EXPERIMENT 4

Student Name: Ruchi Thakur **UID:** 22BET10239

Branch: BE – IT **Section/Group:** BET_KRG_IOT-3B

Semester: 6th **Date:** 18/02/2025

Subject Name: PBLJ With Lab **Subject Code:** 22ITH-359

Problem 1:

Aim: 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.

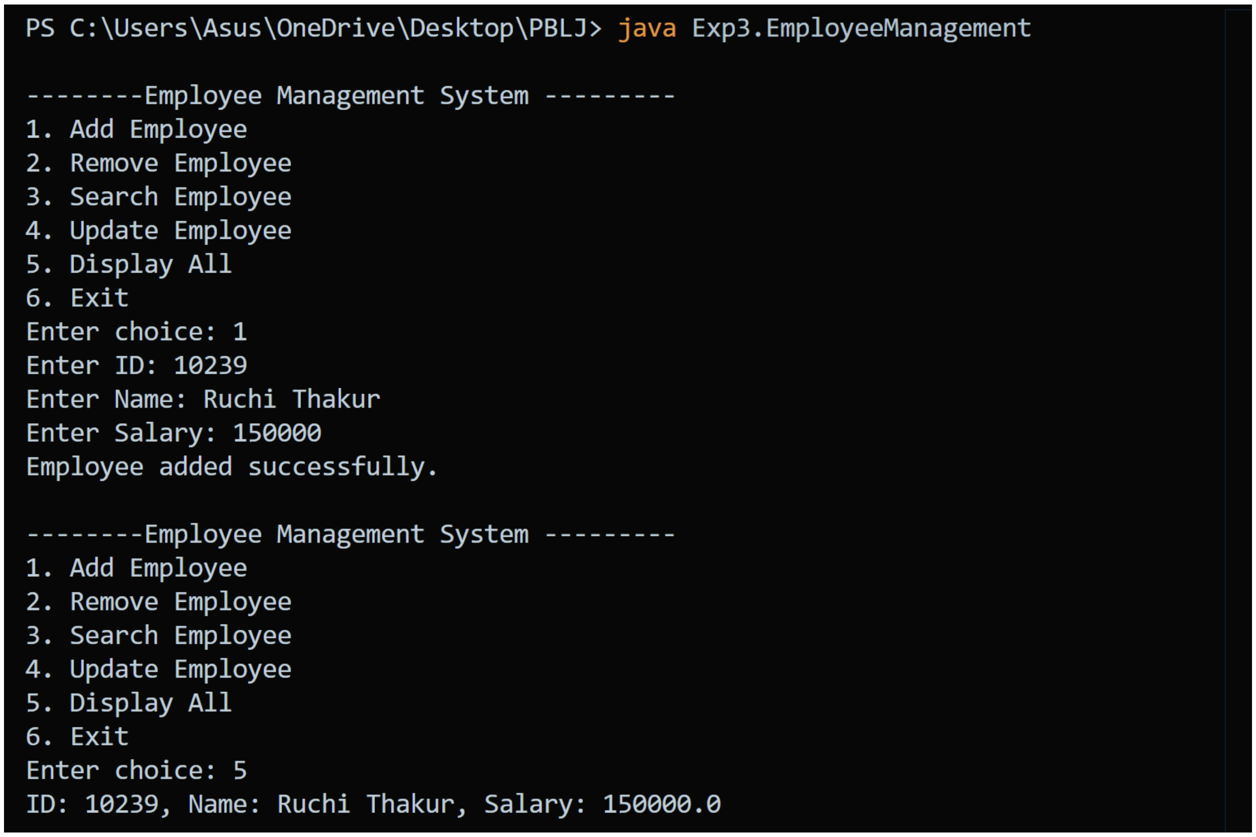
Objective:

- To create a Java program to manage employee information (ID, Name, Salary) using an ArrayList.
- To enable users to add, update, delete, and search for employee records.
- To ensure efficient access and modification of employee details.

