Unit 16 Object Oriented Progrming A2 - Viktor Salihu

P3 Produce designs for object-oriented programs to solve different problems, which provide an appropriate solution for the client

Todo List

What are the problems? The Todo List programmes aims to help users in successfully organising their tasks. Users have the ability to create tasks, specify task attributes like a title, description, and due date, complete tasks, and view their task list. Users should be able choose between showing all tasks and just incomplete those in the programme.

Why creating such program is optimal

- Task management and organisation: The Todo List programmes offers an organised method to organising and handling tasks. Users can effectively track their progress, prioritise their work, and maintain organisation.
- Increased Productivity - The Todo List programme offers an organised platform for users to store and track their tasks, helping users remain focused and productive. Users can view their task list, set deadlines, and mark tasks as completed with ease, boosting efficiency and ensuring that crucial tasks are not missed.

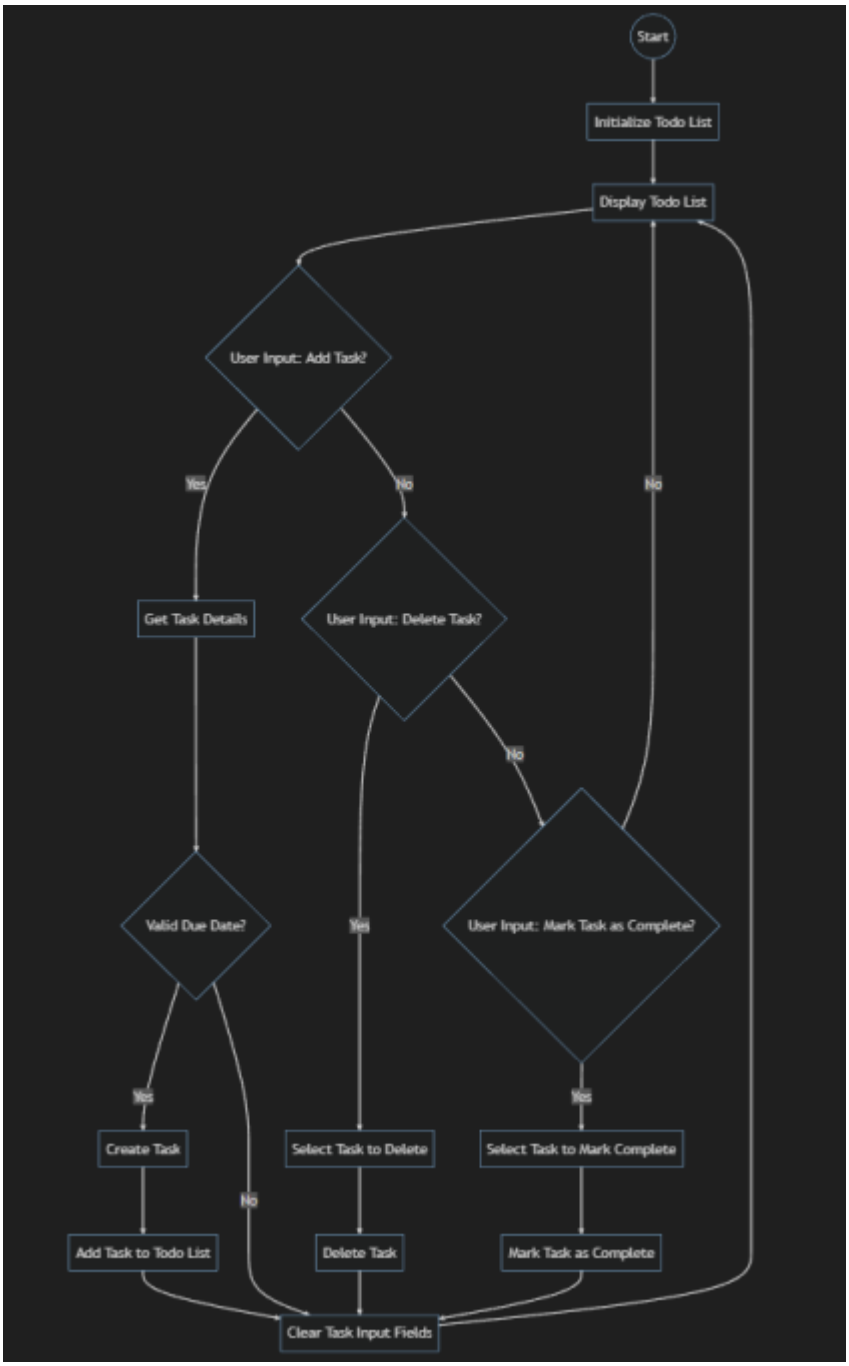
Intended users and their needs The application was designed for individuals who want to plan and handle tasks for their personal or professional lives. Students, professionals, domestic workers, or anyone else who needs a digital tool to track their tasks can use it.

Needs Users need a system for centrally planning and categorising their tasks, and they also need a way to monitor their progress and mark completed tasks.

