Designing Object-Oriented Programs: Solutions for Different Problems

To-Do List Program:

Problem Description

The To-Do List program aims to help users organize their tasks effectively. Users can create tasks with attributes such as title, description, and due date. They can also mark tasks as complete and view their task list. The program should allow users to toggle between displaying all tasks and only incomplete tasks.

Benefits of the To-Do List Program:

1. **Task Management and Organization:** Provides an organized method for managing tasks, helping users track progress, prioritize work, and stay organized.
2. **Increased Productivity:** Offers a platform for storing and tracking tasks, setting deadlines, and marking tasks as completed, boosting efficiency and ensuring critical tasks are not missed.

Intended Users:

- Individuals needing to manage tasks for personal or professional purposes, such as students, professionals, and homemakers.

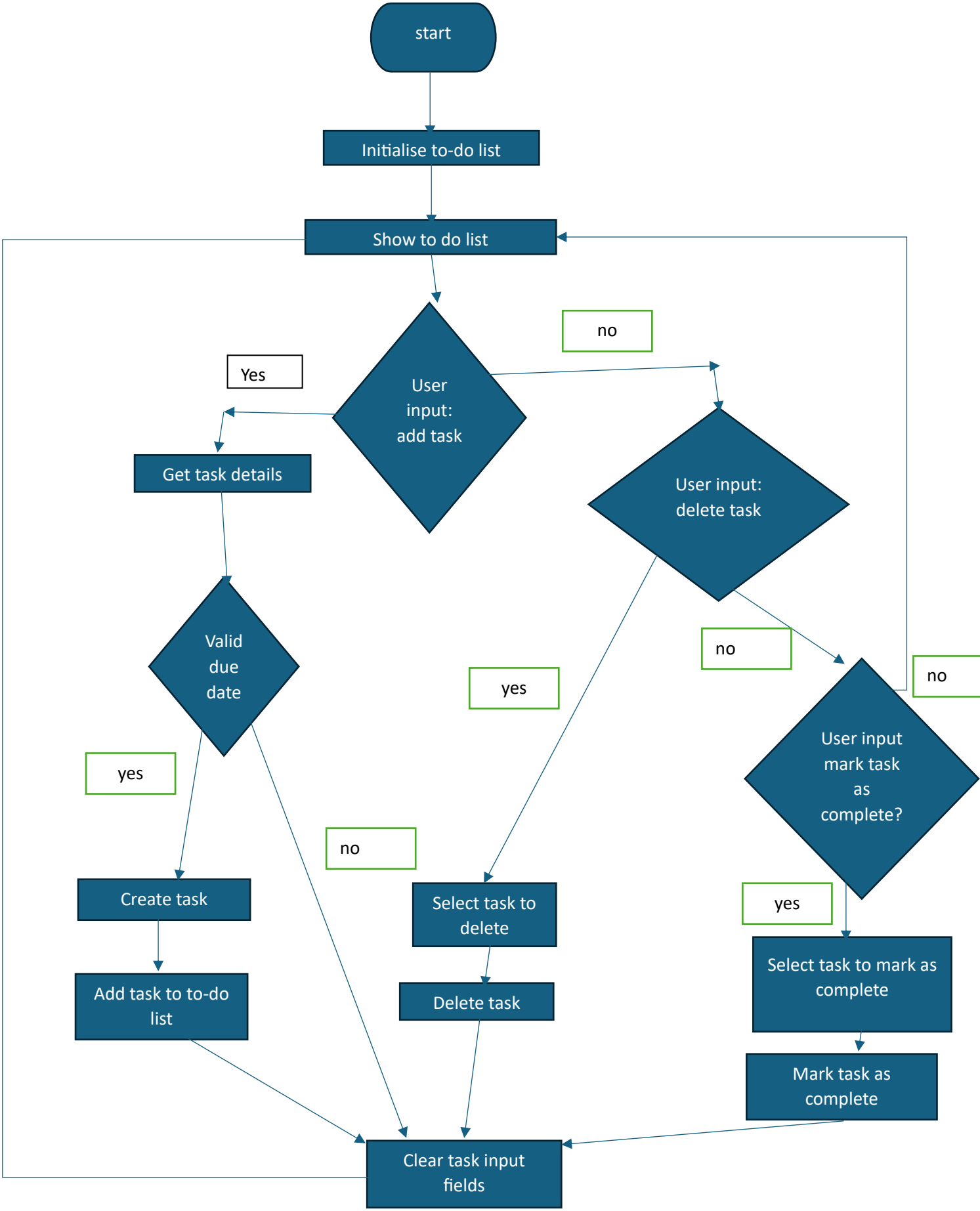
User Needs:

- A central system for planning and categorizing tasks.
- A way to monitor progress and mark tasks as completed.

Constraints:

- Limited time and budget for development.
- Essential time management skills for project completion.

Flowchart:


start
Initialise to-do list
Show to do list
User input: add task
Yes
no
Get task details
Valid due date
yes
no
Create task
Add task to to-do list
User input: delete task
yes
no
Select task to delete
Delete task
User input mark task as complete?
no
yes
Select task to mark as complete
Mark task as complete
Clear task input fields

OOP and GUI solution:

Object-oriented principles:

- **Encapsulation:** The Todo List program demonstrates encapsulation by encapsulating task-related information and actions within the TaskItem class. This class includes getter and setter methods for properties like Title, Description, DueDate, and Completed, which facilitates data abstraction and protects the internal state of the tasks.
- **Inheritance:** Although not explicitly implemented in the current program, inheritance can be used to enhance the functionality of the TaskItem class. Future task types, such as recurring tasks or subtasks, can inherit from the base TaskItem class, acquiring its properties and methods while adding specific functionalities.
- **Polymorphism:** The Todo List program can utilize polymorphism to manage different types of tasks uniformly. By defining a standard interface or base class for tasks, the program can implement specific behaviors for each task type. Functions like AddTask and DeleteTask can accept various task types and perform appropriate actions.
- **Method Overloading/Overriding:** The program can implement method overloading to provide multiple versions of methods with different parameter lists. For example, different versions of the AddTask method can accept various sets of parameters, allowing users to add tasks with different levels of detail. Method overriding can be used when specialized task types derived from the TaskItem class need to override and provide their own implementations for specific methods.

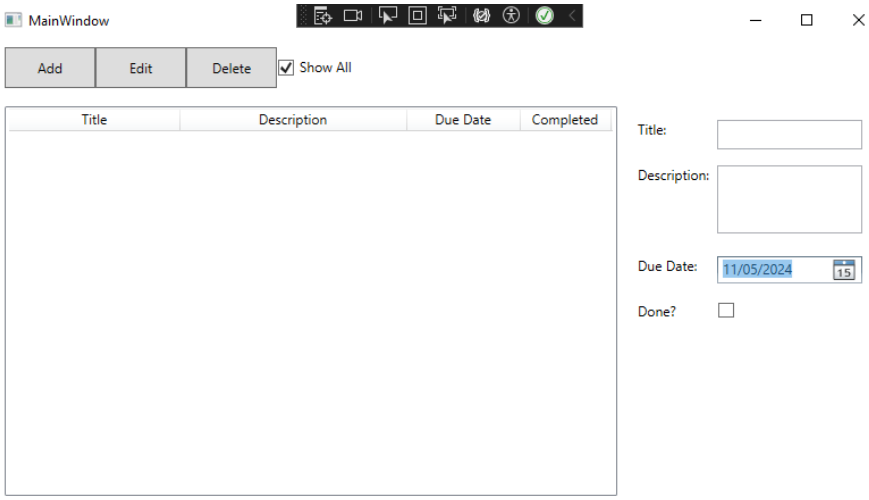
Data types, variables, constants, arithmetic and logical operators, subroutines:

- **Data types:**
    1. **String:** Used to store task titles, descriptions, and other text-based data.
    2. **DateTime:** Used to store task due dates.
- **Variables:**
    - tasks: A collection of TaskItem objects representing tasks.
    - showIncompleteTasks: A boolean indicating whether to show only incomplete tasks.
    - selectedFilter: The selected filter option for task display.
- **Constants:**
    - None explicitly defined in the provided code.
- **Arithmetic and Logical Operators:**
    - No explicit arithmetic or logical operators are used in the given code. However, these operators can be used for calculations and logical comparisons within the program's logic.
- **Subroutines:**
    - AddTask_Click(): Adds a new task to the task list.
    - DeleteTask_Click(): Deletes a task from the task list.
    - MarkAsComplete_Click(): Marks a task as complete.
    - ApplyFilter(): Applies the selected filter option to display tasks.
    - UpdateTaskList(): Updates the task list based on changes made.

