Experiment 4

Student Name: Dheeraj Kumar **UID: 22BCS17304**
Branch: CSE **Section: 22BCS_IOT_632/A**
Semester: 6th **DOP: 11/02/25**
Subject: Java **Subject Code: 22CSH-359**

Problem Statement: 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.

Algorithm:

1. **Initialize an ArrayList** to store Employee objects (ID, Name, Salary).
2. **Display menu options** for user actions (Add, Update, Remove, Search, Display, Exit).
3. **Loop until the user chooses to exit**:
    - **Option 1 (Add Employee)**:
        - Take user input (ID, Name, Salary).
        - Create an Employee object and add it to the list.
    - **Option 2 (Update Employee)**:
        - Take user input for the employee ID to update.
        - Search the list for the ID.
        - If found, update the Name and Salary.
    - **Option 3 (Remove Employee)**:
        - Take user input for the employee ID to remove.
        - Search and remove the matching employee from the list.
    - **Option 4 (Search Employee)**:
        - Take user input for the ID.
        - Search and display the employee details if found.
    - **Option 5 (Display All Employees)**:
        - Iterate and display all employees.
    - **Option 6 (Exit)**:
        - Terminate the loop and exit the program.

Code:

```
import java.util.ArrayList;
import java.util.Scanner;
class Employee {
    int id;
    String name;
    double salary;
    public 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 Main { // Changed class name from EmployeeManager to Main
    public static void main(String[] args) {
        ArrayList<Employee> employees = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("\n1. Add Employee\n2. Update Employee\n3. Remove
Employee\n4. Search Employee\n5. Display All Employees\n6. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Enter ID: ");
                    int id = scanner.nextInt();
                    scanner.nextLine();
                    System.out.print("Enter Name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter 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 ID to Update: ");
                    int updateId = scanner.nextInt();
                    scanner.nextLine();
                    boolean found = 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!");
                found = true;
                break;
            }
        }
        if (!found) System.out.println("Employee Not Found!");
        break;
    case 3:
        System.out.print("Enter ID to Remove: ");
        int removeId = scanner.nextInt();
        employees.removeIf(emp -> emp.id == removeId);
        System.out.println("Employee Removed Successfully!");
        break;
    case 4:
        System.out.print("Enter ID to Search: ");
        int searchId = scanner.nextInt();
        boolean searchFound = false;
        for (Employee emp : employees) {
            if (emp.id == searchId) {
                System.out.println(emp);
                searchFound = true;
            }
        }
        if (!searchFound) System.out.println("Employee Not Found!");
        break;
    case 5:
        System.out.println("Employee List:");
        for (Employee emp : employees) {
            System.out.println(emp);
        }
        break;
```

```
            case 6:
                System.out.println("Exiting...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid Choice! Try Again.");
        }
    }
  }
}
```

Output:

```
Output

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter your choice: 1
Enter ID: 123
Enter Name: Gunjan
Enter Salary: 5000
Employee Added Successfully!
```

Problem Statement: 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.

Algorithm:

- Initialize a HashMap where

- **Key** → Symbol (e.g., Hearts, Spades).
- **Value** → List of card names (e.g., Ace, King).
- **Prepopulate the collection** with a few example cards.
- **Prompt the user** to enter a symbol to search for available cards.
- **Check if the symbol exists** in the HashMap:

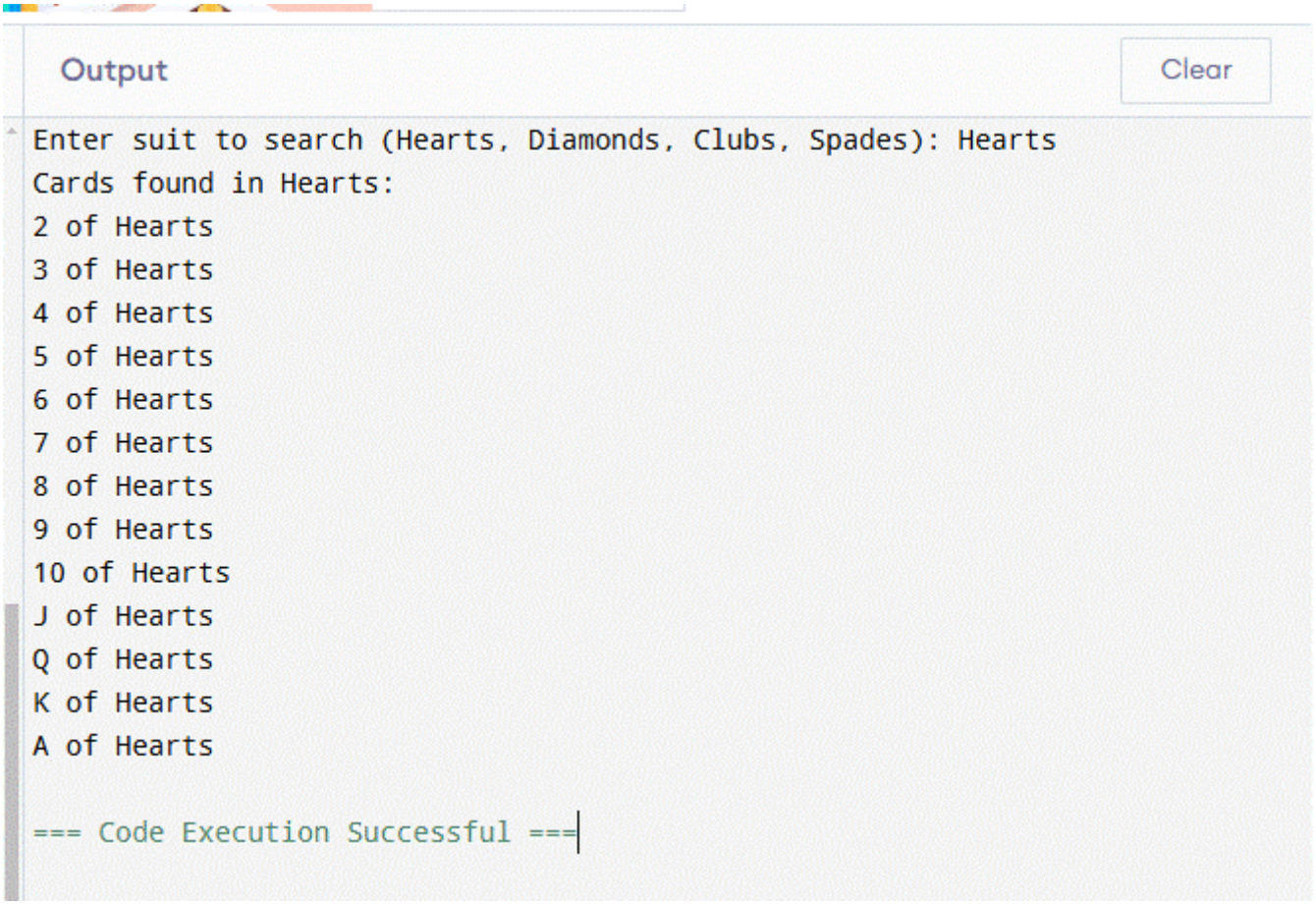
- If found, display the list of cards.
- If not found, show a message indicating no cards are available.
- End the program after displaying results.

