CO3408 – Lab 6: Concurrency and Synchronization

Title: Exploring Concurrency in Java

Learning Outcomes:

By the end of this lab, students should be able to:

1. Understand and demonstrate basic concurrency in Java using threads.
2. Observe and explain the effects of unsynchronized access to shared resources.
3. Apply synchronization to ensure correct concurrent behaviour.

Part (1): Introduction (Discussion)

Before you start coding:

1. What is concurrency, and why is it needed in software systems?
2. What are some examples of concurrent systems in everyday software?
3. What is the difference between concurrency and parallelism?

Discuss in small groups (3–5 students).

Part (2): Basic Thread Creation

Task 1 – Creating and Running Threads

Create a class SimpleThread that extends Thread. Each thread should print its name and a counter.

```java
public class SimpleThread extends Thread {
    private String name;
    public SimpleThread(String name) {
        this.name = name;
    }
    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(name + " - Count: " + i);
            try {
                Thread.sleep(400); // simulate work
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
    public static void main(String[] args) {
        SimpleThread t1 = new SimpleThread("Thread-A");
        SimpleThread t2 = new SimpleThread("Thread-B");
        t1.start();
        t2.start();
    }
}
```

Questions:

1. How do the outputs from both threads appear?
2. Do they always run in the same order? Why or why not?

Part (3): Shared Resource Problem (Race Condition)

Task 2 – Unsynchronized Access

Create a Counter class that is accessed by multiple threads.

```
class Counter {
    private int count = 0;
    public void increment() {
        count++;
        System.out.println(Thread.currentThread().getName() + " count: " + count);
    }
}
public class RaceConditionDemo {
    public static void main(String[] args) {
        Counter counter = new Counter();
        Runnable task = () -> {
            for (int i = 0; i < 5; i++) {
                counter.increment();
                try { Thread.sleep(100); } catch (InterruptedException e) {}
            }
        };
        Thread t1 = new Thread(task, "Thread-1");
        Thread t2 = new Thread(task, "Thread-2");
        t1.start();
        t2.start();
    }
}
```

Observe:

- Is the final count correct (should be 10)?
- Why might it not be correct?

Two threads are updating the same variable without coordination.

Part (4): Synchronizing Access

Task 3 – Using synchronized

Modify the increment() method to be synchronized:

```
public synchronized void increment() {
    count++;
    System.out.println(Thread.currentThread().getName() + " count: " + count);
}
```

Observe Again:

- Is the output now consistent?
- What has changed?

Discussion Questions:

1. What is a *critical section*?
2. What does the synchronized keyword do behind the scenes?
3. Can synchronization affect performance?

Part (5): Higher-Level Concurrency (Executors)

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
public class ExecutorDemo {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(3);
        Runnable task = () -> {
            System.out.println("Running in: " +
Thread.currentThread().getName());
            try { Thread.sleep(500); } catch (InterruptedException e) {}
        };
        for (int i = 0; i < 5; i++) {
            executor.submit(task);
        }
        executor.shutdown();
    }
}
```

Task 4 – Using the Executor Framework

Use a thread pool to run multiple concurrent tasks.

Questions:

- How many threads are actually running?
- Why does the output order vary each time you run it?

Part (6): Reflection (Short Answers)

1. What did you learn about concurrency in Java?
2. What problems can occur without synchronization?