Graphical User Interface (GUI):

- The GUI should be user-friendly and intuitive, featuring an organized task list, simple icons, and clear labels. Common operations such as deleting tasks, marking tasks as complete, and updating task details should be easily accessible.

Here is the GUI:



It is basic but it's very easy and clear to use. Everything works but the show all button needs to be ticked to create ones with unique description and title.

College Library Index System

Problem Description

The Library Book Index System assists a college library in organizing its books. It must create distinct index references for each book by reading information from a saved CSV file and adding these references to a new CSV file. The program should also have a class responsible for allocating serial numbers through an interface.

Benefits of the Library Book Index System

1. **Effective Book Organization:** Automates book indexing, reducing manual data entry and errors.
2. **Quick Information Retrieval:** Enables fast location of books using generated index references, streamlining the search process.

3. **Improved Book Tracking and Management:** Simplifies library management by tracking book availability, circulation history, and overdue status.

Intended Users

- Librarians responsible for book purchases, cataloguing, and maintaining precise library stock records.
- Students who will have to read certain books for some of their assignments.

User Needs

- Efficient planning and resource management for timely project completion.
- Automated book detail reading and index reference creation to save time and minimize mistakes.

OOP and GUI solution:

Object-oriented principles:

- **Encapsulation:** The Book and BookWithID classes in the Library Book Index System illustrate encapsulation by containing all information and functions related to books. These classes include properties like Title, Author, Publisher, and PublicationDate, along with getter and setter methods. Encapsulation aids in data hiding and abstraction by ensuring that the internal workings of the book objects are contained within the classes.
- **Inheritance:** The BookWithID class inherits from the base Book class in the Library Book Index System. This inheritance allows the BookWithID class to add an additional ID property for the distinct index reference while inheriting the properties and behavior of the Book class. Inheritance promotes modularity and code reuse, allowing for the specialization of book objects.
- **Polymorphism:** The Library Book Index System can utilize polymorphism when managing various book-related operations. By defining a common interface or base class for books, the program can implement unique behavior for each type of book, treating different book objects uniformly. This flexibility enables the system to handle various book types or apply different indexing strategies effectively.
- **Method Overloading/Overriding:** The system can implement multiple versions of methods with different parameter lists using method overloading. For instance, different versions of the GenerateIDs method might accept various sets of parameters, such as different book collections or methods for generating unique IDs. Method overriding allows derived classes to enhance the functionality of the Book class by providing specific implementations for inherited methods.

