Experiment 4

Student Name: Sunil Prajapat **UID:** 22BCS11871
Branch: CSE **Section/Group:** 631/A
Semester: 6th **Date of Performance:** 20/02/2025
Subject Name: Project Based Learning in Java with Lab
Subject Code: 22CSH-359

Aim:

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.

Objective:

- To implement a system that stores and manages employee details (ID, Name, Salary) using an ArrayList.
- To allow users to **add, update, remove, search**, and **display** employee records interactively.

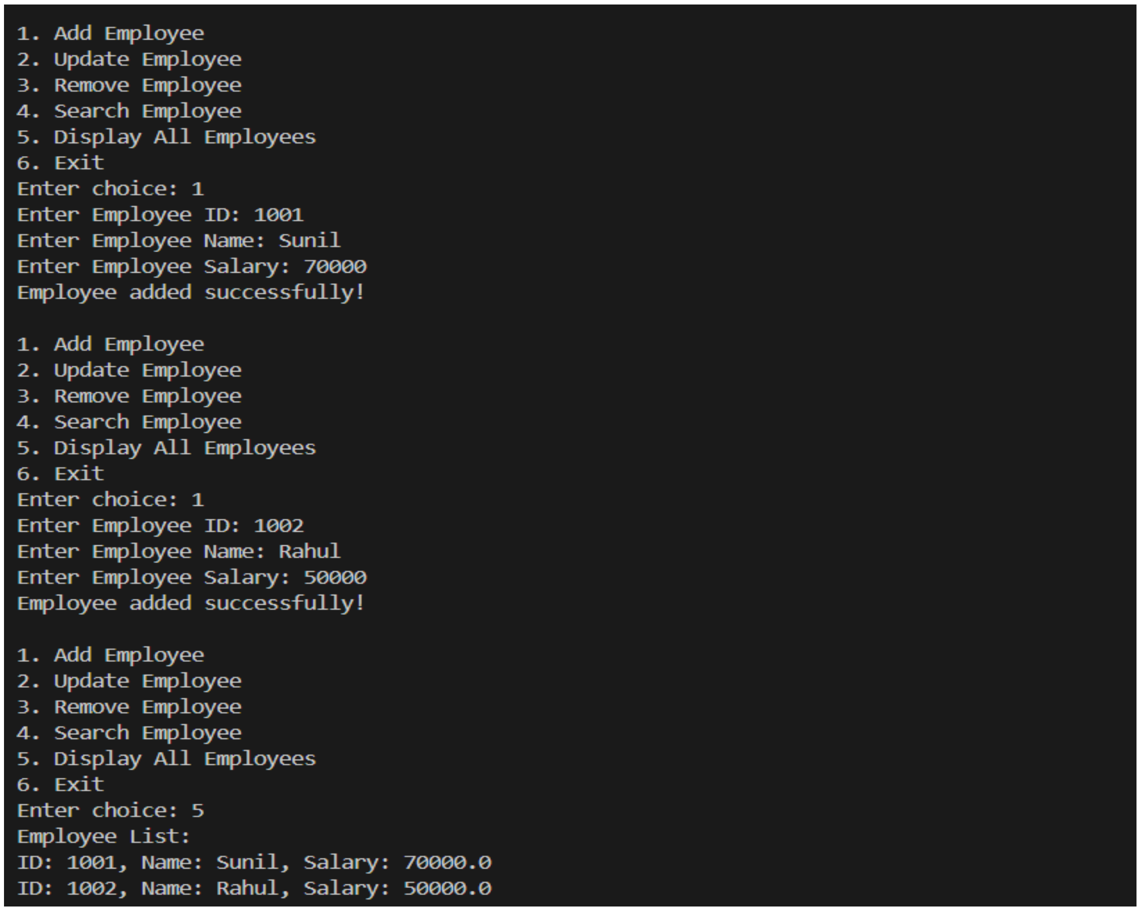
Implementation/Code:

```
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;
    }
    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}

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

    public static void addEmployee() {
        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        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!");
    }

    public static void updateEmployee() {
        System.out.print("Enter Employee ID to update: ");
        int id = scanner.nextInt();
        for (Employee emp : employees) {
            if (emp.id == id) {
                scanner.nextLine(); // Consume newline
                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!");
                return;
            }
        }
        System.out.println("Employee not found!");
    }
```

```
public static void removeEmployee() {
    System.out.print("Enter Employee ID to remove: ");
    int id = scanner.nextInt();
    employees.removeIf(emp -> emp.id == id);
    System.out.println("Employee removed successfully (if exists)!");
}

public static void searchEmployee() {
    System.out.print("Enter Employee ID to search: ");
    int id = scanner.nextInt();
    for (Employee emp : employees) {
        if (emp.id == id) {
            System.out.println("Employee Found: " + emp);
            return;
        }
    }
    System.out.println("Employee not found!");
}

public static void displayEmployees() {
    if (employees.isEmpty()) {
        System.out.println("No employees found!");
        return;
    }
    System.out.println("Employee List:");
    for (Employee emp : employees) {
        System.out.println(emp);
    }
}

public static void main(String[] args) {
    while (true) {
        System.out.println("\n1. 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 choice: ");
        int choice = scanner.nextInt();
```

```
            switch (choice) {
                case 1 -> addEmployee();
                case 2 -> updateEmployee();
                case 3 -> removeEmployee();
                case 4 -> searchEmployee();
                case 5 -> displayEmployees();
                case 6 -> {
                    System.out.println("Exiting...");
                    return;
                }
                default -> System.out.println("Invalid choice! Try again.");
            }
        }
    }
      }
```

