Experiment 4

Student Name: Bavneet Kaur	UID: 22BCS14121
Branch: BE-CSE	Section/Group: DL_903_B
Semester: 6th	Date: 11-02-2025
Subject Name: PBLJ	Subject Code: 22CSH-359

1. Aim: Solving problems under the category of Exception handling in Easy, Medium and Hard

2. Objective: Introduction to Exceptions. Difference between error and exception. Use of try, catch and throw. Difference between throw and throws. Types of Exceptions, Exception handling in Java.

3.Implementation/Code:

1.) 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.

Code:

```
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 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;
    }
}

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

    public static void main(String[] args) {
        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 Employees");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine();
            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! Please try again.");
            }
        }
    }

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

    private static void updateEmployee() {
        System.out.print("Enter Employee ID to update: ");
        int id = scanner.nextInt();
        scanner.nextLine();
        for (Employee emp : employees) {
            if (emp.getId() == id) {
                System.out.print("Enter new Name: ");
                emp.setName(scanner.nextLine());
                System.out.print("Enter new Salary: ");
                emp.setSalary(scanner.nextDouble());
                System.out.println("Employee updated successfully!");
                return;
            }
        }
        System.out.println("Employee not found!");
    }

    private static void removeEmployee() {
        System.out.print("Enter Employee ID to remove: ");
        int id = scanner.nextInt();
        if (employees.removeIf(emp -> emp.getId() == id)) {
            System.out.println("Employee removed successfully!");
        } else {
            System.out.println("Employee not found!");
        }
    }

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

    private static void displayEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees to display.");
        } else {
            System.out.println("\nEmployee List:");
            employees.forEach(System.out::println);
        }
    }
}
```

2.) 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.

Code:

```
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 Main {
    private static final Collection<Card> cards = new ArrayList<>();
    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        while (true) {
            System.out.println("\nCard Collection System");
            System.out.println("1. Add Card");
            System.out.println("2. Search Cards by Symbol");
            System.out.println("3. Display All Cards");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine();
            switch (choice) {
                case 1 -> addCard();
                case 2 -> searchCardsBySymbol();
                case 3 -> displayCards();
                case 4 -> {
                    System.out.println("Exiting...");
                    return;
                }
                default -> System.out.println("Invalid choice! Please try again.");
            }
        }
    }

    private static void addCard() {
        System.out.print("Enter Card Symbol (e.g., Hearts, Spades): ");
        String symbol = scanner.nextLine();
        System.out.print("Enter Card Value (e.g., Ace, King, 10): ");
        String value = scanner.nextLine();

        cards.add(new Card(symbol, value));
        System.out.println("Card added successfully!");
    }

    private static void searchCardsBySymbol() {
        System.out.print("Enter Symbol to Search: ");
        String symbol = scanner.nextLine();
        boolean found = false;

        for (Card card : cards) {
            if (card.getSymbol().equalsIgnoreCase(symbol)) {
                System.out.println(card);
                found = true;
            }
        }

        if (!found) {
            System.out.println("No cards found for the symbol " + symbol);
        }
    }

    private static void displayCards() {
        if (cards.isEmpty()) {
            System.out.println("No cards to display.");
        } else {
            System.out.println("\nCard Collection:");
            for (Card card : cards) {
                System.out.println(card);
            }
        }
    }
}
```

