# Scenario1

# Code

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.Scanner;

class Order {

    private int orderId;

    private String customerName;

    private double totalAmount;

    private String status;

    public Order(int orderId, String customerName, double totalAmount) {

        this.orderId = orderId;

        this.customerName = customerName;

        this.totalAmount = totalAmount;

        this.status = "Pending";

    }

    public int getOrderId() {

        return orderId;

    }

    public String getCustomerName() {

        return customerName;

    }

    public double getTotalAmount() {

        return totalAmount;

    }

    public String getStatus() {

        return status;

    }

    public void setStatus(String status) {

        this.status = status;

    }

    @Override

    public String toString() {

        return "Order{" +

                "orderId=" + orderId +

                ", customerName='" + customerName + '\'' +

                ", totalAmount=" + totalAmount +

                ", status='" + status + '\'' +

                '}';

    }

}

class ArrayListOrderManager {

    private ArrayList<Order> orders;

    private int nextOrderId;

    public ArrayListOrderManager() {

        orders = new ArrayList<>();

        nextOrderId = 1;

        addDefaultOrders();

    }

    private void addDefaultOrders() {

        orders.add(new Order(nextOrderId++, "Ali ahmad", 125.50));

        orders.add(new Order(nextOrderId++, "Ahmad Ali", 89.99));

        orders.add(new Order(nextOrderId++, "javed", 210.75));

        orders.add(new Order(nextOrderId++, "Abc", 56.25));

        orders.add(new Order(nextOrderId++, "xyz", 310.00));

    }

    public void addOrder(String customerName, double totalAmount) {

        Order newOrder = new Order(nextOrderId++, customerName, totalAmount);

        orders.add(newOrder);

        System.out.println("Order added successfully with ID: " + newOrder.getOrderId());

    }

    public void processOldestOrder() {

        if (orders.isEmpty()) {

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

            return;

        }

        Order processed = orders.remove(0);

        System.out.println("Processed oldest order: " + processed);

    }

    public void deliverOrderById(int orderId) {

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

            if (orders.get(i).getOrderId() == orderId) {

                Order delivered = orders.remove(i);

                System.out.println("Delivered order: " + delivered);

                return;

            }

        }

        System.out.println("Order with ID " + orderId + " not found.");

    }

    public void viewMostRecentOrder() {

        if (orders.isEmpty()) {

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

            return;

        }

        System.out.println("Most recent order: " + orders.get(orders.size() - 1));

    }

    public void findOrderById(int orderId) {

        for (Order order : orders) {

            if (order.getOrderId() == orderId) {

                System.out.println("Found order: " + order);

                return;

            }

        }

        System.out.println("Order with ID " + orderId + " not found.");

    }

    public void displayAllOrders() {

        if (orders.isEmpty()) {

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

            return;

        }

        System.out.println("\nAll Orders:");

        for (Order order : orders) {

            System.out.println(order);

        }

    }

    public int getOrderCount() {

        return orders.size();

    }

}

class LinkedListOrderManager {

    private LinkedList<Order> orders;

    private int nextOrderId;

    public LinkedListOrderManager() {

        orders = new LinkedList<>();

        nextOrderId = 1;

        addDefaultOrders();

    }

    private void addDefaultOrders() {

        orders.add(new Order(nextOrderId++, "Ali ahmad", 125.50));

        orders.add(new Order(nextOrderId++, "Ahmad Ali", 89.99));

        orders.add(new Order(nextOrderId++, "javed", 210.75));

        orders.add(new Order(nextOrderId++, "Abc", 56.25));

        orders.add(new Order(nextOrderId++, "xyz", 310.00));

    }

    public void addOrder(String customerName, double totalAmount) {

        Order newOrder = new Order(nextOrderId++, customerName, totalAmount);

        orders.add(newOrder);

        System.out.println("Order added successfully with ID: " + newOrder.getOrderId());

    }

    public void processOldestOrder() {

        if (orders.isEmpty()) {

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

            return;

        }

        Order processed = orders.removeFirst();

        System.out.println("Processed oldest order: " + processed);

    }

    public void deliverOrderById(int orderId) {

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

            if (orders.get(i).getOrderId() == orderId) {

                Order delivered = orders.remove(i);

                System.out.println("Delivered order: " + delivered);

                return;

            }

        }

        System.out.println("Order with ID " + orderId + " not found.");

    }

    public void viewMostRecentOrder() {

        if (orders.isEmpty()) {

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

            return;

        }

        System.out.println("Most recent order: " + orders.getLast());

    }

    public void findOrderById(int orderId) {

        for (Order order : orders) {

            if (order.getOrderId() == orderId) {

                System.out.println("Found order: " + order);

                return;

            }

        }

        System.out.println("Order with ID " + orderId + " not found.");

    }

    public void displayAllOrders() {

        if (orders.isEmpty()) {

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

            return;

        }

        System.out.println("\nAll Orders:");

        for (Order order : orders) {

            System.out.println(order);