Output



```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter choice: 1
Enter Employee ID: 1001
Enter Employee Name: Sunil
Enter Employee Salary: 70000
Employee added successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter choice: 1
Enter Employee ID: 1002
Enter Employee Name: Rahul
Enter Employee Salary: 50000
Employee added successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter choice: 5
Employee List:
ID: 1001, Name: Sunil, Salary: 70000.0
ID: 1002, Name: Rahul, Salary: 50000.0
```

Aim:

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. give me the code for both

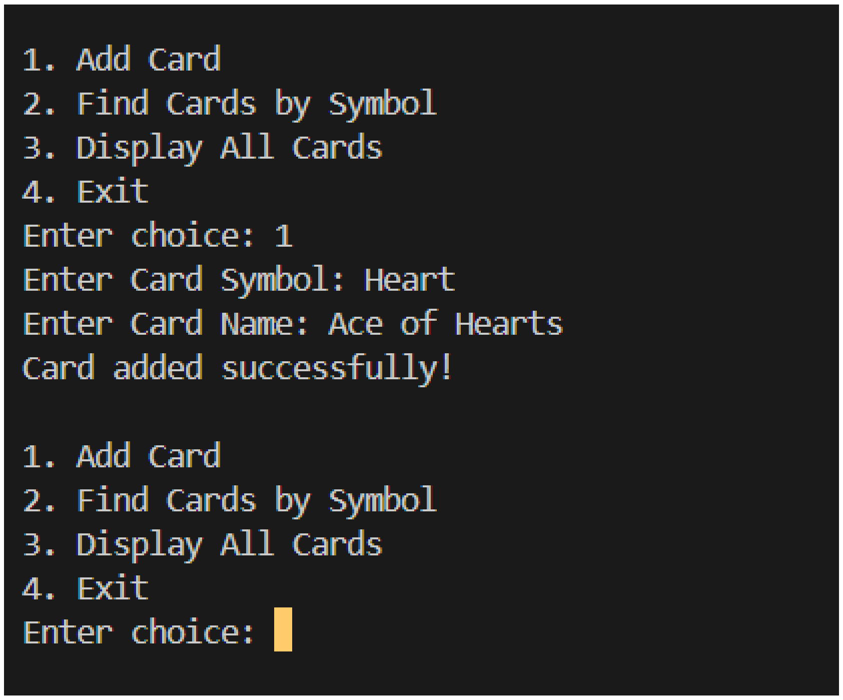
Objective:

- To create a system that stores playing cards categorized by symbols using Java Collections (HashMap).
- To allow users to add cards, search for cards by symbol, and display all stored cards efficiently.

Code:

```
import java.util.*;

class CardCollection {
    private Map<String, List<String>> cardMap;

    public CardCollection() {
        cardMap = new HashMap<>();
    }

    public void addCard(String symbol, String cardName) {
        cardMap.computeIfAbsent(symbol, k -> new ArrayList<>()).add(cardName);
        System.out.println("Card added successfully!");
    }

    public void findCardsBySymbol(String symbol) {
        if (cardMap.containsKey(symbol)) {
            System.out.println("Cards with symbol '" + symbol + "': " + cardMap.get(symbol));
        } else {
            System.out.println("No cards found with this symbol.");
        }
    }

    public void displayAllCards() {
        if (cardMap.isEmpty()) {
            System.out.println("No cards available!");
            return;
        }
        System.out.println("All Cards:");
        for (Map.Entry<String, List<String>> entry : cardMap.entrySet()) {
            System.out.println("Symbol: " + entry.getKey() + " -> Cards: " + entry.getValue());
        }
    }
}

public class CardManager {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        CardCollection collection = new CardCollection();

        while (true) {
            System.out.println("\n1. 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 choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1 -> {
                    System.out.print("Enter Card Symbol: ");
                    String symbol = scanner.nextLine();
                    System.out.print("Enter Card Name: ");
                    String name = scanner.nextLine();
                    collection.addCard(symbol, name);
                }
                case 2 -> {
                    System.out.print("Enter Symbol to Search: ");
                    String symbol = scanner.nextLine();
                    collection.findCardsBySymbol(symbol);
                }
                case 3 -> collection.displayAllCards();
                case 4 -> {
                    System.out.println("Exiting...");
                    return;
                }
                default -> System.out.println("Invalid choice! Try again.");
            }
        }
    }
  }
```

Output:



```
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Enter choice: 1
Enter Card Symbol: Heart
Enter Card Name: Ace of Hearts
Card added successfully!

1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Enter choice: 
```

Learning Outcome

- **Understanding Java Collections** – Learn how to use `ArrayList` for dynamic storage and `HashMap` for key-value-based storage, improving data handling efficiency.
- **Implementing CRUD Operations** – Gain hands-on experience in creating, reading, updating, and deleting records using Java programming constructs.
- **Enhancing User Interaction** – Develop interactive, menu-driven applications that take user inputs and process them effectively.
- **Applying Object-Oriented Programming (OOP) Principles** – Understand how to design and manage data using classes, objects, and encapsulation.
- **Efficient Data Retrieval and Management** – Learn techniques to efficiently search, update, and manage structured data using Java's built-in collections framework.