Experiment - 4

Student Name: Pranav UID: 22BCS50037
Branch: B.ECSE Section: IOT-642-A
Semester: 6th DOP: 24/02/25
Subject: PBLJ Subject Code: 22CSH-359

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

2) Problem Statement:
   a. Write a Java program to implement an Array List that stores employee details (ID, Name, and Salary). Allow users to add, update, remove, and search employees.
   b. 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.
   c. 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.

3) Algorithm:

a. Employee Management (Using Array List)

- Initialize an Array List to store employees.
- Display a menu with options: Add, Update, Remove, Search, and Exit.
- Add Employee:
  - Take user input for ID, Name, and Salary.
  - Create an Employee object and add it to the list.
- Update Employee:
  - Ask for the Employee ID.
  - If found, update Name and Salary.
- Remove Employee:
  - Ask for the Employee ID.
  - Remove matching employee from the list.
- Search Employee:
  - Ask for the Employee ID.
  - If found, display details.
- Repeat until the user chooses to exit.

b. Card Collection (Using Collections)

- Initialize an Array List to store Card objects.
- Display a menu with options: Add Card, Find Cards by Symbol, and Exit.
- Add Card:
  - Ask for card symbol (e.g., Hearts, Diamonds).

- Ask for card value(A,2,3,...J,Q,K).
- Create a Card object and store it in the list.

 Find Cards by Symbol:

- Ask for a symbol.
- Search and display all cards with that symbol.

 Repeat until the user chooses to exit.

c. Ticket Booking System (Multithreading)

 Create a Ticket Booking System with a limited number of seats.
 Implement synchronized booking to prevent double booking.
 Create Customer threads with different priorities (VIP first).
 Each Customer thread:

- Tries to book a ticket.
- If seats are available, booking is confirmed, and the seat count decreases.
- If not, booking fails.

 Start all customer threads and process bookings.
 Stop when all threads have completed execution.

4) Program:

a. Employee Management:

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

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

   // Constructor to initialize employee details     public
Employee(int id, String name, double salary) {
this.id = id;         this.name = name;         this.salary =
salary;
   }

   // Method to display employee details
   public void display() {
      System.out.println("ID: " + id + ", Name: " + name + ", Salary: " + salary);
   }
   // Method to update employee details     public void
updateDetails(String name, double salary) {
this.name = name;         this.salary = salary;
   }
}

public class EmployeeManagement {
   private static ArrayList<Employee> employeeList = new ArrayList<>();
private static Scanner scanner = new Scanner(System.in);

   public static void main(String[] args) {
      while (true) {
         System.out.println("\n--- Employee Management ---");
         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("Choose an option: ");
int choice = scanner.nextInt();
         scanner.nextLine(); // Consume newline character

         switch (choice) {
case 1:
               addEmployee();
               break;
case 2:
               updateEmployee();
               break;
case 3:
               removeEmployee();
break;              case 4:
               searchEmployee();
               break;
case 5:
               displayAllEmployees();
               break;
            case 6:
               System.out.println("Exiting...");
               return;
default:
```

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

    // Method to add employee     private static
void addEmployee() {
System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline character
        System.out.print("Enter Employee Name: ");
        String name = scanner.nextLine();
System.out.print("Enter Employee Salary: ");
        double salary = scanner.nextDouble();

        Employee newEmployee = new Employee(id, name, salary);
employeeList.add(newEmployee);
        System.out.println("Employee added successfully.");
    }

    // Method to update employee
private static void updateEmployee() {
        System.out.print("Enter Employee ID to update: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline character

        for (Employee employee : employeeList) {
            if (employee.id == id) {
                System.out.print("Enter new Name: ");
                String name = scanner.nextLine();
System.out.print("Enter new Salary: ");                double
salary = scanner.nextDouble();
employee.updateDetails(name, salary);
System.out.println("Employee details updated.");
return;
            }
        }
        System.out.println("Employee not found.");
    }

    // Method to remove employee
private static void removeEmployee() {
        System.out.print("Enter Employee ID to remove: ");
        int id = scanner.nextInt();

        for (Employee employee : employeeList) {
            if (employee.id == id) {
employeeList.remove(employee);
System.out.println("Employee removed.");
return;
            }
        }
        System.out.println("Employee not found.");
    }

    // Method to search employee by ID
private static void searchEmployee() {
        System.out.print("Enter Employee ID to search: ");
        int id = scanner.nextInt();

        for (Employee employee : employeeList) {
            if (employee.id == id) {
employee.display();
                return;
            }
        }
        System.out.println("Employee not found.");
    }

    // Method to display all employees
private static void displayAllEmployees() {
if (employeeList.isEmpty()) {
            System.out.println("No employees to display.");
        } else {
            for (Employee employee : employeeList) {
                employee.display();
            }
        }
    }
}
```

