Unit 16 - A2 Object-Oriented Programming

I. Design

Producing designs according to the client requirements.

A. Requirements

1. Todo List

This application is meant to demonstrate GUI(Graphical User Interface). Requirements:

- [ ] Creating, Deleting tasks
- [ ] Tracking state, and allowing user to change state(Complete, Incomplete)
- [ ] Support for titles, descriptions, due dates(mutable), completion comments
- [ ] Displaying tasks in a list
- [ ] Filtering tasks by state

2. Index System

This application is meant to demonstrate the use of a database.

Requirements:

- [ ] The program should be able to read a CSV file
- [ ] Generate unique index reference for each item
- [ ] Write the entry to a new CSV file
- [ ] A separate class responsible for allocation of serial numbers as an interface to allow alternative future implementations

B. Design

1. Todo List

a. Problem summary:

The problem is to develop a Todo List application that allows users to create, delete, and manage tasks. The application should support features like tracking the completion status of tasks, setting due dates, and displaying a list of tasks. Users should be able to toggle between displaying all tasks or only incomplete tasks.

b. Complexity:

The complexity of the problem is moderate. It involves managing task data, implementing CRUD (Create, Read, Update, Delete) operations, and providing user-friendly interactions.

c. Constraints:

- Task should have properties like title, description, due date, completion status, and completion comments

- Users should be able to modify task details like: title, description, due date, completion status, and completion comments

d. Intended users:

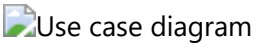
The intended users are people who want to manage their tasks. The application should be easy to use and provide a good user experience.

e. Required Interactivity:

The Todo List application should provide the following interactivity:

- Creating new tasks by entering task details.
- Deleting tasks from the list.
- Tracking the completion status of tasks.
- Modifying task properties such as title, description, due date, and completion status.
- Displaying a list of tasks with filtering options to toggle between all tasks and incomplete tasks.

f. Use case diagram:

 Use case diagram

Data Dictionary - Todo List:

1. Data structures:

- Task:
    - title: string
    - description: string
    - due_date: date
    - completion_status: boolean
    - completion_comments: string
- TaskList: A collection (List) of Task objects

2. UI:

- CreateTask: A button which creates an empty task and adds it to the TaskList
- InputTaskDetails: A form which allows users to enter task details(title, description, due date, completion status, and completion comments)
- DeleteTask: A button which deletes the selected task from the TaskList
- ModifyTask: A button which allows users to modify the selected task

- FilterTasks: A button which allows users to toggle between displaying all tasks and displaying only incomplete tasks

3. Data Storage:

- TaskList: A collection (List) of Task objects stored in memory during runtime
- Implementation of persistance storage is out of scope for this project

2. Index System:

a. Problem summary:

The problem is to develop an Index System application that allows users to generate unique index references for items. The application should support features like reading a CSV file, generating unique index references, and writing the entries to a new CSV file.

b. Complexity:

The complexity of the problem is moderate. It involves reading data from a CSV file, generating unique index references, and writing the indexed data to a new CSV file.

c. Constraints:

- The book details are stored in a CSV file without headings.
- The index references should be unique for each book.

d. Intended users:

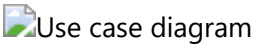
The intended users are the staff or administrators of the college library responsible for managing book indexing.

e. Required Interactivity:

The Index System application should provide the following interactivity:

- Reading book details from a CSV file.
- Generating unique index references for each book.
- Writing the indexed data to a new CSV file.

f. Use case diagram:

 Use case diagram

Data Dictionary - Index System:

1. Data structures:

- Book:
    - Properties:
        - Name: string

- Title: string
- Place published: string
- Publisher: string
- Date of publication: date
- Index reference: string

2. Control structures:

- CSVReader: A class which reads data from a CSV file and returns a list of Book objects
- SerialNumberAllocator: A class which implements an interface to allocate serial numbers
- CSVWriter: A class which writes data to a CSV file

3. Data Storage:

- Input CSV file: A CSV file containing book details(name, title, place published, publisher, date of publication)
- Output CSV file: A CSV file containing book details(name, title, place published, publisher, date of publication, index reference)

4. Pre-defined Code:

- CSV Parsing Library: Utilize a library or built-in functionality for parsing CSV files and extracting book details from the input file and writing book details to the output file.

Algorithm design - Todo List:

Pseudo code:

