Unit number and title	**Unit 19: Data Structure & Algorithms**		
Submission date	20/7/2025		
Student Name	Le Van Thuong	**Student ID**	**001366804**
Class	CO3244	**Assessor name**	Nguyen The Nghia
❒ **Summative Feedback:** **Feedback:**			
Grade:	**Assessor Signature:**	**Date:**	

UNIVERSITY of GREENWICH

Alliance with FPT Education

Contents

1. Introduction

1.1 Overview of the Online Bookstore Project

This program functions as a console-based simulation of an online bookstore, enabling users to browse inventory, place orders, and manage records through a text-based interface. Its primary objective is to implement core data structures and algorithms without relying on Java's built-in collections. Instead, it employs custom implementations of linked lists, dynamic arrays, and stacks, along with manually coded sorting and searching algorithms.

The project serves as a practical exercise in data structure design and system performance. By constructing abstract data types (ADTs) from the ground up, students gain a deeper understanding of how data organization influences computational efficiency and how theoretical concepts can be translated into functional system design.

1.2 Objectives of the Coursework

- This coursework seeks to achieve the following objectives:
- Construct custom data structures tailored to a bookstore system.
- Develop a command-line application that allows users to manage book inventories and place customer orders.
- Apply sorting and searching algorithms to ensure efficient data processing.
- Evaluate and compare the effectiveness of various data structures in a realistic context.
- Perform testing on algorithm performance in terms of time and space complexity and provide justifications for selected approaches.

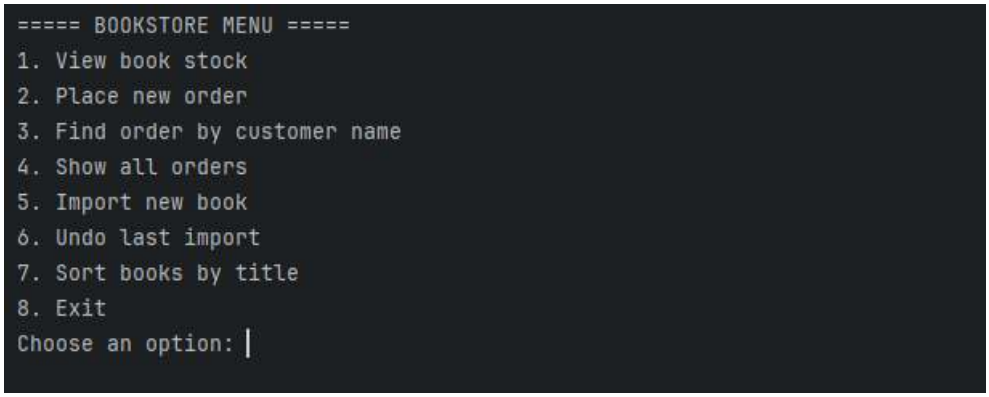
1.3 Scope and Limitations

Scope:

- The system allows users to place, process, and manage book orders through a console interface.
- Order records and inventory are handled using custom-designed data structures such as linked lists and stacks.
- Sorting (via Bubble Sort) and searching (via Linear Search) algorithms are implemented manually.
- Features include real-time stock reduction, undoing book imports, and searching for specific customer orders.

Limitations:

- The application does not include a graphical user interface (GUI); it is strictly text-based.
- All data is stored in memory, meaning information is lost upon program termination.
- The book catalog is predefined and does not allow for the dynamic addition of new book titles at runtime.
- Input validation is minimal, and invalid inputs may result in runtime errors.



- *Figure 1: Bookstore menu*

2. Design Specification

2.1 Book Inventory Management – ArrayListADT

Structure: Dynamically resizing array.

Key Operations: add(), get(), set(), size(), resize().

Rationale: Constant-time random access (O(1)), well-suited for maintaining a relatively stable list of inventory items.

Application: Managing the book inventory in the BookList class.

The BookList class manages inventory using a dynamic array, enabling fast index access and efficient updates. As the collection grows, the array resizes automatically, ensuring consistent performance without manual memory handling—ideal for frequent lookups and modifications.