b. Card Collection:

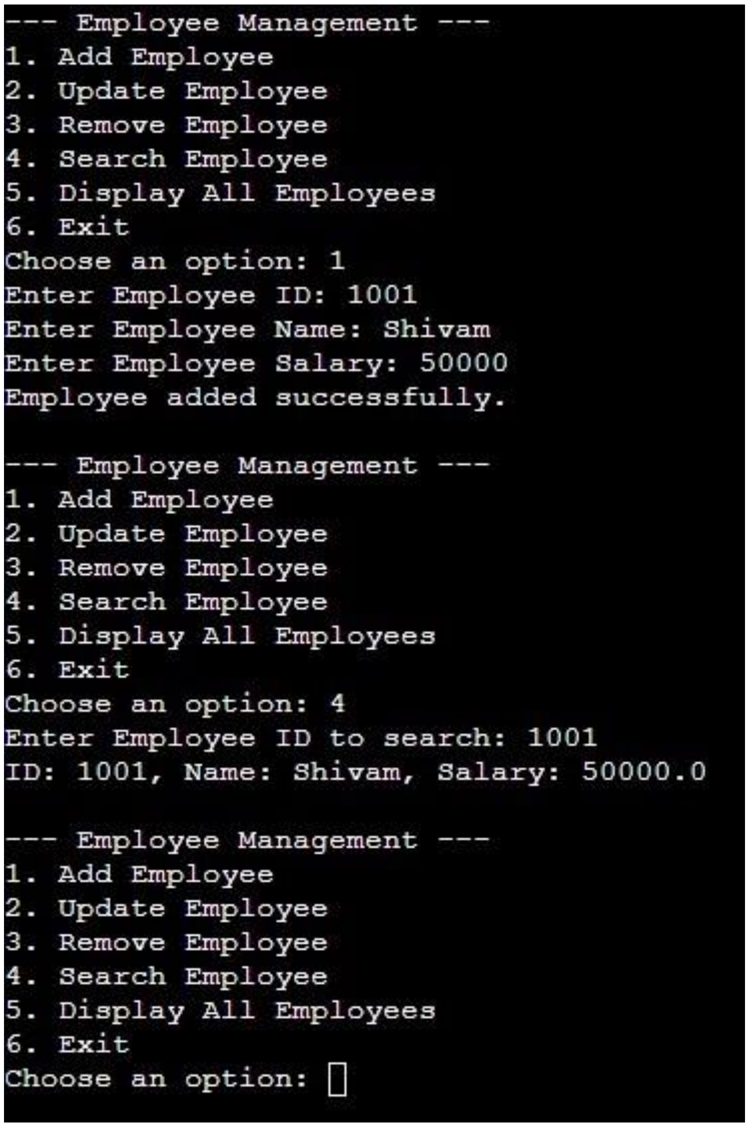
```
import  java.util.ArrayList;
import java.util.Collection;
import java.util.Scanner;

class Card {
private String suit;
   private String rank;

   // Constructor to initialize card details
public Card(String suit, String rank) {
       this.suit = suit;
this.rank = rank;
   }

   // Getter for suit
public String getSuit() {
       return suit;
   }

   // Getter for rank
public String getRank() {
       return rank;
   }

   // Display card details
public void displayCard() {
       System.out.println(rank + " of " + suit);
   }
}

class Deck {
   private Collection<Card> cards;

   // Constructor to initialize the deck
public Deck() {
       cards = new ArrayList<>();
       createDeck();
   }

   // Method to create a standard deck of 52 cards
private void createDeck() {
       String[] suits = {"Hearts", "Diamonds", "Clubs", "Spades"};
       String[] ranks = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King",
"Ace"};

       for (String suit : suits) {
for (String rank : ranks) {
                cards.add(new Card(suit, rank));
            }
       }
   }

   // Method to find cards by suit
   public Collection<Card> findCardsBySuit(String suit) {
Collection<Card> result = new ArrayList<>();
       for (Card card : cards) {
            if (card.getSuit().equalsIgnoreCase(suit)) {
                result.add(card);
            }
}
       return result;
   }

   // Method to display all cards in the deck
public void displayAllCards() {         for
(Card card : cards) {
            card.displayCard();
       }
   }
}

public class Main {
   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);
       Deck deck = new Deck();

       while (true) {
            System.out.println("\n--- Card Deck Management ---");
            System.out.println("1. Display all cards");
            System.out.println("2. Find cards by suit");
            System.out.println("3. Exit");
System.out.print("Choose an option: ");
int choice = scanner.nextInt();
```

```
            scanner.nextLine(); // Consume the newline character

            switch (choice) {
case 1:
                    System.out.println("\nDisplaying all cards in the deck:");
deck.displayAllCards();
                    break;
case 2:
                    System.out.print("Enter the suit (Hearts, Diamonds, Clubs, Spades): ");
                    String suit = scanner.nextLine();
                    Collection<Card> foundCards = deck.findCardsBySuit(suit);
if (foundCards.isEmpty()) {
                        System.out.println("No cards found for the suit: " + suit);
} else {
                        System.out.println("\nCards found with suit " + suit + ":");
for (Card card : foundCards) {
                            card.displayCard();
                        }
}
break;
case 3:
                    System.out.println("Exiting...");
scanner.close();
                    System.out.println("\nMade by Shivam_22BCS50010"); // Display the author
message
                    return;
default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
}
    c. Ticket Booking System:
    class SeatBooking {
        private boolean[] seats;  // Array to represent available seats (true means booked)

