CSIT751 (Core Java)

Module III – Exception Handling Interface and Thread in Java

Programming Exercises for Practice (Practical Home Assignment)

Q1. ATM Withdrawal System – Handling Insufficient Balance

Scenario:
In a banking system, when a user tries to withdraw more money than their account balance, the system should throw an exception and display an appropriate message.

Program Structure:

- Class: `ATM`
    - Field: `balance`
    - Method: `withdraw(double amount)` — throws exception if amount > balance
- Exception: Use **`throw`** and **`try-catch`**
- Use **finally block** to display "Transaction Completed".

Input Format:

```
Enter balance: 5000
Enter amount to withdraw: 6000
```

Expected Output:

```
Exception: Insufficient Balance
Transaction Completed
```

Q2. Student Age Validation – User Defined Exception

Scenario:
While registering students, the age must be above 17. If not, throw a user-defined exception.

Program Structure:

- Class: `Student`
- User Defined Exception: `InvalidAgeException`
- Method: `registerStudent(int age)` — throws `InvalidAgeException` if age < 17
- Handle using try-catch.

Input Format:

```
Enter name: Rohan
Enter age: 15
```

Expected Output:

```
Exception: Age must be above 17 for registration
```

Q3. Array Division – Handling Arithmetic and Array Index Exception (Multiple Catch)

Scenario:
A program takes two numbers and an array index. It divides array[index] by the number. You need to handle both `ArithmeticException` (if divisor = 0) and `ArrayIndexOutOfBoundsException` (if index invalid).

Program Structure:

- Class: `ArrayDivision`
- Use **multiple catch blocks**.

Input Format:

```
Array: [10, 20, 30, 40]
Enter index: 5
Enter divisor: 2
```

Expected Output:

```
Exception: Array index out of bounds
```

(If divisor = 0 → "Division by zero not allowed")

Q4. Student Marks Entry – Input Validation

Scenario:
If a user enters a negative mark or mark greater than 100, throw an exception. Also, ensure that program **always displays a thank you message** using `finally`.

Program Structure:

- Class: `StudentMarks`
- Method: `enterMarks(int marks)` — throws exception for invalid range
- Use try-catch-finally.

Input Format:

```
Enter marks: 105
```

Expected Output:

```
Exception: Marks must be between 0 and 100
Thank you for using the system
```

Q5. University Login System – Null Pointer Handling

Scenario:
In a university login system, if a user tries to log in with a null username, a `NullPointerException` occurs. Handle this gracefully.

Program Structure:

- Class: `UniversityLogin`
- Method: `login(String username)`
- Throw or let `NullPointerException` occur and handle it using try-catch.

Input Format:

```
Enter username: null
```

Expected Output:

```
Exception: Username cannot be null
```

Q6. Online Shopping – Minimum Purchase Amount Exception

Scenario:
An online shopping system requires a minimum cart value of ₹500 for placing an order. If below that, throw a user-defined exception.

Program Structure:

- Class: `OnlineShopping`
- User Defined Exception: `MinimumAmountException`
- Method: `placeOrder(int amount)` — throws exception if amount < 500

Input Format:

```
Enter cart amount: 300
```

Expected Output:

```
Exception: Minimum cart value must be ₹500
```

Q7. Railway Ticket Booking – Synchronization of Threads

Scenario:
In a railway booking system, multiple users are trying to book tickets simultaneously. Only a limited number of seats are available. Synchronization must ensure that no two users book the same seat.

Program Structure:

- Class: `TicketBooking` (Shared Resource)
  - Field: `availableSeats`
  - Method: `bookSeat(int seats)` — synchronized
- Thread classes: `User1`, `User2` extending `Thread`
- Use synchronization to prevent race conditions.

Input Format:

```
Available Seats: 2
User1 wants to book: 1
User2 wants to book: 2
```

Expected Output:

```
User1 booked 1 seat(s) successfully
User2 booking failed. Not enough seats
```

Q8. University Printing System – Multiple Threads

Scenario:
A university has multiple printers. Students can send print jobs simultaneously. Each print job runs on a separate thread.

Program Structure:

- Class: `PrinterJob` implements `Runnable`
- Create multiple threads for different students.
- Simulate printing using `Thread.sleep()`.

Input Format:

```
Number of print jobs: 3
```

Expected Output:

```
Printing job 1 by Student A
Printing job 2 by Student B
Printing job 3 by Student C
```

Q9. IoT Sensor Monitoring System

Scenario: An IoT sensor generates a random number every second.

- If even → Square is calculated by one thread.
- If odd → Cube is calculated by another thread.

Menu:

```
1. Start Simulation
2. Exit
```

Program Structure:

- `RandomNumberGenerator` (Thread) → generates numbers.
- `SquareCalculator` (Thread) → processes even numbers.
- `CubeCalculator` (Thread) → processes odd numbers.
- Uses `Thread.sleep()`, multithreading, and conditional checks.

Input:

```
1
```

Expected Output:

```
Generated: 4
Square: 16
Generated: 7
Cube: 343
Generated: 10
Square: 100
```

Q10. Bank Transaction System – Thread Priorities

Scenario:
In a banking application, high-value transactions must be processed with higher priority than low-value transactions.

Program Structure:

- Class: `BankTransaction` extends `Thread`
- Create threads for different transactions and set priorities: `MAX_PRIORITY` for high-value, `MIN_PRIORITY` for low-value.

Input Format:

```
Transaction1: ₹5000
Transaction2: ₹50000
```

Expected Output:

```
High-value transaction processed first
Low-value transaction processed later
```