UUM

Universiti Utara Malaysia

UNIVERSITI UTARA MALAYSIA

SECOND SEMESTER SESSION A242

STIWK 3014 REAL TIME PROGRAMMING

(GROUP A)

Week 10 Class Activity

Lecturer: Dr. Ruzita binti Ahmad

Name: Chong Mun Kei

Matric No: 298767

Github Link: https://github.com/Chong0508/RealTime.git

VolatileFlagExample.java

```
public class VolatileFlagExample {

   // Shared flag between threads
   private static volatile boolean running = true;

   public static void main(String[] args) {
       // Thread that runs continuously until 'running' becomes
false
       Thread worker = new Thread(() -> {
           System.out.println("Worker thread started...");
           while(running) {
               // simulate work
           }
           System.out.println("Worker thread stopped...");
       });

       worker.start();

       // Main thread sleeps for a bit then signals stop
       try {
           Thread.sleep(2000); // Let the worker run for 2
seconds
       } catch (InterruptedException e) {
           Thread.currentThread().interrupt();
       }
```

```
    }
}
```

ProducerConsumerDemo.java

```
package Chapter8_Concurrency_Collection;

public class ProducerConsumerDemo {

    static class SharedData {
        private boolean dataReady = false;
        private String data;

        // Producer thread
        public synchronized void produce() {
            try {
                System.out.println("Producer: Preparing
data...");
                Thread.sleep(1000);
                data = "Hello from producer";
                dataReady = true;
                System.out.println("Producer: Data is ready.");
                notify(); // Notify waiting consumer
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
```

```
        // Consumer thread
        public synchronized void consume() {
            try {
                while(!dataReady) {
                    System.out.println("Consumer: Waiting for
data...");
                    wait(); // Release lock and wait to be
notified
                }
                System.out.println("Consumer: Received -> " +
data);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }

    public static void main(String[] args) {
        SharedData sharedData = new SharedData();

        // Create consumer thread
        Thread consumerThread = new Thread(() ->
sharedData.consume());

        // Create producer thread
        Thread producerThread = new Thread(() ->
sharedData.produce());
```

```
        // Start consumer thread first
        consumerThread.start();

        // Delay starting the producer thread
        try {
            Thread.sleep(500); // Let the consumer start and wait
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }

        producerThread.start();
    }
}
```

1. Compare the memory consistency by using volatile() & synchronized()

Feature	**volatile()**	**synchronized()**
Visibility	Ensure visibility of changes to variables across threads	Ensures visibility of changes as it flushes changes to and from main memory
Atomicity	Does not guarantee atomicity	Guarantees atomicity
Locking	No locking	Implicit lock used
Performance	Faster	Slower
Use Case	Best for simple flags or status updates	Best for complex operations on shared data

2. Modify the ProducerConsumerDemo by using notifyAll() and compare the memory consistency

```
package Chapter8_Concurrency_Collection;


public class ProducerConsumerDemo_notifyAll {
   static class SharedData {
       private boolean dataReady = false;
       private String data;


       public synchronized void produce() {
           try {
               System.out.println("Producer: Preparing
data...");
               Thread.sleep(1000);
               data = "Hello from producer";
               dataReady = true;
               System.out.println("Producer:  Data is
ready.");
               notifyAll();
           } catch (InterruptedException e) {
               Thread.currentThread().interrupt();
           }
       }


       public synchronized void consume() {
           try {
               while(!dataReady) {
```

```
                    System.out.println("Consumer: Waiting
for data...");
                    wait();
                }
                System.out.println("Consumer: Received -> "
+ data);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }

    public static void main(String[] args) {
        SharedData sharedData = new SharedData();

        Thread consumerThread1 = new
Thread(sharedData::consume);
        Thread consumerThread2 = new
Thread(sharedData::consume);

        consumerThread1.start();
        consumerThread2.start();

        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
```

```
        Thread producerThread = new
Thread(sharedData::produce);

        producerThread.start();

    }

}
```

Feature	notify()	notifyAll()
Notifies	Wakes up one waiting thread	Wakes up all waiting threads
Fairness	Lead to missed notifications	Reduces missed notifications
Use Case	When only one consumer should proceed	When multiple consumers may need to act
Memory Consistency	Same	Same