**1ST ASSIGNMENT**

**Data Structure and Algorithm (Java)**

**L1F23BSSE0389**

SCENARIO 01 (E-COMMERCE WEBSITE)

Implementation Using ArrayList

import java.util.ArrayList;

import java.util.Scanner;

class CustomerOrder {

    int id;

    double amount;

    public CustomerOrder(int id, double amount) {

        this.id = id;

        this.amount = amount;

    }

    public void displayDetails() {

        System.out.println("Order ID: " + id);

        System.out.println("Total Amount: $" + amount);

    }

}

class EcommerceOrderSystem {

    ArrayList<CustomerOrder> orderQueue;

    public EcommerceOrderSystem() {

        orderQueue = new ArrayList<>();

    }

    public void insertOrder(CustomerOrder order) {

        orderQueue.add(0, order); // Adds at the front (FIFO)

        System.out.println("New order placed successfully.");

    }

    public void handleOrder() {

        if (orderQueue.isEmpty()) {

            System.out.println("No orders to process.");

        } else {

            CustomerOrder lastOrder = orderQueue.get(orderQueue.size() - 1);

            lastOrder.displayDetails();

            orderQueue.remove(orderQueue.size() - 1); // Remove from end (FIFO)

            System.out.println("Order processed.");

        }

    }

    public void viewLatestOrder() {

        if (orderQueue.isEmpty()) {

            System.out.println("No recent order found.");

        } else {

            orderQueue.get(0).displayDetails(); // Most recent order

        }

    }

}

public class OrderSystemArrayList {

    public static void main(String[] args) {

        EcommerceOrderSystem system = new EcommerceOrderSystem();

        Scanner input = new Scanner(System.in);

        int option;

        System.out.println("\n=== Welcome to E-Shop Order Manager ===");

        while (true) {

            System.out.println("\n1. Add New Order");

            System.out.println("2. Process Next Order");

            System.out.println("3. View Recent Order");

            System.out.println("4. Exit");

            System.out.print("Choose your action: ");

            option = input.nextInt();

            switch (option) {

                case 1:

                    System.out.print("Enter Order ID: ");

                    int id = input.nextInt();

                    System.out.print("Enter Total Amount: ");

                    double amt = input.nextDouble();

                    system.insertOrder(new CustomerOrder(id, amt));

                    break;

                case 2:

                    System.out.println("Processing the next order...");

                    system.handleOrder();

                    break;

                case 3:

                    System.out.println("Most recent order information:");

                    system.viewLatestOrder();

                    break;

                case 4:

                    System.out.println("Thank you for using E-Shop Order Manager!");

                    return;

                default:

                    System.out.println("Invalid choice. Try again.");

            }

        }

    }

}

Implementation Using LinkedList

import java.util.LinkedList;

import java.util.Scanner;

class PurchaseOrder {

    int orderNum;

    double price;

    public PurchaseOrder(int orderNum, double price) {

        this.orderNum = orderNum;

        this.price = price;

    }

    public void showOrderInfo() {

        System.out.println("Order Number: " + orderNum);

        System.out.println("Order Total: $" + price);

    }

}

class OrderHandler {

    LinkedList<PurchaseOrder> orderRecords;

    public OrderHandler() {

        orderRecords = new LinkedList<>();

    }

    public void queueOrder(PurchaseOrder order) {

        orderRecords.addFirst(order); // FIFO: Add to the start

        System.out.println("Order successfully queued.");

    }

    public void dispatchOrder() {

        if (orderRecords.isEmpty()) {

            System.out.println("Order queue is empty.");

        } else {

            PurchaseOrder last = orderRecords.getLast();

            last.showOrderInfo();

            orderRecords.removeLast(); // Remove from end (FIFO)

            System.out.println("Order dispatched.");

        }

    }

    public void latestOrderInfo() {

        if (orderRecords.isEmpty()) {

            System.out.println("No recent orders available.");

        } else {

            orderRecords.getFirst().showOrderInfo(); // Most recent order

        }

    }

}

public class OrderSystemLinkedList {

    public static void main(String[] args) {

        OrderHandler handler = new OrderHandler();

        Scanner scanner = new Scanner(System.in);

        int userChoice;