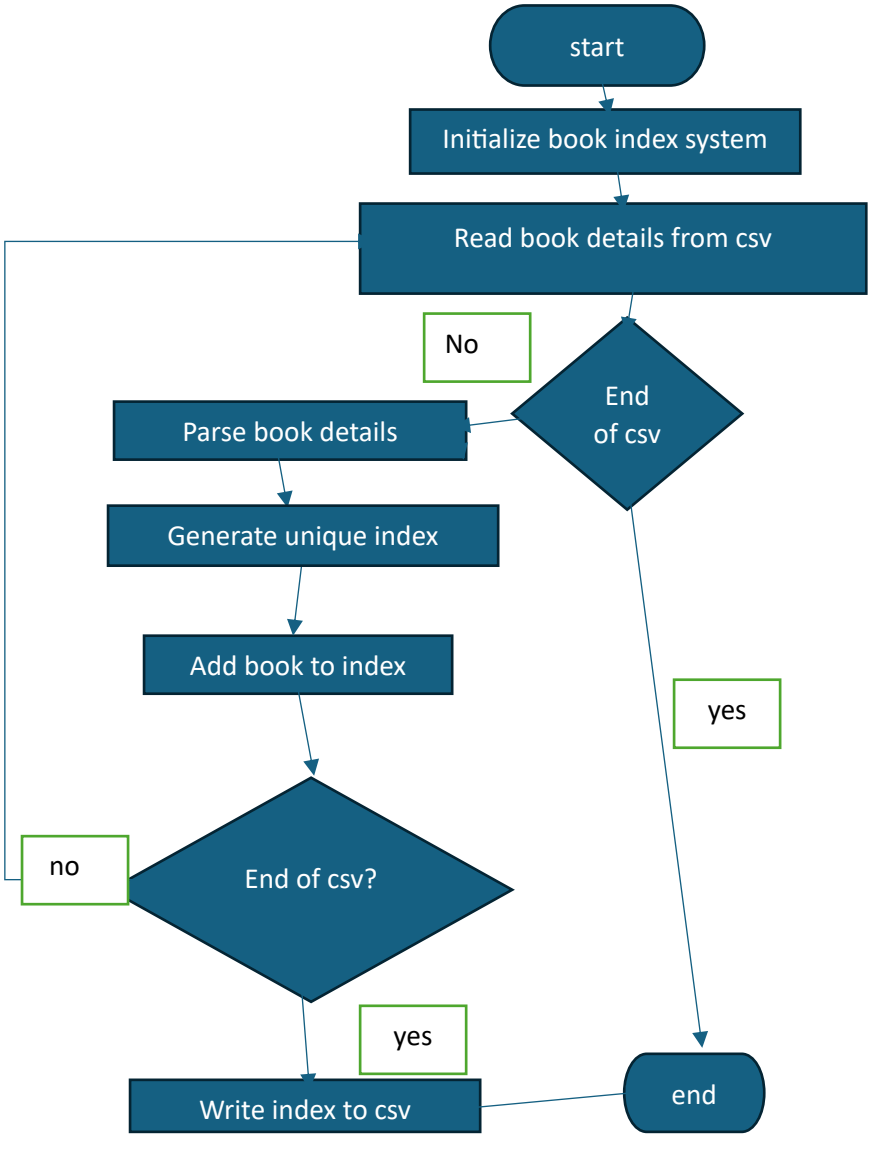
Data types, variables, constants, arithmetic and logical operators, subroutines:

- **Data Types:**
    1. **String:** Used to store book titles, authors, publishers, and other text-based data.

2. **DateTime:** Used to store book publication dates.

- **Variables:**
  - books: A collection of Book objects representing books read from the CSV file.
  - booksWithID: A collection of BookWithID objects representing books with generated index references.
  - idMap: A dictionary mapping books to their generated index references.
- **Constants:**
  - inputFilePath: Stores the constant path to the input CSV file.
  - outputFilePath: Stores the constant path to the output CSV file.
- **Arithmetic and Logical Operators:**
  - While no explicit arithmetic or logical operators are used in the given code, these operators can be utilized for calculations and logical comparisons within the program's logic.
- **Subroutines:**
  - ReadBooksFromCSV(): Reads book details from the input CSV file.
  - GenerateIDs(): Generates unique index references for books.
  - WriteBooksToCSV(): Writes books with index references to the output CSV file.
  - GetCsvConfiguration(): Returns a CsvConfiguration object for CSV file handling.

Flowchart:



Graphical User Interface (GUI)

- The GUI should automate book detail reading and index reference creation, featuring error handling and validation mechanisms. Users should import book data from a CSV file easily.