        }

    }

    public int getOrderCount() {

        return orders.size();

    }

}

public class scenario1 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Choose implementation:");

        System.out.println("1. ArrayList Implementation");

        System.out.println("2. LinkedList Implementation");

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

        int implChoice = scanner.nextInt();

        scanner.nextLine();

        if (implChoice == 1) {

            ArrayListOrderManager manager = new ArrayListOrderManager();

            runMenu(scanner, manager, "ArrayList");

        } else if (implChoice == 2) {

            LinkedListOrderManager manager = new LinkedListOrderManager();

            runMenu(scanner, manager, "LinkedList");

        } else {

            System.out.println("Invalid choice, exiting...");

        }

    }

    private static void runMenu(Scanner scanner, Object manager, String type) {

        boolean exit = false;

        while (!exit) {

            System.out.println("\n===== MENU =====");

            System.out.println("1. Display All Orders");

            System.out.println("2. Place New Order");

            System.out.println("3. Process Oldest Order");

            System.out.println("4. Deliver Order by ID");

            System.out.println("5. View Most Recent Order");

            System.out.println("6. Find Order by ID");

            System.out.println("7. Check Order Count");

            System.out.println("0. Exit");

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

            int choice = scanner.nextInt();

            scanner.nextLine();

            switch (choice) {

                case 0:

                    System.out.println("Exiting the system...");

                    exit = true;

                    break;

                case 1:

                    if (type.equals("ArrayList")) {

                        ((ArrayListOrderManager) manager).displayAllOrders();

                    } else {

                        ((LinkedListOrderManager) manager).displayAllOrders();

                    }

                    break;

                case 2:

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

                    String customerName = scanner.nextLine();

                    System.out.print("Enter order amount: $");

                    double amount = scanner.nextDouble();

                    scanner.nextLine();

                    if (type.equals("ArrayList")) {

                        ((ArrayListOrderManager) manager).addOrder(customerName, amount);

                    } else {

                        ((LinkedListOrderManager) manager).addOrder(customerName, amount);

                    }

                    break;

                case 3:

                    if (type.equals("ArrayList")) {

                        ((ArrayListOrderManager) manager).processOldestOrder();

                    } else {

                        ((LinkedListOrderManager) manager).processOldestOrder();

                    }

                    break;

                case 4:

                    System.out.print("Enter order ID to deliver: ");

                    int deliverId = scanner.nextInt();

                    scanner.nextLine();

                    if (type.equals("ArrayList")) {

                        ((ArrayListOrderManager) manager).deliverOrderById(deliverId);

                    } else {

                        ((LinkedListOrderManager) manager).deliverOrderById(deliverId);

                    }

                    break;

                case 5:

                    if (type.equals("ArrayList")) {

                        ((ArrayListOrderManager) manager).viewMostRecentOrder();

                    } else {

                        ((LinkedListOrderManager) manager).viewMostRecentOrder();

                    }

                    break;

                case 6:

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

                    int findId = scanner.nextInt();

                    scanner.nextLine();

                    if (type.equals("ArrayList")) {

                        ((ArrayListOrderManager) manager).findOrderById(findId);

                    } else {

                        ((LinkedListOrderManager) manager).findOrderById(findId);

                    }

                    break;

                case 7:

                    int count;

                    if (type.equals("ArrayList")) {

                        count = ((ArrayListOrderManager) manager).getOrderCount();

                    } else {

                        count = ((LinkedListOrderManager) manager).getOrderCount();

                    }

                    System.out.println("Current order count: " + count);

                    break;

                default:

                    System.out.println("Invalid option, please try again.");

            }

        }

    }

}

# Time Complexity Analysis for Order Management:

Adding Orders

- ArrayList:  
 • Time Complexity: O(1) (amortized) – Adding at the end.  
 • Details: If the internal array is full, it resizes (costly), but infrequent.

- LinkedList:  
 • Time Complexity: O(1) – Appending at the end (uses tail pointer internally).  
 • Details: No resizing overhead.

Removing Orders

- Removing Oldest Order (i.e., from front):  
 • ArrayList: O(n) – All elements must be shifted left.  
 • LinkedList: O(1) – removeFirst() is a constant time operation.

- Removing by ID (middle):  
 • ArrayList & LinkedList: O(n) – Both require traversal to find the order by ID.

## Comparison Table

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | ArrayList (Time) | LinkedList (Time) | Winner |
| Add Order (End) | O(1)\* | O(1) | Tie |
| Remove Oldest Order | O(n) | O(1) | LinkedList |
| Remove by ID | O(n) | O(n) | Tie |
| View Recent Order | O(1) | O(1) | Tie |
| Find Order by ID | O(n) | O(n) | Tie |

\* Amortized: occasional resizing makes some adds more expensive.

**Conclusion**

Best Choice: LinkedList

- Handles frequent processing of oldest orders more efficiently (O(1) vs O(n)).  
- Adding to end is equally efficient.  
- Slightly more memory overhead, but worth it for performance in large-scale order queues