        // Constructor to initialize all seats as available
    public SeatBooking(int numberOfSeats) {
            seats = new boolean[numberOfSeats];
        }

        // Synchronized method to book a seat
```

```
   public synchronized boolean bookSeat(int seatNumber)
{        if (seatNumber < 0 || seatNumber >= seats.length) {
System.out.println("Invalid seat number.");             return
false;
        }

        if (seats[seatNumber]) {
            System.out.println("Seat " + seatNumber + " is already booked.");
return false;
        }

        // Book the seat
        seats[seatNumber] = true;
        System.out.println(Thread.currentThread().getName() + " successfully booked seat " +
     seatNumber);
        return true;
    }

    // Method to display available seats
public void displayAvailableSeats() {
System.out.print("Available seats: ");
for (int i = 0; i < seats.length; i++) {
            if (!seats[i]) {
                System.out.print(i + " ");
            }
        }
        System.out.println();
    }
}
class BookingThread extends Thread {
private SeatBooking seatBooking;
private int seatNumber;

    // Constructor
    public BookingThread(SeatBooking seatBooking, int seatNumber, String name)
{        super(name); // Set thread name (VIP or regular)        this.seatBooking =
seatBooking;
        this.seatNumber = seatNumber;
    }
   @Override
public void run() {
try {
            // Simulate some processing time
            Thread.sleep(1000);
        } catch (InterruptedException e) {
e.printStackTrace();
        }

        // Attempt to book the seat
        if (!seatBooking.bookSeat(seatNumber)) {
System.out.println(Thread.currentThread().getName() + " failed to book seat " +
seatNumber);
        }
    }
}
public class Main {
    public static void main(String[] args) {
        SeatBooking seatBooking = new SeatBooking(10);  // Create a booking system with 10
     seats

        // Create threads for VIP and regular customers
        BookingThread vipCustomer = new BookingThread(seatBooking, 2, "VIP Customer");
        BookingThread regularCustomer1 = new BookingThread(seatBooking, 2, "Regular
     Customer 1");
        BookingThread regularCustomer2 = new BookingThread(seatBooking, 3, "Regular
     Customer 2");

        // Set thread priorities (VIP has higher priority)
        vipCustomer.setPriority(Thread.MAX_PRIORITY);  // VIP gets highest priority
regularCustomer1.setPriority(Thread.NORM_PRIORITY);  // Regular customers get normal
priority
        regularCustomer2.setPriority(Thread.NORM_PRIORITY);  // Regular customers get
     normal priority

        // Display available seats before booking
        seatBooking.displayAvailableSeats();

        // Start the threads
vipCustomer.start();
regularCustomer1.start();
regularCustomer2.start();
try {
            // Wait for all threads to finish
vipCustomer.join();
regularCustomer1.join();
regularCustomer2.join();         } catch
(InterruptedException e) {
e.printStackTrace();
        }

        // Display available seats after booking
        seatBooking.displayAvailableSeats();

        // Print author information
        System.out.println("\nMade by Shivam_22BCS50010");
    }
     }
```