Code:

```
package Exp3;

import java.util.*;

class Employee {

    private int id;

    private String name;

    private double salary;

    public Employee(int id, String name, double salary) {

        this.id = id;

        this.name = name;

        this.salary = salary;

    }

    public int getId() { return id; }
    public String getName() { return name; }
    public double getSalary() { return salary; }
    public void setName(String name) { this.name = name; }
    public void setSalary(double salary) { this.salary = salary; }
    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}
class EmployeeManagement {
    private List<Employee> employees = new ArrayList<>();

    public void addEmployee(Employee employee) {
        employees.add(employee);
    }
    public void removeEmployee(int id) {
        employees.removeIf(emp -> emp.getId() == id);
    }
    public Employee searchEmployee(int id) {
        return employees.stream().filter(emp -> emp.getId() == id).findFirst().orElse(null);
    }
    public void updateEmployee(int id, String name, double salary) {
        for (Employee emp : employees) {
            if (emp.getId() == id) {
                emp.setName(name);
                emp.setSalary(salary);
            }
        }
    }
    public void displayEmployees() {
        employees.forEach(System.out::println);
    }
}
public class EmployeeManagementSystem {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        EmployeeManagement system = new EmployeeManagement();
        int choice;
        do {
           System.out.println("--------Employee Management System ---------- \n1. Add Employee\n2. 
Remove Employee\n3. Search Employee\n4. Update Employee\n5. Display All\n6. Exit");
            System.out.print("Enter choice: ");
            choice = sc.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Enter ID: ");
                    int id = sc.nextInt();
                    sc.nextLine();
                    System.out.print("Enter Name: ");
                    String name = sc.nextLine();
                    System.out.print("Enter Salary: ");
                    double salary = sc.nextDouble();
                    system.addEmployee(new Employee(id, name, salary));
                    break;
                case 2:
                    System.out.print("Enter ID to remove: ");
                    system.removeEmployee(sc.nextInt());
                    break;
                case 3:
                    System.out.print("Enter ID to search: ");
                    Employee emp = system.searchEmployee(sc.nextInt());
                    System.out.println(emp != null ? emp : "Employee not found");
                    break;
                case 4:
                    System.out.print("Enter ID to update: ");
                    int updateId = sc.nextInt();
                    sc.nextLine();
                    System.out.print("Enter New Name: ");
                    String newName = sc.nextLine();
                    System.out.print("Enter New Salary: ");
                    double newSalary = sc.nextDouble();
                    system.updateEmployee(updateId, newName, newSalary);
                    break;
                case 5:
                    system.displayEmployees();
                    break;
            }
        } while (choice != 6);
        sc.close();
    }
}
```

Output:



```
PS C:\Users\Asus\OneDrive\Desktop\PBLJ> java Exp3.EmployeeManagement

--------Employee Management System ---------
1. Add Employee
2. Remove Employee
3. Search Employee
4. Update Employee
5. Display All
6. Exit
Enter choice: 1
Enter ID: 10239
Enter Name: Ruchi Thakur
Enter Salary: 150000
Employee added successfully.

--------Employee Management System ---------
1. Add Employee
2. Remove Employee
3. Search Employee
4. Update Employee
5. Display All
6. Exit
Enter choice: 5
ID: 10239, Name: Ruchi Thakur, Salary: 150000.0
```

Learning Outcomes:

- Gained knowledge on utilizing ArrayList for dynamically storing and managing employee records.
- Learned the methods for adding, updating, deleting, and searching elements in an ArrayList.
- Learnt implementing search functionality using switch-case statements, loops, and conditions.

Problem 2:

Aim: 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.

Objective:

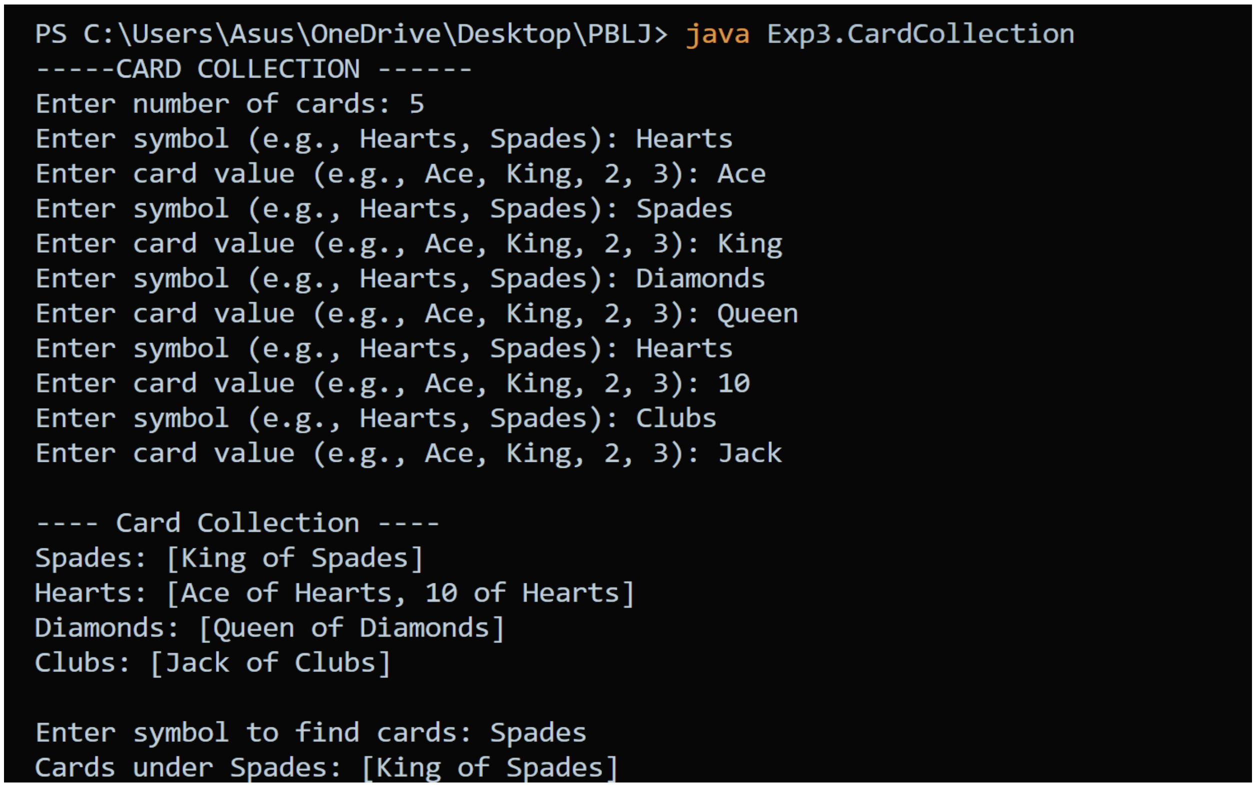
- To use the Java Collection Interface to effectively store and manage card information.
- To implement symbol-based searching to allow users to find all cards linked to a specific symbol.

- To ensure organized storage and retrieval by using suitable data structures such as HashSet or HashMap.

Code:

```
package Exp3;

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;

    }

    public String getValue() {

        return value;

    }

    @Override

    public String toString() {

        return value + " of " + symbol;

    }

}

class CardCollectionManager {

    private HashMap<String, List<Card>> cardCollection;


    public CardCollectionManager() {

        this.cardCollection = new HashMap<>();
    }
    public void addCard(String symbol, String value) {
        cardCollection.putIfAbsent(symbol, new ArrayList<>());
        cardCollection.get(symbol).add(new Card(symbol, value));
    }
    public List<Card> getCardsBySymbol(String symbol) {
        return cardCollection.getOrDefault(symbol, new ArrayList<>());
    }
    public void displayCards() {
        System.out.println("\n---- Card Collection ----");
        for (Map.Entry<String, List<Card>> entry : cardCollection.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
    }
}
public class CardCollection {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CardCollectionManager manager = new CardCollectionManager();
        System.out.print("-----CARD COLLECTION ------ \nEnter number of cards: ");
        int n = sc.nextInt();
        sc.nextLine(); // Consume newline

        for (int i = 0; i < n; i++) {
            System.out.print("Enter symbol (e.g., Hearts, Spades): ");
            String symbol = sc.nextLine();
            System.out.print("Enter card value (e.g., Ace, King, 2, 3): ");
            String value = sc.nextLine();
            manager.addCard(symbol, value);
        }
        manager.displayCards();
        System.out.print("\nEnter symbol to find cards: ");
        String findSymbol = sc.nextLine();
        List<Card> cards = manager.getCardsBySymbol(findSymbol);
        System.out.println("Cards under " + findSymbol + ": " + cards);
        sc.close();
    }
}
```

Output:



```
PS C:\Users\Asus\OneDrive\Desktop\PBLJ> java Exp3.CardCollection
-----CARD COLLECTION ------
Enter number of cards: 5
Enter symbol (e.g., Hearts, Spades): Hearts
Enter card value (e.g., Ace, King, 2, 3): Ace
Enter symbol (e.g., Hearts, Spades): Spades
Enter card value (e.g., Ace, King, 2, 3): King
Enter symbol (e.g., Hearts, Spades): Diamonds
Enter card value (e.g., Ace, King, 2, 3): Queen
Enter symbol (e.g., Hearts, Spades): Hearts
Enter card value (e.g., Ace, King, 2, 3): 10
Enter symbol (e.g., Hearts, Spades): Clubs
Enter card value (e.g., Ace, King, 2, 3): Jack

---- Card Collection ----
Spades: [King of Spades]
Hearts: [Ace of Hearts, 10 of Hearts]
Diamonds: [Queen of Diamonds]
Clubs: [Jack of Clubs]

Enter symbol to find cards: Spades
Cards under Spades: [King of Spades]
```