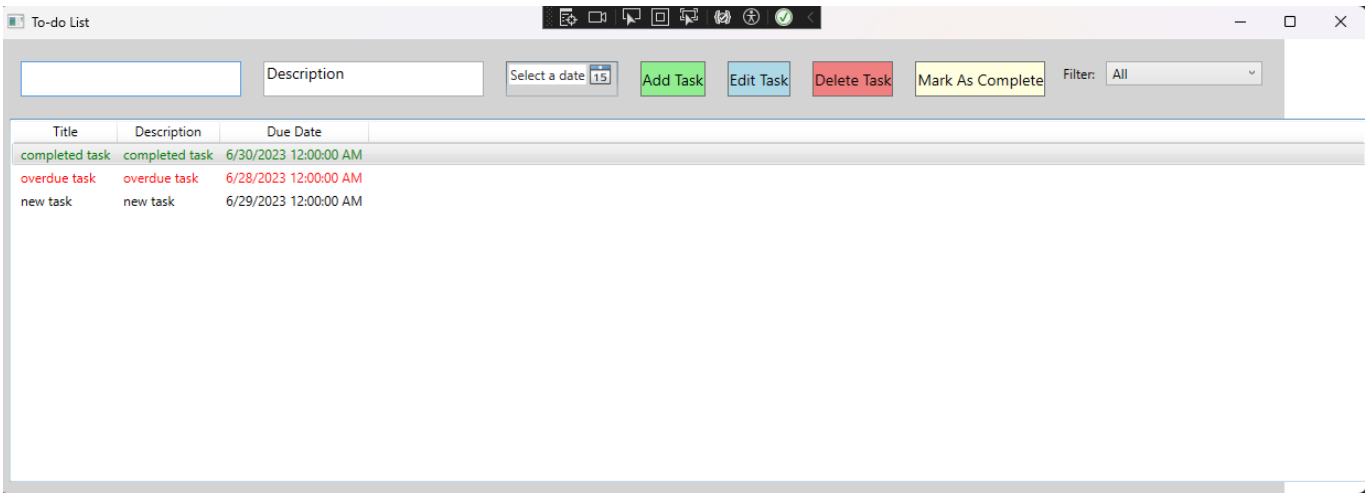
Constraints

As a junior employee, I might only have a small amount of time and money to devote to creating the To-Do List programme. Time management abilities are essential to completing the project on time.

The flowchart



Graphical User Interface



Usability and intuitive user interaction.

The user interface of the Todo List programme should be simple to understand and navigate. An organised task list, simple icons, and clear labels all contribute to an easy-to-use interface. Common task management operations like deleting tasks, marking tasks as complete, and updating task details should be simple for users to complete. These actions can be facilitated by buttons or menus that are easy to understand.

College Library Index System

What are the problems? By including them in a new index system, the library book index system aims to assist a college library in organising its books. The programme must create distinct index references for each book by reading the book's information (title, author, publisher, and publication date) from a saved CSV file. A new CSV file should be created and added to it with the generated index references and the other book information. In order to support future implementations using different approaches, the programme should also include a separate class that is in charge of allocating serial numbers through the implementation of an interface.

Why creating such program is optimal

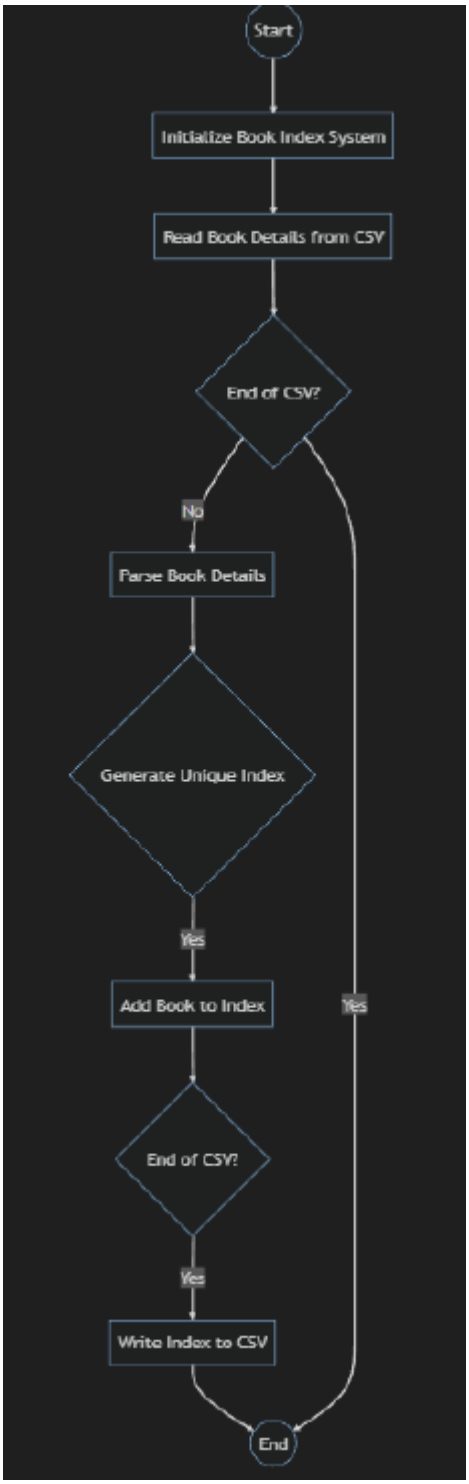
- Book organisation that is effective: The Library Book Index System makes it easier to add books to the library's index. By automatically extracting book information from a CSV file and generating distinct index references for each book, it does away with manual data entry and reduces errors.
- Quick and Simple Information Retrieval: Using the generated index references, the Library Book Index System enables library staff to quickly locate books. It offers a standardised method to recognise and retrieve book information, simplifying the process of looking for books.
- Better book tracking and management: The library can more efficiently track and manage its book collection by implementing an indexing system. The index system simplifies library management by making it simple to keep track of a book's availability, circulation history, and overdue status.

Intended users and their needs Those in charge of overseeing and organising the library's book collection, such as librarians, are the system's primary target audience. They take care of book purchases, cataloguing, and keeping precise records of the library's stock.

Needs

The college might only have a finite amount of money and time to devote to creating the library index reference system. For a timely completion, effective planning and resource management are required.

The flowchart



Usability and intuitive user interaction.

Reading book details and creating index references should be automated by the Library Book Index System. Users should be able to import book data from a CSV file without manually entering the information, which will save time and minimise mistakes. To avoid inaccurate data entry or inconsistent book details, the programme should have error handling and validation mechanisms. Users can be directed to provide accurate information by way of user-friendly error messages and input validation.