This is they CSV file the users use to import book data:

	Name	Title	Place publication	Publisher	D
2	Orwell, George	England your England	London	Penguin	20
3	Orwell, George	England your England	London	Penguin	20
4	Orwell, George	The road to Wigan Pier	London	Penguin	20
5	Orwell, George	The road to Wigan Pier	London	Harvill Secker	20
6	Orwell, George	The road to Wigan Pier	London	HarperCollins	20
7	Orwell, George	The road to Wigan Pier	London	HarperCollins	20
8	Orwell, George	The road to Wigan Pier	Oxford	Oxford University Press	20
9	Orwell, George	The road to Wigan Pier	London	Arcturus	20
10	Orwell, George	The road to Wigan Pier	London	HarperCollins	20
11	Orwell, George	The road to Wigan Pier	London	Macmillan	20
12	Orwell, George	The road to Wigan Pier	London	Arcturus Digital	20
13	Orwell, George	The road to Wigan Pier	London	Penguin	20
14	Orwell, George	Orwell and politics : Animal Farm in the context of essays, reviews and letters selected from The complete works of George Orwell	London	Penguin	20
15	Orwell, George	Fascism and democracy	London	Penguin	20
16	Orwell, George	Fascism and democracy	London	Penguin	20
17	Orwell, George	Notes on nationalism	London	Penguin	20
18	Orwell, George	Orwell on freedom	London	Harvill Secker	20

Review of Program Designs and the code itself

To-Do List Program

Feedback:

- **Oliver:** The user interface is intuitive and visually appealing. The program efficiently handles task management actions and provides sorting/filtering options. Visual cues enhance the user experience.
- **Adam:** Code organization is excellent, with robust error handling ensuring data integrity. The user-friendly interface could benefit from task prioritization.

Requirements Fulfilment:

- Creation and deletion of tasks.
- Tracking task completion with visual progress indication.
- Support for task details (title, description, due date).
- Displaying task list with the option to toggle between all tasks and incomplete tasks.

Error Handling and Validation:

- Handles invalid dates and empty titles.
- Confirms task deletion to prevent accidental deletions.
- Validates user input for task details, ensuring data consistency.
- Uses exception handling for unexpected errors, providing user-friendly messages.

Library Book Index System

Feedback:

- **Oliver:** Effectively reads book details from a CSV file and generates unique index references. The search functionality allows quick book location, with clear indicators for book availability and loan status.
- **Adam:** Well-implemented functionality with a user-friendly interface. Notable for quick book information retrieval and efficient book editing.

Requirements Fulfilment:

- Automated book detail retrieval from a CSV file.
- Unique index reference generation for accurate identification and organization.
- Separate class for serial number allocation, supporting alternative implementations.
- Efficient data manipulation and file operations ensuring accurate book indexing and storage.

Error Handling and Validation:

- CSV file read error handling.
- Data integrity validation for imported book details.
- Index reference generation validation for uniqueness and proper formatting.
- Error reporting and exception handling with meaningful messages.

Testing and Results

To-Do List Program

Test Plan:

1. **Creation of Tasks:** Task created with valid details.
2. **Task Deletion:** Selected task removed from the list.
3. **Toggling Task Completion:** Task completion status updated.
4. **Displaying Tasks:** Task list updated based on selected filter.

Results: All tests passed, ensuring the program meets the functional requirements.

Library Book Index System

Test Plan:

1. **Reading Book Details from CSV:** Successfully reads book details.
2. **Generating Unique Index References:** Ensures no duplicate indexes.
3. **Writing Books with Indexes to CSV:** Writes book details with index references to a new CSV file.
4. **Performance and Scalability:** Handles a large dataset without performance degradation.

Results: All tests passed, confirming the program meets the functional requirements.

Conclusion

Both the To-Do List Program and the Library Book Index System are well-designed and meet client requirements. They feature robust error handling, intuitive user interfaces, and efficient functionality, ensuring users can manage tasks and library books effectively. Here are some expanded details on their design and features:

To-Do List Program:

1. **Robust Error Handling:**

   The program includes comprehensive error handling mechanisms to manage various user inputs and system errors. For instance, it validates task details before adding them to the list, ensuring that mandatory fields like Title and DueDate are not left empty.

   It catches and manages exceptions, such as date format errors or null values, providing user-friendly error messages to guide the user in correcting their input.

2. **Intuitive User Interface:**

   The user interface is designed to be clean and straightforward, allowing users to easily add, view, edit, and delete tasks.

   Tasks are displayed in a well-organized list format, with clear labels and indicators for task status (e.g., completed or pending).

   The program includes interactive elements like buttons for adding new tasks, marking tasks as complete, and applying filters to view specific tasks.