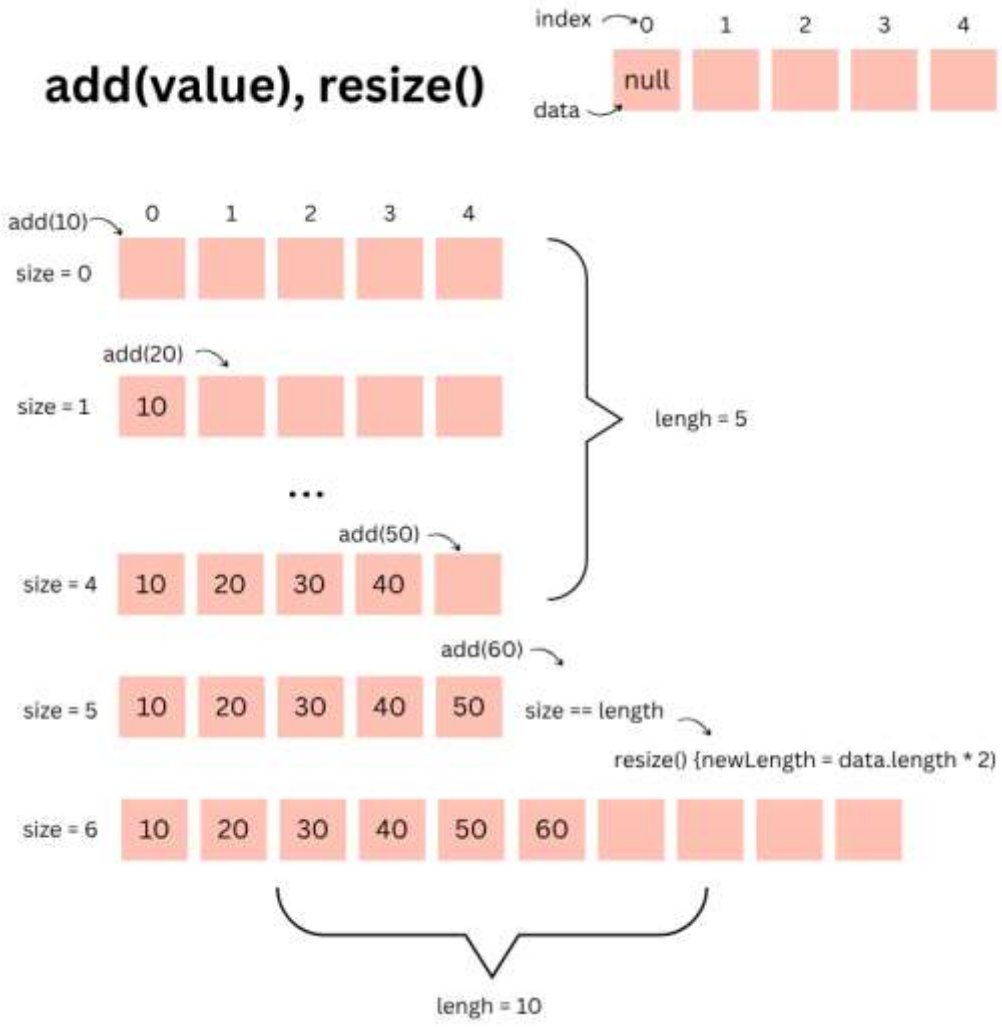


Figure 2:Explain the data flow of the add() and resize() functions

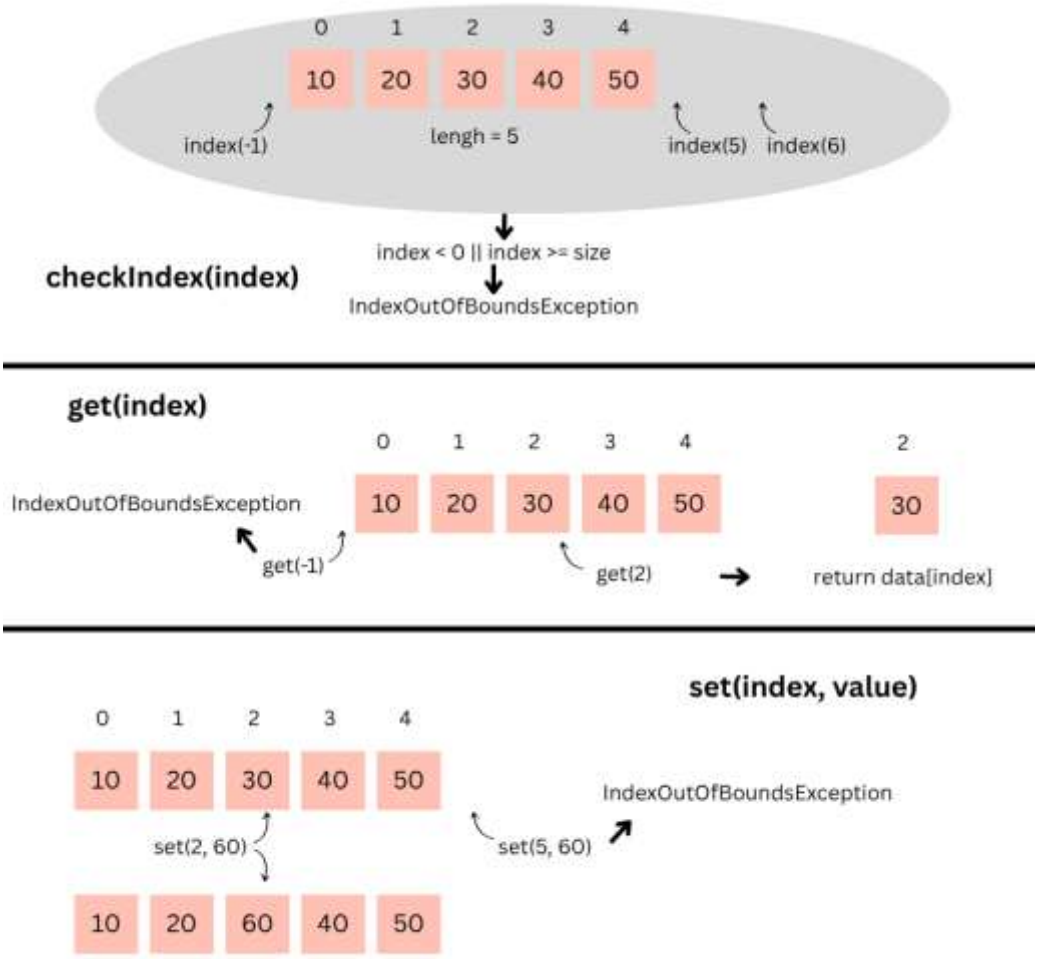


Figure 3:Explain the data flow of the get(index), set(index, value) and checkIndex(index) functions

2.2 Order Storage – LinkedListADT

Structure: Singly linked list.

Key Operations: addLast(), get(), set(), removeAt(), swap(), size().

Rationale: Optimized for dynamic insertions and deletions without shifting elements, providing flexibility in list manipulation.

Application: Used to store customer orders and books within orders.

The singly linked list supports sequential data storage and is well-suited for dynamic datasets, as its node-based structure allows efficient insertions and deletions without requiring contiguous memory allocation.