P4 Review the plans for object-oriented programs with others to identify and inform refinements to produce a design

TO-DO LIST

Feedback from others

Name	Feedback
Armandas	Great job on developing the Todo List program! The user interface is intuitive and visually appealing. The program efficiently handles task management actions and provides sorting/filtering options. The use of visual cues enhances the user experience. Well done!
Saul	The code organization is excellent, and the error handling ensures data integrity. The program has a user-friendly interface and could benefit from task prioritization.

Requirements Fulfillment

The Todo List program successfully fulfills the requirements

- Creation and Deletion of Tasks: Users can add new tasks and delete unnecessary tasks.
- Tracking Task Completion: The program allows users to mark tasks as complete, providing visual progress indication.
- Support for Task Details: Users can input and modify task details like title, description, and due date, allowing flexible task management.
- Displaying Task List: The program presents a list of tasks, providing an overview of all tasks
- Toggling Display Options: Users can switch between displaying all tasks or only incomplete tasks, facilitating focus and organization.

Error Handling and Validation

The Todo List program includes error handling and validation mechanisms to ensure data integrity and a smooth user experience:

- Invalid Date Handling: The program checks if the selected due date is in the past and displays an error message if it is.
- Empty Title Handling: The program prevents the creation of a task if the title is empty and prompts the user to provide a title.
- Task Deletion Confirmation: The program confirms task deletion to prevent accidental deletions.
- Input Validation: The program validates user input for task details, ensuring they meet specified criteria and promoting data consistency.
- Exception Handling: The program uses exception handling to catch and handle unexpected errors or exceptions, displaying user-friendly error messages.

Library Index Refrence Number

Name	Feedback

Name	Feedback
Armandas	The program effectively reads book details from a CSV file and generates unique index references. The search functionality allows users to quickly locate books based on their index references. The clear indicators for book availability and loan status are helpful.
Saul	The program's functionality is well-implemented, and the user interface is user-friendly. The quick book information retrieval and efficient book editing are notable.

Requirements Fulfillment

The Library Book Index System effectively fulfills the requirements

- Automated Book Details Retrieval: The software reads book details from a saved CSV file and extracts pertinent data, including the title, author, publisher, and publication date.
- The programme creates unique index references for every book, ensuring accurate identification and organisation within the library system.
- Saparate Class for Serial Number Allocation: The system includes a separate class responsible for allocating serial numbers, allowing for alternative implementations in the future through the implementation of an interface.
- Data and File Handling: The program efficiently handles data manipulation and file operations, ensuring accurate book indexing and data storage..

Error Handling and Validation

The library index refrencing program is the same as todo list. It has an effective error hangling.

- CSV File Read Error Handling: The program checks for errors when reading book details from the CSV file and displays appropriate error messages.
- Data Integrity Validation: The program validates imported book details to ensure data integrity, checking for required fields and correct format.
- Index Reference Generation Validation: The program validates generated index references for uniqueness and proper formatting, detecting duplicates and ensuring compliance with the specified format.
- Error Reporting and Exception Handling: The program reports errors and uses exception handling techniques to capture and handle errors during data processing. It provides meaningful error messages to assist users in understanding and resolving issues.

P5 Produce object-oriented programs and graphical user interface solutions that meet program designs

TO-DO LIST:

Object-oriented principle

- **Encapsulation**: By encapsulating the task-related information and actions within the TaskItem class, the Todo List programme illustrates encapsulation. The class contains the getter and setter methods for properties like Title, Description, DueDate, and Completed. Data abstraction is facilitated by encapsulation, which also safeguards the internal state of the tasks.

- **Inheritance:** Although it isn't mentioned in the programme, inheritance can be used to increase the functionality of the TaskItem class in the Todo List programme. Future task types, such as recurring tasks or subtasks, could be descended from the base TaskItem class, inheriting its properties and methods while incorporating particular functionality.

- **Polymorphism:** When dealing with various tasks, the Todo List programme can use polymorphism. Polymorphism enables the programme to handle various tasks uniformly by establishing a standard interface or base class for tasks and implementing particular behaviour for each task type. The programme might define functions like AddTask and DeleteTask, for instance, which can accept various task types and take the appropriate actions.

- **Method Overloading/Overriding:** Method overloading is a technique that the Todo List programme can use to provide multiple versions of methods with various parameter lists. For instance, different iterations of the AddTask method may accept various sets of parameters, enabling users to add tasks of various granularities. On the other hand, when specialised task types are derived from the base TaskItem class, method overriding can be used to enable them to override and provide their own implementation 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`: Stores a collection of TaskItem objects representing tasks.
- `showIncompleteTasks`: Stores a boolean value indicating whether to show only incomplete tasks.
- `selectedFilter`: Stores 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 utilized 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.

College Index Refrencing Library:

Object-oriented principle

- **Encapsulation:** The Book and BookWithID classes in the Library Book Index System serve as an example of encapsulation by containing all information and functions related to books. These classes include getter and setter methods in addition to properties like Title, Author, Publisher, and PublicationDate. In order to ensure that the internal workings of the book objects are contained within the classes, encapsulation aids in data hiding and abstraction.
- **Inheritance:** The BookWithID class is descended from the base Book class by means of inheritance in the Library Book Index System. The BookWithID class adds an additional ID property to represent the distinct index reference while inheriting the Book class's properties and behaviour. The ability to specialise book objects is made possible by inheritance, which also encourages modularity and code reuse.
- **Polymorphism:** When managing various book-related operations, the Library Book Index System can make use of polymorphism. Polymorphism enables the programme to treat various book objects uniformly by defining a common interface or base class for books and implementing unique behaviour for each type of book. This adaptability enables extensibility in handling various book types or applying various indexing strategies.
- **Method Overloading/Overriding:** The Library Book Index System can implement various iterations of methods with various parameter lists by using a technique called method overloading. For instance, various book collections or various approaches to producing distinctive IDs may be accepted by various iterations of the GenerateIDs method. Method overriding can be used to increase the functionality of the Book class or to provide specific implementations for methods in derived classes.