        System.out.println("\n=== Online Store Order System ===");

        while (true) {

            System.out.println("\n1. Place Order");

            System.out.println("2. Complete Next Order");

            System.out.println("3. Show Latest Order");

            System.out.println("4. Exit");

            System.out.print("Enter your choice: ");

            userChoice = scanner.nextInt();

            switch (userChoice) {

                case 1:

                    System.out.print("Order Number: ");

                    int number = scanner.nextInt();

                    System.out.print("Order Price: ");

                    double price = scanner.nextDouble();

                    handler.queueOrder(new PurchaseOrder(number, price));

                    break;

                case 2:

                    System.out.println("Dispatching next order...");

                    handler.dispatchOrder();

                    break;

                case 3:

                    System.out.println("Details of latest order:");

                    handler.latestOrderInfo();

                    break;

                case 4:

                    System.out.println("Session ended. Goodbye!");

                    return;

                default:

                    System.out.println("Please enter a valid option.");

            }

        }

    }

}

Time Complexity Analysis

|  |  |  |
| --- | --- | --- |
| Operation | ArrayList | LinkedList |
| AddFirst(Order) | **O(n)** because shifting required | **O(1)** as NO shifting required |
| RemoveLast() | O(1) | O(1) |
| getMostRecentOrder() | O(1) | O(1) |

Which implementation is better?

**LinkedList is better** for FIFO order processing because it takes **O(1)** time complexity for addition at first.

Scenario 02 (Stock Market Application)

Implementation Using ArrayList

import java.util.ArrayList;

import java.util.Scanner;

class PriceTracker {

    ArrayList<Double> stockPrices;

    public PriceTracker() {

        stockPrices = new ArrayList<>();

    }

    public void insertPrice(double price) {

        stockPrices.add(price);  // O(1) amortized

        System.out.println("New stock price recorded!");

    }

    public void deleteOldestPrice() {

        if (stockPrices.isEmpty()) {

            System.out.println("No prices to delete.");

        } else {

            stockPrices.remove(0);  // O(n) due to shifting

            System.out.println("Old price entry removed.");

        }

    }

    public double fetchMaxPrice() {

        if (stockPrices.isEmpty()) return -1;

        double max = Double.MIN\_VALUE;

        for (double price : stockPrices) {

            if (price > max) max = price;

        }

        return max;

    }

    public double fetchMinPrice() {

        if (stockPrices.isEmpty()) return -1;

        double min = stockPrices.get(0);

        for (double price : stockPrices) {

            if (price < min) min = price;

        }

        return min;

    }

}

public class ArrayListStockTracker {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        PriceTracker tracker = new PriceTracker();

        int option;

        System.out.println("📈 Welcome to Live Stock Price Monitor 📈");

        while (true) {

            System.out.println("\n1. Add Stock Price");

            System.out.println("2. Remove Oldest Price");

            System.out.println("3. Show Highest Price");

            System.out.println("4. Show Lowest Price");

            System.out.println("5. Exit");

            System.out.print("Choose an option: ");

            option = scanner.nextInt();

            switch (option) {

                case 1:

                    System.out.print("Enter stock price: ");

                    double price = scanner.nextDouble();

                    tracker.insertPrice(price);

                    break;

                case 2:

                    tracker.deleteOldestPrice();

                    break;

                case 3:

                    double max = tracker.fetchMaxPrice();

                    if (max == -1)

                        System.out.println("No data available.");

                    else

                        System.out.println("Highest recorded price: " + max);

                    break;

                case 4:

                    double min = tracker.fetchMinPrice();

                    if (min == -1)

                        System.out.println("No data available.");

                    else

                        System.out.println("Lowest recorded price: " + min);

                    break;

                case 5:

                    System.out.println("📉 Exiting Stock Price Monitor. Goodbye!");

                    return;

                default:

                    System.out.println("Invalid input. Try again.");

            }

        }

    }

}

Implementation Using LinkedList

import java.util.LinkedList;

import java.util.Scanner;

class PriceMonitor {

    LinkedList<Double> priceHistory;

    public PriceMonitor() {

        priceHistory = new LinkedList<>();

    }

    public void logPrice(double stockPrice) {

        priceHistory.add(stockPrice);  // O(1)

        System.out.println("Price logged successfully!");