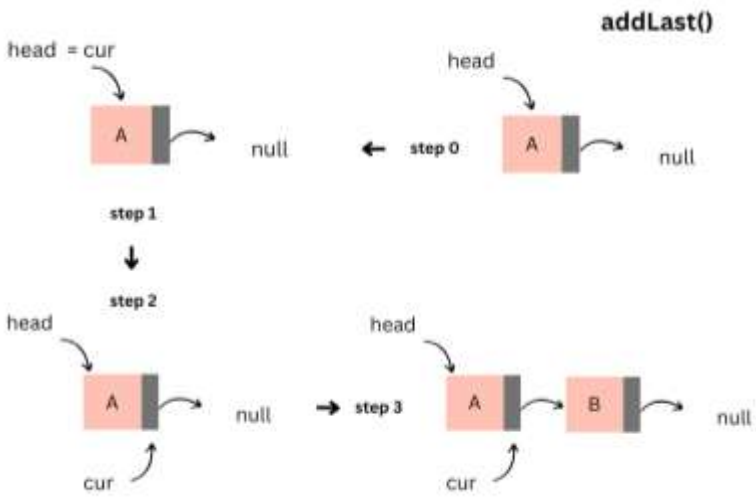


Figure 4: Explain the data flow of the addlast() functions

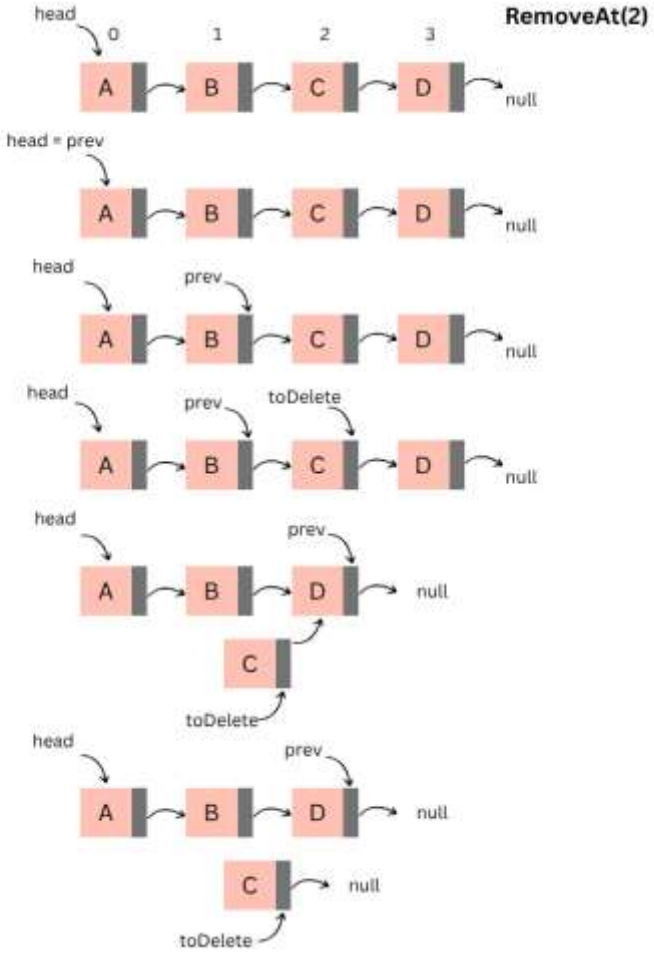


Figure 5:Explain the data flow of the removeAt() functions

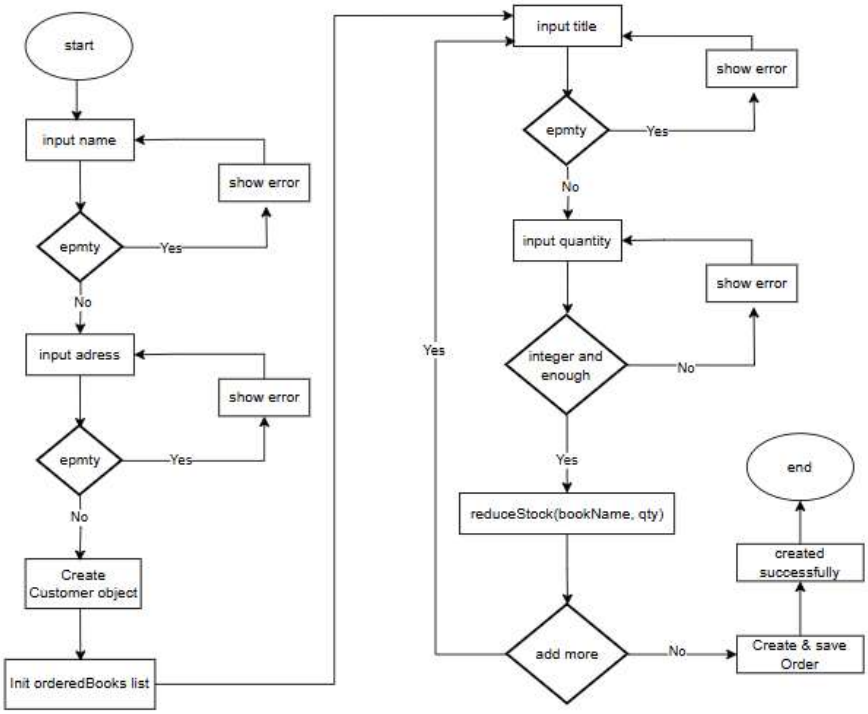


Figure 6: Flowchart: Place Order Function

The following flowchart illustrates the logic of placing a customer order in the Online Bookstore. It includes input validation (e.g., empty fields, valid quantity), real-time stock checking, and support for undoing the last book added using a stack-based mechanism.

2.3 Undo History – LinkedStackADT

Structure: Stack implemented using a singly linked list (LIFO).

Key Operations: push(), pop(), peek(), isEmpty(), size().

Rationale: Naturally supports last-in-first-out behavior, which aligns with undo functionality.

Application: Tracks book import history for reversal operations.

The undo mechanism for book imports is implemented via a custom stack, enabling efficient reversal of the most recent operations with minimal overhead.

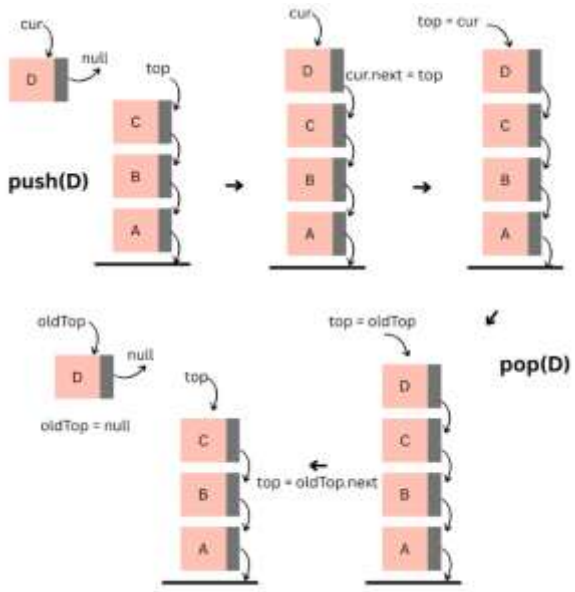


Figure 7:Explain the data flow of the push() and pop() functions

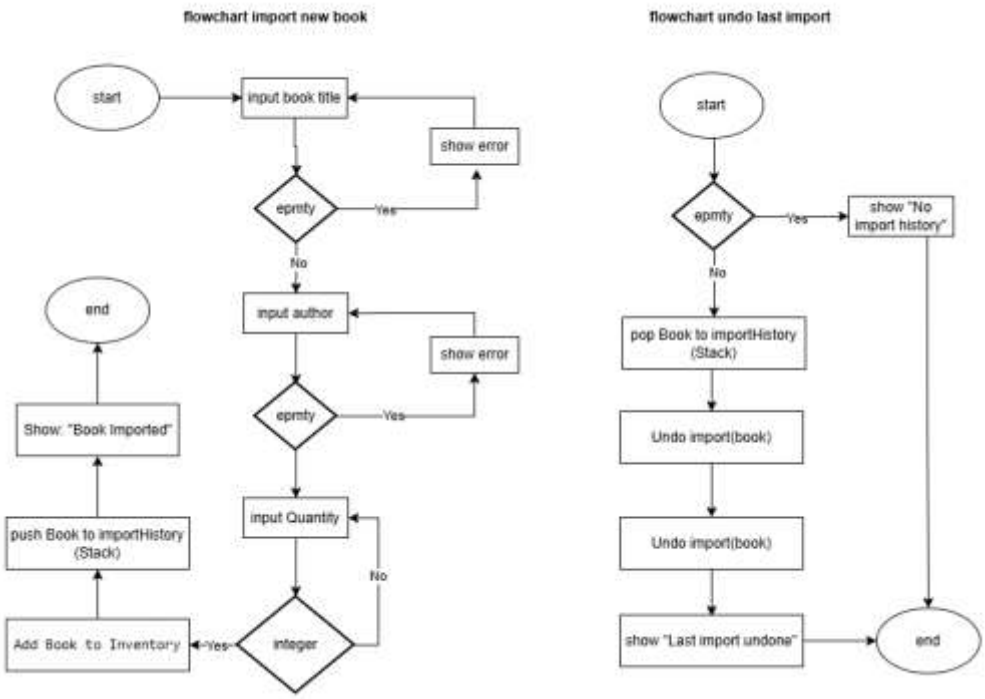


Figure 8: flowchart funtion import new book end undo last import

This diagram demonstrates how the system handles Undo actions by checking the import history stack and removing the last imported book if available.

