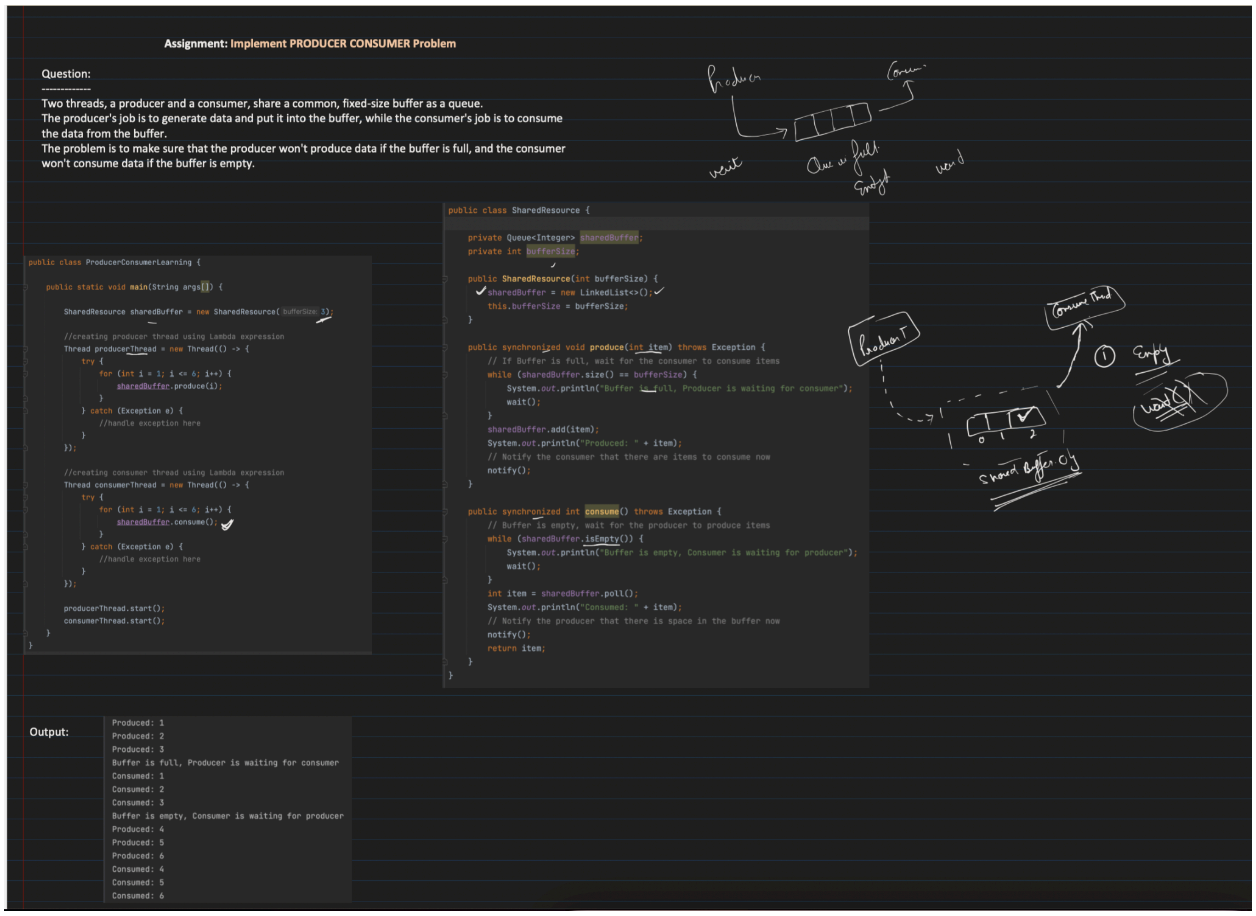
Java: Thread Joining, Daemon



Assignment: Implement PRODUCER CONSUMER Problem

Question:

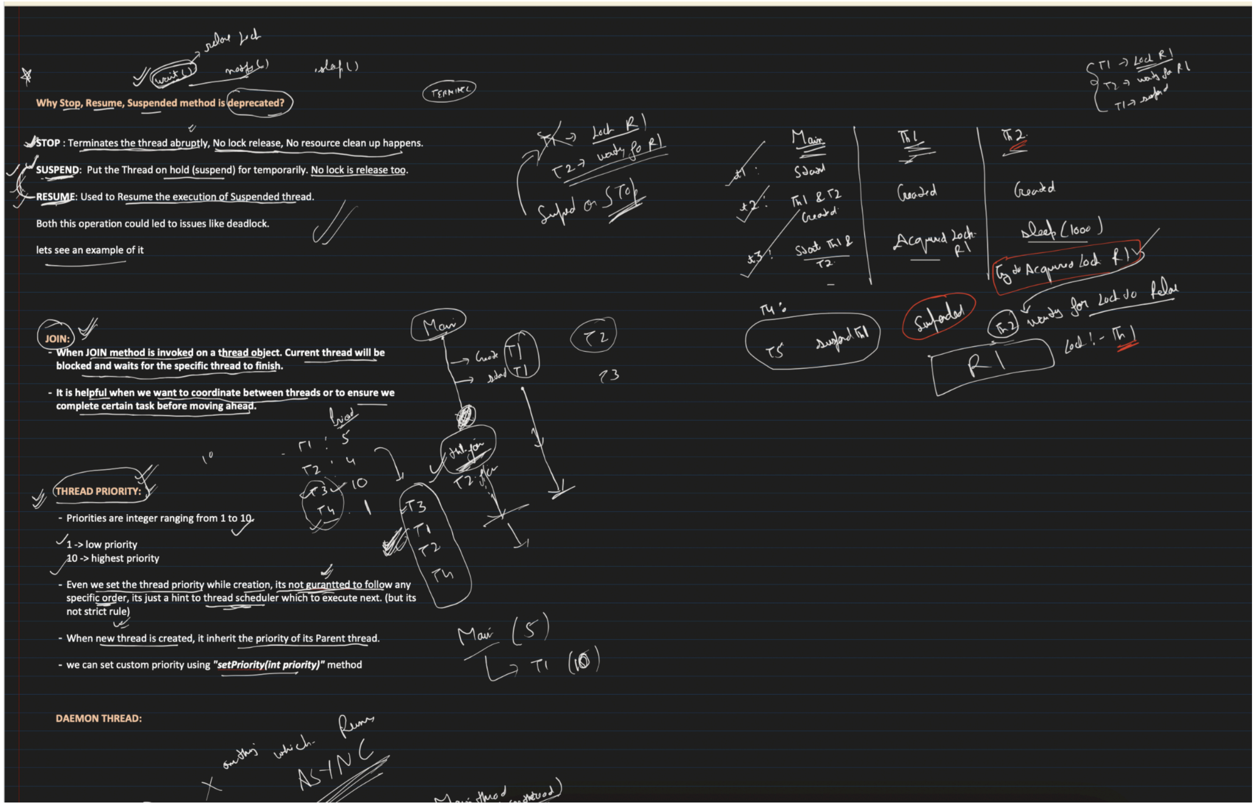
Two threads, a producer and a consumer, share a common, fixed-size buffer as a queue.
The producer's job is to generate data and put it into the buffer, while the consumer's job is to consume the data from the buffer.
The problem is to make sure that the producer won't produce data if the buffer is full, and the consumer won't consume data if the buffer is empty.

```java
public class ProducerConsumerLearning {

    public static void main(String args[]) {

        SharedResource sharedBuffer = new SharedResource( bufferSize: 3);

        //creating producer thread using Lambda expression
        Thread producerThread = new Thread(() -> {
            try {
                for (int i = 1; i <= 6; i++) {
                    sharedBuffer.produce(i);
                }
            } catch (Exception e) {
                //handle exception here
            }
        });

        //creating consumer thread using Lambda expression
        Thread consumerThread = new Thread(() -> {
            try {
                for (int i = 1; i <= 6; i++) {
                    sharedBuffer.consume();
                }
            } catch (Exception e) {
                //handle exception here
            }
        });

        producerThread.start();
        consumerThread.start();
    }
}
```

```java
public class SharedResource {

    private Queue<Integer> sharedBuffer;
    private int bufferSize;

    public SharedResource(int bufferSize) {
        sharedBuffer = new LinkedList<>();
        this.bufferSize = bufferSize;
    }

    public synchronized void produce(int item) throws Exception {
        // If Buffer is full, wait for the consumer to consume items
        while (sharedBuffer.size() == bufferSize) {
            System.out.println("Buffer is full, Producer is waiting for consumer");
            wait();
        }
        sharedBuffer.add(item);
        System.out.println("Produced: " + item);
        // Notify the consumer that there are items to consume now
        notify();
    }

    public synchronized int consume() throws Exception {
        // Buffer is empty, wait for the producer to produce items
        while (sharedBuffer.isEmpty()) {
            System.out.println("Buffer is empty, Consumer is waiting for producer");
            wait();
        }
        int item = sharedBuffer.poll();
        System.out.println("Consumed: " + item);
        // Notify the producer that there is space in the buffer now
        notify();
        return item;
    }
}
```

Output:

```
Produced: 1
Produced: 2
Produced: 3
Buffer is full, Producer is waiting for consumer
Consumed: 1
Consumed: 2
Consumed: 3
Buffer is empty, Consumer is waiting for producer
Produced: 4
Produced: 5
Produced: 6
Consumed: 4
Consumed: 5
Consumed: 6
```



Why Stop, Resume, Suspended method is deprecated?

STOP : Terminates the thread abruptly, No lock release, No resource clean up happens.

SUSPEND: Put the Thread on hold (suspend) for temporarily. No lock is release too.

RESUME: Used to Resume the execution of Suspended thread.

Both this operation could led to issues like deadlock.

lets see an example of it

JOIN:

- **When JOIN method is invoked on a thread object. Current thread will be blocked and waits for the specific thread to finish.**
- **It is helpful when we want to coordinate between threads or to ensure we complete certain task before moving ahead.**

THREAD PRIORITY:

- Priorities are integer ranging from 1 to 10.

  1 -> low priority
  10 -> highest priority

- Even we set the thread priority while creation, its not gurantted to follow any specific order, its just a hint to thread scheduler which to execute next. (but its not strict rule)
- When new thread is created, it inherit the priority of its Parent thread.
- we can set custom priority using ***"setPriority(int priority)"*** method

DAEMON THREAD: