Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering
Semester: (Summer, Year:2021), B.Sc. in CSE (Day)

Lab Report NO #3

Course Title:Object Oriented Programming

Course Code: CSE202 Section:242_d2

Lab Experiment Name: Threading

Student Details

Name		ID
1.	Tushar Rajbongshi	242002140

Submission Date : 21/12/2025
Course Teacher's Name : Sabbir Hosen Mamun

Lab Report Status

Marks: …… **Signature:.....**
Comments:..... **Date:.....**

1. TITLE OF THE LAB REPORT EXPERIMENT:Threading

2.OBJECTIVES:

1. We can demonstrate the use of **Multithreading** in Java.
2. We can create a thread by **extending the Thread class**.
3. We can create another thread by **implementing the Runnable interface**.
4. We can run two threads at the same time to demonstrate **concurrent execution**.
5. We can use a **500-millisecond delay** between each output.
6. We can use a **shared counter variable**.
7. We can prevent **data inconsistency** by applying **synchronization**.

3.Theory:

Threading in Java allows a program to perform multiple tasks concurrently, improving performance and responsiveness. Threads are lightweight sub-processes that share the same memory, making communication efficient.

Importance of Threading in Java:

Improved Performance: Threads run in parallel on multi-core processors, reducing execution time.

Responsive Applications: Time-consuming tasks run in separate threads, keeping the user interface responsive.

Resource Sharing: Threads share memory and resources with less overhead than processes.

Simplified Design: Large tasks can be divided into smaller, manageable parts.
Creating Threads in Java

Threads can be created in two ways:

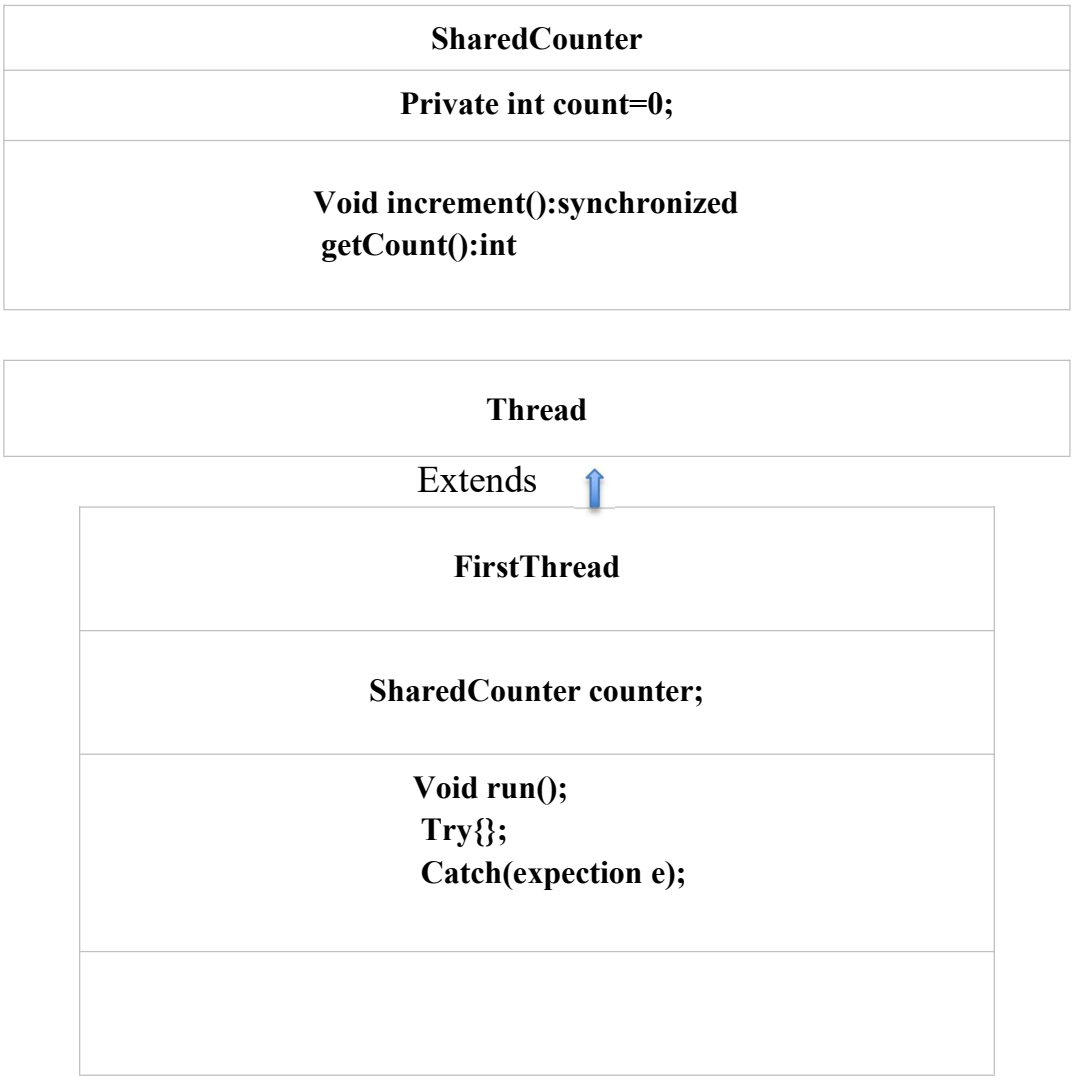
Extending the Thread Class: Override the run() method and start the thread using start().