2.4 Sorting Books – Bubble Sort

Structure: Comparison-based sorting algorithm.

Key Operation: Repeatedly compare and swap adjacent elements based on book title.

Rationale: Straightforward to implement; sufficient for small datasets with limited performance requirements.

Application: Alphabetical sorting of books in the inventory.

Bubble sort, despite its O(n2)$O(n2)$ complexity, is used for its simplicity and pedagogical value. Given the small dataset, performance drawbacks are minimal, making it suitable for illustrating fundamental algorithmic concepts.

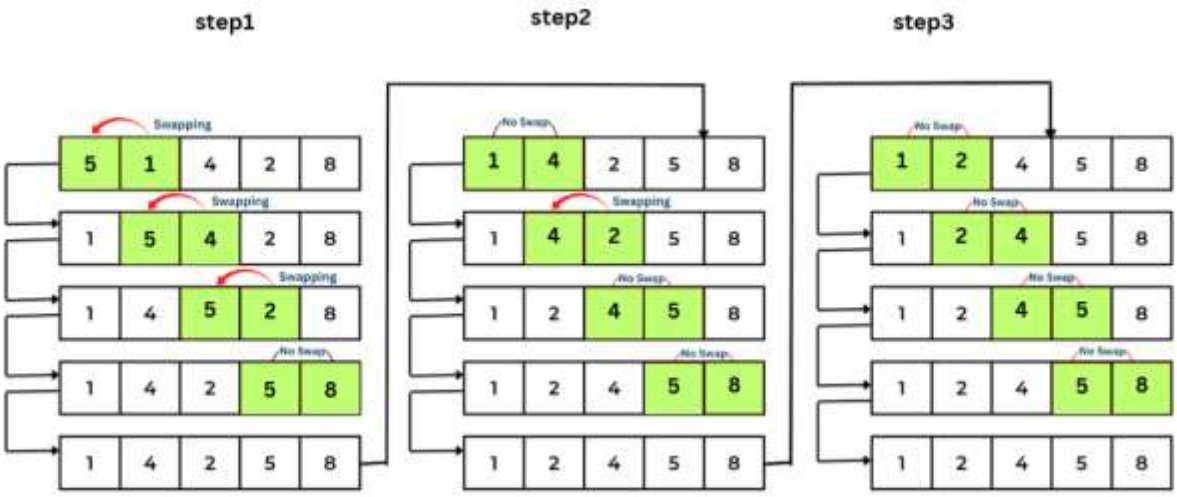


Figure 9: bubble sort algorithm explained

2.5 Searching Orders – Linear Search

Structure: Sequential search through list elements.

Key Operation: Match input customer name against stored order records.

Rationale: Easy to implement, requires no pre-sorting or additional indexing structures.

Application: Locate specific orders by customer name.

A linear search is used to locate orders by scanning each element sequentially. While suboptimal for large datasets, it is sufficient given the system's limited scale.

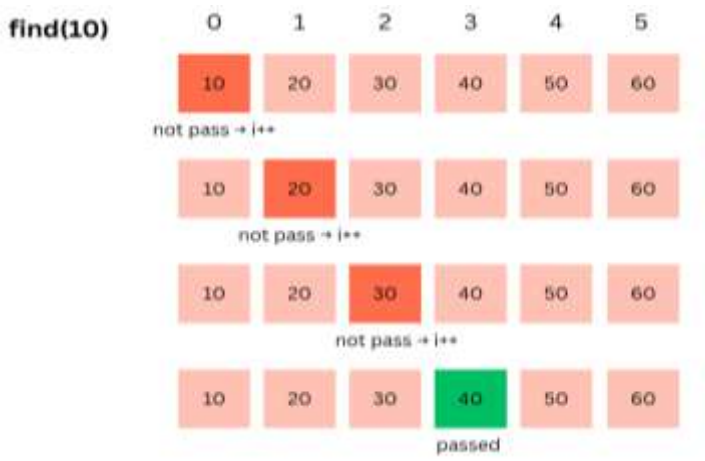


Figure 10: search algorithm explained

2.6 Role-Based Access Control (RBAC)

To ensure a more realistic simulation, the system implements a login mechanism that distinguishes between 'admin' and 'user' roles.

Based on the login input, the main menu dynamically shows only the functions permitted to that role.

- **Admins** can access all system functions including inventory management and order history.

- **Users** are restricted to browsing books, placing orders, and sorting inventory.

This helps simulate multi-user systems and enforce access boundaries in real-life software systems.

Figure 11: folowchart login funtion

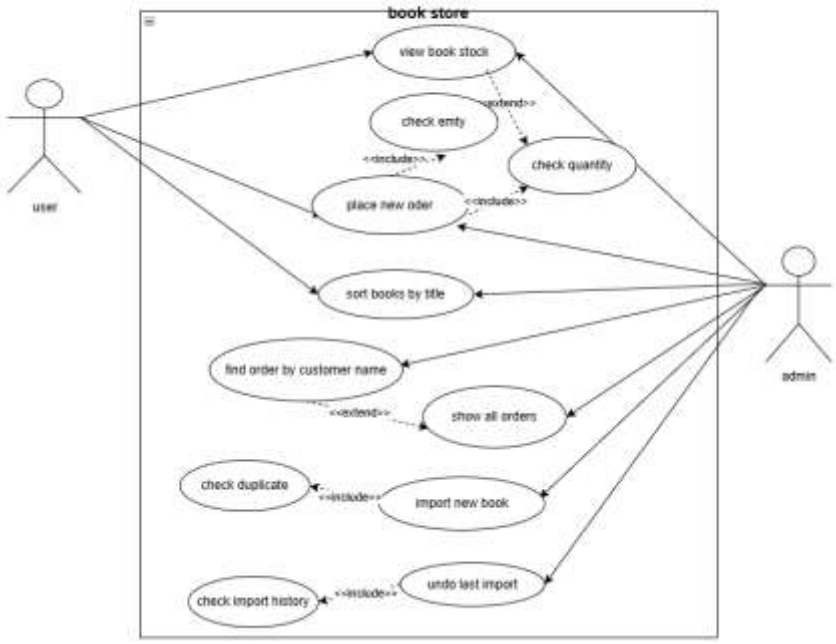


Figure 12: Use Case Diagram of Bookstore Processing System

The diagram above illustrates the flow of the login system. After the user inputs their role (admin or user), the system dynamically displays a customized menu. Unauthorized access attempts are blocked with warning messages, ensuring secure role-based navigation.

