2023F-BSE-064
AIMA KHAN

# LAB # 08

# Open Ended Lab

## Open-Ended Lab Task: GitHub Integration and Concurrency in Java

**Objective**: In this lab, you will explore essential practices for using GitHub repository and understand multithreading and inter-thread communication in Java.

1. **Concurrency with Multithreading Mechanisms**
   - Create a class with multiple threads that perform different tasks (e.g., one thread handles calculations, another handles data logging).
   - Implement the `start`, `sleep`, and `stop` functionalities to demonstrate different thread lifecycle states.
   - Use `join()` where necessary to ensure one thread completes before another begins, simulating dependency between threads.
   - **Deliverable**: Code implementing multithreading with clear comments explaining the use of each concurrency method (`start`, `sleep`, `stop`, and `join`).

## Additional Challenge:

1. Experiment with `ReentrantLock` to replace synchronized blocks, allowing for finer control over thread locking.

**CODE:**

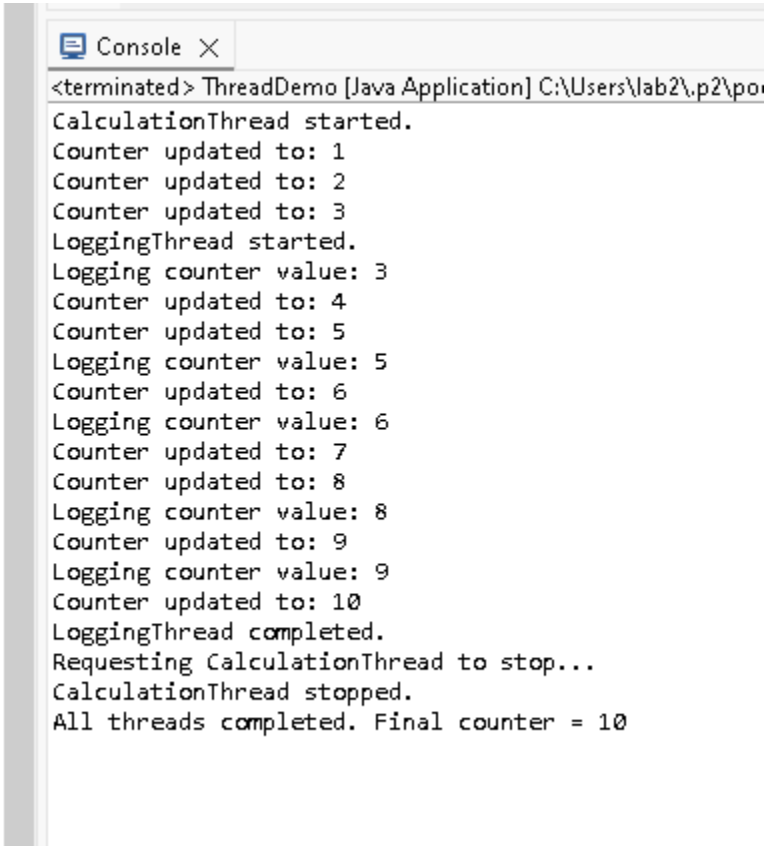
```java
package AIMA;

import java.util.concurrent.locks.ReentrantLock;

class SharedResource {
    private int counter = 0;

    // ReentrantLock gives finer control than synchronized
    private final ReentrantLock lock = new ReentrantLock();

    public void increment() {
        lock.lock();  // Acquire lock manually
        try {
            counter++;
            System.out.println("Counter updated to: " + counter);
        } finally {
            lock.unlock(); // Always unlock to avoid deadlocks
        }
    }

    public int getCounter() {
        return counter;
    }
}

```

```java
package AIMA;

class CalculationThread extends Thread {
    private SharedResource resource;
    private volatile boolean running = true;

    public CalculationThread(SharedResource resource) {
        this.resource = resource;
    }

    // Safe stop method using a flag—avoid deprecated Thread.stop()
    public void requestStop() {
        running = false;
    }

    @Override
    public void run() {
        System.out.println("CalculationThread started.");

        while (running) {
            resource.increment();

            try {
                // Sleep shows Timed Waiting state
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println("CalculationThread interrupted.");
            }
        }

        System.out.println("CalculationThread stopped.");
    }
}
```

```java
package AIMA;

class LoggingThread extends Thread {
    private SharedResource resource;

    public LoggingThread(SharedResource resource) {
        this.resource = resource;
    }

    @Override
    public void run() {
        System.out.println("LoggingThread started.");

        // Just log the value 5 times
        for (int i = 1; i <= 5; i++) {
            System.out.println("Logging counter value: " + resource.getCounter());

            try {
                Thread.sleep(700);  // Sleep to slow logging
            } catch (InterruptedException e) {
                System.out.println("LoggingThread interrupted.");
            }
        }

        System.out.println("LoggingThread completed.");
    }
}
```

```java
package AIMA;

public class ThreadDemo {
    public static void main(String[] args) {
        SharedResource resource = new SharedResource();

        CalculationThread calcThread = new CalculationThread(resource);
        LoggingThread logThread = new LoggingThread(resource);

        calcThread.start();


        try {
            Thread.sleep(1500); // Let counter update a few times
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        logThread.start();

        try {
            logThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Requesting CalculationThread to stop...");

        // ➤ Safe stop using a flag
        calcThread.requestStop();

        try {
            calcThread.join(); // Ensure calc thread finishes before exiting program
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("All threads completed. Final counter = " + resource.getCounter());
    }
}
```

