**Scenario 2**

# Code

import java.util.ArrayList;

import java.util.Collections;

import java.util.LinkedList;

import java.util.Scanner;

class Stock {

    private String name;

    private double price;

    public Stock(String name, double price) {

        this.name = name;

        this.price = price;

    }

    public String getName() {

        return name;

    }

    public double getPrice() {

        return price;

    }

    @Override

    public String toString() {

        return name + " (Rs. " + price + ")";

    }

}

class StockTracker {

    private ArrayList<Stock> stockArrayList;

    private LinkedList<Stock> stockLinkedList;

    private boolean useArrayList = true;

    private Scanner scanner;

    public StockTracker() {

        stockArrayList = new ArrayList<>();

        stockLinkedList = new LinkedList<>();

        scanner = new Scanner(System.in);

        addDefaultEntries();

    }

    private void addDefaultEntries() {

        addStock(new Stock("Pakistan State Oil", 120.50));

        addStock(new Stock("Habib Bank Limited", 135.75));

        addStock(new Stock("Engro Corporation", 290.25));

        addStock(new Stock("K-Electric", 3.45));

        addStock(new Stock("Lucky Cement", 725.60));

    }

    public void run() {

        int choice;

        do {

            displayMenu();

            choice = getIntInput("Enter your choice: ");

            switch (choice) {

                case 1:

                    toggleDataStructure();

                    break;

                case 2:

                    addNewStock();

                    break;

                case 3:

                    removeStock();

                    break;

                case 4:

                    findHighestPrice();

                    break;

                case 5:

                    findLowestPrice();

                    break;

                case 6:

                    displayAllStocks();

                    break;

                case 7:

                    System.out.println("Exiting program. Goodbye!");

                    break;

                default:

                    System.out.println("Invalid choice. Please try again.");

            }

            System.out.println();

        } while (choice != 7);

        scanner.close();

    }

    private void displayMenu() {

        System.out.println("\t\t\t\t\t\t\t============================================");

        System.out.println("\t\t\t\t\t\t\t\tSTOCK MARKET TRACKING SYSTEM");

        System.out.println("\t\t\t\t\t\t\t============================================");

        System.out.println("\t\t\t\t\t\t\t\tCurrent Data Structure: " + (useArrayList ? "ArrayList" : "LinkedList"));

        System.out.println("1. Toggle Data Structure (ArrayList/LinkedList)");

        System.out.println("2. Add New Stock Price");

        System.out.println("3. Remove Stock");

        System.out.println("4. Find Highest Price");

        System.out.println("5. Find Lowest Price");

        System.out.println("6. Display All Stocks");

        System.out.println("7. Exit");

        System.out.println("========================================");

    }

    private void toggleDataStructure() {

        useArrayList = !useArrayList;

        System.out.println("Switched to " + (useArrayList ? "( ArrayList )" : "( LinkedList )"));

    }

    private void addNewStock() {

        String name = getStringInput("Enter stock name: ");

        double price = getDoubleInput("Enter stock price: ");

        Stock stock = new Stock(name, price);

        addStock(stock);

        System.out.println("Stock added successfully!");

    }

    private void addStock(Stock stock) {

        stockArrayList.add(stock);

        stockLinkedList.add(stock);

    }

    private void removeStock() {

        if (isEmpty()) return;

        displayAllStocks();

        int index = getIntInput("Enter index of stock to remove: ");

        if (index >= 0 && index < (useArrayList ? stockArrayList.size() : stockLinkedList.size())) {

            if (useArrayList) {

                Stock removed = stockArrayList.remove(index);

                stockLinkedList.remove(removed);

                System.out.println("Removed: " + removed);

            } else {

                Stock removed = stockLinkedList.remove(index);

                stockArrayList.remove(removed);

                System.out.println("Removed: " + removed);

            }

        } else {

            System.out.println("Invalid index!");

        }

    }

    private void findHighestPrice() {

        if (isEmpty()) return;

        Stock highest;

        if (useArrayList) {

            highest = Collections.max(stockArrayList, (s1, s2) -> Double.compare(s1.getPrice(), s2.getPrice()));