3. System Implementation Overview

The system is developed using modular object-oriented principles in Java, encapsulating data within custom structures and maintaining a clear separation of concerns across functional layers:

- **Model Layer:** Contains representations for Book, Customer, and Order.
- **Data Structure Layer (ADT):** Custom implementations of ArrayListADT, LinkedListADT, and LinkedStackADT.
- **Algorithm Layer:** Implements bubble sort and linear search logic.
- **Application Layer:** BookstoreApp.java coordinates user interactions and system responses via a console-based menu.

All functionalities are accessible through a user-friendly text interface, allowing inventory management, order processing, and administrative tasks such as sorting and undoing imports.

```
Login as (admin/user): adfg
[!] Invalid role. Please enter 'admin' or 'user'.
Login as (admin/user):
```

Figure 13:login if the user enters incorrectly

```
Login as (admin/user): user

===== BOOKSTORE MENU =====
1. View book stock
2. Place new order
7. Sort books by title
8. Exit
Choose an option:
```

Figure 14: login if the user is a user

```
Login as (admin/user): admin

===== BOOKSTORE MENU =====
1. View book stock
2. Place new order
3. Find order by customer name
4. Show all orders
5. Import new book
6. Undo last import
7. Sort books by title
8. Exit
Choose an option:
```

Figure 15:login if the user is admin

```
===== BOOKSTORE MENU =====
1. View book stock
2. Place new order
3. Find order by customer name
4. Show all orders
5. Import new book
6. Undo last import
7. Sort books by title
8. Exit
Choose an option:
```

Figure 16: Bookstore menu

```
Choose an option: 1

=== 📚 Book Stock ===
1. Java Programming Basics             | Author: John A. Nguyen      | Quantity: 10
2. Data Structures and Algorithms      | Author: Alice Tran          | Quantity: 8
3. Advanced Algorithms                 | Author: Charles Pham        | Quantity: 12
4. System Analysis and Design          | Author: Diana Le            | Quantity: 7
5. Computer Networks                   | Author: Edward Ngo          | Quantity: 5
6. Operating Systems                   | Author: Fiona Dang          | Quantity: 9
7. Database Systems                    | Author: George Truong       | Quantity: 11
8. Object-Oriented Programming         | Author: Hannah Vo           | Quantity: 6
9. Information Security                | Author: Isaac Nguyen        | Quantity: 13
10. Artificial Intelligence             | Author: Jennifer Doan       | Quantity: 10
```

Figure 17: view book stock

```
Choose an option: 2
Place a New Order
Customer name:
[!] Name cannot be empty. Please enter again.
Customer name: thuong
Address:
[!] Address cannot be empty. Please enter again.
Address: da nang
Book title to buy:
[!] Book title cannot be empty.
Book title to buy: Artificial Intelligence
Quantity (positive number): -1
[!] Please enter a valid number.
Quantity (positive number): 3
Book added to order.
Add more books? (y/n): y
Book title to buy: Information Security
Quantity (positive number): 3
Book added to order.
Add more books? (y/n): n
🎉 Order created successfully for thuong
```

Figure 18: Order function

```
Choose an option: 3
Customer name: thuong
📦 Order ID: 1000, Customer: thuong, Address: Da Nang
- Object-Oriented Programming         | Author: Hannah Vo            | Quantity: 2
```

Figure 19: Find order by customer name

```
All Orders:
Order ID: 1000, Customer: thuong, Address: Da Nang
  - Object-Oriented Programming         | Author: Hannah Vo            | Quantity: 2
Order ID: 1001, Customer: Quan, Address: Quang Tri
  - Artificial Intelligence             | Author: Jennifer Doan        | Quantity: 2
```

Figure 20: Show all orders

```
Choose an option: 5
Book title: Conan
Author: Aoyama
Quantity: 4
Book imported and saved to history.
```

Figure 21: Import new book

```
8. Object-Oriented Programming         | Author: Hannah Vo            | Quantity: 4
9. Information Security                | Author: Isaac Nguyen         | Quantity: 13
10. Artificial Intelligence             | Author: Jennifer Doan         | Quantity: 8
11. Conan                               | Author: Aoyama               | Quantity: 4
```

Figure 22: Book stock after import a new book

```
Choose an option: 6
Last import undone.
```

Figure 23: Undoing last book import using Stack

```
 7. Database Systems                   | Author: George Truong        | Quantity: 11
 8. Object-Oriented Programming        | Author: Hannah Vo            | Quantity: 4
 9. Information Security               | Author: Isaac Nguyen         | Quantity: 13
 10. Artificial Intelligence            | Author: Jennifer Doan         | Quantity: 8
```

Figure 24: Book stock after undo last import

```
Choose an option: 7
Books sorted by title.
```

Figure 25: Sorting book stock by title (Bubble Sort)

Figure 26: book stock after sort by title

4. Testing and Evaluation

4.1 Functionality Testing

The system underwent black-box and white-box testing to validate the core functionalities:

- **Inventory display** correctly lists all available books.
- **Order placement** accurately deducts stock and creates records.
- **Undo mechanism** reliably reverts the most recent book import.
- **Search functionality** returns the correct order given a matching customer name.
- **Sorting** rearranges book records alphabetically as expected.
- **Log in** when the user is user or admin.

Appendix C: Sample Test Case.

Test Case	Functionality	Result
Login	Login admin or user	Pass
View Stock	Displays inventory	Pass
Place Order	Create order with details	Pass
Search by Customer	Locate order via name	Pass
Import Book	Add new stock to inventory	Pass
Undo Import	Revert last book added	Pass
Sort Inventory	Alphabetical title sort	Pass

Table 1: test case

```
Login as (admin/user): adfg
[!] Invalid role. Please enter 'admin' or 'user'.
Login as (admin/user):
```

Figure 27:login if the user enters incorrectly

```
Login as (admin/user): user

===== BOOKSTORE MENU =====
1. View book stock
2. Place new order
7. Sort books by title
8. Exit
Choose an option:
```

Figure 28: login if the user is a user

```
Login as (admin/user): admin

===== BOOKSTORE MENU =====
1. View book stock
2. Place new order
3. Find order by customer name
4. Show all orders
5. Import new book
6. Undo last import
7. Sort books by title
8. Exit
Choose an option:
```

