Experiment 4

Student Name: Rudraksh Mishra **UID: 22BCS10607**
Branch: CSE **Section/Group: 22BCS_KPIT-901**
Semester: 6 **Date of Performance: 21 / 2 / 25**
Subject Name: Project Based Learning with Java **Subject Code: 22CSH–359**

1. **Aim:** Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.
   a. 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.
   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.
2. **Objective:** to develop Java programs that demonstrate the efficient use of core programming concepts such as data structures, collections, and multithreading for effective data management and manipulation.
3. **Implementation/Code:**
   a. **ArrayList for Employee.**

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

public class Ex4_1 {
    static final Scanner sc = new Scanner(System.in);
    static int e_id = 1;

    static class employee {
        private final int    id;
        private       String name;
        private       int    salary;

        employee(String name, int salary) {
            this.id     = e_id++;
            this.name   = name;
            this.salary = salary;
        }

        public int get_id() { return id; }
        public String get_name()   { return name;   }
        public int    get_salary() { return salary; }

        public void set_name(String name)  { this.name   = name;   }
        public void set_salary(int salary) { this.salary = salary; }

        @Override
        public String toString() {
            return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
        }
    }
    private static final ArrayList<employee> employees = new ArrayList<>();

    public static void main(String[] args) {
        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("Choose an option: ");
            int choice = sc.nextInt();

            sc.nextLine();

            switch (choice) {
                case 1 -> add_employee();
                case 2 -> update_employee();
                case 3 -> remove_employee();
                case 4 -> search_employee();
                case 5 -> display_employees();
                case 6 -> {
                    System.out.println("Exiting...");
                    return;
                }
                default -> System.out.println("Invalid option! Try again.");
            }
        }
    }

    private static void add_employee() {
        System.out.print("Enter name: ");
        String name = sc.nextLine();
        System.out.print("Enter salary: ");
        int salary = sc.nextInt();

        employees.add(new employee(name, salary));
        System.out.println("Employee added successfully.");
    }

    private static void update_employee() {
        System.out.print("Enter Employee ID to update: ");
        int id = sc.nextInt();
        sc.nextLine();

        for (employee emp : employees) {
            if (emp.get_id() == id) {
                System.out.print("Enter new name: ");
                String name = sc.nextLine();
                System.out.print("Enter new salary: ");
                int salary = sc.nextInt();

                emp.set_name(name);
                emp.set_salary(salary);
                System.out.println("Employee updated successfully.");
                return;
            }
        }
        System.out.println("Employee not found!");
    }
    private static void remove_employee() {
        System.out.print("Enter Employee ID to remove: ");
        int id = sc.nextInt();

        employees.removeIf(emp -> emp.get_id() == id);
        System.out.println("Employee removed successfully.");
    }

    private static void search_employee() {
        System.out.print("Enter Employee ID to search: ");
        int id = sc.nextInt();

        for (employee emp : employees) {
            if (emp.get_id() == id) {
                System.out.println(emp);
                return;
            }
        }
        System.out.println("Employee not found!");
    }

    private static void display_employees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
        } else {
            employees.forEach(System.out::println);
        }
    }
}
```

b. Cards using Collection Interface

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

public class Ex4_2 {
    private static final Scanner          SC = new Scanner(System.in);
    private static final Collection<card> cards = new ArrayList<>();

    static class card {
        private final String symbol;
        private final String value;

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

        public String get_symbol() { return symbol; }

        @Override
        public String toString() {
            return "[" + value + " of " + symbol + "]";
        }
    }
    private static void add_card() {
        System.out.print("Enter Card Symbol (e.g., Hearts, Diamonds, Clubs, 
Spades): ");
        String symbol = SC.nextLine();

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

        cards.add(new card(symbol, value));

        System.out.println("Card added successfully.");
    }

    private static void find_cards_by_symbol() {
        System.out.print("Enter Symbol to search (e.g., Hearts, Diamonds): ");
        String symbol = SC.nextLine();

        boolean found = false;
        for (card c : cards) {
            if (c.get_symbol().equalsIgnoreCase(symbol)) {
                System.out.println(c);
                found = true;
            }
        }
        if (!found) {
            System.out.println("No cards found for the given symbol.");
        }
    }

    private static void display_cards() {
        if (cards.isEmpty()) {
            System.out.println("No cards available.");
        } else {
            System.out.println("All Cards:");
            cards.forEach(System.out::println);
        }
    }

    public static void main(String[] args) {
        while (true) {
            System.out.println("\n1. Add Card\n2. Find Cards by Symbol\n3. Display 
All Cards\n4. Exit");
            System.out.print("Choose an option: ");
            int choice = SC.nextInt();
            SC.nextLine();

            switch (choice) {
                case 1 -> add_card();
                case 2 -> find_cards_by_symbol();
                case 3 -> display_cards();
                case 4 -> {
                    System.out.println("Exiting...");
                    return;
                }
                default -> System.out.println("Invalid choice! Try again.");
            }
        }
    }
}
```

