Experiment 4

Student Name: Amika **UID: 22BCS14180**
Branch: B.E CSE **Section: IOT-643-A**
Semester: 6th **DOP:24/02/25**
Subject: PBLJ **Subject Code: 22CSH-359**

Aim:

Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

Problem Statement :

1) Write a Java program to implement an ArrayList that stores employee details (ID, Name, and Salary). Allow users to add, update, remove, and search employees.

2) Create a program to collect and store all the cards to assist the users in finding all the cards in a given symbol using Collection interface.

3) Develop a ticket booking system with synchronized threads to ensure no double booking of seats. Use thread priorities to simulate VIP bookings being processed first.

Program :

1. Employee Management:

```
import java.util.ArrayList;
import java.util.Scanner;

class Employee {
   int id;
   String name;
   double salary;

   Employee(int id, String name, double salary) {
      this.id = id;
      this.name = name;
      this.salary = salary;
   }

   public String toString() {
```

```
        return "ID: " + id + ", Name: " + name + ", Salary: $" + salary;
    }
}

public class EmployeeManager {
    public static void main(String[] args) {
        ArrayList<Employee> employees = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\nEmployee Management System");
            System.out.println("1. Add Employee");
            System.out.println("2. Update Employee");
            System.out.println("3. Remove Employee");
            System.out.println("4. Search Employee");
            System.out.println("5. Display All Employees");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Enter Employee ID: ");
                    int id = scanner.nextInt();
                    scanner.nextLine();

                    System.out.print("Enter Employee Name: ");
                    String name = scanner.nextLine();

                    System.out.print("Enter Employee Salary: ");
                    double salary = scanner.nextDouble();

                    employees.add(new Employee(id, name, salary));
                    System.out.println("Employee added successfully!");
                    break;

                case 2:
                    System.out.print("Enter Employee ID to update: ");
                    int updateId = scanner.nextInt();
                    scanner.nextLine();
```

```
            boolean foundUpdate = false;
            for (Employee emp : employees) {
                if (emp.id == updateId) {
                    System.out.print("Enter New Name: ");
                    emp.name = scanner.nextLine();
                    System.out.print("Enter New Salary: ");
                    emp.salary = scanner.nextDouble();
                    System.out.println("Employee updated successfully!");
                    foundUpdate = true;
                    break;
                }
            }
            if (!foundUpdate) System.out.println("Employee not found!");
            break;

        case 3:
            System.out.print("Enter Employee ID to remove: ");
            int removeId = scanner.nextInt();
            boolean removed = employees.removeIf(emp -> emp.id == removeId);

            if (removed) {
                System.out.println("Employee removed successfully!");
            } else {
                System.out.println("Employee not found!");
            }
            break;

        case 4:
            System.out.print("Enter Employee ID to search: ");
            int searchId = scanner.nextInt();
            boolean found = false;

            for (Employee emp : employees) {
                if (emp.id == searchId) {
                    System.out.println("Employee Found: " + emp);
                    found = true;
                    break;
                }
            }
            if (!found) System.out.println("Employee not found!");
            break;

        case 5:
```

```
                    if (employees.isEmpty()) {
                        System.out.println("No employees found!");
                    } else {
                        System.out.println("Employee List:");
                        for (Employee emp : employees) {
                            System.out.println(emp);
                        }
                    }
                    break;

                case 6:
                    System.out.println("Exiting program...");
                    scanner.close();
                    return;

                default:
                    System.out.println("Invalid choice! Try again.");
                }
            }
        }
    }}
    System.out.println("Employee not found.");
}
public static void main(String[] args) {
    while (true) {
        System.out.println("\n1. Add Employee\n2. Update Employee\n3. Remove
Employee\n4. Search Employee\n5. Exit");
        System.out.print("Choose an option: ");
        int choice = scanner.nextInt();
        switch (choice) {
            case 1 -> addEmployee();
            case 2 -> updateEmployee();
            case 3 -> removeEmployee();
            case 4 -> searchEmployee();
            case 5 -> System.exit(0);
            default -> System.out.println("Invalid choice! Try again.");
    } } } }
```

