# Producer-Consumer Pattern Lab Report

**Amo Samuel**

## Introduction

In this document, I'll explain the producer-consumer problem, a classic synchronization problem in concurrent programming. I'll also discuss its solutions, focusing on the implementation I've created using Java.

### *What is the Producer-Consumer Problem*

The producer-consumer problem involves two types of processes:

- **Producers:** Create items and add them to a shared buffer
- **Consumers:** Remove items from the shared buffer and process them

### *The main challenge is to ensure that:*

- Producers don't add items when the buffer is full
- Consumers don't remove items when the buffer is empty
- Multiple producers or consumers don't interfere with each other

### *My Solution: The Sleeping Barber Problem*

I've implemented a variation of the producer-consumer problem called the "Sleeping Barber Problem." In this scenario:

- The barber represents the consumer
- The customers represent the producers
- The barbershop chairs represent the shared buffer

Key Components of My Implementation

- BarberShop: Manages the shared resources and synchronization
- Barber: Represents the consumer (service provider)
- Customer: Represents the producer (service requester)
- Main: Orchestrates the simulation

***Synchronization Mechanisms Used***

I've used Java's Semaphore class to handle synchronization:

- waitingSeats: Limits the number of customers in the waiting area
- customers: Signals the barber when a customer is waiting
- barberReady: Signals a customer when the barber is ready

***How My Solution Works***

- The barbershop initializes with a fixed number of chairs.
- Customers arrive and try to occupy a waiting seat.
- If a seat is available, the customer waits for the barber.
- If no seat is available, the customer leaves.
- The barber continuously checks for waiting customers.
- When a customer is present, the barber provides the haircut service.

***Advantages of This Approach***

- Prevents deadlocks and race conditions
- Efficiently manages shared resources
- Simulates real-world scenarios effectively

**Conclusion**

The producer-consumer problem is crucial in concurrent programming. My implementation demonstrates how to solve this problem using semaphores in Java, providing a practical example through the Sleeping Barber Problem.