        } else {

            highest = Collections.max(stockLinkedList, (s1, s2) -> Double.compare(s1.getPrice(), s2.getPrice()));

        }

        System.out.println("Highest Price Stock: " + highest);

    }

    private void findLowestPrice() {

        if (isEmpty()) return;

        Stock lowest;

        if (useArrayList) {

            lowest = Collections.min(stockArrayList, (s1, s2) -> Double.compare(s1.getPrice(), s2.getPrice()));

        } else {

            lowest = Collections.min(stockLinkedList, (s1, s2) -> Double.compare(s1.getPrice(), s2.getPrice()));

        }

        System.out.println("Lowest Price Stock: " + lowest);

    }

    private void displayAllStocks() {

        if (isEmpty()) return;

        System.out.println("Current Stocks:");

        if (useArrayList) {

            for (int i = 0; i < stockArrayList.size(); i++) {

                System.out.println(i + ": " + stockArrayList.get(i));

            }

        } else {

            for (int i = 0; i < stockLinkedList.size(); i++) {

                System.out.println(i + ": " + stockLinkedList.get(i));

            }

        }

    }

    private boolean isEmpty() {

        boolean empty = useArrayList ? stockArrayList.isEmpty() : stockLinkedList.isEmpty();

        if (empty) {

            System.out.println("No stocks available!");

        }

        return empty;

    }

    private int getIntInput(String prompt) {

        int value;

        while (true) {

            System.out.print(prompt);

            String input = scanner.nextLine().trim();

            try {

                value = Integer.parseInt(input);

                if (value >= 1 && value <= 7) {

                    return value;

                } else {

                    System.out.println("Please enter a number between 1 and 7.");

                }

            } catch (NumberFormatException e) {

                System.out.println("Invalid input. Please enter a valid number.");

            }

        }

    }

    private double getDoubleInput(String prompt) {

        double value;

        while (true) {

            System.out.print(prompt);

            String input = scanner.nextLine().trim();

            try {

                value = Double.parseDouble(input);

                return value;

            } catch (NumberFormatException e) {

                System.out.println("Invalid number. Please enter a valid price.");

            }

        }

    }

    private String getStringInput(String prompt) {

        System.out.print(prompt);

        return scanner.nextLine().trim();

    }

}

public class scenario2 {

    public static void main(String[] args) {

        StockTracker tracker = new StockTracker();

        tracker.run();

    }

}

# Time Complexity Analysis and Data Structure Comparison

## Time Complexity Comparison

|  |  |  |
| --- | --- | --- |
| Operation | ArrayList | LinkedList |
| Insertion (at end) | O(1) amortized | O(1) |
| Insertion (at position) | O(n) | O(n) |
| Deletion (by index) | O(n) | O(n) |
| Search for Min/Max | O(n) | O(n) |
| Random Access (get i) | O(1) | O(n) |
| Iteration | O(n) | O(n) |

## Observations

- Both ArrayList and LinkedList require a linear scan (O(n)) to find the minimum or maximum stock price, as the data is unsorted.  
- Insertions at the end are fast for both structures, but inserting/deleting in the middle is slower.

## Would a Sorted Data Structure Improve Performance?

Yes, using a sorted data structure (like TreeSet, PriorityQueue, or a self-balancing BST) can improve performance if min/max operations are frequent.

## Unsorted List vs Sorted Data Structure

|  |  |  |
| --- | --- | --- |
| Feature | Unsorted List | Sorted Data Structure (e.g., TreeSet) |
| Add new element | O(1) | O(log n) |
| Delete an element | O(n) | O(log n) |
| Find Min/Max | O(n) | O(1) |
| Search by price | O(n) | O(log n) |

## Conclusion

- If you frequently need to find min/max stock prices or need range queries, a sorted data structure is better.  
- For a simple stock tracker with occasional min/max search and mostly append operations, ArrayList is sufficient.  
- For a large-scale real-time stock system, using PriorityQueue, TreeMap, or custom sorted trees is recommended.