Output:

input

```
Enter Employee Name: Riya
Enter Employee Salary: 50000
Employee added successfully!

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 2
Enter Employee ID to update: 180
Enter New Name: Shreya
Enter New Salary: 60000
Employee updated successfully!

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 6
Exiting program...
```

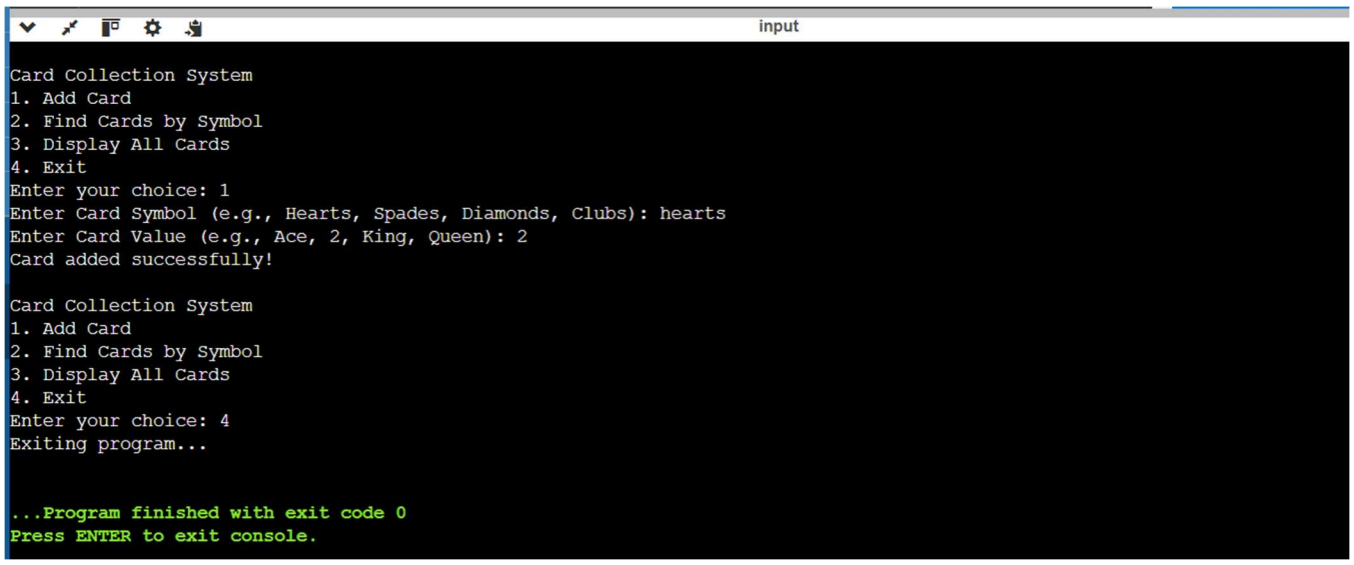
2. Card Collection :

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class Card {
    String symbol;
    String value;

    Card(String symbol, String value) {
        this.symbol = symbol;
        this.value = value;
    }

    public String toString() {
        return value + " of " + symbol;
    }
}
public class CardCollection {
    public static void main(String[] args) {
        List<Card> cards = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);
      while (true) {
            System.out.println("\nCard Collection System");
            System.out.println("1. Add Card");
            System.out.println("2. Find Cards by Symbol");
            System.out.println("3. Display All Cards");
            System.out.println("4. Exit");
```

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

int choice = scanner.nextInt();
scanner.nextLine();

if (choice == 1) {
    System.out.print("Enter Card Symbol (e.g., Hearts, Spades, Diamonds, Clubs): ");
    String symbol = scanner.nextLine();

    System.out.print("Enter Card Value (e.g., Ace, 2, King, Queen): ");
    String value = scanner.nextLine();

    cards.add(new Card(symbol, value));
    System.out.println("Card added successfully!");

} else if (choice == 2) {
    System.out.print("Enter Symbol to search: ");
    String searchSymbol = scanner.nextLine();
    boolean found = false;

    for (Card card : cards) {
        if (card.symbol.equalsIgnoreCase(searchSymbol)) {
            System.out.println(card);
            found = true;
        }
    }
    if (!found) {
        System.out.println("No cards found for this symbol.");
    }

} else if (choice == 3) {
    if (cards.isEmpty()) {
        System.out.println("No cards in the collection.");
    } else {
        System.out.println("All Cards:");
        for (Card card : cards) {
            System.out.println(card);
        }
    }

} else if (choice == 4) {
    System.out.println("Exiting program...");
    scanner.close();
    break;
```

```
            } else {
                System.out.println("Invalid choice. Try again.");
            }
        }
    }
}
```