3. **Efficient Functionality:**

   Users can quickly add new tasks with all necessary details such as Title, Description, and DueDate.

   The filtering functionality allows users to view tasks based on their completion status or due date, making it easier to prioritize and manage tasks.

   The program supports task modification, enabling users to update task details or mark tasks as complete with minimal effort.

Library Book Index System:

1. **Robust Error Handling:**

The system ensures data integrity by validating book information during data entry and import from CSV files. It checks for missing or incorrect fields, such as ensuring that every book has a Title, Author, and PublicationDate.

It handles file-related errors gracefully, such as issues with reading from or writing to CSV files, providing informative error messages to the user.

2. **Intuitive User Interface:**

   The interface is designed for ease of use, with clear options for importing books, generating unique IDs, and exporting indexed books.

   Books are presented in a structured format, making it easy for users to browse and manage the library inventory.

   The interface includes functionalities to search for specific books, filter the book list based on different criteria, and view detailed information about each book.

3. **Efficient Functionality:**

   The system efficiently reads book data from CSV files, processes it, and generates unique index references for each book, ensuring a well-organized library index.

   It supports exporting the indexed book data back to a CSV file, facilitating easy sharing and backup of the library database.

   The program maintains a mapping of books to their generated index references, allowing quick lookups and efficient management of the library catalogue.

Overall, both the To-Do List Program and the Library Book Index System are designed with user experience in mind, providing reliable and user-friendly tools for managing tasks and library books, respectively. Their robust error handling, intuitive interfaces, and efficient functionality ensure they meet the needs of their users effectively.

Justification

To-Do List Program

Design Decisions and Justifications

1. **Graphical User Interface (GUI)**

Explanation: Implementing a GUI enhances the user experience by making the program more intuitive and visually appealing. Users can easily create, manage, and track tasks through interactive elements such as input fields, buttons, checkboxes, and list views. This visual interaction simplifies task management and improves usability.

Fit for Purpose: A user-friendly GUI ensures that the program is accessible and efficient, meeting the client's need for an easy-to-use task management tool.

2. **ObservableCollection**

Explanation: The use of ObservableCollection allows for dynamic updates to the task list. It automatically notifies the UI when tasks are added, removed, or modified, ensuring the task list is always current. This leads to a responsive application that enhances the user experience.

Fit for Purpose: This choice ensures the program remains responsive and user-friendly, crucial for maintaining an up-to-date task list without requiring manual refreshes.

Meeting Client Needs

1. **Task Management**

Explanation: The program enables users to add, update, and delete tasks, providing a comprehensive task management system. Users can monitor task progress and remove completed or irrelevant tasks, ensuring effective organization.

Fit for Purpose: The ability to manage tasks efficiently meets the client's requirement for a robust task management solution.

2. **Flexibility**

Explanation: Users can assign a title, description, and due date to each task, allowing them to schedule and prioritize effectively. This level of detail caters to diverse user needs and preferences.

Fit for Purpose: Customizable task details make the program adaptable to various user requirements, fulfilling the client's need for detailed task information.

3. **Task Filtering**

Explanation: Users can filter tasks to display only incomplete ones, helping them focus on pending tasks. This functionality enhances productivity by allowing users to concentrate on what needs to be done.

Fit for Purpose: Task filtering improves task management efficiency, addressing the client's need for a streamlined task view.

Specific Functionalities

1. **Task Creation and Deletion**

   Explanation: Users can add tasks with titles, descriptions, and due dates, and remove tasks when they are completed or no longer needed. This facilitates better organization and task management.

   Fit for Purpose: These functionalities ensure users can manage their tasks effectively, enhancing the program's utility.

2. **Task Tracking and Completion**

   Explanation: Users can mark tasks as completed, providing a visual indication of progress. This helps users keep track of their accomplishments and remaining tasks.

   Fit for Purpose: Visual task tracking aids users in monitoring their progress, fulfilling the client's need for a comprehensive task management tool.

3. **Task Display and Filtering**

   Explanation: The program displays tasks with all relevant details and allows users to switch between viewing all tasks and only those that are incomplete. This helps users maintain focus and prioritize tasks.

