EXPERIMENT 05

Objective

To create a Java program demonstrating advanced error-handling techniques using exception handling and multithreading.

Theory

Java exception handling allows developers to manage unexpected events at runtime using try-catch-finally blocks. Multithreading enables concurrent execution of two or more parts of a program for better resource utilization and efficiency.

Key Concept	Description
Exception Handling	Managing errors through structured blocks (try/catch).
Thread	A lightweight subprocess that can run concurrently.
Runnable Interface	Provides abstraction for thread behavior.
InterruptedException	Signals that a thread has been interrupted.

Main Code: Runnable with Exception

```
// TaskRunner.java
class TaskRunner implements Runnable {
    @Override
    public void run() {
        try {
            System.out.println("Thread Task Started");
            Thread.sleep(1000);
            throw new InterruptedException("Simulated interruption occurred!");
        } catch (InterruptedException ex) {
            System.out.println("Handled Exception: " + ex.getMessage());
        }
    }
}

// ThreadDemo.java
public class ThreadDemo {

    public static void main(String[] args) {
        Thread taskThread = new Thread(new TaskRunner());
        taskThread.start();
    }
}
```

Sample Output

Thread Task Started
Handled Exception: Simulated interruption occurred!

Output Explanation

The Runnable task initiates a thread and simulates a delay using Thread.sleep(). An artificial exception is thrown and successfully handled inside the catch block.

Coding Problem A: Negative Amount Exception

```
// InvalidTransactionException.java

class InvalidTransactionException extends Exception {
    public InvalidTransactionException(String message) {
        super(message);
    }
}

// TransactionValidator.java

public class TransactionValidator {
    public static void validateAmount(double amount) throws InvalidTransactionException {
        if (amount < 0) {
            throw new InvalidTransactionException("Invalid amount: Negative balance detected :" +
amount);
        } else {
            System.out.println("Transaction Amount Accepted: " + amount);
        }
    }
    public static void main(String[] args) {
        try {
            validateAmount(2500.75);
            validateAmount(-1200.00);
        } catch (InvalidTransactionException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

Output

Transaction Amount Accepted: 2500.75

Exception: Invalid amount: Negative balance detected : -1200.0

Explanation

This program uses a custom checked exception to validate if a transaction amount is negative. If the value is less than 0, an `InvalidTransactionException` is thrown and caught.

Coding Problem B: Even and Odd Threads with Sleep Timing

```
// EvenThread.java

class EvenThread extends Thread {

    @Override
    public void run() {
        for (int number = 2; number <= 10; number += 2) {
            System.out.println("Even Thread: " + number);
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                System.out.println("Even Thread Error: " +
e.getMessage());
            }
        }
    }
}
```

```


// OddThread.java

class OddThread extends Thread {
    @Override
    public void run() {
        for (int number = 1; number < 10; number += 2) {
            System.out.println("Odd Thread: " + number);
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                System.out.println("Odd Thread Error: " + e.getMessage());
```

```
            }
        }
    }
}
```

```
// ThreadManager.java

public class ThreadManager {
    public static void main(String[] args) {
        EvenThread evenThread = new EvenThread();
        OddThread oddThread = new OddThread();
        evenThread.start();
        oddThread.start();
    }
}
```

Output

Odd Thread: 1

Even Thread: 2

Even Thread: 4

Even Thread: 6

Odd Thread: 3

Even Thread: 8

Even Thread: 10

Odd Thread: 5

Odd Thread: 7

Odd Thread: 9

Explanation

Two threads print even and odd numbers with different sleep intervals. This showcases thread synchronization and timing control using `Thread.sleep()` inside custom threads.