Learning Outcomes:

- Understand the Collection Interface and how to implement it for managing card data.
- Explored different Collection types like List, Set, or Map based on the use case.
- Learned how to choose the appropriate Collection implementation for different scenarios.

Problem 3:

Aim: To 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.

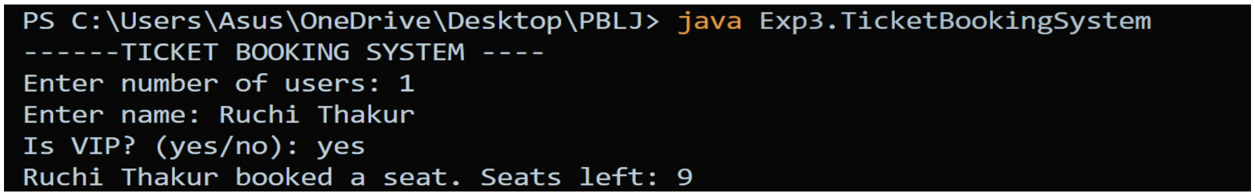
Objective:

- To use synchronized threads to avoid multiple users booking the same seat at the same time.
- To implement locks or synchronized methods to ensure thread safety.
- To assign higher thread priority to VIP bookings to ensure they are processed first.

Code:

```
package Exp3;

import java.util.*;

import java.util.concurrent.*;

// Interface for booking

interface Bookable {

    void bookSeat();

}

// Class for handling individual ticket bookings

class TicketBooking implements Runnable, Bookable {

    private static int availableSeats = 10;

    private final String name;

    private final boolean isVIP;

    public TicketBooking(String name, boolean isVIP) {

        this.name = name;

        this.isVIP = isVIP;

    }

    public boolean isVIP() {

        return isVIP;

    }

    public String getName() {
        return name;
    }
    @Override
    public synchronized void bookSeat() {
        if (availableSeats > 0) {
            System.out.println(name + " booked a seat. Seats left: " + (--availableSeats));
        } else {
            System.out.println(name + " booking failed. No seats available.");
        }
    }
    @Override
    public void run() {
        bookSeat();
    }
}
// Ticket Manager class to handle booking system logic
class TicketManager {
    private final PriorityQueue<TicketBooking> queue;
    private final ExecutorService executor;
    public TicketManager() {
        this.queue = new
PriorityQueue<>(Comparator.comparing(TicketBooking::isVIP).reversed());
        this.executor = Executors.newSingleThreadExecutor(); // Ensures sequential VIP execution
    }
    public void addBooking(String name, boolean isVIP) {
        queue.add(new TicketBooking(name, isVIP));
    }
    public void processBookings() {
        while (!queue.isEmpty()) {
            executor.execute(queue.poll());
        }
        executor.shutdown();
    }
}
// Main class for execution
public class TicketBookingSystem {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        TicketManager manager = new TicketManager();
        System.out.print("------TICKET BOOKING SYSTEM ---- \nEnter number of users: ");
        int n = sc.nextInt();
        sc.nextLine(); // Consume newline
        for (int i = 0; i < n; i++) {
            System.out.print("Enter name: ");
            String name = sc.nextLine();
            System.out.print("Is VIP? (yes/no): ");
            boolean isVIP = sc.nextLine().equalsIgnoreCase("yes");
            manager.addBooking(name, isVIP);
        }
        manager.processBookings();
        sc.close();
    }
}
```

Output:



```
PS C:\Users\Asus\OneDrive\Desktop\PBLJ> java Exp3.TicketBookingSystem
------TICKET BOOKING SYSTEM ----
Enter number of users: 1
Enter name: Ruchi Thakur
Is VIP? (yes/no): yes
Ruchi Thakur booked a seat. Seats left: 9
```

Learning Outcomes:

- Gained knowledge on creating and managing multiple threads by understanding the thread lifecycle and its various states.
- Learnt how to set and manage thread priorities.
- Understood how to set thread priorities to control the order of execution.