    }

    public void discardOldPrice() {

        if (priceHistory.isEmpty()) {

            System.out.println("Price list is already empty.");

        } else {

            priceHistory.removeFirst();  // O(1)

            System.out.println("Oldest price entry removed.");

        }

    }

    public double highestPrice() {

        if (priceHistory.isEmpty()) return -1;

        double max = Double.MIN\_VALUE;

        for (double price : priceHistory) {

            if (price > max) max = price;

        }

        return max;

    }

    public double lowestPrice() {

        if (priceHistory.isEmpty()) return -1;

        double min = priceHistory.getFirst();

        for (double price : priceHistory) {

            if (price < min) min = price;

        }

        return min;

    }

}

public class LinkedListPriceMonitor {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        PriceMonitor monitor = new PriceMonitor();

        int choice;

        System.out.println("📊 Real-Time Stock Tracker System 📊");

        while (true) {

            System.out.println("\n1. Insert Price");

            System.out.println("2. Remove Old Entry");

            System.out.println("3. View Maximum Price");

            System.out.println("4. View Minimum Price");

            System.out.println("5. Exit");

            System.out.print("Enter your selection: ");

            choice = input.nextInt();

            switch (choice) {

                case 1:

                    System.out.print("Input current stock price: ");

                    double val = input.nextDouble();

                    monitor.logPrice(val);

                    break;

                case 2:

                    monitor.discardOldPrice();

                    break;

                case 3:

                    double highest = monitor.highestPrice();

                    if (highest == -1)

                        System.out.println("No price data found.");

                    else

                        System.out.println("Maximum stock price: " + highest);

                    break;

                case 4:

                    double lowest = monitor.lowestPrice();

                    if (lowest == -1)

                        System.out.println("No price data found.");

                    else

                        System.out.println("Minimum stock price: " + lowest);

                    break;

                case 5:

                    System.out.println("📉 Logging out from Stock Tracker. Stay updated!");

                    return;

                default:

                    System.out.println("Oops! Invalid choice.");

            }

        }

    }

}

Time Complexity Analysis

|  |  |  |
| --- | --- | --- |
| Operation | ArrayList | LinkedList |
| addPrice(price) | O(1) amortized | O(1) |
| removeFirst() | O(n) as shifting required | O(1) |
| Get minimum/maximum | O(n) | O(n) |

Would a Sorted Data Structure improve performance?

**Definitely Yes**, because we can access the minimum price by just “list.get(0)” that takes O(1) time complexity, and the maximum by just “list.get(size – 1)” that also takes O(1) time complexity.

Scenario 03 (Train Booking System)

Implementation Using Array

import java.util.Scanner;

class FixedSeatReservation {

    String[] seatChart;

    int capacity;

    public FixedSeatReservation(int totalSeats) {

        capacity = totalSeats;

        seatChart = new String[totalSeats];

    }

    public void reserveSeat(int seatNo, String name) { // O(1)

        if (seatNo >= 0 && seatNo < capacity) {

            if (seatChart[seatNo] == null) {

                seatChart[seatNo] = name;

                System.out.println("Seat " + seatNo + " reserved for " + name);

            } else {

                System.out.println(" Seat already taken.");

            }

        } else {

            System.out.println("Invalid seat number.");

        }

    }

    public void cancelReservation(int seatNo) { // O(1)

        if (seatNo >= 0 && seatNo < capacity) {

            if (seatChart[seatNo] != null) {

                seatChart[seatNo] = null;

                System.out.println("Reservation for seat " + seatNo + " canceled.");

            } else {

                System.out.println("Seat was never reserved.");

            }

        } else {

            System.out.println("Invalid seat number.");

        }

    }

    public void displayVacantSeats() { // O(n)

        boolean empty = true;

        System.out.print("Available Seats: ");

        for (int i = 0; i < capacity; i++) {

            if (seatChart[i] == null) {

                System.out.print(i + " ");

                empty = false;

            }

        }

        if (empty) System.out.println("All seats are occupied.");

        System.out.println();

    }

}

public class TrainBookingArrayVersion {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        FixedSeatReservation bookingSystem = new FixedSeatReservation(10);

        int option;

        System.out.println("Fixed Seat Train Reservation System ");

        while (true) {

            System.out.println("\n1. Reserve Seat");