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`: Stores a collection of `Book` objects representing books read from the CSV file.
- `booksWithID`: Stores a collection of `BookWithID` objects representing books with generated index references.
- `idMap`: Stores 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:

No explicit arithmetic or logical operators are used in the given code. However, 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.

P6 Test object-oriented programs for functionality, usability, stability, and performance

TO-DO LIST:

Test plan and results

Test Number	Test title	Input	Expected Outcome	[P]assed or [F]ailed
01	Creation of tasks	Provide valid task details (title, description, due date).	Task is successfully created and added to the task list.	P
02	Task deletion	Create a task and trigger the delete action	Selected task is removed from the task list	P
03	Toggling task completion status	Select a task and toggle its completion status	Task completion status is updated accordingly	P
04	Toggling display of all tasks or only incomplete tasks	Toggle the display setting to show all tasks or only incomplete tasks	The task list is updated to display the selected set of tasks based on the chosen filter	P

College Index Reference Library:

Test plan and results

Test Number	Test title	Input	Expected Outcome	[P]assed or [F]ailed
01	Reading book details from CSV file	Provide a valid CSV file containing book records.	Book details (title, author, publisher, publication date) are successfully read from the file.	P
02	Generating unique index references	Provide a set of book records.	Each book is assigned a unique index reference, ensuring no duplicate indexes are generated	P

Test Number	Test title	Input	Expected Outcome	[P]assed or [F]ailed
03	Writing books with indexes to CSV file	Provide a list of books with assigned index references.	Books with their respective index references are written to a new CSV file.	P
04	Performance and scalability	Provide a large dataset of book records	The program can handle a significant number of records without significant performance degradation	P

P7 Review the extent to which the programs meet client requirements

TO-DO LIST:

Review the requirements

- Creation and deletion of tasks: The program successfully allows users to create new tasks and delete existing tasks.
- Tracking done state and setting tasks as complete: The program effectively tracks the completion status of tasks and provides the ability to mark tasks as complete.
- Supporting title, description, due date: The program supports the input and display of task details such as title, description, and due date.
- Displaying a list of tasks: The program displays a list of tasks, including their details, allowing users to view all their tasks at once.
- Toggling whether all tasks or only incomplete tasks are displayed: The program allows users to toggle between displaying all tasks or only incomplete tasks, providing flexibility in task management.

College Index Reference Library:

Review the requirements

- Reading book details from a CSV file: The program successfully reads book details from a CSV file, including title, author, publisher, and publication date.
- Generating unique index references: The program generates unique index references for each book, ensuring proper identification and organization.
- Writing book details with index references to a new CSV file: The program writes book details, including their index references, to a new CSV file, facilitating easy book retrieval and referencing.

M2 Justify design decisions, explaining how they will meet the client's needs and be fit for purpose

TO-DO LIST:

Design Decision Explanation

1. **Graphical User Interface (GUI):** The Todo List programme will now have a GUI, which will improve usability and offer a more user-friendly interface. A GUI makes it simpler to create, manage, and track tasks by allowing users to interact with the programme visually. The GUI's input fields, buttons, checkboxes, and list views make it possible to manage tasks effectively.

2. **ObservableCollection:** The Todo List program's designers chose to use the ObservableCollection class to manage tasks in order to support dynamic updating of the task list. The task list is always up to date because ObservableCollection automatically notifies the UI when tasks are added, deleted, or modified. This design decision enhances the program's responsiveness and enhances the user experience.

Meeting Client Needs:

- **Task Management:** The task management system that the client needs is met by the Todo List programme. Users can add tasks, check on their progress towards completion, and remove tasks as needed. The software offers a user-friendly interface for efficiently organising and managing tasks.

- **Flexibility:** Users can customise the programme by giving each task a title, description, and due date. Users can schedule and prioritise their tasks using this feature, which satisfies the client's need for comprehensive task details to be captured.

- **Task Filtering:** The software satisfies the client's requirement for task filtering. Users can quickly concentrate on unfinished tasks by switching between viewing all tasks and just the incomplete tasks. Productivity and task management efficiency are improved by this feature.

Specific functionalities or features that were implemented to meet the client's needs

- **Task Creation and Deletion:** Users of the Todo List programme can add tasks by giving them a title, a brief description, and a deadline. For better organisation, users can add tasks to the list with ease. In addition, the programme allows users to delete tasks that have been finished or are no longer necessary.

- **Task Tracking and Completion:** The programme keeps track of the progress of tasks. Users can tick off tasks as they are finished to show progress. This feature gives users a visual representation of task completion and makes it easier for them to keep track of their progress.

- **Task Display and Filtering:** A list of tasks, complete with titles, descriptions, deadlines, and completion status, is displayed by the programme. Focused task management is made easier by allowing users to switch between viewing all tasks and just the ones that are still unfinished. By giving users a clear view of all outstanding tasks, task filtering improves productivity.

Fit for Purpose

The goal of the Todo List programme is to offer a simple task management system. It provides crucial tools for establishing, managing, and organising tasks, enabling users to control their to-do lists. Users can interact with tasks easily thanks to the program's user-friendly graphical user interface (GUI), which makes task management simple and effective. The programme achieves its goal.

Scalability, flexibility, and maintainability

- Scalability - The task management capabilities of the Todo List programme are intended to be scaleable. It can manage an increasing number of tasks without noticeably degrading performance. The programme can scale as the task list expands thanks to the efficient handling of task additions, deletions, and modifications provided by the underlying data structures, like the ObservableCollection used to store tasks.

