Generalized Data Structures and Algorithms Library (Java)

Project Overview

This project is a comprehensive Java-based library demonstrating the fundamental implementations of various **linear** and **non-linear data structures**, along with essential **searching** and **sorting algorithms**. It is designed as an academic and practical resource, focusing on a clean, object-oriented approach for clarity and reusability.

Author: Shardul Tapkire

Date: 29/09/2025

File Details

File Name	Description
Generalised_Data_Structures_library.java	A merged Java file containing modular class implementations for all data structures and algorithms, including Queue, Stack, various Linked Lists, BST, Sorting, and Searching.

Features

1. Linear Data Structures

Class	Description	Key Methods	Constraint/Type
QueueX<T>	Generic Queue implementation.	enqueue(), dequeue(), display(), count()	First-In, First-Out (FIFO)
StackX<T>	Generic Stack implementation.	push(), pop(), display(), count()	Last-In, First-Out (LIFO)

Class	Description	Key Methods	Constraint/Type
SinglyLL<T extends Number>	Generic Singly Linear Linked List.	insertFirst(), insertLast(), deleteFirst(), deleteLast(), display()	Linear, Non-Circular
DoublyLL<T extends Number>	Generic Doubly Linear Linked List.	insertFirst(), insertLast(), deleteFirst(), deleteLast(), display()	Linear, Non-Circular
SinglyCLL<T extends Number>	Generic Singly Circular Linked List.	insertFirst(), insertLast(), display()	Circular
DoublyCLL<T extends Number>	Generic Doubly Circular Linked List.	insertFirst(), insertLast(), display()	Circular

2. Non-Linear Data Structure

Class	Description	Key Methods	Traversal Orders
BST<T extends Comparable<T>>	Generic Binary Search Tree implementation.	insert(), search()	inorder(), preorder(), postorder()

3. Algorithms

Class	Type	Algorithm	Time Complexity (Average)
Sorting	Sorting	bubbleSort()	O(n2)
Sorting	Sorting	selectionSort()	O(n2)
Searching	Searching	linearSearch()	O(n)
Searching	Searching	binarySearch()	O(logn)

4. Mathematical Utilities (Linked Lists)

The Linked List classes (SinglyLL, DoublyLL, SinglyCLL, DoublyCLL) include extra logic to perform number property checks and manipulations on their integer elements. This is enforced by the **T extends Number** generic constraint.

Method	Description
displayPerfect()	Displays all **Perfect Numbers** in the list.
displayPrime()	Displays all **Prime Numbers** in the list.
sumOfDigits()	Displays the **sum of digits** for each element.
reverseDigits()	Modifies the list by **reversing the digits** of each element (e.g., 123 becomes 321).

Technical Implementation Details

Generic Usage

All data structure classes (QueueX, StackX, BST, and all Linked Lists) are **Generic** (<T>), allowing them to store objects of any type.

- **Linked Lists** are constrained with T extends Number to enable number-specific utility methods (isPerfect, isPrime, etc.).
- **BST** is constrained with T extends Comparable<T> to ensure elements can be compared for proper ordering within the tree.

Linked List Structure

All linked list implementations use an internal private class Node to represent elements.

- **Doubly** lists include a prev reference.
- **Circular** lists ensure that the last node points back to the first node (last.next = first).

Error Handling

The QueueX and StackX classes use robust error handling by throwing a **NoSuchElementException** when attempting to dequeue or pop from an empty structure.

Project Flow and Execution

Compilation and Run

Since all classes are contained within a single file, the compilation and execution steps are straightforward:

1. **Compile:**

Bash

```
javac Generalised_Data_Structures_library.java
```

2. **Run:**

Bash

```
java Generalised_Data_Structures_library
```

Main Demonstration (main method)

The Generalised_Data_Structures_library class contains a main method that serves as a demonstration and testing harness for all implemented data structures and algorithms. It provides examples for:

- Queue/Stack push/pop/display operations.
- Insertion, traversal, and number property checks for all four Linked List types.
- BST insertion, search, and all three traversal methods (Inorder, Preorder, Postorder).
- Sorting examples (Bubble and Selection Sort).
- Searching examples (Linear and Binary Search).

Future Enhancements

The current implementation provides a solid foundation. Potential future work includes:

- Implementing more advanced sorting algorithms (e.g., **Quick Sort**, **Merge Sort**, Heap Sort).
- Implementing **Graph** data structures and algorithms (e.g., BFS, DFS, Dijkstra's Algorithm).
- Adding hash-based structures like **HashMap** and **HashTable**.
- Creating a robust suite of **JUnit tests** for validation.
- Packaging the classes into a reusable **Java Archive (JAR)** library.

Learning Outcomes

By completing and exploring this project, I have learned:

1. Gain hands-on experience with **object-oriented programming (OOP) principles** in Java.
2. Understand the **internal working and implementation** of fundamental data structures (Stack, Queue, Linked Lists, Trees).
3. Learn how to design and use **generic classes and constraints** in Java for type safety and reusability.
4. Explore the **time and space complexity** of searching and sorting algorithms.

5. Understand how **circular and doubly-linked structures** differ from linear ones in terms of design and usage.
6. Gain knowledge of **tree-based structures (BSTs)** and traversal strategies (Inorder, Preorder, Postorder).
7. Apply **mathematical utilities** on linked list data structures to connect number theory with programming.
8. Develop skills in **error handling and exception management** in Java.
9. Build confidence in compiling, running, and testing modular Java code in a single-file project setup.
10. Lay a strong foundation for moving towards **advanced data structures and algorithms** like Graphs, Heaps, and Hashing.