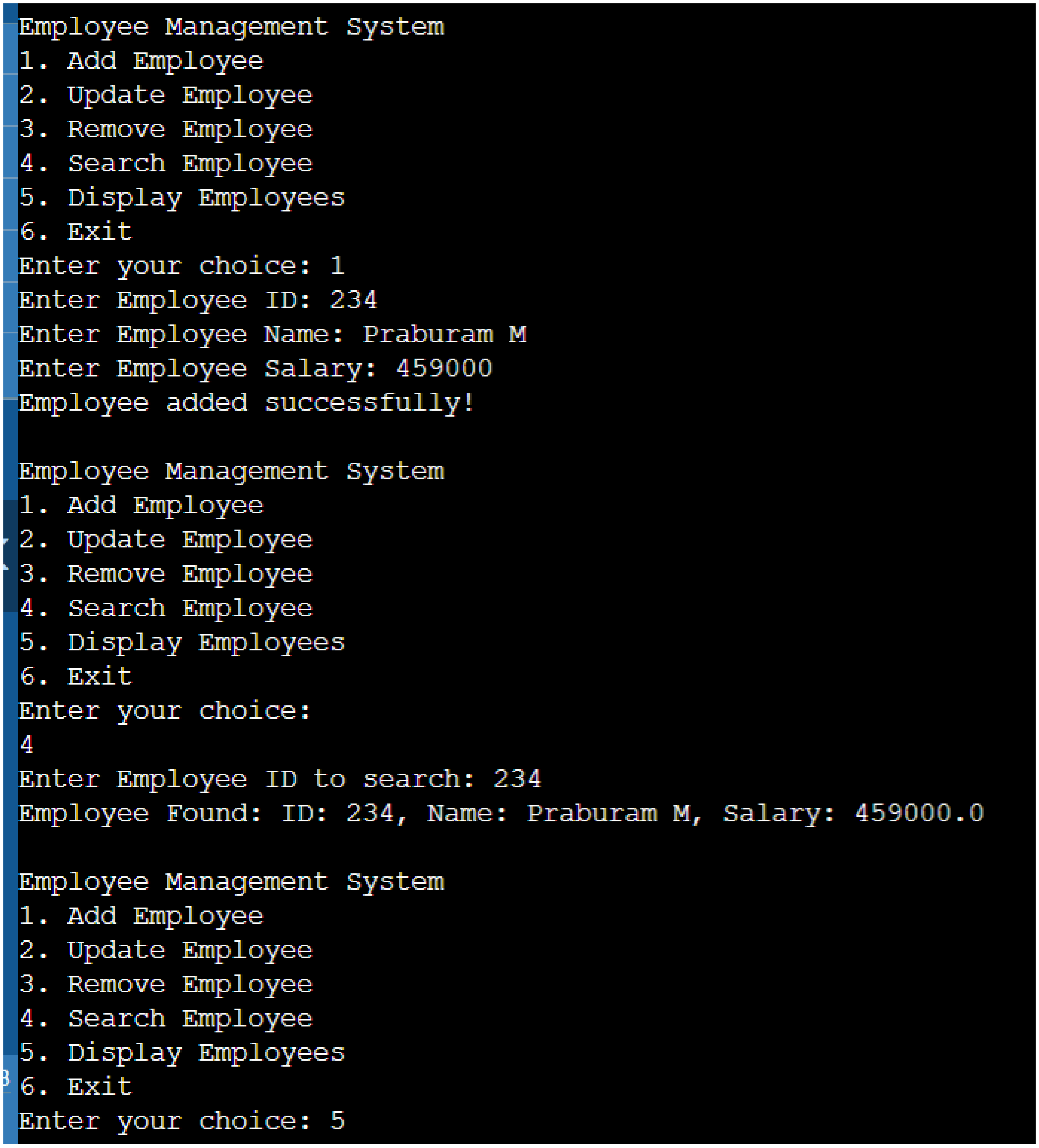
3.) Hard: Develop a ticket booking system with synchronized threads to ensure no double booking of seats. Use thread priorities to simulate VIP bookings being processed first.

Code:

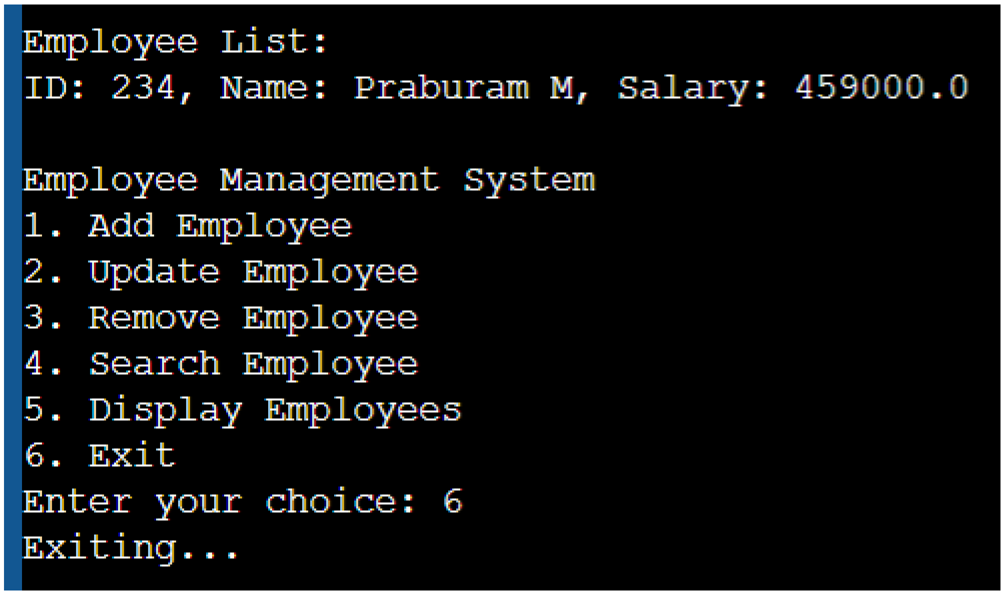
```
import java.util.*;

class TicketBookingSystem {
    private final boolean[] seats;

    public TicketBookingSystem(int numSeats) {
        seats = new boolean[numSeats];
    }

    public synchronized boolean bookSeat(int seatNumber) {
        if (seatNumber < 0 || seatNumber >= seats.length) {
            System.out.println("Invalid seat number.");
            return false;
        }
        if (!seats[seatNumber]) {
            seats[seatNumber] = true;
            System.out.println(Thread.currentThread().getName() + " successfully booked seat " + seatNumber);
            return true;
        } else {
            System.out.println(Thread.currentThread().getName() + " failed to book seat " + seatNumber + " (Already booked)");
            return false;
        }
    }
}

class BookingThread extends Thread {
    private final TicketBookingSystem system;
    private final int seatNumber;

    public BookingThread(TicketBookingSystem system, int seatNumber, String name, int priority) {
        super(name);
        this.system = system;
        this.seatNumber = seatNumber;
        setPriority(priority);
    }

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

public class Main {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem(10);
        Thread vip1 = new BookingThread(system, 5, "VIP-1", Thread.MAX_PRIORITY);
        Thread vip2 = new BookingThread(system, 5, "VIP-2", Thread.MAX_PRIORITY);
        Thread user1 = new BookingThread(system, 5, "User-1", Thread.NORM_PRIORITY);
        Thread user2 = new BookingThread(system, 5, "User-2", Thread.NORM_PRIORITY);

        vip1.start();
        vip2.start();
        user1.start();
        user2.start();
    }
}
```