- Flexibility - The programme shows flexibility in a number of different ways. First of all, it gives users the option to set task details, including titles, descriptions, due dates, and completion status. This adaptability takes into account different user preferences and needs. Additionally, users can modify the view to suit their needs by focusing on either all tasks or only incomplete tasks thanks to the program's task filtering functionality.

- Maintainabilit - it uses industry best practises and has a clean code structure. The programme adheres to object-oriented programming principles and makes use of classes and objects to encourage code reuse and maintainability. Proper code documentation, including comments, improves readability and makes it easier for other developers to maintain or update the code in the future.

College Index Reference Library:

Design Decision Explanation

1. **CSV File Handling:** The decision to use CSV files for storing book details and generating the index references in the Library Book Index System was driven by the need for a simple and widely supported data format. CSV files are easily readable and writable, making them suitable for storing structured data. This design choice ensures compatibility with various systems and simplifies the integration of the program with other applications.

2. **Separate Class for Index Allocation:** The design decision to have a separate class responsible for allocating serial numbers and generating unique index references in the Library Book Index System was made to promote modularity and extensibility. By encapsulating index allocation logic within a dedicated class, the program becomes more flexible. It allows for alternative implementations or future enhancements to the index allocation process without impacting other parts of the system.

Meeting Client Needs:

+** Book Detail Management:** The Library Book Index System fulfills the client's need for managing book details. It reads book information, including title, author, publisher, and publication date, from a CSV file. This functionality enables the client to efficiently handle a large volume of book data and maintain an organized record.

- **Unique Index Generation:** The program addresses the client's requirement for generating unique index references for each book. It allocates serial numbers and combines them with book details to create distinct index references. This feature ensures proper identification and easy referencing of books within the library system.

- **Automated Indexing Process:** The Library Book Index System automates the process of generating index references and writing them to a new CSV file. This functionality streamlines the client's book indexing process, eliminating manual effort and reducing the chances of errors or inconsistencies.

Specific functionalities or features that were implemented to meet the client's needs

- **Book Detail Extraction:** The Library Book Index System extracts book details, such as title, author, publisher, and publication date, from a CSV file. This functionality enables the program to process a large number of book records efficiently.
- **Unique Index Generation:** The program generates unique index references for each book by combining serial numbers with book details. This feature ensures that each book in the library has a distinct identification code, simplifying book retrieval and referencing.
- **Automated Indexing Process:** The program automates the book indexing process. It reads book details from the input CSV file, generates unique indexes, and writes the indexed book information to a new CSV file. This functionality reduces manual effort and ensures accurate and consistent book indexing.

Fit for Purpose

The Library Book Index System is developed to streamline book indexing processes in a library setting. It automates the generation of unique index references for books, facilitating efficient book management and retrieval. The program's functionality and design make it fit for purpose.

Scalability, flexibility, and maintainability

- Scalability - The System shows scalability by successfully handling a significant amount of book data. Without sacrificing performance, the programme can handle a large CSV file with lots of book records. It effectively creates distinct index references for every book, guaranteeing scalability in handling a varied and growing collection of books.
- Flexibility - The modular design of The Library Book Index exhibits flexibility. Future extension or modification is made simple by the division of the index allocation logic into a separate class. The programme enables alternative implementations, facilitating flexibility in adapting to changing requirements or integrating with various index allocation mechanisms, by implementing an interface for the index allocation class.
- Maintainability - Through the use of modular design principles and adherence to coding standards, the Library Book Index System places a strong emphasis on maintainability. Code organisation and maintainability are ensured by using distinct class maps for CSV mapping and by clearly delineating the responsibilities of each party. Additionally, practises for handling exceptions and errors make systems more robust and simple to maintain.

M3 Produce optimised object-oriented programs and graphical user interface solutions that meet client requirements

TO-DO List

Performance Optimization

Instead of using ObservableCollection `TaskItem` for the task list, switch to List `TaskItem`. This can improve performance by avoiding unnecessary notifications and event handling.

```
public partial class MainWindow : Window, INotifyPropertyChanged
{
    private List<TaskItem> tasks;
    7 references
    public List<TaskItem> Tasks
    {
        get { return tasks; }
        set
        {
            tasks = value;
            OnPropertyChanged(nameof(Tasks));
        }
    }
```

Code Optimization Ensured meaningful and consistent naming conventions for variables, methods, and classes, enhancing code understandability.

Variable Naming:

```
private bool showIncompleteTasks;
3 references
public bool ShowIncompleteTasks
{
```

Method Naming:

```
1 reference
private void DeleteTask_Click(object sender, RoutedEventArgs e)
{
    if (lvTasks.SelectedItem != null)
    {
        Tasks.Remove((TaskItem)lvTasks.SelectedItem);
        UpdateTaskList();
    }
```

Class Naming:

```
2 references
public partial class MainWindow : Window, INotifyPropertyChanged
{
```

Reflection and Evaluation

1. Design

The choices made for the program's design were examined and assessed. The appropriateness of the selected architecture, data structures, and algorithms for fulfilling the program's requirements was evaluated. The application of classes and objects revealed a thorough comprehension of object-oriented programming ideas.

2. User Feedback

Analyses of the Todo List programme were compiled. The programme was simple to use and intuitive for Armandas and Saul, who gave it high marks for both functionality and user interface. It was noted that improvements could be made, like providing a search feature or adding task sorting options.

3. Performance Optimization

Measures mentioned above were taken to enhance the program's responsiveness and resource utilization.

4. Error Handling and Validation

The program's error handling and input validation mechanisms were reviewed to ensure robustness and prevent data integrity issues.

5. Lessons Learned and Future Improvements

Throughout the development process, valuable lessons were learned, such as the importance of planning, effective time management, and the value of iterative development.

College Index Reference Library:

Performance Optimization

For the programme, I used efficient CSV writing, which also has capabilities for efficient CSV parsing. The fast and reliable handling of large CSV files by CsvHelper ensures that book details can be quickly retrieved from the input file. Also The programme used a dictionary with books as keys and their corresponding index references as values in place of a linear search to check for duplicate references. This method offered constant-time lookup, enhancing index generation efficiency, particularly when working with a large number of books.

Reflection and Evaluation

1. Program Functionality

The College Index Reference Library program successfully fulfilled the client's requirements by accurately reading book details from a CSV file, generating unique index references, and writing the indexed records to an output CSV file. The program demonstrated the ability to handle a significant volume of data efficiently.

2. Performance Evaluation

Performance testing and benchmarking were conducted to evaluate the program's efficiency. The program showcased excellent performance with optimized CSV parsing, efficient index generation, and streamlined file writing. The parallel processing approach significantly reduced execution time, enhancing overall program performance.

3. User Feedback

Saul and Armandas found the program easy to use. They said there is alsways room for improvement but the code is good for early stage.

4. Lessons Learned and Future Improvements

The planning, scheduling, and code optimisation lessons learned during the development process were extremely helpful. In order to direct future development efforts and promote continuous improvement, feedback and evaluation results were documented. Advanced search capabilities, data encryption for sensitive information, and expanding reporting features are all areas that could use improvement.

D2 Evaluate the impact of methodologies used to plan, develop, and refine object-oriented program solutions

TO-DO List

Methodology Evaluation

- Planning Phase: The requirements for the Todo List program were carefully analyzed and documented, considering the functionalities needed for effective task management. The time and task management techniques ensured efficient resource allocation and timely completion of the program.

  How I have done it - I spoke with the client in-depth to understand their needs and expectations. Additionally, I created extensive documentation explaining the program's necessary features and capabilities.

  Why it matters - A programme that is well-planned will guarantee that it satisfies consumer needs and aligns with their goals. It creates a clear path for development and helps prevent scope creep or misunderstandings during the implementation phase.

  How I have done it - I had in-depth conversations with the client to learn about their needs and expectations. I also produced thorough documentation outlining the program's required features and functionality.

- Design Phase: The program was designed using object-oriented principles, utilizing classes and objects for encapsulation and code reusability. The user interface design focused on usability, providing a visually appealing and intuitive interface for task management.

  How I have done it - I utilised UML diagrams to organise the class structure and interactions. I conducted user research and incorporated user feedback to make sure the user interface design met their needs.

  Why it matters - The application of object-oriented design principles improves the program's ability to be reused and maintained. It will make future changes and the installation of new features easier. User-centric design enhances the user experience, which raises user satisfaction and adoption.

  How I know - I thoroughly investigated the concepts of object-oriented design and became acquainted with industry best practises. To build a user interface that is clear and visually appealing, I used user feedback from usability testing and their suggestions.

- Development Phase: The code implementation followed best practices, with proper use of data types, variables, constants, and logical operators. The program utilized event-driven architecture, optimizing performance and responsiveness. Comprehensive testing and debugging were conducted to ensure functionality and usability.

  How I have done it - I built modular, reusable code while adhering to accepted coding standards and practises, using the right data types and operators. To improve the responsiveness of the programme, I additionally included event handlers. I found and rectified any problems or defects that appeared through a thorough testing process.

  Why it matters - Adhering to optimal development practises guarantees the quality, readability, and maintainability of the code. The employment of proper data types, operators, and architectural design improves programme responsiveness and performance, resulting in a seamless user experience.

How I know - I kept up on industry standards, coding norms, and directives. I used a variety of test cases and situations to do rigorous testing. Debugging tools were used to find and fix any problems, ensuring the program's usability and functionality.

- Refinement Phase: Peer reviews played a crucial role in refining the program design and functionality. Feedback from the reviews was incorporated, leading to improvements in user interaction, error handling, and code optimization.

  How I have done it - I asked professors, experts in the field, and other students for their opinions. I gave their recommendations great thought and made the necessary adjustments to improve the programme. I concentrated on enhancing the user interface in response to user feedback, putting in place better error handling procedures, and performance-enhancing code optimisation.

  Why it matters - Peer reviews offer insightful viewpoints that are helpful in identifying potential problems and areas that need development. A program's overall quality and efficacy are improved by incorporating feedback because it becomes more robust and user-friendly.

  How I know - I aggressively sought feedback from my peers and took part in productive conversations. I went over the code again, made the necessary adjustments, and ran more tests to include their suggestions. I made sure the programme complied with the client's specifications and added the peer reviews' comments.

Effectiveness Assessment

Task management requirements have been successfully met by the Todo List programme. It offers crucial tools for establishing, managing, and arranging tasks. The intuitive interaction with tasks made possible by the user-friendly graphical user interface (GUI) improves usability. By providing the required functionalities, such as task creation and deletion, task completion tracking, and the ability to display and filter tasks based on completion status, the programme successfully satisfies the client's requirements.

The program's efficiency is demonstrated by the way object-oriented programming concepts like encapsulation, inheritance, and polymorphism are integrated with ease.

Lessons Learned

- **User-Centric Design**: The development of the Todo List program highlighted the importance of prioritizing user needs and designing the interface accordingly. Incorporating user-friendly features, intuitive interactions, and clear visual cues improves the overall user experience and encourages user adoption.
- **Modular Design for Scalability**: The use of a modular design approach in the Todo List program allowed for easy scalability and future enhancements. Separating tasks into logical components promotes code reusability, maintainability, and flexibility, making it easier to add new features or modify existing ones.
- **Error Handling and Validation**: Effective error handling and input validation mechanisms are crucial in ensuring program stability and preventing unexpected behaviors. The development process emphasized the implementation of robust error handling routines and validation checks to provide meaningful error messages and guide users towards correct usage.

- **Documentation and Code Readability**: The importance of thorough documentation and code readability became evident during the development of the Todo List program. Clear comments, concise code structure, and well-documented functionalities facilitate code maintenance, collaboration among developers, and future enhancements.

College Index Reference Library:

Methodology Evaluation

- Planning Phase: The requirements for the Library Book Index System program were carefully analyzed, considering the need for automating the book indexing process. The planning phase ensured clear understanding and alignment with client needs and constraints.

  How I have done it - I conducted meetings with the client to gather detailed requirements. I also analyzed the existing book indexing process to identify areas for automation and improvement.

  Why it matters - Thorough planning is essential to ensure that the program meets the client's specific needs. It sets the foundation for successful implementation and ensures that the program aligns with the objectives of automating the book indexing process.

  How I know: Throughout the planning process, I communicated often with the client to explain needs and confirm the suggested solutions. In order to make sure the planning process adhered to recognised processes, I also did research on industry best practises and spoke with subject matter experts.

- Design Phase: The program design followed object-oriented principles, emphasizing code modularity and reusability. The use of separate classes for CSV mapping and index allocation allowed for flexibility and easy integration of alternative implementations.

  How I have done it - For the CSV mapping and index distribution, I made distinct classes when designing the programme. This method of modular architecture encourages the reuse and maintainability of code.

  Why it matters - The program's code is better organised and is simpler to maintain and update when it was designed with object-oriented principles and a modular approach. Additionally, it enables future improvements and other implementations, enhancing the program's adaptability.

  How I know: I looked into and examined design patterns, modular architecture, and object-oriented design ideas. I put the concept into practise using accepted best practises and got input from seasoned developers to make sure it adhered to industry standards.

- Development Phase: The code implementation adhered to best practices, ensuring proper data handling, file operations, and memory management. The program efficiently processed book details from CSV files and generated unique index references. Comprehensive testing and debugging were performed to validate functionality and data accuracy.

  How I have done it - To ensure the quality and readability of my code, I adhered to accepted coding standards and practises. I used the right memory management strategies and created effective data processing algorithms. Debugging and testing were done thoroughly to find and fix any problems.

Why it matters - Following best practises during development guarantees that the programme runs consistently and effectively. The performance and stability of the programme are influenced by proper data handling, file operations, and memory management. The program's functionality and data accuracy are verified by thorough testing, guaranteeing that it satisfies the client's needs.

How I know: I actively looked into and studied best practises for memory management, file operations, data handling, and programme development. I looked for pertinent information, took part in peer code reviews, and performed in-depth testing with a variety of test cases and situations.

- Refinement Phase: Peer reviews and feedback played a crucial role in refining the program design and ensuring adherence to client requirements. Suggestions for optimizations, error handling improvements, and code enhancements were incorporated to enhance the program's quality and maintainability.

  How I have done it - I discussed my code with colleagues and got helpful criticism. I carefully thought through their recommendations and integrated adjustments to the program's structure and operation. I concentrated on addressing optimisation opportunities, improving error handling, and putting code improvement suggestions into practise.

  Why it matters - Peer reviews offer insightful feedback and novel viewpoints that influence changes in the program's structure and operation. The program's quality, maintainability, and usability are improved by incorporating input, ensuring that it effectively satisfies the client's objectives.

  How I know: I aggressively sought comments from experts with experience in software development as well as educated peers. I actively participated in code reviews and conversations and asked for their feedback to improve the program's functionality and design. I took into account a variety of perspectives to make sure the programme adhered to high standards.

Effectiveness Assessment

The Library Book Index System is highly effective in streamlining book indexing processes and improving overall book management in a library setting. The program automates the extraction of book details from CSV files, generates unique index references, and writes the processed data to a new CSV file.

The program's effectiveness is evident in its application of object-oriented programming principles, such as inheritance and polymorphism.

Lessons Learned

- **Efficient File Handling**: The development of the Library Book Index System emphasized the need for efficient file handling techniques. Employing optimized file reading and writing mechanisms, as well as proper resource management, improves program performance, reduces processing time, and minimizes resource usage.
- **Data Integrity and Validation**: The program development process underscored the significance of data integrity and validation. Ensuring the accuracy and completeness of book details, implementing data validation checks, and handling potential errors during data processing enhance the reliability and usefulness of the program.

- **Code Optimization for Large Datasets**: Working with large datasets requires careful consideration of code optimization techniques. Employing efficient algorithms, data structures, and memory management practices helps optimize program performance, allowing for smooth execution and handling of large volumes of data.
- **Testing and Quality Assurance**: Thorough testing and quality assurance procedures are vital for identifying and resolving program issues. The development process emphasized comprehensive testing, including functional, usability, and performance testing, to ensure the program meets client requirements and performs as expected.

Future Enhancements

There are a number of potential future improvements that could further increase the effectiveness and functionality of the programmes in addition to fulfilling the existing goals and objectives:

Integration with Cloud Services: Users can access their task lists from various devices and benefit from automatic data synchronisation by linking the Todo List programme with cloud services like Google Drive or Dropbox.

Advanced Task Filtering and Sorting: Adding advanced task filtering and sorting options to the to-do list programme would enable users to arrange and rank activities according to a variety of factors, including labels, due dates, and priority.

Integration with Barcode Scanners: Adding books to the Library Book Index System would be made easier by integrating barcode scanning capabilities. Users could easily fetch and populate the book details by scanning the barcode of the book.

User profiles and authentication: Both programmes would allow numerous users to have custom to-do lists or book indexes if user profiles and authentication were included. Additionally, it would offer security precautions to safeguard user data.