**OUTPUT:**




2. **Inter-Thread Communication Using Synchronization**
    - Develop a program where two threads communicate via shared resources.
    - Use synchronization techniques (such as synchronized methods or blocks) to ensure that shared resources are accessed safely by each thread.
    - Example scenario: Implement a "Producer-Consumer" pattern, where one thread (Producer) adds items to a buffer, and another thread (Consumer) removes them, using `wait()` and `notify()` methods.
    - **Deliverable**: A fully functional inter-thread communication program demonstrating correct use of synchronization.

## Additional Challenge:

Add logging to record thread actions (such as start, wait, notify, stop) to a file for a clear view of thread operations over time.

# CODE

```java
package AIMY;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.LinkedList;
import java.util.Queue;

class Logger {
    private static PrintWriter writer;

    static {
        try {
            writer = new PrintWriter(new FileWriter("thread_log.txt", true));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static synchronized void log(String message) {
        writer.println(message);
        writer.flush();
        System.out.println(message);
    }
}
```

```java
package AIMY;

import java.util.LinkedList;
import java.util.Queue;

class Buffer {
    private Queue<Integer> queue = new LinkedList<>();
    private int capacity;

    public Buffer(int capacity) {
        this.capacity = capacity;
    }

    public synchronized void produce(int item) throws InterruptedException {
        while (queue.size() == capacity) {
            Logger.log("Buffer full - Producer WAITING...");
            wait();   // Producer waits
        }

        queue.add(item);
        Logger.log("Produced: " + item + " | Buffer size: " + queue.size());

        notify();   // Notify consumer waiting
        Logger.log("Producer issued NOTIFY to Consumer.");
    }

    public synchronized int consume() throws InterruptedException {
        while (queue.isEmpty()) {
            Logger.log("Buffer empty - Consumer WAITING...");
            wait();   // Consumer waits
        }

        int item = queue.remove();
        Logger.log("Consumed: " + item + " | Buffer size: " + queue.size());

        notify();  // Notify producer waiting
        Logger.log("Consumer issued NOTIFY to Producer.");

        return item;
    }
}
```