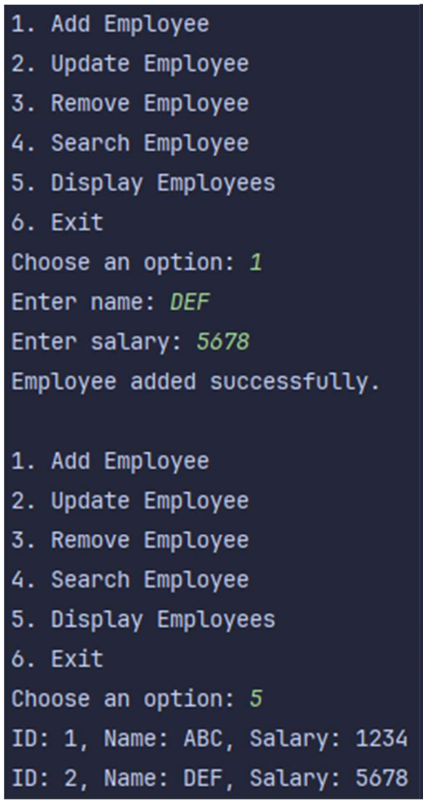
c. **Ticket Booking System with Synchronization and Prioritization**

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

public class Ex_4_3 {
    static class Customer implements Runnable {
        private final Ticket_Booking_System booking_system;
        private final String customerName;

        public Customer(Ticket_Booking_System system, String name) {
            this.booking_system = system;
            this.customerName = name;
        }

        @Override
        public void run() {
            booking_system.book_Seat(customerName);
        }
    }

    static class Ticket_Booking_System {
        private int availableSeats;

        public Ticket_Booking_System(int seats) {
            this.availableSeats = seats;
        }

        public synchronized void book_Seat(String name) {
            if (availableSeats > 0) {
                System.out.println(
                    name + " booked a seat. Remaining: " + (--availableSeats)
                );
            } else {
                System.out.println(name + " failed to book. No seats available.");
            }
        }
    }

    static class Booking_Request {
        Customer customer;
        int priority;

        public Booking_Request(Customer customer, int priority) {
            this.customer = customer;
            this.priority = priority;
        }
    }

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

        System.out.print("Enter total number of seats available: ");
        int total_seats = scanner.nextInt();
        scanner.nextLine();
        Ticket_Booking_System booking_system = new
Ticket_Booking_System(total_seats);
```

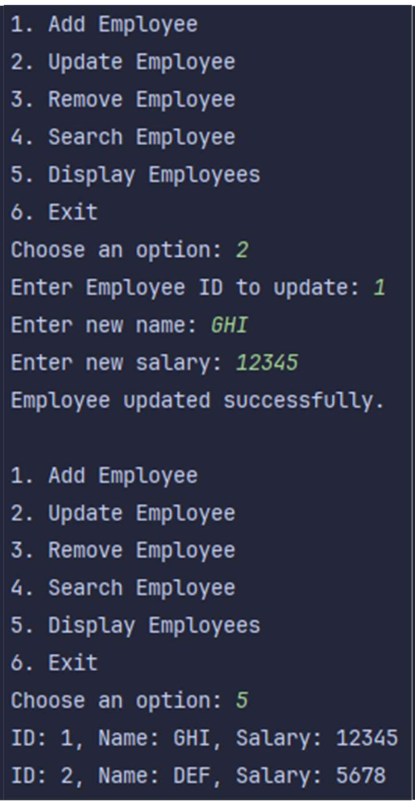
```
        System.out.print("Enter the number of customers: ");
        int customer_count = scanner.nextInt();
        scanner.nextLine();