Figure 29:login if the user is admin

```
===== BOOKSTORE MENU =====
1. View book stock
2. Place new order
3. Find order by customer name
4. Show all orders
5. Import new book
6. Undo last import
7. Sort books by title
8. Exit
Choose an option: 1

=== 📚 Book Stock ===
1. Java Programming Basics            | Author: John A. Nguyen      | Quantity: 10
2. Data Structures and Algorithms     | Author: Alice Tran          | Quantity: 8
3. Advanced Algorithms                | Author: Charles Pham        | Quantity: 12
4. System Analysis and Design         | Author: Diana Le            | Quantity: 7
5. Computer Networks                  | Author: Edward Ngo          | Quantity: 5
6. Operating Systems                  | Author: Fiona Dang          | Quantity: 9
7. Database Systems                   | Author: George Truong       | Quantity: 11
8. Object-Oriented Programming        | Author: Hannah Vo           | Quantity: 6
9. Information Security               | Author: Isaac Nguyen        | Quantity: 13
10. Artificial Intelligence            | Author: Jennifer Doan       | Quantity: 10
```

Figure 30: Book stock

```
Choose an option: 2
Customer name: thuong
Address: Da Nang
Book title to buy: Artificial Intelligence
Quantity: 2
Add more books? (y/n): y
Book title to buy: Information Security
Quantity: Author: Isaac Nguyen
[!] Please enter a number.
Quantity: 2
Add more books? (y/n): n
Order created for thuong
```

Figure 31: Create a valid and invalid order

```
All Orders:
Order ID: 1000, Customer: thuong, Address: Da Nang
  - Artificial Intelligence           | Author: Jennifer Doan       | Quantity: 2
  - Information Security              | Author: Isaac Nguyen        | Quantity: 2
```

Figure 32: show all order

```
Choose an option: 5
Book title:
[!] Title cannot be empty. Please re-enter.
Book title: test
Author:
[!] Author cannot be empty. Please re-enter.
Author: thuong
Quantity: -1
[!] Quantity must be a positive number.
Quantity: ă
[!] Quantity must be a positive number.
Quantity: 1
✅ Book imported and saved to history.
Last import undone.
```

Figure 33: import a new book

```
=== 📚 Book Stock ===
1. Java Programming Basics            | Author: John A. Nguyen      | Quantity: 10
2. Data Structures and Algorithms     | Author: Alice Tran          | Quantity: 8
3. Advanced Algorithms                | Author: Charles Pham        | Quantity: 12
4. System Analysis and Design         | Author: Diana Le            | Quantity: 7
5. Computer Networks                  | Author: Edward Ngo          | Quantity: 5
6. Operating Systems                  | Author: Fiona Dang          | Quantity: 9
7. Database Systems                   | Author: George Truong       | Quantity: 11
8. Object-Oriented Programming        | Author: Hannah Vo           | Quantity: 6
9. Information Security               | Author: Isaac Nguyen        | Quantity: 11
10. Artificial Intelligence            | Author: Jennifer Doan       | Quantity: 8
11. test                               | Author: thuong              | Quantity: 1
```

Figure 34: Book stock after import a new book

```
Choose an option: 6
Last import undone.
```

Figure 35: undo last import

```
=== 📚 Book Stock ===
1. Java Programming Basics            | Author: John A. Nguyen      | Quantity: 10
2. Data Structures and Algorithms     | Author: Alice Tran          | Quantity: 8
3. Advanced Algorithms                | Author: Charles Pham        | Quantity: 12
4. System Analysis and Design         | Author: Diana Le            | Quantity: 7
5. Computer Networks                  | Author: Edward Ngo          | Quantity: 5
6. Operating Systems                  | Author: Fiona Dang          | Quantity: 9
7. Database Systems                   | Author: George Truong       | Quantity: 11
8. Object-Oriented Programming        | Author: Hannah Vo           | Quantity: 6
9. Information Security               | Author: Isaac Nguyen        | Quantity: 11
10. Artificial Intelligence            | Author: Jennifer Doan       | Quantity: 8
```

Figure 36: book stock after undo

```
Choose an option: 6
No import history.
```

Figure 37: undo without last import

```
Choose an option: 3
Customer name: thuong
 Order ID: 1000, Customer: thuong, Address: Da Nang
- Artificial Intelligence            | Author: Jennifer Doan       | Quantity: 2
- Information Security               | Author: Isaac Nguyen        | Quantity: 2
```

Figure 38: find order by customer name

```
Choose an option: 7
Books sorted by title.
```

Figure 39: sort book stock by book title

```
=== Book Stock ===
1. Advanced Algorithms              | Author: Charles Pham        | Quantity: 12
2. Artificial Intelligence          | Author: Jennifer Doan       | Quantity: 8
3. Computer Networks                | Author: Edward Ngo          | Quantity: 5
4. Data Structures and Algorithms   | Author: Alice Tran          | Quantity: 8
5. Database Systems                 | Author: George Truong       | Quantity: 11
6. Information Security             | Author: Isaac Nguyen        | Quantity: 11
7. Java Programming Basics          | Author: John A. Nguyen      | Quantity: 10
8. Object-Oriented Programming      | Author: Hannah Vo           | Quantity: 6
9. Operating Systems                | Author: Fiona Dang          | Quantity: 9
10. System Analysis and Design       | Author: Diana Le            | Quantity: 7
```

Figure 40: book stock after sort

4.2 Algorithmic Efficiency.

a. **Algorithm Evaluation Table**.

Bubble Sort was chosen for its simplicity and pedagogical value. To assess its suitability, its performance was compared with Merge Sort. As shown in the execution time chart, Bubble Sort performs adequately on arrays up to 1,000 elements—consistent with the system's scale. While Merge Sort offers better efficiency for larger datasets, its added complexity makes Bubble Sort a balanced choice in terms of clarity, traceability, and performance.

Table 2:Algorithm Evaluation Table

Algorithm	Applied To	Time Complexity	Reason for Selection	Advantages
Bubble Sort	LinkedListADT<Book> (Sort by Title)	$O(n^2)$	Although inefficient for large datasets, Bubble Sort is easy to implement and debug	- Simple logic for beginners - Easy to trace and debug

			manually with custom LinkedList.	- Sufficient for small book inventory
Linear Search	LinkedListADT<Order> (Find Order by Name)	O(n)	Suitable for unsorted linked lists where indexing is not optimized. Allows straightforward implementation on custom ADTs.	- Works without pre-sorting - Easy to understand and implement - Ideal for small datasets
Merge Sort	not used	O(n log n)	Efficient sorting of large datasets	Stable sort, consistent performance, suitable for linked lists