   Fit for Purpose: Clear task display and filtering options enhance productivity and task management efficiency.

Scalability, Flexibility, and Maintainability

1. **Scalability**

   Explanation: The program can handle an increasing number of tasks without performance degradation, thanks to efficient data structures like ObservableCollection.

   Fit for Purpose: This scalability ensures the program remains effective as the task list grows, meeting future client needs.

2. **Flexibility**

   Explanation: Users can customize task details and filter tasks, adapting the program to their specific needs. This adaptability is crucial for meeting diverse user requirements.

Fit for Purpose: Flexibility in task management and viewing options ensures the program can cater to a wide range of user preferences.

3. **Maintainability**

   Explanation: The program follows object-oriented principles, using classes and objects to promote code reuse and maintainability. Proper documentation and adherence to coding standards make future updates and maintenance easier.

   Fit for Purpose: Maintainable code ensures the program can be easily updated or modified, addressing the client's long-term needs.

Library Book Index System

Design Decisions and Justifications

1. **CSV File Handling**

   Explanation: Using CSV files for storing book details and generating index references ensures compatibility with various systems and simplifies data handling. CSV is a widely supported format that is easy to read and write.

   Fit for Purpose: This design choice ensures the program can integrate smoothly with other applications, meeting the client's need for a simple and reliable data format.

2. **Separate Class for Index Allocation**

   Explanation: Encapsulating index allocation logic in a separate class promotes modularity and extensibility. This design makes it easier to update or change the indexing process without affecting other parts of the program.

   Fit for Purpose: A modular design enhances the program's flexibility and maintainability, fulfilling the client's need for a robust and adaptable indexing system.

Meeting Client Needs

1. **Book Detail Management**

   Explanation: The program reads book information from CSV files, allowing efficient handling of large volumes of data. It maintains organized records of book details such as title, author, publisher, and publication date.

   Fit for Purpose: Efficient book detail management meets the client's requirement for handling and organizing a large volume of book data.

2. **Unique Index Generation**

   Explanation: The program generates unique index references by combining serial numbers with book details. This ensures each book has a distinct identification code, facilitating easy retrieval and referencing.

   Fit for Purpose: Unique index generation ensures proper identification of books, addressing the client's need for a reliable indexing system.

3. **Automated Indexing Process**

   Explanation: Automating the indexing process reduces manual effort and minimizes errors. The program reads book details, generates indexes, and writes the indexed information to a new CSV file.

   Fit for Purpose: Automation streamlines the indexing process, enhancing efficiency and accuracy.

Specific Functionalities

1. **Book Detail Extraction**

   Explanation: The program extracts book details from CSV files, allowing it to process a large number of records efficiently.

   Fit for Purpose: Efficient extraction and processing of book details meet the client's need for handling large volumes of data.

2. **Unique Index Generation**

   Explanation: By combining serial numbers with book details, the program ensures each book has a unique index, simplifying retrieval and management.

   Fit for Purpose: Unique index generation addresses the client's requirement for distinct identification codes.

3. **Automated Indexing Process**

   Explanation: The program automates the process of reading book details, generating unique indexes, and writing indexed information to a new CSV file. This reduces manual effort and errors.

   Fit for Purpose: Automation enhances the efficiency and accuracy of the indexing process, fulfilling the client's needs.

Scalability, Flexibility, and Maintainability

1. **Scalability**

Explanation: The program can handle large CSV files with numerous book records without performance issues. It efficiently generates unique index references for a growing collection of books.

Fit for Purpose: Scalability ensures the program remains effective as the volume of data increases, meeting future client needs.

2. **Flexibility**

Explanation: The modular design allows for easy updates and changes to the index allocation process. Implementing an interface for the index allocation class facilitates alternative implementations.

Fit for Purpose: Flexibility in design ensures the program can adapt to changing requirements, meeting the client's need for an adaptable system.

3. **Maintainability**

Explanation: The program follows modular design principles and coding standards, ensuring code organization and ease of maintenance. Proper exception handling and documentation further enhance maintainability.

Fit for Purpose: Maintainable code ensures the program can be easily updated or modified, addressing the client's long-term needs.