Experiment 4

Student Name: Shashwat Kumar **UID:22BCS50117**
Branch: CSE **Section/Group:DL_904/B**
Semester: 6th **DOP: 17/02/25**
Subject: Java Lab **Subject Code: 22CSH-359**

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

Easy
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.

Medium
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.

Hard
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

1)Easy Problem

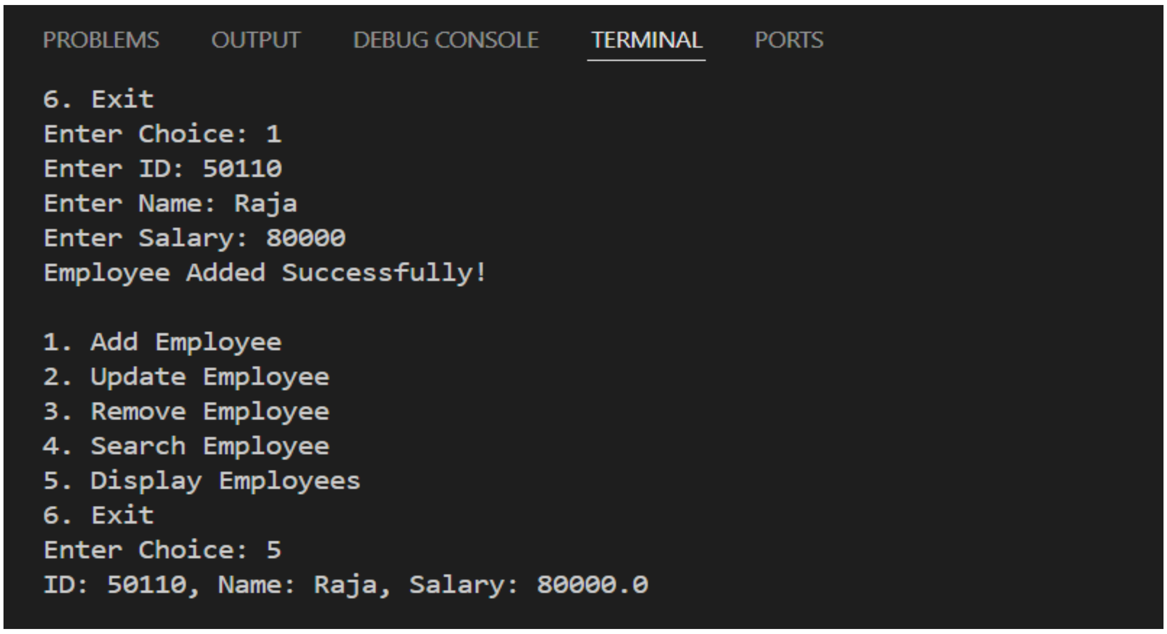
Code:

```
// Source code is decompiled from a .class file using FernFlower decompiler.
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;

public class EmployeeManagement {
  static ArrayList<Employee> employees = new ArrayList();
  static Scanner scanner;

  public EmployeeManagement() {
  }

  public static void addEmployee() {
    System.out.print("Enter ID: ");
    int var0 = scanner.nextInt();
    scanner.nextLine();
    System.out.print("Enter Name: ");
    String var1 = scanner.nextLine();
  System.out.print("Enter Salary: ");
  double var2 = scanner.nextDouble();
  employees.add(new Employee(var0, var1, var2));
  System.out.println("Employee Added Successfully!");
}

public static void updateEmployee() {
  System.out.print("Enter Employee ID to Update: ");
  int var0 = scanner.nextInt();
  Iterator var1 = employees.iterator();

  Employee var2;
  do {
    if (!var1.hasNext()) {
      System.out.println("Employee Not Found!");
      return;
    }

    var2 = (Employee)var1.next();
  } while(var2.id != var0);

  scanner.nextLine();
  System.out.print("Enter New Name: ");
  var2.name = scanner.nextLine();
  System.out.print("Enter New Salary: ");
  var2.salary = scanner.nextDouble();
  System.out.println("Employee Updated Successfully!");
}

public static void removeEmployee() {
  System.out.print("Enter Employee ID to Remove: ");
  int var0 = scanner.nextInt();
  employees.removeIf((var1) -> {
    return var1.id == var0;
  });
  System.out.println("Employee Removed Successfully!");
}
```

```
public static void searchEmployee() {
  System.out.print("Enter Employee ID to Search: ");
  int var0 = scanner.nextInt();
  Iterator var1 = employees.iterator();

  Employee var2;
  do {
    if (!var1.hasNext()) {
      System.out.println("Employee Not Found!");
      return;
    }

    var2 = (Employee)var1.next();
  } while(var2.id != var0);

  System.out.println(var2);
}

public static void displayEmployees() {
  if (employees.isEmpty()) {
    System.out.println("No Employees Found!");
  } else {
    Iterator var0 = employees.iterator();

    while(var0.hasNext()) {
      Employee var1 = (Employee)var0.next();
      System.out.println(var1);
    }

  }
}

public static void main(String[] var0) {
  while(true) {
    System.out.println("\n1. Add Employee\n2. Update Employee\n3. Remove
Employee\n4. Search Employee\n5. Display Employees\n6. Exit");
    System.out.print("Enter Choice: ");
```

```
                int var1 = scanner.nextInt();
                switch (var1) {
                    case 1:
                        addEmployee();
                        break;
                    case 2:
                        updateEmployee();
                        break;
                    case 3:
                        removeEmployee();
                        break;
                    case 4:
                        searchEmployee();
                        break;
                    case 5:
                        displayEmployees();
                        break;
                    case 6:
                        System.exit(0);
                        break;
                    default:
                        System.out.println("Invalid Choice! Try Again.");
                }
            }
        }

        static {
            scanner = new Scanner(System.in);
        }
    }
```