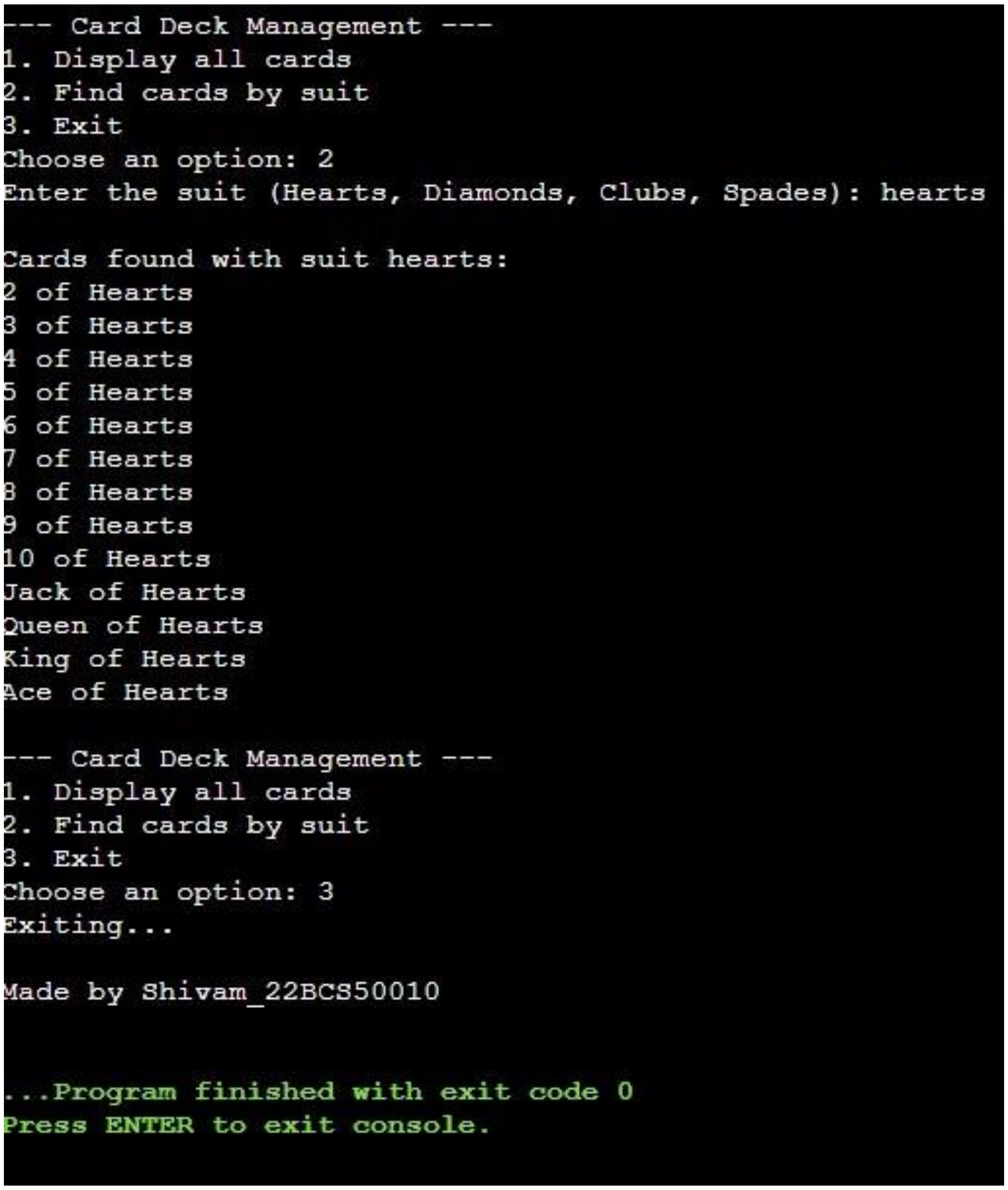
5) OUTPUT:

1. Employee Management:



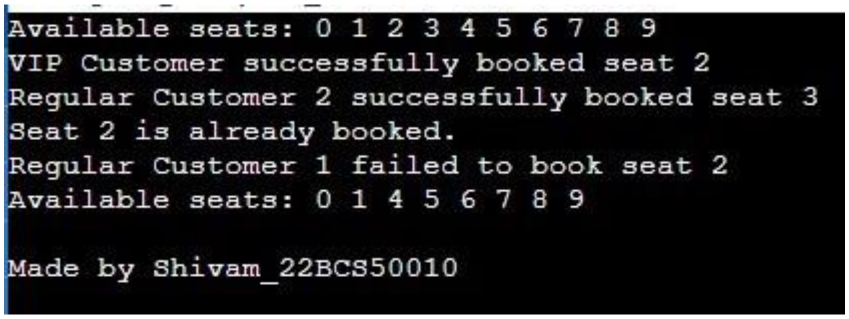
```
--- Employee Management ---
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 1
Enter Employee ID: 1001
Enter Employee Name: Shivam
Enter Employee Salary: 50000
Employee added successfully.

--- Employee Management ---
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 4
Enter Employee ID to search: 1001
ID: 1001, Name: Shivam, Salary: 50000.0

--- Employee Management ---
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option:
```

2. Card Collection:



```
--- Card Deck Management ---
1. Display all cards
2. Find cards by suit
3. Exit
Choose an option: 2
Enter the suit (Hearts, Diamonds, Clubs, Spades): hearts

Cards found with suit hearts:
2 of Hearts
3 of Hearts
4 of Hearts
5 of Hearts
6 of Hearts
7 of Hearts
8 of Hearts
9 of Hearts
10 of Hearts
Jack of Hearts
Queen of Hearts
King of Hearts
Ace of Hearts

--- Card Deck Management ---
1. Display all cards
2. Find cards by suit
3. Exit
Choose an option: 3
Exiting...

Made by Shivam_22BCS50010


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

3. Ticket Booking System:



```
Available seats: 0 1 2 3 4 5 6 7 8 9
VIP Customer successfully booked seat 2
Regular Customer 2 successfully booked seat 3
Seat 2 is already booked.
Regular Customer 1 failed to book seat 2
Available seats: 0 1 4 5 6 7 8 9

Made by Shivam_22BCS50010
```

6) Learning Outcomes:

a. Object-Oriented Design(Classes for real-world entities)

b. Core Programming Skills(Loops,conditionals,methodsforinventoryoperations)
c. DataStructureUsage(ArrayListfordynamicdatamanagement)
d. User-FriendlySystems(Intuitiveinterfacewitherrorhandling)