Code:

```
import java.util.*;
class Card {
   String suit;
   String rank;
   public Card(String suit, String rank) {
      this.suit = suit;
      this.rank = rank;
   }
   public String toString() {
      return rank + " of " + suit;
   }
}
public class Main {
   public static void main(String[] args) {
      List<Card> deck = new ArrayList<>();
      String[] suits = { "Hearts", "Diamonds", "Clubs", "Spades" };
      String[] ranks = { "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A" };
      for (String suit : suits) {
         for (String rank : ranks) {
            deck.add(new Card(suit, rank));
         }
      }
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter suit to search (Hearts, Diamonds, Clubs, Spades): ");
      String suitToSearch = scanner.nextLine();
      System.out.println("Cards found in " + suitToSearch + ":");
      for (Card card : deck) {
         if (card.suit.equalsIgnoreCase(suitToSearch)) {
            System.out.println(card);
         }
      }

      scanner.close();
   }
}
```

Output:



Problem Statement: 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.

Algorithm:

1. Initialize a shared ticket booking system with a limited number of seats.
2. Create a lock mechanism to ensure only one thread can book a seat at a time.
3. Define a BookingThread class that:
    - Takes a user name and priority.
    - Calls the synchronized bookTicket() method to attempt booking.
4. Inside bookTicket():
    - Check if seats are available.
    - If available, book a seat and decrement the count.
    - If not, display a message indicating no availability.
5. Create multiple threads, setting VIP users with higher priority.
6. Start the threads to simulate concurrent booking.
7. Threads execute and book seats, prioritizing VIP users first.
8. End the program after all threads finish execution.

Code:

```
import java.util.*;
class Ticket {
    int seatNumber;
    boolean isBooked;
    public Ticket(int seatNumber) {
        this.seatNumber = seatNumber;
        this.isBooked = false;
    }
}
class TicketSystem {
    private List<Ticket> seats = new ArrayList<>();
    public TicketSystem(int totalSeats) {
        for (int i = 1; i <= totalSeats; i++) {
            seats.add(new Ticket(i));
        }
    }
    public synchronized void bookTicket(int seatNumber, String passengerName) {
        if (seatNumber < 1 || seatNumber > seats.size()) {
            System.out.println(passengerName + ": Invalid seat number!");
            return;
        }
        Ticket ticket = seats.get(seatNumber - 1);
        if (!ticket.isBooked) {
            ticket.isBooked = true;
```

```
            System.out.println(passengerName + " booked seat #" + seatNumber);

        } else {

            System.out.println(passengerName + ": Seat #" + seatNumber + " is already booked.");

        }

    }

}

public class Main {

    public static void main(String[] args) {

        TicketSystem ticketSystem = new TicketSystem(10);

        Thread vipBooking = new Thread(() -> ticketSystem.bookTicket(3, "VIP"));

        Thread normalBooking = new Thread(() -> ticketSystem.bookTicket(3, "Regular
Passenger"));

        vipBooking.setPriority(Thread.MAX_PRIORITY);

        normalBooking.setPriority(Thread.MIN_PRIORITY);

        vipBooking.start();

        normalBooking.start();

    }

}
```

Output:

```
Output

VIP booked seat #3
Regular Passenger: Seat #3 is already booked.

=== Code Execution Successful ===
```

Learning Outcomes:

1. Understand and apply object-oriented programming principles to solve real-world problems.
2. Gain hands-on experience in designing and managing a video rental inventory system.
3. Learn to handle user inputs and implement error handling in a Java application.
4. Enhance problem-solving skills by implementing renting and returning functional