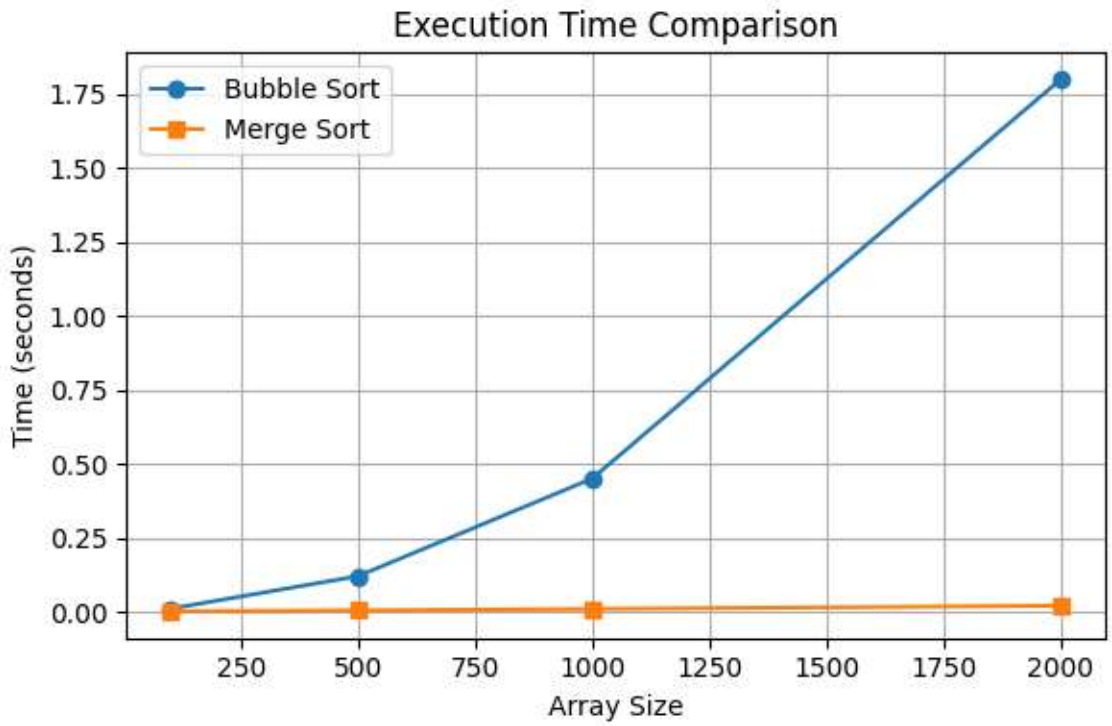


Figure 41: Execution Time Comparison

To test the scalability of LinkedListADT and linear search, a test program was written to automatically generate 2000 customer orders. Finding a specific order (e.g. "Customer_999") using linear search takes about 17ms and bubble sort takes about 170ms on a typical laptop. This test confirms that while the current implementation is

sufficient for small to medium sized data sets, using a HashMap or Binary Search Tree would be more appropriate for larger systems.

b. **Data Structure Evaluation Summary**.

Table 3:Data Structure Evaluation Summary

Data Structure	Used For	Reason for Use	Advantages
ArrayListADT	Storing core list of books	Allows random access and indexing, useful for accessing books directly by index	- Fast lookup - Resizable array - Familiar API
LinkedListADT	Managing orders and ordered books	Efficient insertion/deletion at ends, no need to shift elements	- Good for dynamic datasets - Simple node-based logic - Avoids resizing overhead
LinkedStackADT	Undo import functionality	LIFO logic fits perfectly for reversing the most recent book import	- Natural fit for undo - Simple to manage history - Lightweight implementation

4.3 Edge Case Testing

To ensure the robustness and reliability of the system, several critical edge cases were thoroughly tested and validated:

Invalid Index Handling:

Attempts to access or modify elements using invalid indices in the get() and set() methods correctly trigger an IndexOutOfBoundsException. This behavior safeguards the system from potential memory corruption and ensures data integrity is maintained.

Code:

```
    //      Test 7: Invalid index in ArrayListADT
System.out.println("\n Test 7: Invalid Index Access in ArrayListADT");
ArrayListADT<String> myList = new ArrayListADT<>();
myList.add("Book A");
myList.add("Book B");

try {
   System.out.println("Trying to get index 5...");
   String book = myList.get(5); // Index >=lengh
```

```
    } catch (IndexOutOfBoundsException e) {
        System.out.println("[ERROR] Cannot retrieve element: " + e.getMessage());
    }

    try {
        System.out.println("Trying to set index -1...");
        myList.set(-1, "Book Z"); // Index <0
    } catch (IndexOutOfBoundsException e) {
        System.out.println("[ERROR] Invalid index: " + e.getMessage());
    }
```

- **Test Case :** Import New Book

- Input: choose case 2.
- Expected Output:
  - Trying to get index 5...
  - [ERROR] Cannot retrieve element: Index out of bounds: 5
  - Trying to set index -1...
  - [ERROR] Invalid index: Index out of bounds: -1

✅ Passed

```
 Test 7: Invalid Index Access in ArrayListADT
Trying to get index 5...
[ERROR] Cannot retrieve element: Index out of bounds: 5
Trying to set index -1...
[ERROR] Invalid index: Index out of bounds: -1
```

Figure 42: test Case: Import New Book

Duplicate Book Import Handling:

When a book with an existing title is re-imported, the system updates the quantity of the existing record rather than duplicating it, ensuring data consistency and preventing redundancy.

Test Case : Add books already in the book list.

- Data: In the book list, there is the book "Java Programming Basics" with author "John A. Nguyen" and quantity is 10.

- Input: 5 more books of the same type
- Expected Output:

1. Java Programming Basics | Author: John A. Nguyen | Quantity: 15 Passed

Figure 43: Inventory before importing duplicate book

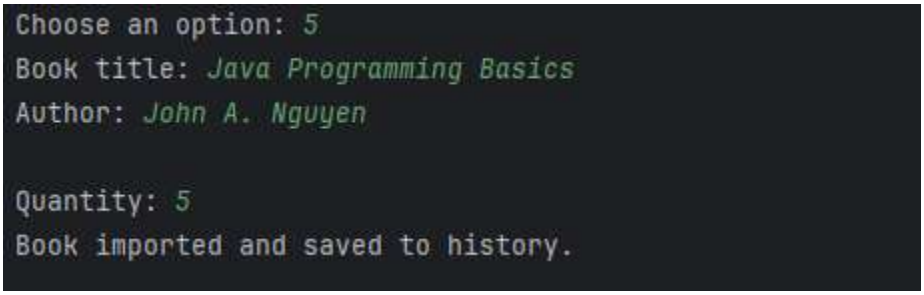


Figure 44: Console output during duplicate import

Figure 45: Inventory after updating quantity of existing book

Undo Operation on Empty History:

If no prior import operations exist, invoking the undo function triggers a user-friendly error message. This ensures graceful failure handling and prevents system instability.

Test Case : Undo Operation on Empty History

- Input: no previous history.
- Expected Output: No import history.

Figure 46: Undo Operation on Empty History

4.4 Limitations

- The system is console-based and lacks a graphical interface.
- Sorting and searching algorithms are not optimized for scalability.
- Error handling is basic and can be expanded with more robust validations.