Output:



3. Ticket Booking System:

```
import java.util.concurrent.PriorityBlockingQueue;

class Ticket implements Comparable<Ticket> {
    int seatNumber;
    String passengerName;
    boolean isVip;

    Ticket(int seatNumber, String passengerName, boolean isVip) {
        this.seatNumber = seatNumber;
        this.passengerName = passengerName;
        this.isVip = isVip;
    }

    public String toString() {
        return "Seat " + seatNumber + " booked for " + passengerName + (isVip ? " (VIP)" : "");
    }

    public int compareTo(Ticket other) {
        return Boolean.compare(other.isVip, this.isVip);
```

```
    }
}

class TicketBookingSystem {
    private final boolean[] seats;
    private final PriorityBlockingQueue<Ticket> bookingQueue = new
PriorityBlockingQueue<>();

    TicketBookingSystem(int totalSeats) {
        seats = new boolean[totalSeats];
    }

    public synchronized void bookTicket(int seatNumber, String passengerName, boolean isVip)
{
        if (seatNumber < 1 || seatNumber > seats.length || seats[seatNumber - 1]) {
            System.out.println(passengerName + " booking failed for seat " + seatNumber);
            return;
        }

        seats[seatNumber - 1] = true;
        bookingQueue.add(new Ticket(seatNumber, passengerName, isVip));
        System.out.println(passengerName + " successfully booked seat " + seatNumber + (isVip ?
" (VIP)" : ""));
    }

    public void processBookings() {
        while (!bookingQueue.isEmpty()) {
            Ticket ticket = bookingQueue.poll();
            System.out.println(ticket);
        }
    }
}

class BookingThread extends Thread {
    private final TicketBookingSystem system;
    private final int seatNumber;
    private final String passengerName;
    private final boolean isVip;

    BookingThread(TicketBookingSystem system, int seatNumber, String passengerName,
boolean isVip, int priority) {
        this.system = system;
        this.seatNumber = seatNumber;
        this.passengerName = passengerName;
        this.isVip = isVip;
        setPriority(priority);
```

```
    }

    public void run() {
        system.bookTicket(seatNumber, passengerName, isVip);
    }
}

public class TicketBookingApp {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem(5);

        BookingThread t1 = new BookingThread(system, 1, "Alice", false,
Thread.MIN_PRIORITY);
        BookingThread t2 = new BookingThread(system, 2, "Bob", false,
Thread.MIN_PRIORITY);
        BookingThread t3 = new BookingThread(system, 3, "Charlie", true,
Thread.MAX_PRIORITY);
        BookingThread t4 = new BookingThread(system, 4, "David", true,
Thread.MAX_PRIORITY);
        BookingThread t5 = new BookingThread(system, 5, "Eve", false,
Thread.NORM_PRIORITY);

        t3.start();
        t4.start();
        t1.start();
        t2.start();
        t5.start();

        try {
            t1.join();
            t2.join();
            t3.join();
            t4.join();
            t5.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("\nFinal Booking List:");
        system.processBookings();
    }
}
```

Output:

input

```
Charlie successfully booked seat 3 (VIP)
Bob successfully booked seat 2
Eve successfully booked seat 5
Alice successfully booked seat 1
David successfully booked seat 4 (VIP)

Final Booking List:
Seat 3 booked for Charlie (VIP)
Seat 4 booked for David (VIP)
Seat 1 booked for Alice
Seat 5 booked for Eve
Seat 2 booked for Bob


...Program finished with exit code 0
Press ENTER to exit console.
```

Learning Outcomes:

- **Object-Oriented Design** (Classes for real-world entities)
- **Core Programming Skills** (Loops, conditionals, methods for inventory operations)
- **Data Structure Usage** (ArrayList for dynamic data management)
- **User-Friendly Systems** (Intuitive interface with error handling)