5. Output

1.) Easy problem: Square root calculator

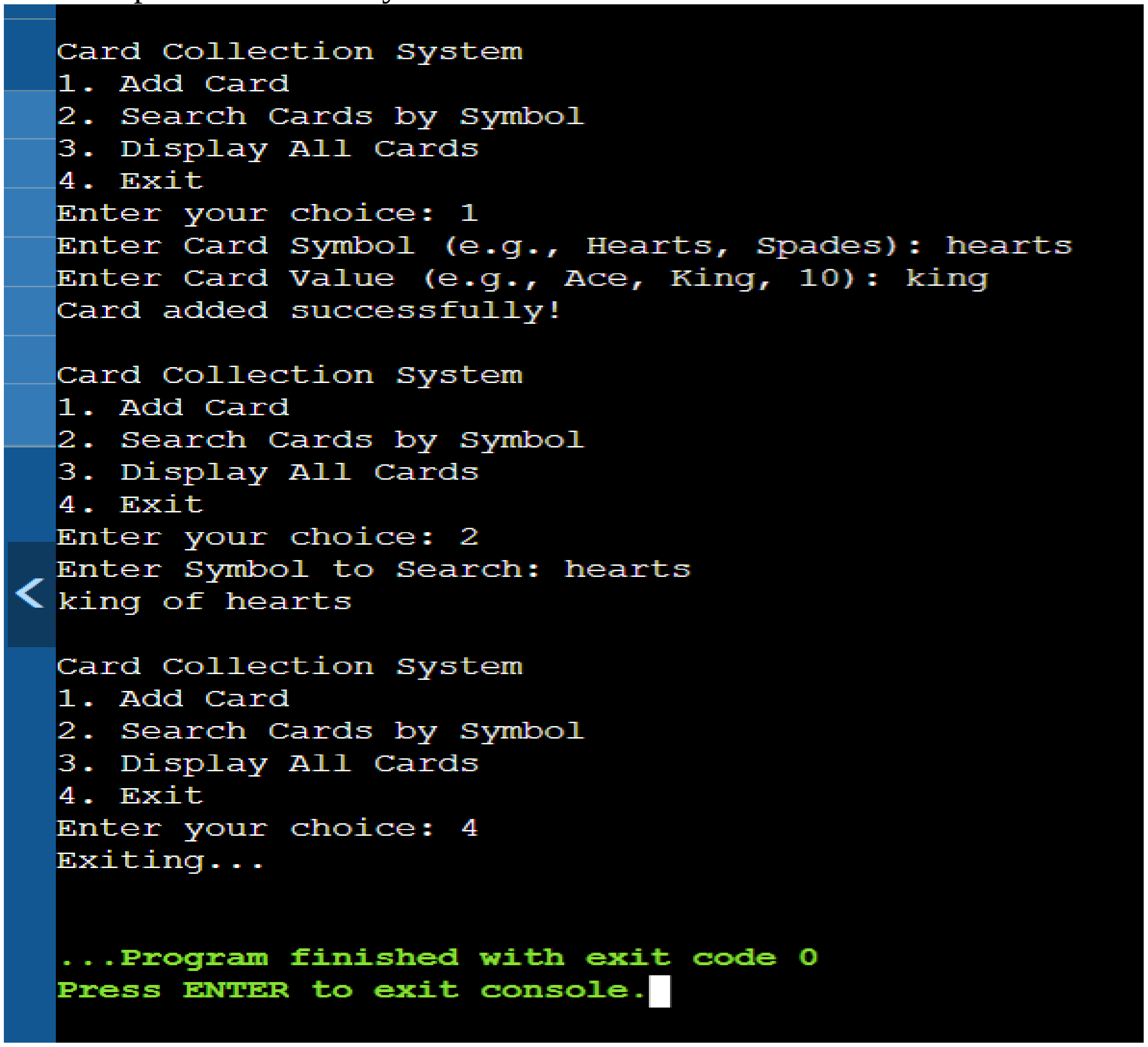


```
Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Enter your choice: 1
Enter Employee ID: 234
Enter Employee Name: Praburam M
Enter Employee Salary: 459000
Employee added successfully!

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Enter your choice:
4
Enter Employee ID to search: 234
Employee Found: ID: 234, Name: Praburam M, Salary: 459000.0

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Enter your choice: 5
```



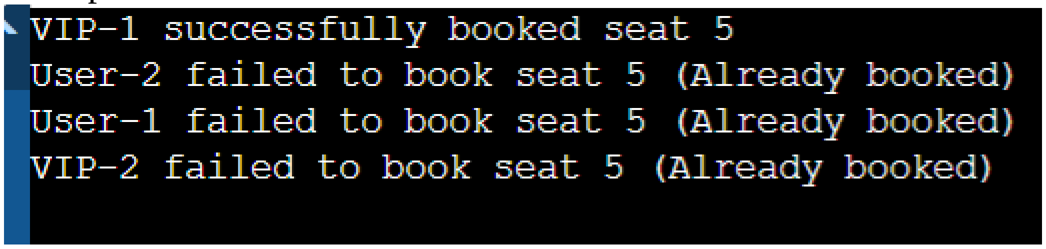
```
Employee List:
ID: 234, Name: Praburam M, Salary: 459000.0

Employee Management System
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display Employees
6. Exit
Enter your choice: 6
Exiting...
```

2.) Medium problem: ATM System



```
Card Collection System
1. Add Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Enter your choice: 1
Enter Card Symbol (e.g., Hearts, Spades): hearts
Enter Card Value (e.g., Ace, King, 10): king
Card added successfully!

Card Collection System
1. Add Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Enter your choice: 2
Enter Symbol to Search: hearts
king of hearts

Card Collection System
1. Add Card
2. Search Cards by Symbol
3. Display All Cards
4. Exit
Enter your choice: 4
Exiting...


...Program finished with exit code 0
Press ENTER to exit console.
```

3.) Hard problem:



```
VIP-1 successfully booked seat 5
User-2 failed to book seat 5 (Already booked)
User-1 failed to book seat 5 (Already booked)
VIP-2 failed to book seat 5 (Already booked)
```

6. Learning Outcomes

1. Learned to use classes and objects for organizing employee and designation data in Java.
2. Implemented salary calculations using switch-case and array data handling.
3. Practiced input handling with the Scanner class and validating user input.
4. Gained experience in searching arrays and structuring conditional logic.
5. Displayed formatted output for real-world applications like employee management systems.