Hanoi University of Science and Technology
The School of Information and Communication Technology

IT3100E-147839
Object-oriented Programming

Mini-Project
Demonstration of basic operations on List, Stack and Queue

[Group 21]
Tran Huu Dao 20220061
Dam Quang Duc 20225483
Vu Huu An 20225467

Ha Noi, June 2024

I. Mini-project description

In the world of computer science, List, Stack, and Queue are three foundational and widely used types of data structures. These structures provide essential tools for organizing and manipulating data efficiently. Our group has developed an application specifically designed to showcase the fundamental features of these data structures: creating, inserting, sorting, finding, and deleting elements.

II. Requirement

The main menu of our application showcases three buttons together with the general description of the three fundamental data structures: List, Stack, and Queue. By selecting any of these buttons, users can create a data structure of their chosen type and proceed to explore its basic operations, including insertion, sorting, finding, and deletion.

For assistance, a Help menu is available, accessible by clicking the Help button. User can also close the application at any time by using the Exit button.

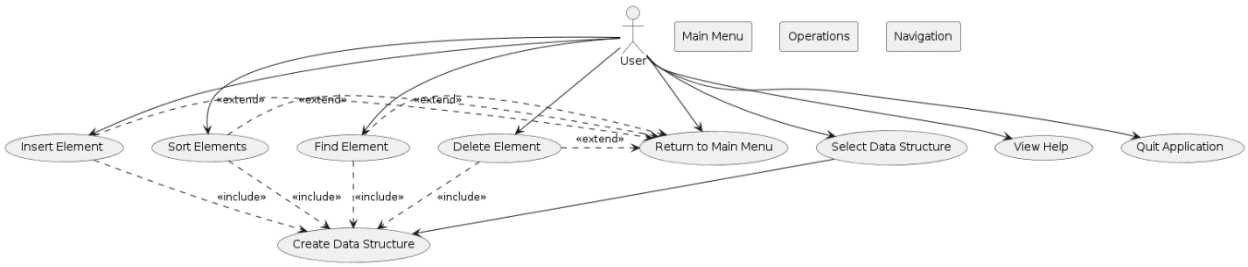
The requirements of the project can be understood through the analysis of the use case diagram provided.

Use Case Diagram Analysis

- Actors:

    - User: Interacts with the application to perform various operations on data structures.
- Use Cases:
    - Insert Element: Allows the user to add an element to a data structure.
    - Sort Elements: Enables the user to sort elements in a list.
    - Find Element: Provides functionality to search for an element within the data structure.
    - Delete Element: Allows the user to remove an element from a data structure.
    - Create Data Structure: The initial step where the user creates the desired data structure (List, Stack, or Queue).
    - Return to Main Menu: Provides navigation back to the main menu.
    - Select Data Structure: The user can choose the type of data structure to work with.
    - View Help: Displays help information about using the application.
    - Quit Application: Allows the user to exit the application.
- Relationships:
    - Include: Indicates mandatory inclusion of use cases like creating a data structure before performing operations like insert, delete, etc.

- Extend: Shows optional use cases that extend the basic operations, such as sorting elements.



Use case diagram

III. Design

We structure our application into two main packages: datastructure and GUI.