            System.out.println("2. Cancel Seat");

            System.out.println("3. Show Available Seats");

            System.out.println("4. Exit");

            System.out.print("Select choice: ");

            option = input.nextInt();

            switch (option) {

                case 1:

                    System.out.print("Enter seat number (0-9): ");

                    int seatNo = input.nextInt();

                    input.nextLine(); // clear buffer

                    System.out.print("Enter passenger name: ");

                    String name = input.nextLine();

                    bookingSystem.reserveSeat(seatNo, name);

                    break;

                case 2:

                    System.out.print("Enter seat number to cancel: ");

                    int cancelNo = input.nextInt();

                    bookingSystem.cancelReservation(cancelNo);

                    break;

                case 3:

                    bookingSystem.displayVacantSeats();

                    break;

                case 4:

                    System.out.println("Thank you for using the reservation system!");

                    return;

                default:

                    System.out.println(" Invalid selection.");

            }

        }

    }

}

Implementation Using ArrayList

import java.util.ArrayList;

import java.util.Scanner;

class DynamicSeatBooking {

    ArrayList<String> seatList;

    public DynamicSeatBooking(int totalSeats) {

        seatList = new ArrayList<>(totalSeats);

        for (int i = 0; i < totalSeats; i++) {

            seatList.add(null);

        }

    }

    public void assignSeat(int seatIndex, String passenger) { // O(1)

        if (seatIndex >= 0 && seatIndex < seatList.size()) {

            if (seatList.get(seatIndex) == null) {

                seatList.set(seatIndex, passenger);

                System.out.println(" Seat " + seatIndex + " successfully assigned to " + passenger);

            } else {

                System.out.println(" Seat already assigned.");

            }

        } else {

            System.out.println(" Invalid seat number.");

        }

    }

    public void removeBooking(int seatIndex) { // O(1)

        if (seatIndex >= 0 && seatIndex < seatList.size()) {

            if (seatList.get(seatIndex) != null) {

                seatList.set(seatIndex, null);

                System.out.println("Booking removed from seat " + seatIndex);

            } else {

                System.out.println("Seat was already free.");

            }

        } else {

            System.out.println("Invalid seat number.");

        }

    }

    public void showEmptySeats() { // O(n)

        System.out.print("Seats available: ");

        boolean found = false;

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

            if (seatList.get(i) == null) {

                System.out.print(i + " ");

                found = true;

            }

        }

        if (!found) System.out.println("No empty seats left!");

        System.out.println();

    }

}

public class TrainBookingArrayListVersion {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        DynamicSeatBooking trainSystem = new DynamicSeatBooking(10);

        int menuChoice;

        System.out.println(" Dynamic Train Seat Booking System 🚆");

        while (true) {

            System.out.println("\n1. Book Seat");

            System.out.println("2. Cancel Seat");

            System.out.println("3. View Empty Seats");

            System.out.println("4. Exit");

            System.out.print("Choose an action: ");

            menuChoice = scanner.nextInt();

            switch (menuChoice) {

                case 1:

                    System.out.print("Seat number (0–9): ");

                    int seat = scanner.nextInt();

                    scanner.nextLine(); // clear buffer

                    System.out.print("Passenger name: ");

                    String user = scanner.nextLine();

                    trainSystem.assignSeat(seat, user);

                    break;

                case 2:

                    System.out.print("Seat number to cancel: ");

                    int cancelSeat = scanner.nextInt();

                    trainSystem.removeBooking(cancelSeat);

                    break;

                case 3:

                    trainSystem.showEmptySeats();

                    break;

                case 4:

                    System.out.println("Goodbye from Train Booking System!");

                    return;

                default:

                    System.out.println(" Invalid selection. Try again.");

            }

        }

    }

}

Time Complexity Analysis

|  |  |  |
| --- | --- | --- |
| Operation | ArrayList | LinkedList |
| bookSeat(seatNumber) | O(1) | O(1) |
| cancelSeat(seatNumber) | O(1) | O(1) |
| searchAvailableSeats() | O(n) | O(n) |

Which approach is better for a large-scale system?

