown.

7. Store Data to Files (100 LOC):

- **Save to File:** Store the list of RAM items into a binary file (RAMModules.dat) to preserve data between sessions.
- **Load from File:** Load data from the RAMModules.dat file at the start of the program to ensure continuity.

8. Quit Menu (10 LOC):

- Provide an option for the user to safely exit the program.
- Ensure that all data is saved before the program closes.

Additional Considerations:

- **User Experience:** Ensure the program is **user-friendly**, with clear instructions and error messages.
- **Error Handling: Implement robust error handling** to manage invalid inputs, file I/O errors, and ensure the system is reliable.
- **Extensibility:** Design the system to allow for **easy future enhancements**, such as adding more attributes to RAM modules or expanding the search capabilities.

Program Flow:

- The program should display a menu after each task and wait for the user to select another option until they choose to exit.
- Each menu choice should invoke the corresponding function to perform the selected operation.

Final Notes:

- **Testing:** Ensure the program is **thoroughly tested**, including edge cases like **empty files, duplicate entries, and invalid input formats.**
- **Documentation:** Include **clear comments** and documentation to **explain how the program works** and how it **meets the project requirements**.