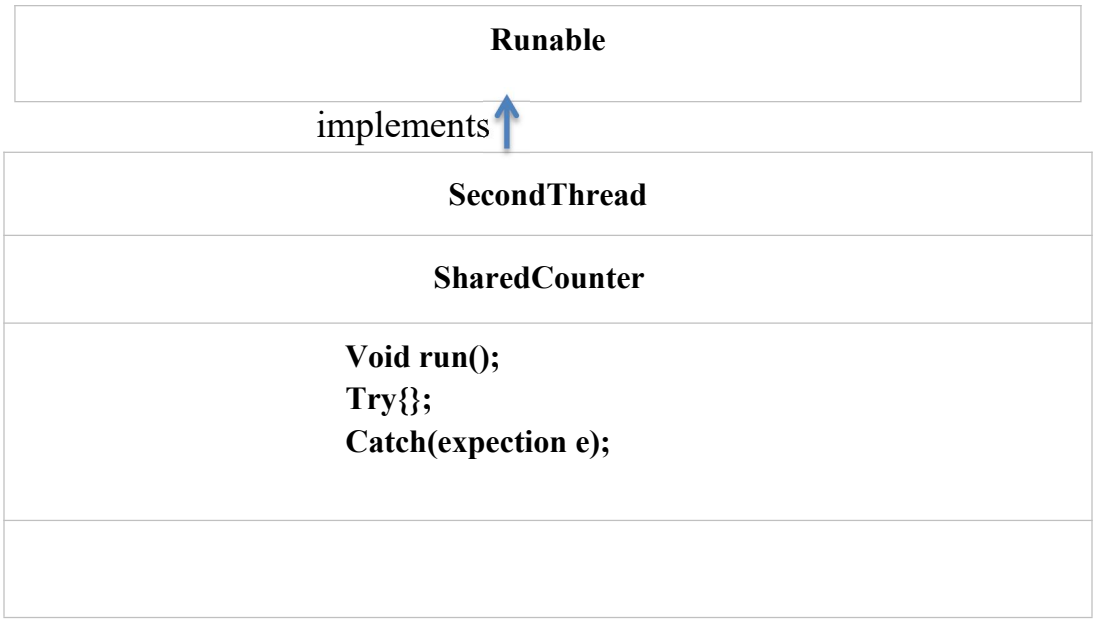
Implementing the Runnable Interface: Define the task in the run() method and pass it to a Thread object.

Key Points:
Always use start() to run a thread concurrently.
Thread execution order is controlled by the operating system and is unpredictable.

4.PROCEDURE / ANALYSIS / DESIGN :

4.1: UML class diagram:





5. IMPLEMENTATION:

```
class SharedCounter {
    private int count = 0;
    public synchronized void increment() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

```
class FirstThread extends Thread{
    SharedCounter counter;
    public FirstThread(SharedCounter counter){
        this.counter = counter;
    }
    public void run(){
        try{
        for(int i=1; i<=10; i++){
            System.out.println("First Thread: " + i);
            counter.increment();
            System.err.println("SharedCounter count: "+counter.getCount());
            Thread.sleep(500);
        }
    } catch(Exception e){
```

```
        System.out.println(e);
    }
}
}
class SecondThread implements Runnable{
    SharedCounter counter;
    public SecondThread(SharedCounter counter){
        this.counter = counter;
    }

    public void run(){
         try {
        for(int i=1; i<=10; i++){
            System.out.println("Square Thread: " + (i*i));
            counter.increment();
            Thread.sleep(500);
        }
    } catch(InterruptedException e){
        System.out.println(e);
}
}
}
```

```
public class Threading {
    public static void main(String[] args) {
        SharedCounter counter = new SharedCounter();
        FirstThread t1 = new FirstThread(counter);
        Thread t2 = new Thread(new SecondThread(counter));
        t1.start();
        t2.start();
    }
}
```

6.TEST RESULT / OUTPUT:

```
[Running] cd "c:\Users\USER\Downloads\New folder\" && javac Threading.java &&
java Threading
Square Thread: 1
First Thread: 1
SharedCounter count: 2
Square Thread: 4
First Thread: 2
SharedCounter count: 4
Square Thread: 9
```

```
First Thread: 3
SharedCounter count: 6
Square Thread: 16
First Thread: 4
SharedCounter count: 8
Square Thread: 25
First Thread: 5
SharedCounter count: 10
Square Thread: 36
First Thread: 6
SharedCounter count: 12
Square Thread: 49
First Thread: 7
SharedCounter count: 14
Square Thread: 64
First Thread: 8
SharedCounter count: 16
Square Thread: 81
First Thread: 9
SharedCounter count: 18
Square Thread: 100
First Thread: 10
SharedCounter count: 20

[Done] exited with code=0 in 5.661 seconds
```

7. ANALYSIS AND DISCUSSION:

This program demonstrates the use of **Multithreading** in Java, where two threads operate simultaneously. It illustrates the two standard methods of thread creation: one thread is created by extending the **Thread class**, and the other is created by implementing the **Runnable interface**.

The first thread prints numbers from 1 to 10, while the second thread prints the squares of those numbers. A 500-millisecond delay is applied between each output, causing the outputs of both threads to interleave.

Both threads utilize a **shared counter variable**. To prevent data inconsistency, the method responsible for updating the counter has been **synchronized**, ensuring that only one thread can modify the counter at a time.

In summary, this program clearly explains the importance of multithreading, concurrent execution, and synchronization.

8. GitHub Repository Link:

https://github.com/Rajbongshi2005/LabReport03-Threading.git