Output:



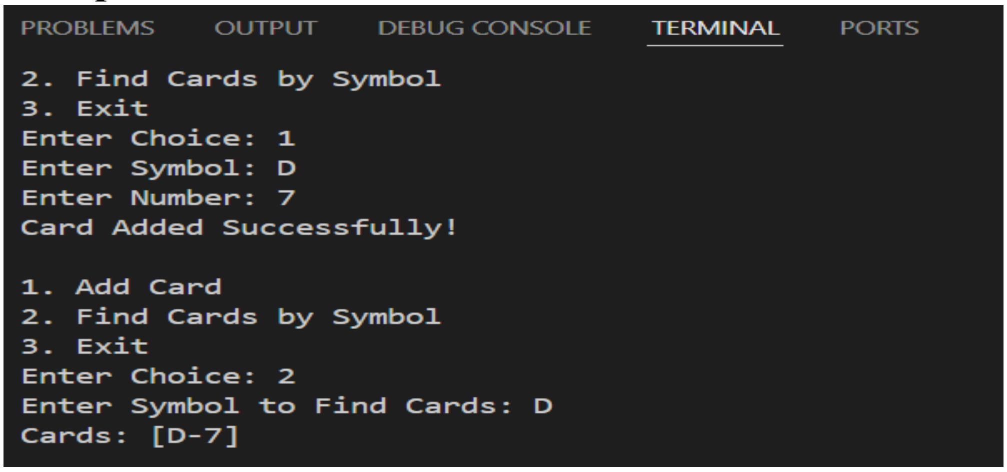
```
6. Exit
Enter Choice: 1
Enter ID: 50110
Enter Name: Raja
Enter Salary: 80000
Employee Added Successfully!

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Enter Choice: 5
ID: 50110, Name: Raja, Salary: 80000.0
```

1)Medium Level Problem

Code:

```
import java.util.*;

class Card {
    String symbol;
    int number;

    public Card(String symbol, int number) {
        this.symbol = symbol;
        this.number = number;
    }

    @Override
    public String toString() {
        return symbol + "-" + number;
    }
}

public class CardCollection {
    static Map<String, List<Card>> cardMap = new HashMap<>();
    static Scanner scanner = new Scanner(System.in);

    public static void addCard() {
        System.out.print("Enter Symbol: ");

        String symbol = scanner.next();
        System.out.print("Enter Number: ");
        int number = scanner.nextInt();
        cardMap.putIfAbsent(symbol, new ArrayList<>());
        cardMap.get(symbol).add(new Card(symbol, number));

        System.out.println("Card Added Successfully!");
    }

    public static void findCards() {
        System.out.print("Enter Symbol to Find Cards: ");
        String symbol = scanner.next();
        if (cardMap.containsKey(symbol)) {
            System.out.println("Cards: " + cardMap.get(symbol));
        } else {
            System.out.println("No Cards Found for This Symbol!");
        }
    }

    public static void main(String[] args) {
        while (true) {
            System.out.println("\n1. Add Card\n2. Find Cards by Symbol\n3. Exit");
            System.out.print("Enter Choice: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1 -> addCard();
                case 2 -> findCards();
                case 3 -> System.exit(0);
                default -> System.out.println("Invalid Choice! Try Again.");
            }
        }
    }
}
```

Output:



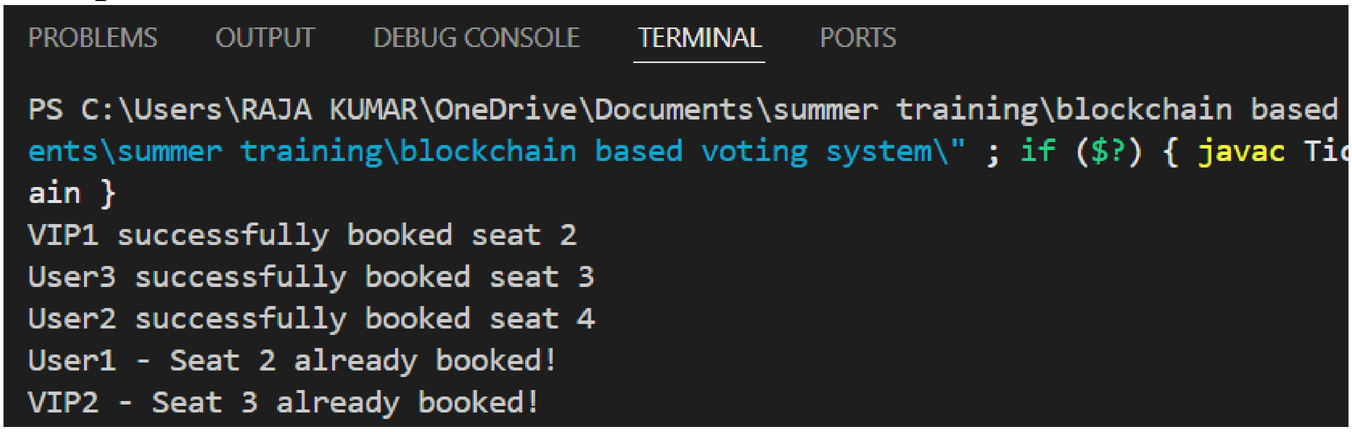
```
2. Find Cards by Symbol
3. Exit
Enter Choice: 1
Enter Symbol: D
Enter Number: 7
Card Added Successfully!

1. Add Card
2. Find Cards by Symbol
3. Exit
Enter Choice: 2
Enter Symbol to Find Cards: D
Cards: [D-7]
```

3) **Hard Problem**

Code:

```
import java.util.concurrent.*;


class TicketBookingSystem {
    private final boolean[] seats = new boolean[10]; // 10 seats
    private final Object lock = new Object();

    public void bookSeat(String passenger, int seatNumber) {
        synchronized (lock) {
            if (seatNumber < 0 || seatNumber >= seats.length) {
                System.out.println(passenger + " - Invalid seat number!");
                return;
            }
            if (seats[seatNumber]) {
                System.out.println(passenger + " - Seat " + seatNumber + " already booked!");
            } else {
                seats[seatNumber] = true;
                System.out.println(passenger + " successfully booked seat " + seatNumber);
            }


       }
    }
}


class Passenger extends Thread {
    private final TicketBookingSystem system;
    private final String name;
    private final int seatNumber;

    public Passenger(TicketBookingSystem system, String name, int seatNumber, int priority) {
        this.system = system;
        this.name = name;
        this.seatNumber = seatNumber;
        setPriority(priority); // VIP passengers get high priority
    }

    @Override
    public void run() {
        system.bookSeat(name, seatNumber);
    }
}

public class TicketBookingMain {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem();
        ExecutorService executor = Executors.newFixedThreadPool(5);

        Passenger vip1 = new Passenger(system, "VIP1", 2, Thread.MAX_PRIORITY);
        Passenger vip2 = new Passenger(system, "VIP2", 3, Thread.MAX_PRIORITY);

        Passenger user1 = new Passenger(system, "User1", 2, Thread.NORM_PRIORITY);
        Passenger user2 = new Passenger(system, "User2", 4, Thread.NORM_PRIORITY);
        Passenger user3 = new Passenger(system, "User3", 3, Thread.MIN_PRIORITY);

        executor.execute(vip1);
        executor.execute(vip2);
        executor.execute(user1);
        executor.execute(user2);
        executor.execute(user3);
        executor.shutdown();
    }
}
```

Output:



```
PS C:\Users\RAJA KUMAR\OneDrive\Documents\summer training\blockchain based
ents\summer training\blockchain based voting system\" ; if ($?) { javac Ti
ain }
VIP1 successfully booked seat 2
User3 successfully booked seat 3
User2 successfully booked seat 4
User1 - Seat 2 already booked!
VIP2 - Seat 3 already booked!
```

Learning Outcomes:

- **Inheritance**: Use of base and derived classes for shared attributes and methods.
- **Method Overriding**: Custom implementation of methods in subclasses.
- **Constructor**: Initializing object attributes using constructors.
- **Encapsulation**: Storing and manipulating data within objects.
- **Polymorphism**: Different behavior of `calculateInterest()` based on object type.
- **Interest Calculation**: Implementing FD and RD interest formulas.
- **Class Interaction**: Creating objects and calling methods to display details.