Experiment 2.1

Student Name: Varnika Srivastava **UID: 22BCS11670**
Branch: CSE **Section: 22KPIT-901/A**
Semester: 6th **Date of Performance:21/02/2025**
Subject: Project Based Learning in 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.

2. **Objective 1: Easy Level**
   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.

3. **Code/Implementation:**

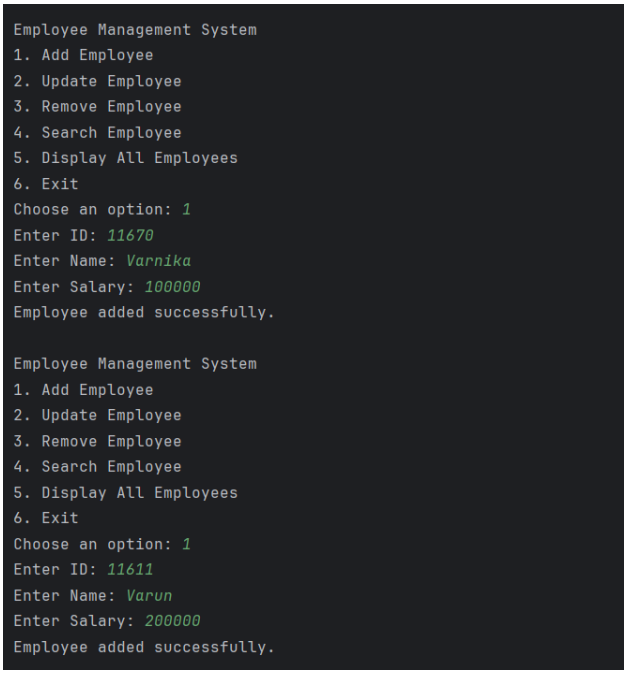
```
import java.util.ArrayList;
import java.util.Scanner;

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 "Employee [ID=" + id + ", Name=" + name + ", Salary=" + salary + "]";
    }
}

public class EmployeeManager {
    private static final ArrayList<Employee> employees = new ArrayList<>();

    public static void addEmployee(int id, String name, double salary) {
        employees.add(new Employee(id, name, salary));
        System.out.println("Employee added successfully.");
    }
```

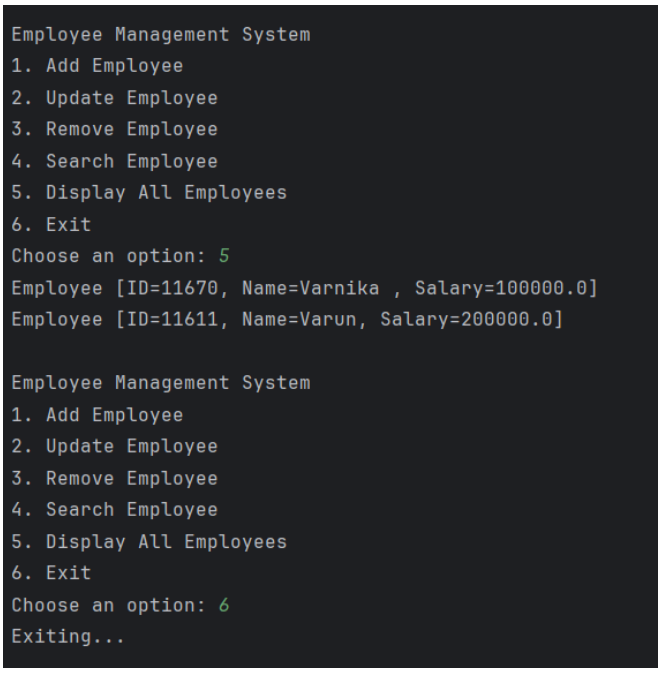
```
    public static void updateEmployee(int id, String newName, double newSalary) {
        for (Employee emp : employees) {
            if (emp.id == id) {
                emp.name = newName;
                emp.salary = newSalary;
                System.out.println("Employee updated successfully.");
                return;
            }
        }
        System.out.println("Employee not found.");
    }

    public static void removeEmployee(int id) {
        employees.removeIf(emp -> emp.id == id);
        System.out.println("Employee removed successfully.");
    }

    public static void searchEmployee(int id) {
        for (Employee emp : employees) {
            if (emp.id == id) {
                System.out.println(emp);
                return;
            }
        }
        System.out.println("Employee not found.");
    }

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

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("\nEmployee Management System");
            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();
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();
        addEmployee(id, name, salary);
        break;
    case 2:
        System.out.print("Enter Employee ID to update: ");
        int updateId = scanner.nextInt();
        scanner.nextLine();
        System.out.print("Enter New Name: ");
        String newName = scanner.nextLine();
        System.out.print("Enter New Salary: ");
        double newSalary = scanner.nextDouble();
        updateEmployee(updateId, newName, newSalary);
        break;
    case 3:
        System.out.print("Enter Employee ID to remove: ");
        int removeId = scanner.nextInt();
        removeEmployee(removeId);
        break;
    case 4:
        System.out.print("Enter Employee ID to search: ");
        int searchId = scanner.nextInt();
        searchEmployee(searchId);
        break;
    case 5:
        displayEmployees();
        break;
    case 6:
        System.out.println("Exiting...");
        scanner.close();
        return;
    default:
        System.out.println("Invalid option. Please try again.");
}
```

```
        }
    }
     }
```

Output:





4. Objective 2: Medium Level

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.

