JAVA ASSIGNMENT – 3

Course: Java Programming (ENCS201)
Name: CHIRAG SHARMA
Roll No.: 2401730183

Q1. Student Result Management System using Exception Handling

Code:

```
import
java.util.InputMismatchException;
import java.util.Scanner;
class InvalidMarksException extends Exception {
    public InvalidMarksException(String message) {
        super(message);
    }
}
class Student {
    private int rollNumber;
    private String studentName;
    private int[] marks = new int[3];
    public Student(int rollNumber, String studentName, int[] marks) throws InvalidMarksException {
        if (studentName == null || studentName.trim().isEmpty()) {
            throw new NullPointerException("Student name cannot be null or empty.");
        }
        this.rollNumber = rollNumber;
        this.studentName = studentName;
        this.marks = marks;
        validateMarks();
    }
    public void validateMarks() throws InvalidMarksException {
        for (int m : marks) {
            if (m < 0 || m > 100) {
                throw new InvalidMarksException("Marks should be between 0 and 100.");
            }
        }
    }
    public double calculateAverage() {
        int total = 0;
        for (int m : marks) total += m;
```

```
return total / 3.0;
```

```
}
    public int getRollNumber() {
        return rollNumber;
    }
    public void displayResult() {
        System.out.println("\n----- Student Result       ");
        System.out.println("Roll Number : " + rollNumber);
        System.out.println("Name        : " + studentName);
        System.out.println("Marks       : " + marks[0] + ", " + marks[1] + ", " + marks[2]);
        double avg = calculateAverage();
        System.out.println("Average     : " + avg);
        System.out.println("Result  : " + (avg >= 40 ? "PASS" : "FAIL"));
        System.out.println("                              \n");
    }
}
public class ResultManager {
    private static Student[] students = new Student[100];
    private static int count = 0;
    private static Scanner sc = new Scanner(System.in);
    public static void addStudent() {
        try {
            System.out.print("Enter Roll Number: ");
            int roll = sc.nextInt();
            sc.nextLine();
            System.out.print("Enter Student Name: ");
            String name = sc.nextLine();
            int[] marks = new int[3];
            System.out.println("Enter marks for 3
            subjects:");

            for (int i = 0; i < 3; i++) {
                System.out.print("Subject " + (i + 1) + ": ");
                marks[i] = sc.nextInt();
            }
            Student s = new Student(roll, name, marks);
            students[count++] = s;
            System.out.println("Student added successfully.\n");
```

```
    } catch (InputMismatchException e) {
        System.out.println("Invalid input. Enter numbers only.");
        sc.nextLine();
    } catch (InvalidMarksException e) {
        System.out.println(e.getMessage());
    } catch (NullPointerException e) {
        System.out.println(e.getMessage());
    }
}
public static void main(String[] args) {
    try {
        int choice;
        do {
            System.out.println("===== Student Result Management =====");
            System.out.println("1. Add Student");
            System.out.println("2. Show Student Details");
            System.out.println("3. Exit");
            System.out.print("Enter choice: ");
            choice = sc.nextInt();
            switch (choice) {
                case 1:
                    addStudent();
                    break;
                case 2:
                    System.out.print("Enter Roll Number: ");
                    int r = sc.nextInt();
                    boolean found = false;

                    for (int i = 0; i < count; i++) {
                        if (students[i].getRollNumber() == r) {
                            students[i].displayResult();
                            found = true;
                            break;
                        }
                    }
```

```
                if (!found) {
                    System.out.println("Student not found.\n");
                }
                break;
            case 3:
                System.out.println("Exiting system...");
                break;
            default:
                System.out.println("Invalid choice!");
        }
    } while (choice != 3);
  } finally {
    System.out.println("Closing application. Scanner released.");
    sc.close();
  }
 }
}
```

2. Explain Life cycle of Thread. Develop a multithreaded Java application where two threads print numbers from 1 to 10. Use synchronization to ensure one thread prints only odd numbers and the other prints only even numbers in sequence.

ANSWER:

A thread in Java goes through several states during its execution. These states are defined in the Thread.State enumeration. The thread life cycle describes how a thread is created, scheduled, executed, paused, and terminated.

1. New (Born State)

When a thread object is created using the Thread class but the start() method is not called, it is in the **New** state.

2. Runnable (Ready-to-Run State)

After calling the start() method, the thread enters the **Runnable** state.
In this state, the thread is ready to run and is waiting for the CPU scheduler to allocate time.

3. Running (Active State)

When the thread scheduler picks a thread from the runnable pool, it enters the **Running** state and starts executing the run() method.

4. Blocked / Waiting / Timed Waiting

A thread moves to **Blocked**, **Waiting**, or **Timed Waiting** when it cannot proceed temporarily.

- **Blocked:** waiting for a lock (e.g., synchronized block)
- **Waiting:** waiting indefinitely for another thread to notify
- **Timed Waiting:** waiting for a specified period (e.g., sleep(ms))

5. Terminated (Dead State)

When the thread finishes execution of the run() method or stops due to an exception, it reaches the **Terminated** state. A terminated thread cannot be restarted.

CODE:

```
class NumberPrinter {
   private int number = 1;
   public synchronized void printOdd() {
      while (number <= 10) {
         if (number % 2 == 1) {
            System.out.println("Odd Thread: " + number);
            number++;
            notify();
         } else {
            try { wait(); } catch (Exception e) {}
         }
```

```
        }
        notify();
    }
    public synchronized void printEven() {
        while (number <= 10) {
            if (number % 2 == 0) {
                System.out.println("Even Thread: " + number);
                number++;
                notify();
            } else {
                try { wait(); } catch (Exception e) {}
            }
        }
        notify();
    }
}
class OddThread extends Thread {
    NumberPrinter np;
    OddThread(NumberPrinter np) { this.np = np; }
    public void run() { np.printOdd(); }
}
class EvenThread extends Thread {
    NumberPrinter np;
    EvenThread(NumberPrinter np) { this.np = np; }
    public void run() { np.printEven(); }
}
public class Main {
    public static void main(String[] args) {
        NumberPrinter np = new NumberPrinter();
        Thread t1 = new OddThread(np);
        Thread t2 = new EvenThread(np);
        t1.start();
        t2.start();
    }
}
```