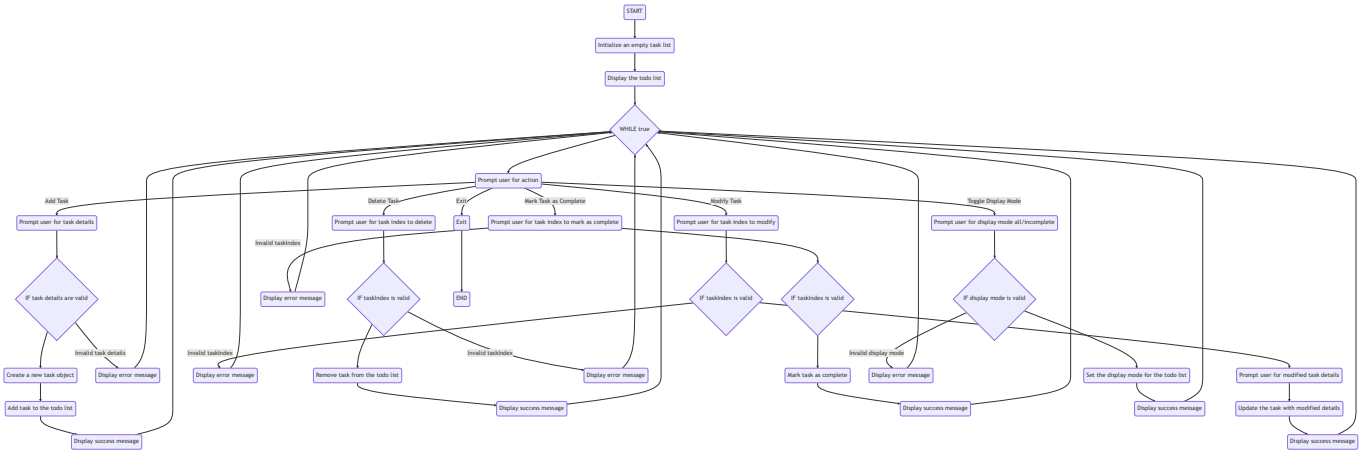
```
START

// Initialize an empty task list
todoList = []

// Display the todo list
DISPLAY_TODO_LIST(todoList)

// User interaction loop
WHILE true
    // Prompt user for action
    action = PROMPT_USER_FOR_ACTION()

    IF action is "Add Task"
        // Prompt user for task details
        taskDetails = PROMPT_USER_FOR_TASK_DETAILS()

        // Create a new task object
        task = CREATE_TASK(taskDetails)

        // Add task to the todo list
        todoList.ADD(task)
```

```
    ELSE IF action is "Modify Task"
        // Prompt user for task index to modify
        taskIndex = PROMPT_USER_FOR_TASK_INDEX(todoList)

        IF taskIndex is valid
            // Prompt user for modified task details
            modifiedTaskDetails =
PROMPT_USER_FOR_MODIFIED_TASK_DETAILS(todoList[taskIndex])

            // Update the task with modified details
            UPDATE_TASK(todoList[taskIndex], modifiedTaskDetails)

    ELSE IF action is "Delete Task"
        // Prompt user for task index to delete
        taskIndex = PROMPT_USER_FOR_TASK_INDEX(todoList)

        IF taskIndex is valid
            // Remove task from the todo list
            todoList.REMOVE(taskIndex)

    ELSE IF action is "Mark Task as Complete"
        // Prompt user for task index to mark as complete
        taskIndex = PROMPT_USER_FOR_TASK_INDEX(todoList)

        IF taskIndex is valid
            // Mark task as complete
            todoList[taskIndex].SET_COMPLETED(true)

    ELSE IF action is "Toggle Display Mode"
        // Prompt user for display mode (all/incomplete)
        displayMode = PROMPT_USER_FOR_DISPLAY_MODE()

        // Set the display mode for the todo list
        todoList.SET_DISPLAY_MODE(displayMode)

    ELSE IF action is "Exit"
        BREAK // Exit the user interaction loop

    // Display the updated todo list
    DISPLAY_TODO_LIST(todoList)

END
```

Flowchart:

```
graph TD

    A(START) --> B(Initialize an empty task list)
    B --> C(Display the todo list)
    C --> D{WHILE true}
    D --> E(Prompt user for action)
```

```
    E --> |Add Task| F(Prompt user for task details)
    F --> G{IF task details are valid}
    G --> H(Create a new task object)
    H --> I(Add task to the todo list)
    I --> J(Display success message)
    J --> D
    G --> |Invalid task details| K(Display error message)
    K --> D
    E --> |Modify Task| L(Prompt user for task index to modify)
    L --> M{IF taskIndex is valid}
    M --> N(Prompt user for modified task details)
    N --> O(Update the task with modified details)
    O --> P(Display success message)
    P --> D
    M --> |Invalid taskIndex| Q(Display error message)
    Q --> D
    E --> |Delete Task| R(Prompt user for task index to delete)
    R --> S{IF taskIndex is valid}
    S --> T(Remove task from the todo list)
    T --> U(Display success message)
    U --> D
    S --> |Invalid taskIndex| V(Display error message)
    V --> D
    E --> |Mark Task as Complete| W(Prompt user for task index to mark as
complete)
    W --> X{IF taskIndex is valid}
    X --> Y(Mark task as complete)
    Y --> Z(Display success message)
    Z --> D
    W --> |Invalid taskIndex| AA(Display error message)
    AA --> D
    E --> |Toggle Display Mode| BB(Prompt user for display mode all/incomplete)
    BB --> CC{IF display mode is valid}
    CC --> DD(Set the display mode for the todo list)
    DD --> EE(Display success message)
    EE --> D
    CC --> |Invalid display mode| FF(Display error message)
    FF --> D
    E --> |Exit| GG(Exit)
    GG --> HH(END)
```


START
Initialize an empty task list
Display the todo list
WHILE true
Prompt user for action
Add Task
Prompt user for task details
IF task details are valid
Invalid task details
Create a new task object
Display error message
Add task to the todo list
Display success message
Delete Task
Prompt user for task index to delete
IF taskIndex is valid
Invalid taskIndex
Remove task from the todo list
Display success message
Invalid taskIndex
Display error message
Exit
END
Mark Task as Complete
Prompt user for task index to mark as complete
IF taskIndex is valid
Invalid taskIndex
Display error message
Display success message
Modify Task
Prompt user for task index to modify
IF taskIndex is valid
Prompt user for modified task details
Update the task with modified details
Display success message
Toggle Display Mode
Prompt user for display mode all/incomplete
IF display mode is valid
Invalid display mode
Display error message
Set the display mode for the todo list
Display success message