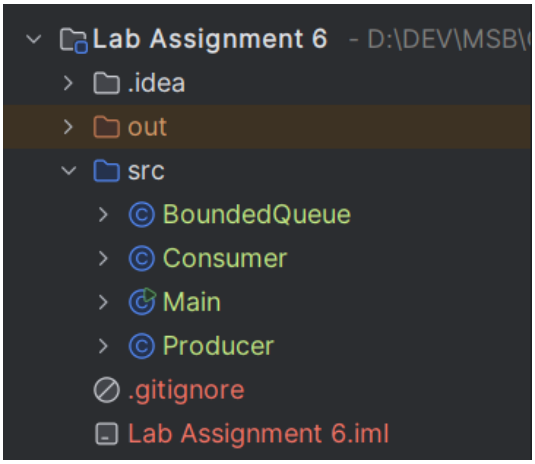
Lab Assignment 6

Name: Manobal Singh Bagady
SID: 21104129
Semester: 8
Branch: Electrical Engineering
Due Date: Mar 31, 2025

File Structure:



Output:

```
"C:\Program Files\Java\jdk-21\bin\java.exe"
Enter the size of the queue: 5
Produced: 16
Consumed: 16
Produced: 35
Consumed: 35
Produced: 67
Produced: 83
Consumed: 67
Produced: 94
Produced: 49
Consumed: 83
Produced: 37
Consumed: 94
Produced: 42
Produced: 29
Consumed: 49
Produced: 32
Consumed: 37
Consumed: 42
Consumed: 29
Consumed: 32
Producer and Consumer have finished
```

BoundedQueue.java :

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

public class BoundedQueue {
    private final Queue<Integer> queue;
    private final int capacity;

    public BoundedQueue(int capacity) {
        this.queue = new LinkedList<>();
        this.capacity = capacity;
    }

    public synchronized void enqueue(int value) throws
InterruptedException {
        // while queue is full, producer must wait
        while (queue.size() == capacity) wait();

        // add new item
        queue.offer(value);

        // notify waiting threads that there is data available
        notifyAll();
    }

    public synchronized int dequeue() throws InterruptedException
{
        // while queue is empty, consumer must wait
        while (queue.isEmpty()) wait();

        // remove an item
        int value = queue.poll();

        // notify waiting threads that there is space available
        notifyAll();

        return value;
    }
}
```

Features of BoundedQueue Class:

- Maintains an internal `Queue<Integer>` and a fixed `capacity`.

- `enqueue(int value)`
  - If the queue is full (`size == capacity`), the calling thread waits.
  - Otherwise, it adds the new value, prints a message, and calls `notifyAll()` to signal that an item is available.

- `dequeue()`
  - If the queue is empty (`size == 0`), the calling thread waits.
  - Otherwise, it removes the item, prints a message, and calls `notifyAll()` to signal that space is available.

Producer.java :

```
import java.util.Random;

public class Producer implements Runnable {
    private final BoundedQueue queue;
    private final Random random = new Random();

    private final int numberOfItemsToProduce;

    public Producer(BoundedQueue queue, int
numberOfItemsToProduce) {
        this.queue = queue;
        this.numberOfItemsToProduce = numberOfItemsToProduce;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < numberOfItemsToProduce; i++) {
                int value = random.nextInt(100);
                System.out.println("Produced: " + value);
                queue.enqueue(value);

                Thread.sleep(random.nextInt(1000));
            }
```

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

Producer

- Continuously (or for a certain number of items) generates random numbers and calls `queue.enqueue(...)`.

Consumer.java :

```
import java.util.Random;

public class Consumer implements Runnable {
    private final BoundedQueue queue;
    private final int numberOfItemsToConsume;
    private final Random random = new Random();

    public Consumer(BoundedQueue queue, int
numberOfItemsToConsume) {
        this.queue = queue;
        this.numberOfItemsToConsume = numberOfItemsToConsume;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < numberOfItemsToConsume; i++) {
                int value = queue.dequeue();
                System.out.println("Consumed: " + value);

                Thread.sleep(random.nextInt(1000));
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
```

```
    }
}
```

Consumer

- Continuously (or for a certain number of items) calls `queue.dequeue()` and prints (or processes) the consumed item.

Main.java :

```
import java.util.Scanner;

public class Main {
    static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        // read queue size
        System.out.print("Enter the size of the queue: ");
        int queueSize = scanner.nextInt();

        // create a bounded queue
        BoundedQueue sharedQueue = new BoundedQueue(queueSize);

        // create a producer and a consumer
        Producer producer = new Producer(sharedQueue, queueSize *
2);
        Consumer consumer = new Consumer(sharedQueue, queueSize *
2);

        // create and start producer and consumer threads
        Thread producerThread = new Thread(producer,
"ProducerThread");
        Thread consumerThread = new Thread(consumer,
"ConsumerThread");

        producerThread.start();
        consumerThread.start();

        // wait for the threads to finish
        try {
```

```
            producerThread.join();
            consumerThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Producer and Consumer have finished");

    }
}
```

Main Class

- Creates a `BoundedQueue` of a given size.
- Creates a Producer and Consumer with the shared `BoundedQueue`.
- Starts both threads, and optionally waits for them to finish using `join()`.