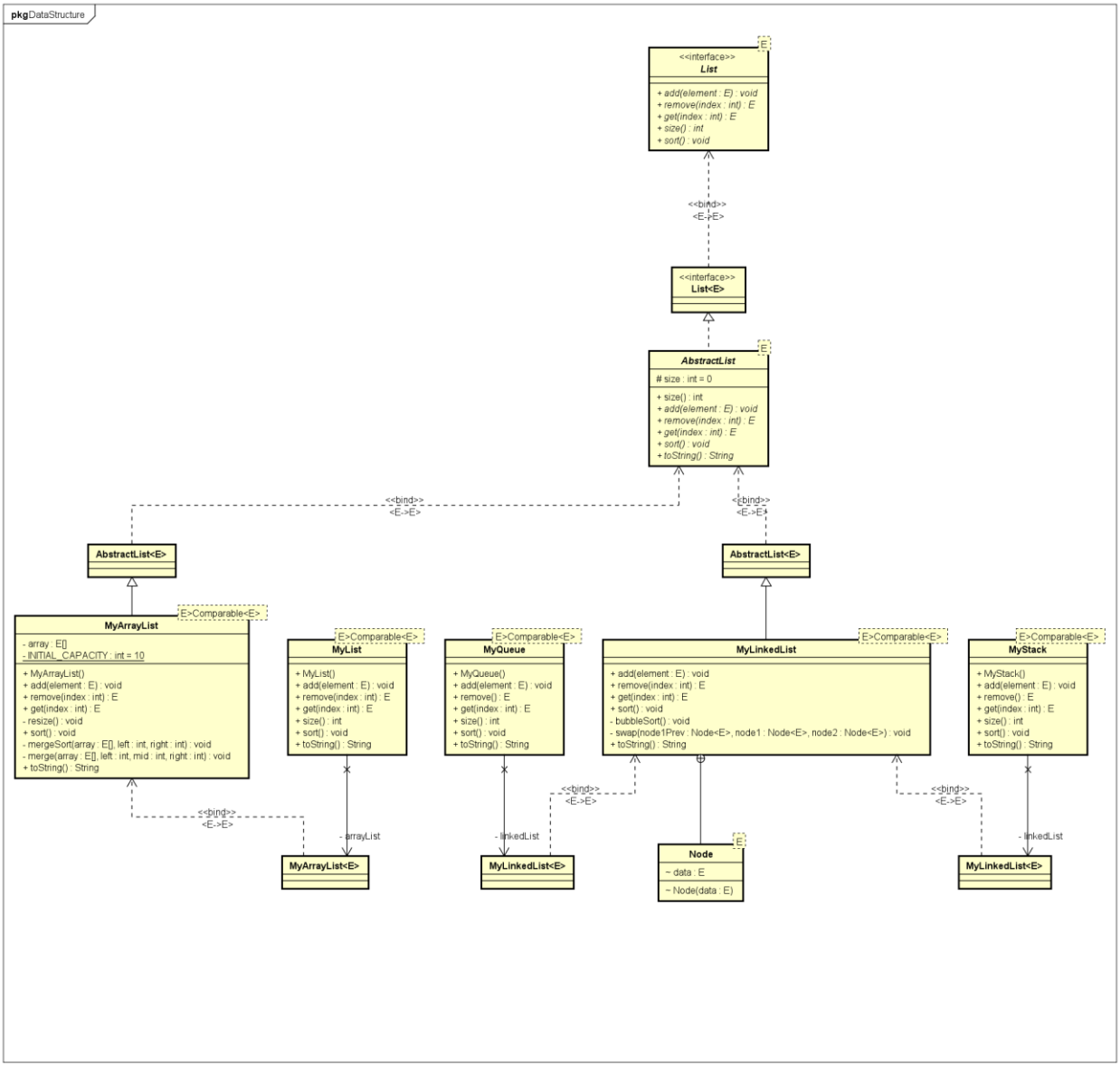
1. Package datastructure

This package contains classes and interfaces representing different data structures and their operations.

Class Diagram Analysis

1. Interface List<E extends Comparable<E>>
    - Methods: void insert(E element), void sort(), int find(E element), void delete(E element), E get(int index), void set(int index, E element)
2. Abstract Class AbstractList<E extends Comparable<E>>
    - Implements the List interface with abstract methods that need to be overridden by concrete classes.

3. Class MyLinkedList<E extends Comparable<E>>
    - Extends AbstractList and implements basic linked list operations using bubble sort.
4. Class MyList<E extends Comparable<E>>
    - Extends AbstractList and implements basic list operations using merge sort.
5. Class MyQueue<E extends Comparable<E>>
    - Extends MyLinkedList and provides queue-specific operations such as enqueue and dequeue.
6. Class MyStack<E extends Comparable<E>>
    - Extends MyLinkedList and provides stack-specific operations such as push and pop.



Datastructure package Class Diagram

2. Package GUI

This package handles the graphical user interface, providing a visual representation of the data structures and their operations. Key classes include:

1. AnimationController

- Responsible for animating the operations on data structures.
- Methods: animateLinkedList, enqueue, dequeue, push, pop

2. MainGUI
    - Initializes and manages the main user interface.
    - Contains two main parts: demonstration and animation.
    - Methods: initializeDemonstrationLayout, showQueueOperation, showStackOperation
    - Enhanced with additional methods: find, sort, delete, add, get, set

3. Relationship between two packages

Dependency: The GUI package depends on the datastructure package to perform operations on data structures. The AnimationController class interacts with data structure classes to visualize operations like insertion, deletion, etc.

IV. Implementation

1. Encapsulation

- Encapsulation is achieved by defining private attributes and providing public methods to access and modify these attributes. For instance, the MyLinkedList class encapsulates its elements and provides methods for insertion, deletion, and other operations.

2. Abstraction

- Abstraction is implemented by defining abstract data structures and operations that can be performed on them. The AbstractList class serves as an abstract base class for more specific structures like MyLinkedList and MyList.

3. Inheritance

- Inheritance is used to create specialized data structures. The MyQueue and MyStack classes inherit from the MyLinkedList class, extending its functionality with queue-specific and stack-specific methods.

4. Polymorphism

- Polymorphism is demonstrated through method overriding. For example, the enqueue and dequeue methods in the MyQueue class override the basic insert method of the MyLinkedList class to provide queue-specific behavior, while the push and pop methods in the MyStack class provide stack-specific behavior.

V. Conclusion

This mini-project successfully demonstrates the basic operations on List, Stack, and Queue data structures through a well-designed GUI. The project highlights the use of object-oriented principles such as encapsulation, abstraction, inheritance, and polymorphism to create a modular and maintainable codebase. The visual representation of data structure operations enhances user understanding and interaction. The GUI is enhanced with a stylesheet (style.css) for a consistent and visually appealing user experience.

VI. References

1. Idea for the general structure of the program: OOP Lab excercises – AIMS Project.

2. Idea for the queue implementation: Introduction and Array Implementation of Queue | geeksforgeeks

3. Idea for the Help menu: Java Tutorial | Oracle