        List<Booking_Request> booking_list = new ArrayList<>();

        for (int i = 0; i < customer_count; i++) {
            System.out.print("Enter Customer Name: ");
            String name = scanner.nextLine();

            System.out.print("Enter Priority (1 = Regular, 2 = VIP): ");
            int priority = scanner.nextInt();
            scanner.nextLine();

            Customer customer = new Customer(booking_system, name);
            booking_list.add(new Booking_Request(customer, priority));
        }

        booking_list.sort((a, b) -> Integer.compare(b.priority, a.priority));

        int threadPoolSize = Math.min(customer_count, total_seats);
        ExecutorService executor = Executors.newFixedThreadPool(threadPoolSize);

        for (Booking_Request request : booking_list) {
            executor.execute(request.customer);
        }

        executor.shutdown();
        try {
            executor.awaitTermination(10, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Booking Process Completed!");
    }
}
```

Output

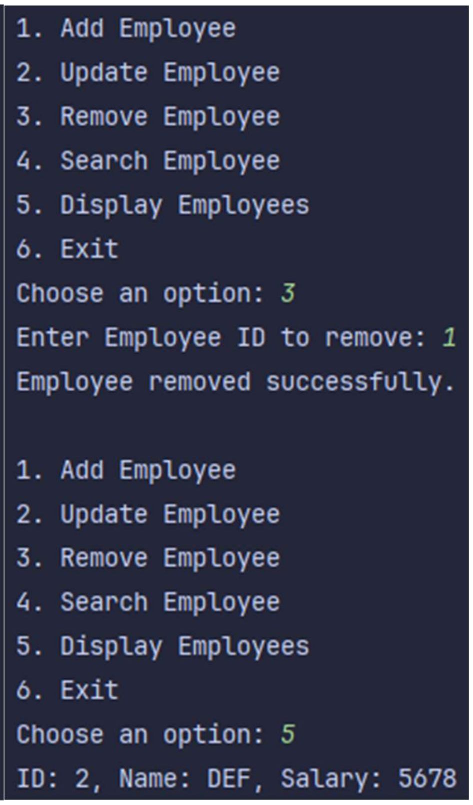
Part 1)



```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 1
Enter name: DEF
Enter salary: 5678
Employee added successfully.

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 5
ID: 1, Name: ABC, Salary: 1234
ID: 2, Name: DEF, Salary: 5678
```

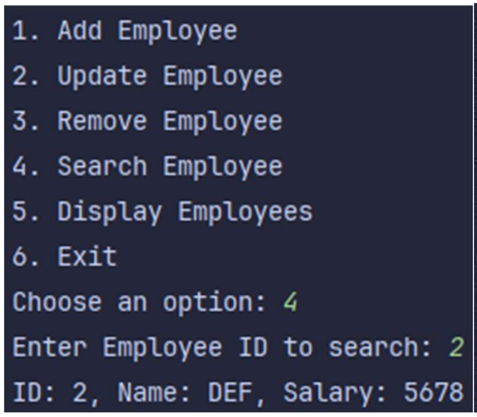


```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 2
Enter Employee ID to update: 1
Enter new name: GHI
Enter new salary: 12345
Employee updated successfully.