5. Code/Implementation:

```
import java.util.*;

class Card {
    private String symbol;
    private String value;

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

    public String getSymbol() {
        return symbol;
    }

    @Override
```

```
    public String toString() {
        return value + " of " + symbol;
    }
}

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

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

    public void addCard(String symbol, String value) {
        cardMap.putIfAbsent(symbol, new ArrayList<>());
        cardMap.get(symbol).add(new Card(symbol, value));
    }

    public List<Card> getCardsBySymbol(String symbol) {
        return cardMap.getOrDefault(symbol, Collections.emptyList());
    }

    public void displayAllCards() {
        for (Map.Entry<String, List<Card>> entry : cardMap.entrySet()) {
            System.out.println("Cards with symbol " + entry.getKey() + ": " +
     entry.getValue());
        }
    }

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

        while (true) {
            System.out.println("\nCard Management System");
            System.out.println("1. 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("Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Enter Card Symbol: ");
```

```
                String symbol = scanner.nextLine();
                System.out.print("Enter Card Value: ");
                String value = scanner.nextLine();
                collection.addCard(symbol, value);
                System.out.println("Card added successfully.");
                break;
            case 2:
                System.out.print("Enter Symbol to Search: ");
                String searchSymbol = scanner.nextLine();
                List<Card> cards = collection.getCardsBySymbol(searchSymbol);
                if (cards.isEmpty()) {
                    System.out.println("No cards found for symbol: " + searchSymbol);
                } else {
                    System.out.println("Cards: " + cards);
                }
                break;
            case 3:
                collection.displayAllCards();
                break;
            case 4:
                System.out.println("Exiting...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid option. Please try again.");
        }
    }
}
 }
```

Output:

```
Card Management System
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Choose an option: 1
Enter Card Symbol: J
Enter Card Value: 10
Card added successfully.

Card Management System
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
Choose an option: 1
Enter Card Symbol: Q
Enter Card Value: 20
Card added successfully.

Card Management System
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
```

```
Choose an option: 3
Cards with symbol Q: [20 of Q]
Cards with symbol J: [10 of J]

Card Management System
1. Add Card
2. Find Cards by Symbol
3. Display All Cards
4. Exit
```

6. Objective 3 : Hard Level

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.

7. Code/Implementation:

```
import java.util.*;

class TicketBookingSystem {
    private static final int TOTAL_SEATS = 10;
    private boolean[] seats = new boolean[TOTAL_SEATS];

    public synchronized boolean bookSeat(int seatNumber, String passengerName) {
        if (seatNumber < 0 || seatNumber >= TOTAL_SEATS || seats[seatNumber]) {
            return false;
        }
        seats[seatNumber] = true;
        System.out.println(passengerName + " successfully booked seat " + seatNumber);
        return true;
    }
}

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

    public Passenger(TicketBookingSystem bookingSystem, int seatNumber, String name, int
priority) {
```

```
        this.bookingSystem = bookingSystem;
        this.seatNumber = seatNumber;
        this.name = name;
        this.setPriority(priority);
    }

    @Override
    public void run() {
        boolean success = bookingSystem.bookSeat(seatNumber, name);
        if (!success) {
            System.out.println(name + " failed to book seat " + seatNumber + " (already
taken or invalid)");
        }
    }
}

public class TicketBookingMain {
    public static void main(String[] args) {
        TicketBookingSystem bookingSystem = new TicketBookingSystem();

        List<Passenger> passengers = new ArrayList<>();
        passengers.add(new Passenger(bookingSystem, 2, "VIP_Passenger_1",
Thread.MAX_PRIORITY));
        passengers.add(new Passenger(bookingSystem, 2, "Regular_Passenger_1",
Thread.NORM_PRIORITY));
        passengers.add(new Passenger(bookingSystem, 3, "VIP_Passenger_2",
Thread.MAX_PRIORITY));
        passengers.add(new Passenger(bookingSystem, 3, "Regular_Passenger_2",
Thread.NORM_PRIORITY));
        passengers.add(new Passenger(bookingSystem, 4, "VIP_Passenger_3",
Thread.MAX_PRIORITY));

        for (Passenger p : passengers) {
            p.start();
        }

        for (Passenger p : passengers) {
            try {
                p.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("Booking process completed.");
    }
}
```

Output:

```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\Jet
Regular_Passenger_1 successfully booked seat 2
VIP_Passenger_3 successfully booked seat 4
Regular_Passenger_2 successfully booked seat 3
VIP_Passenger_1 failed to book seat 2 (already taken or invalid)
VIP_Passenger_2 failed to book seat 3 (already taken or invalid)
Booking process completed.

Process finished with exit code 0
```

8. Learning Outcomes:

- Learned how to use ArrayList to store and manage employee details.
- Understood how to group and retrieve cards using the Map interface.
- Practiced synchronized threads to prevent double booking in a ticket system.
- Used thread priorities to ensure VIP bookings happen first.
- Fixed input handling issues to avoid errors in multithreaded programs.