5. Reflection, Lessons Learned, and Future Enhancements

5.1 Personal Reflection and Insights

Building this system from scratch deepened my understanding of abstract data types and memory management beyond Java's built-in libraries. Manually implementing LinkedListADT, ArrayListADT, and LinkedStackADT clarified how data is structured and manipulated. I realized that algorithm choice significantly impacts performance, especially in search and sort operations. The project also reinforced key software engineering principles like modularity and separation of concerns. In future modules such as AI or Database Systems, I will apply these insights to optimize data handling, particularly by evaluating trade-offs in algorithm design and improving system scalability.

5.2 Challenges Encountered

- **Manual Implementation:** Developing low-level data structures demanded precision and exposed various logical pitfalls, especially with node manipulation and memory reference errors.
- **Error Handling:** Managing exceptional scenarios like empty stack operations or invalid list indices required extra care to avoid runtime crashes.
- **User Experience via Console:** Implementing a smooth and logical text-based UI flow—especially for nested book order loops—was more intricate than anticipated.

5.3 Skills Acquired

This project enhanced both technical and analytical skillsets, including:

- Mastery of Java syntax and object-oriented design concepts.
- Practical application of custom data structures in real-life scenarios.
- Improved algorithmic thinking, especially concerning complexity trade-offs.
- Ability to test and analyze software functionality in both black-box and white-box contexts.

5.4 Future Enhancements

To advance this system into a more robust, scalable application, the following improvements are proposed:

- **Enhanced Search:** Replace linear search with a hash map or apply binary search after sorting to improve efficiency.
- **GUI Development:** Integrate a graphical interface using JavaFX or Swing to enhance usability and accessibility.
- **Persistent Storage:** Implement file I/O or a lightweight database (e.g., SQLite) to preserve data across sessions.
- **Optimized Sorting:** Introduce Merge Sort or Quick Sort for more efficient handling of large datasets.
- **Advanced Validation:** Add comprehensive input validation and exception management for greater fault tolerance.

This project has not only met its educational goals but has also provided a foundational experience in designing, implementing, and evaluating a non-trivial software system through the lens of core data structure theory.

6.References

Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C., 2009. Introduction to Algorithms. 3rd ed. Cambridge, MA: MIT Press.

Karumanchi, N., 2016. Data Structures and Algorithms Made Easy. 2nd ed. CareerMonk Publications.

Bible, P.W., Moser, L. and Scarlato, M.M., 2023. An Open Guide to Data Structures and Algorithms. PALNI. Available at: https://open.umn.edu/opentextbooks/textbooks/1017 [Accessed 20 July 2025].

Weiss, M.A., 2014. Data Structures and Algorithm Analysis in Java. 3rd ed. Pearson Education.

7. Appendices.

Full Source Code:

Drive: https://drive.google.com/drive/folders/1f3QjxPm7X-ADcdqyz4N3bLjuO8j-VP4n?usp=sharing

Github : https://github.com/thuonglv024/LeVanThuong_1649A.git

Appendix : Key Code Snippets

ArrayListADT.java – A simplified dynamic array structure used to store inventory items.

```
private void resize() {
        int newLength = data.length * 2;
        data = Arrays.copyOf(data, newLength);
    }

    public void add(T value) {
        if (size == data.length) {
            resize();
        }
        data[size] = value;
        size++;
    }

    public T get(int index) {
        checkIndex(index);
        return data[index];
    }

    public void set(int index, T value) {
        checkIndex(index);
        data[index] = value;
    }

    public int size() {
        return size;
    }


    private void checkIndex(int index) {
        if (index < 0 || index >= size)
            throw new IndexOutOfBoundsException("Index out of bounds: " + index);
    }
}
```

LinkedListADT.java – Node-based list for customer orders and order content.

```
public class Sort {
    public static void bubbleSortByTitle(LinkedListADT<Book> list) {
        int n = list.size();
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (list.get(j).getTitle().compareToIgnoreCase(list.get(j +
1).getTitle()) > 0) {
                    list.swap(j, j + 1);
                }
            }
```

```
        }
    }
```

LinkedStackADT.java – Stack used for undo import history.

```
public void push(T item) {
    Node node = new Node(item);
    node.next = top;
    top = node;
    size++;
}

public T pop() {
    if (top == null) return null;
    T item = top.data;
    Node oldTop = top;
    top = top.next;
    oldTop.next = null;

    size--;
    return item;
}
```

Bubble Sort Function – Sort inventory books alphabetically by title.

```
public class Sort {
    public static void bubbleSortByTitle(LinkedListADT<Book> list) {
        int n = list.size();
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (list.get(j).getTitle().compareToIgnoreCase(list.get(j +
1).getTitle()) > 0) {
                    list.swap(j, j + 1);
                }
            }
        }
    }
```

Place order funtion

```
System.out.println("Place a New Order");

String name;
do {
    System.out.print("Customer name: ");
    name = sc.nextLine().trim();
    if (name.isEmpty()) {
        System.out.println("[!] Name cannot be empty. Please enter again.");
    }
} while (name.isEmpty());

String address;
do {
```

```
    System.out.print("Address: ");
    address = sc.nextLine().trim();
    if (address.isEmpty()) {
        System.out.println("[!] Address cannot be empty. Please enter again.");
    }
} while (address.isEmpty());

Customer customer = new Customer(name, address);
LinkedListADT<Book> orderedBooks = new LinkedListADT<>();

while (true) {
    String bookName;
    do {
        System.out.print("Book title to buy: ");
        bookName = sc.nextLine().trim();
        if (bookName.isEmpty()) {
            System.out.println("[!] Book title cannot be empty.");
        }
    } while (bookName.isEmpty());

    int qty = -1;
    while (qty <= 0) {
        System.out.print("Quantity (positive number): ");
        String input = sc.nextLine().trim();
        if (input.matches("\\d+")) {
            qty = Integer.parseInt(input);
            if (qty <= 0) {
                System.out.println("[!] Quantity must be greater than 0.");
            }
        } else {
            System.out.println("[!] Please enter a valid number.");
        }
    }

    if (inventory.reduceStock(bookName, qty)) {
        Book found = inventory.findBook(bookName);
        orderedBooks.addLast(new Book(found.getTitle(), found.getAuthor(), qty));
        System.out.println("Book added to order.");
    } else {
        System.out.println("Not enough stock or book not found.");
    }

    System.out.print("Add more books? (y/n): ");
    if (!sc.nextLine().equalsIgnoreCase("y")) break;
}

orders.addLast(new Order(customer, orderedBooks));
System.out.println("Order created successfully for " + name);

break;
```

Undo last import funtion

```
if (!importHistory.isEmpty()) {
    Book undoBook = importHistory.pop();
```

```
    if (inventory.undoImport(undoBook)) {
        System.out.println("Last import undone.");
    }
} else {
    System.out.println("No import history.");
}
break;
```