ArrayList-based approach is better for large-scale train booking systems because:

1. Scalability: Dynamically grows to accommodate more seats or trains.

2. Flexibility: Supports future enhancements with minimal changes.

Scenario 04 (Social Medial App comment section)

Implementation Using LinkedList

import java.util.LinkedList;

import java.util.Scanner;

class SocialComments {

    private LinkedList<String> commentList;

    public SocialComments() {

        commentList = new LinkedList<>();

    }

    // Adds a new comment at the end (chronological)

    public void postComment(String message) { // O(1)

        commentList.add(message);

        System.out.println("Comment posted: \"" + message + "\"");

    }

    // Edits a specific comment by index

    public void updateComment(int pos, String updatedMessage) { // O(n)

        if (pos >= 0 && pos < commentList.size()) {

            commentList.set(pos, updatedMessage);

            System.out.println("Comment updated at position " + pos);

        } else {

            System.out.println("Invalid comment number.");

        }

    }

    // Deletes a comment

    public void removeComment(int index) { // O(n)

        if (index >= 0 && index < commentList.size()) {

            String removed = commentList.remove(index);

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

        } else {

            System.out.println("Comment does not exist at given position.");

        }

    }

    // Searches for comments by keyword

    public void findComment(String searchText) { // O(n)

        boolean matchFound = false;

        System.out.println("Searching for: \"" + searchText + "\"");

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

            if (commentList.get(i).toLowerCase().contains(searchText.toLowerCase())) {

                System.out.println("Found at [" + i + "]: " + commentList.get(i));

                matchFound = true;

            }

        }

        if (!matchFound) {

            System.out.println("⚠️ No comments matched your search.");

        }

    }

    // Show comments from newest to oldest

    public void showLatestFirst() { // O(n)

        System.out.println("🕘 Recent Comments:");

        for (int i = commentList.size() - 1; i >= 0; i--) {

            System.out.println("[" + i + "] " + commentList.get(i));

        }

    }

    // Show comments in the order they were posted

    public void showChronologically() { // O(n)

        System.out.println("All Comments (Oldest to Newest):");

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

            System.out.println("[" + i + "] " + commentList.get(i));

        }

    }

}

public class Scenario4UsingLinkedList {

    public static void main(String[] args) {

        SocialComments sc = new SocialComments();

        Scanner input = new Scanner(System.in);

        int option;

        System.out.println("=== Social App: Comment Manager ===");

        while (true) {

            System.out.println("\nChoose an action:");

            System.out.println("1. Post Comment");

            System.out.println("2. Edit Comment");

            System.out.println("3. Delete Comment");

            System.out.println("4. Search by Keyword");

            System.out.println("5. Show Recent Comments");

            System.out.println("6. Exit");

            System.out.print("Your choice: ");

            option = input.nextInt();

            input.nextLine(); // Clear buffer

            switch (option) {

                case 1:

                    System.out.print("Enter your comment: ");

                    String comment = input.nextLine();

                    sc.postComment(comment);

                    break;

                case 2:

                    System.out.print("Comment number to edit: ");

                    int editIndex = input.nextInt();

                    input.nextLine();

                    System.out.print("New comment: ");

                    String edited = input.nextLine();

                    sc.updateComment(editIndex, edited);

                    break;

                case 3:

                    System.out.print("Comment number to delete: ");

                    int deleteIndex = input.nextInt();

                    sc.removeComment(deleteIndex);

                    break;

                case 4:

                    System.out.print("Enter keyword to find: ");

                    String keyword = input.nextLine();

                    sc.findComment(keyword);

                    break;

                case 5:

                    sc.showLatestFirst();

                    break;

                case 6:

                    System.out.println("Thanks for using the comment manager!");

                    input.close();

                    return;

                default:

                    System.out.println("Please select a valid option.");

            }

        }

    }

}

Time Complexity Analysis

|  |  |
| --- | --- |
| Operation | ArrayList |
| Insertion | O(1) |
| Deletion | O(n) |
| Searching | O(n) |
| Editing | O(n) |
| Display reverse order | O(n) |

Would an ArrayList be a better choice? Why or why not?

Yes, **ArrayList is a better choice** for this comment section system because of

**Editing Efficiency**: O(1) vs. O(n)   
Explanation: For editing comment, ArrayList takes O(1) time complexity

i.e.: set (index, value) is O(1) with direct index access.