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 5
ID: 1, Name: GHI, Salary: 12345
ID: 2, Name: DEF, Salary: 5678
```

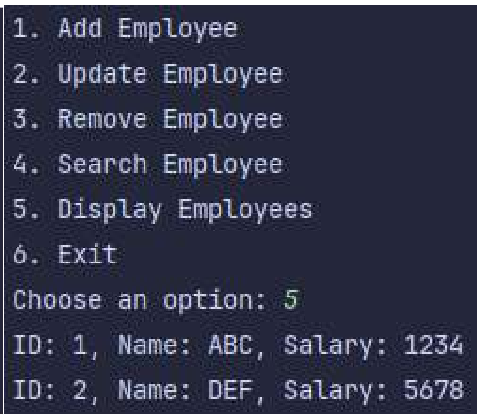


```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 3
Enter Employee ID to remove: 1
Employee removed successfully.

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 5
ID: 2, Name: DEF, Salary: 5678
```

Add, Update, Remove ↑ **Search Display ↓**

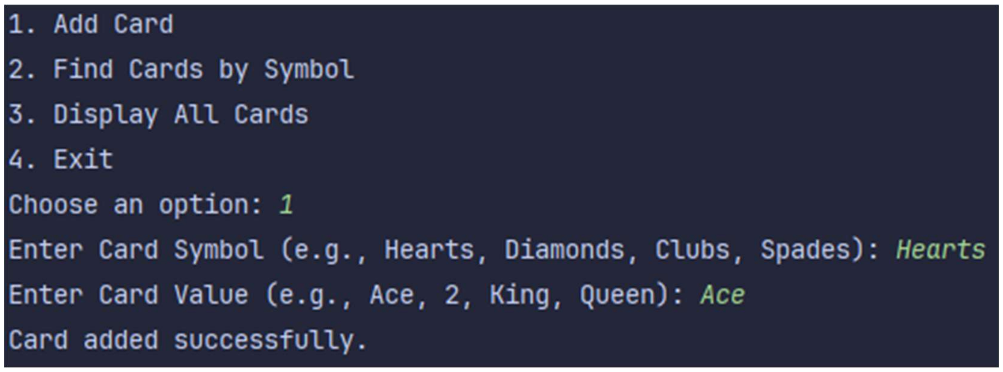


```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 4
Enter Employee ID to search: 2
ID: 2, Name: DEF, Salary: 5678
```



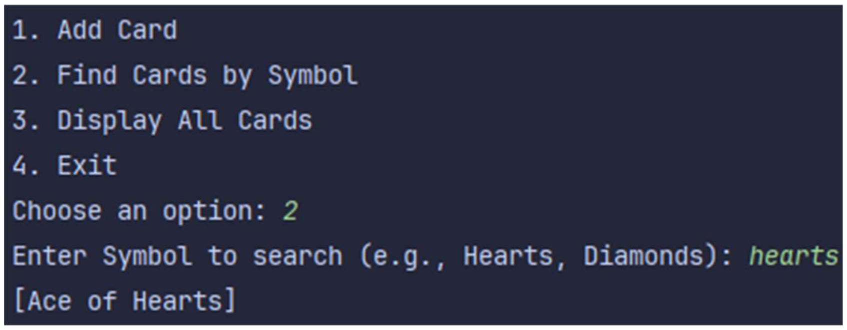
```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Choose an option: 5
ID: 1, Name: ABC, Salary: 1234
ID: 2, Name: DEF, Salary: 5678
```

Part 2)



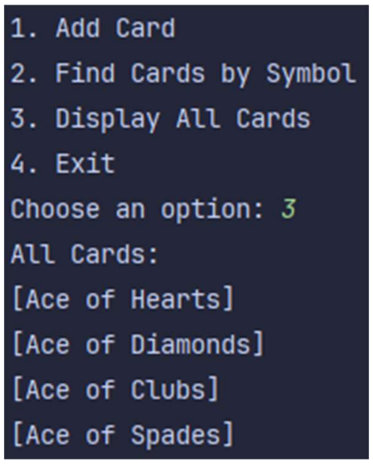
```
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Choose an option: 1
Enter Card Symbol (e.g., Hearts, Diamonds, Clubs, Spades): Hearts
Enter Card Value (e.g., Ace, 2, King, Queen): Ace
Card added successfully.
```

Add



