NUR AZYYATI BINTI ABU BAKAR | 291560

TUTORIAL12: SLIDE 6 AND 8 (CONCURRENCY COLLECTION)

CODE:

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

VolatileFlagExample.java (volatile)

```java
public class VolatileFlagExample {

    //shared flag between threads
    private static volatile boolean running = true;  1 usage

    public static void main(String[] args) {
        //tread 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( millis: 2000);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

THE OUTPUT (volatile):

```
"C:\Program Files\Java\jdk-16.0.1
Worker thread started...
```

VolatileFlagExample.java (synchronized):

```java
public class VolatileFlagExample {

    private static boolean running = true;  2 usages

    //synchronized setter
    public static synchronized void setRunning(boolean value) {  1 usage
        running = value;
    }

    //synchronized getter
    public static synchronized boolean isRunning() {  1 usage
        return running;
    }

    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 (isRunning()) {
                //simulate work
            }
            System.out.println("Worker thread stopped...");
        });

        worker.start();

        //main thread sleeps for a bit then signals stop
        try{
            Thread.sleep( millis: 2000);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }

        setRunning(false); //update the flag
    }
}
```

THE OUTPUT (synchronized):

```
"C:\Program Files\Java\jdk-16.0.1\bin\java.
Worker thread started...
Worker thread stopped...

Process finished with exit code 0
```

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

ProducerConsumerDemo.java (notify)

```java
public class ProducerConsumerDemo {

    static class SharedData {  2 usages
        private boolean dataReady = false;  2 usages
        private String data;  2 usages

        // Producer method
        public synchronized void produce() {  1 usage
            try {
                System.out.println("Producer: Preparing data...");
                Thread.sleep( millis: 1000); // Simulate time to produce data
                data = "Hello from producer!";
                dataReady = true;
                System.out.println("Producer: Data is ready.");
                notify(); // Notify waiting consumer
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }

        // Consumer method
        public synchronized void consume() {  1 usage
            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();

        Thread consumerThread = new Thread(() -> sharedData.consume());
        Thread producerThread = new Thread(() -> sharedData.produce());
        consumerThread.start();
        producerThread.start();

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

THE OUTPUT (notify):

```
"C:\Program Files\Java\jdk-16.0.1\bin\java.ex
Consumer: Waiting for data...
Consumer: Waiting for data...
Producer: Preparing data...
Producer: Data is ready.
Consumer: Received --> Hello from producer!
```

ProducerConsumerDemo.java (notifyAll)

```java
public class ProducerConsumerDemo {

    static class SharedData {  2 usages
        private boolean dataReady = false;  2 usages
        private String data;  2 usages

        // Producer method
        public synchronized void produce() {  1 usage
            try {
                System.out.println("Producer: Preparing data...");
                Thread.sleep( millis: 1000); // Simulate time to produce data
                data = "Hello from producer!";
                dataReady = true;
                System.out.println("Producer: Data is ready.");
                notifyAll(); // Notify waiting consumer
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }

        // Consumer method
        public synchronized void consume() {  2 usages
            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();

        Thread consumerThread1 = new Thread(() -> sharedData.consume());
        Thread consumerThread2 = new Thread(() -> sharedData.consume());

        Thread producerThread = new Thread(() -> sharedData.produce());
        consumerThread1.start();
        consumerThread2.start();
        producerThread.start();

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

THE OUTPUT (notifyAll):