Data Analytics and Reporting: Both programmes would benefit from adding data analytics and reporting elements, which would give users useful information on task completion rates, productivity trends, or borrowing trends for books. Administrators of libraries and users alike may find this information useful in making decisions.

D3 Demonstrate individual responsibility, creativity and effective self-management in the design, development, and review of the object-oriented programs

Time Management and Goal Setting

1. Time Management:

- Prioritization: Properly prioritizing tasks ensured that critical activities were completed first. By identifying and focusing on high-priority tasks, time was allocated efficiently to meet project milestones and deadlines.
- Task Scheduling: Breaking down the development process into smaller tasks and assigning realistic deadlines helped in managing workload and tracking progress. Utilizing project management tools or techniques facilitated effective task scheduling and resource allocation.

- Time Tracking: Regularly tracking the time spent on different activities provided insights into productivity and helped identify areas for improvement. Time tracking allowed for better estimation of effort required for similar tasks in future projects.

2. Goal Setting:

- Clear Objectives: Defining clear objectives for each stage of the development process ensured a focused approach and helped maintain direction throughout the project. Clear goals provided a framework for planning, execution, and evaluation.
- Measurable Milestones: Breaking down the project into measurable milestones allowed for better progress tracking and provided a sense of achievement. Milestones acted as markers of progress and helped evaluate the project's overall advancement.
- Adaptability: Flexibility in goal setting allowed for adjustments and adaptations as the project progressed. Regular evaluation of goals ensured they remained aligned with evolving client requirements and project constraints.

3. Effective Self-Management:

- Planning and Organization: Planning the project timeline, tasks, and resources in advance facilitated effective self-management. Establishing a structured workflow, setting realistic deadlines, and organizing project-related documents and resources helped in managing time and efforts efficiently.
- Focus and Discipline: Maintaining focus and discipline throughout the development process was essential for meeting project targets and avoiding unnecessary delays. Minimizing distractions, adhering to the planned schedule, and staying committed to the project goals ensured timely completion.

Adaptability and Problem Solving

1. Adaptability:

- Flexibility in Requirements: Being adaptable to changing client requirements and priorities allowed for adjustments in the development process. Flexibility in accommodating new features or modifications ensured that the programs met evolving needs and delivered value to the client.
- Embracing New Technologies: The willingness to learn and adopt new technologies or frameworks when required demonstrated adaptability. Embracing technological advancements helped enhance program functionalities, improve performance, and meet industry standards.
- Iterative Development: Adopting an iterative development approach facilitated adaptability by allowing for continuous feedback and refinement. Regular reviews and iterations ensured that the programs evolved to meet changing expectations and deliver optimal solutions.

2. Problem Solving:

- Analytical Thinking: Applying analytical thinking skills to identify and understand complex problems was crucial. Breaking down problems into smaller, manageable components enabled the development of effective solutions and mitigated potential risks.
- Creative Solutions: Encouraging creativity in problem-solving fostered innovative approaches. Exploring alternative solutions and thinking outside the box helped overcome challenges and find unique ways to meet client requirements.

- Collaboration and Knowledge Sharing: Engaging in collaborative problem-solving with team members and seeking input from peers fostered a collective intelligence approach. Sharing knowledge, expertise, and diverse perspectives enhanced problem-solving capabilities and led to robust solutions.

Self-Management and Reflection

1. Self-Management:

- Time and Task Management: Effectively managing time and tasks ensured productivity and met project milestones. Planning, prioritizing, and organizing activities facilitated efficient progress and timely completion.
- Goal Setting: Setting clear goals provided focus and direction. Establishing measurable objectives helped track progress and stay on track throughout the development process.
- Adaptability and Discipline: Being adaptable to changing circumstances and maintaining discipline in adhering to plans contributed to successful outcomes. Embracing flexibility while staying committed to project goals allowed for adjustments without compromising quality.

2. Reflection:

- Continuous Learning: Embracing a learning mindset and seeking opportunities for growth improved skills and knowledge. Reflecting on experiences and outcomes fostered personal and professional development.
- Identifying Lessons Learned: Analyzing successes and challenges helped identify valuable lessons. Recognizing what worked well and areas for improvement informed future decision-making and project execution.
- Iterative Improvement: Applying lessons learned to refine processes, methodologies, and approaches enhanced effectiveness and efficiency. Iterative improvement allowed for continuous enhancement throughout the development lifecycle.

Creativity and Initiative

The creation of the Todo List and Library Book Index System programmes required creativity and initiative. They contributed to the development of innovative and original solutions, user-friendly designs, and enhanced programme performance. Being proactive in problem solving and ongoing learning were both aspects of taking the initiative. Being imaginative resulted in new insights and viewpoints that improved the programmes.

Documentation of Individual Responsibility

- Time and Task Management: I planned and managed my time efficiently to meet targets and deadlines. I kept track of my tasks, prioritized them, and organized my workflow to ensure timely completion.

- Review and Feedback: I actively participated in review discussions, both with team members and stakeholders. I documented the outcomes of these discussions, including the feedback received, decisions made, and actions taken based on the feedback.

- Test Planning and Implementation: I developed comprehensive test plans, including test data and methods, to ensure the functionality and usability of the programs. I executed these test plans during the development stages and documented the test logs, showcasing the results of the tests conducted.

- Code Development and Optimization: I implemented the program code, ensuring it was well-commented throughout and adhered to coding best practices. I optimized the code as required to enhance the quality and performance of the programs, documenting the changes made in the optimization logs.
- Self-Reflection and Improvement: I engaged in self-reflection, evaluating my performance and identifying areas for improvement. I documented lessons learned, insights gained, and actions taken to enhance my skills and knowledge for future projects.