```
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Choose an option: 2
Enter Symbol to search (e.g., Hearts, Diamonds): hearts
[Ace of Hearts]
```

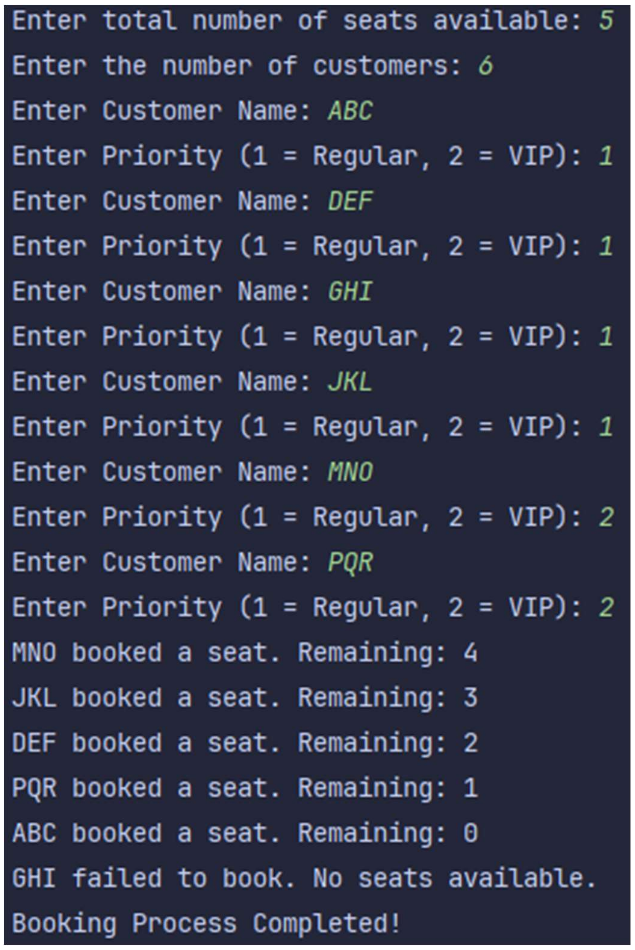
Find



```
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Choose an option: 3
All Cards:
[Ace of Hearts]
[Ace of Diamonds]
[Ace of Clubs]
[Ace of Spades]
```

Display

Part 3)



```
Enter total number of seats available: 5
Enter the number of customers: 6
Enter Customer Name: ABC
Enter Priority (1 = Regular, 2 = VIP): 1
Enter Customer Name: DEF
Enter Priority (1 = Regular, 2 = VIP): 1
Enter Customer Name: GHI
Enter Priority (1 = Regular, 2 = VIP): 1
Enter Customer Name: JKL
Enter Priority (1 = Regular, 2 = VIP): 1
Enter Customer Name: MNO
Enter Priority (1 = Regular, 2 = VIP): 2
Enter Customer Name: PQR
Enter Priority (1 = Regular, 2 = VIP): 2
MNO booked a seat. Remaining: 4
JKL booked a seat. Remaining: 3
DEF booked a seat. Remaining: 2
PQR booked a seat. Remaining: 1
ABC booked a seat. Remaining: 0
GHI failed to book. No seats available.
Booking Process Completed!
```

ABC, DEF, GHI, JKL are regular customers. MNO & PQR are VIP Customers i.e. they will get priority. 2 out of 5 available seats are reserved for them. Rest 3 are for ABC, DEF, GHI, JKL. In this instance GHI failed to book the seat.

Learning Outcomes

- Understand and apply object-oriented programming (OOP) principles.
- Implement data management systems using ArrayList, HashMap, and PriorityQueue.
- Use multithreading and synchronization to ensure safe concurrent execution in applications.
- Apply thread priority and scheduling to simulate real-world scenarios like VIP booking preference.
- Develop scalable and efficient Java applications using the Java Collections Framework.
- Enhance code reusability and maintainability through modular class design.
- Gain hands-on experience in handling user input, sorting, and data retrieval operations.
- Apply synchronized methods and thread pools to manage multiple users efficiently.