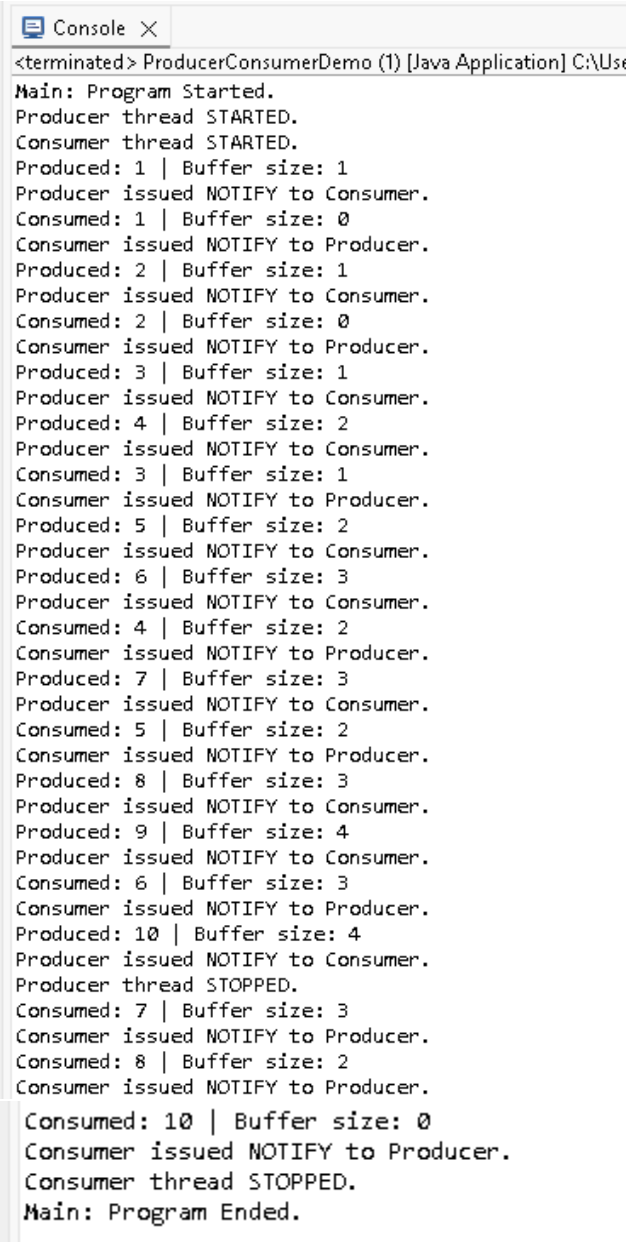
```java
package AIMY;

class Producer extends Thread {
    private Buffer buffer;

    public Producer(Buffer buffer) {
        this.buffer = buffer;
    }

    @Override
    public void run() {
        Logger.log("Producer thread STARTED.");

        for (int i = 1; i <= 10; i++) {
            try {
                buffer.produce(i);
                Thread.sleep(300);  // Simulate production time
            } catch (InterruptedException e) {
                Logger.log("Producer INTERRUPTED.");
            }
        }

        Logger.log("Producer thread STOPPED.");
    }
}
```

```java
package AIMY;

class Consumer extends Thread {
    private Buffer buffer;

    public Consumer(Buffer buffer) {
        this.buffer = buffer;
    }

    @Override
    public void run() {
        Logger.log("Consumer thread STARTED.");

        for (int i = 1; i <= 10; i++) {
            try {
                buffer.consume();
                Thread.sleep(500);  // Simulate consumption time
            } catch (InterruptedException e) {
                Logger.log("Consumer INTERRUPTED.");
            }
        }

        Logger.log("Consumer thread STOPPED.");
    }
}
```

*Logger.java Buffer.java Producer.java Consumer.j

```java
package AIMY;

public class ProducerConsumerDemo {
    public static void main(String[] args) {
        Logger.log("Main: Program Started.");

        Buffer buffer = new Buffer(5);

        Producer producer = new Producer(buffer);
        Consumer consumer = new Consumer(buffer);

        producer.start();
        consumer.start();

        try {
            producer.join();
            consumer.join();
        } catch (InterruptedException e) {
            Logger.log("Main thread INTERRUPTED.");
        }

        Logger.log("Main: Program Ended.");
    }
}
```

**OUTPUT**


Console
<terminated> ProducerConsumerDemo (1) [Java Application] C:\Use
Main: Program Started.
Producer thread STARTED.
Consumer thread STARTED.
Produced: 1 | Buffer size: 1
Producer issued NOTIFY to Consumer.
Consumed: 1 | Buffer size: 0
Consumer issued NOTIFY to Producer.
Produced: 2 | Buffer size: 1
Producer issued NOTIFY to Consumer.
Consumed: 2 | Buffer size: 0
Consumer issued NOTIFY to Producer.
Produced: 3 | Buffer size: 1
Producer issued NOTIFY to Consumer.
Produced: 4 | Buffer size: 2
Producer issued NOTIFY to Consumer.
Consumed: 3 | Buffer size: 1
Consumer issued NOTIFY to Producer.
Produced: 5 | Buffer size: 2
Producer issued NOTIFY to Consumer.
Produced: 6 | Buffer size: 3
Producer issued NOTIFY to Consumer.
Consumed: 4 | Buffer size: 2
Consumer issued NOTIFY to Producer.
Produced: 7 | Buffer size: 3
Producer issued NOTIFY to Consumer.
Consumed: 5 | Buffer size: 2
Consumer issued NOTIFY to Producer.
Produced: 8 | Buffer size: 3
Producer issued NOTIFY to Consumer.
Produced: 9 | Buffer size: 4
Producer issued NOTIFY to Consumer.
Consumed: 6 | Buffer size: 3
Consumer issued NOTIFY to Producer.
Produced: 10 | Buffer size: 4
Producer issued NOTIFY to Consumer.
Producer thread STOPPED.
Consumed: 7 | Buffer size: 3
Consumer issued NOTIFY to Producer.
Consumed: 8 | Buffer size: 2
Consumer issued NOTIFY to Producer.
Consumed: 10 | Buffer size: 0
Consumer issued NOTIFY to Producer.
Consumer thread STOPPED.
Main: Program Ended.