# PROJECT INSTRUCTIONS

# Producer-Consumer Message Processing System in Core Java

This project is a Producer-Consumer simulation in core Java, designed to demonstrate thread-safe interactions between multiple threads sharing a common resource (message queue). The goal is to implement a basic, yet effective, message-driven application that logs processed messages and handles error tracking.

## Project Structure

The following structure represents the organization of files in the project:

ProducerConsumerProject/  
├── src/  
 └── main/  
 └── java/  
 └── ProducerConsumer/  
 ├── Main.java  
 ├── MessageQueue.java  
 ├── Producer.java  
 ├── Consumer.java  
 └── Logger.java

## Instructions to Run the Application and Tests

1. Import the Project:

- Open your preferred IDE (e.g., Eclipse, IntelliJ IDEA, STS) and import the project by selecting File > Import > Existing Projects. Choose the folder where the project files are located and finish the import process.

2. Run the Application:

- Open the Main.java file, right-click on it, and select Run As > Java Application. This will start both the Producer and Consumer threads. You should see the console display messages indicating messages being produced and consumed.

3. Monitor the Output:

- Watch the console for messages like 'Produced: Message 1' and 'Consumed: Message 1', indicating that the Producer and Consumer are functioning as expected.

4. Run Unit Tests :

- To verify functionality, you can run ProducerConsumerTest.java by right-clicking on the file and selecting Run As > JUnit Test. This will simulate the Producer and Consumer running for a short period, validating that the queue is functioning correctly.

## Explanation of Key Concepts

1. Thread Safety and Synchronization:

- The MessageQueue class uses synchronized methods and wait/notifyAll mechanisms to manage access to the queue, ensuring that only one thread interacts with the queue at a time. This prevents issues like race conditions and data inconsistency.

2. Concurrency Management:

- The Producer and Consumer run on separate threads, allowing them to function independently. When the queue is full, the Producer waits; when the queue is empty, the Consumer waits. This approach effectively manages the concurrent processing of messages.

3. Logging Mechanism:

- The Logger class tracks the number of successful and failed message processes, providing a summary at the end of execution. This log helps understand system performance and identify potential issues.

4. Graceful Shutdown:

- The threads are set up to be interrupted after the test or upon application termination, ensuring that resources are released properly and the application exits smoothly without leaving threads hanging.