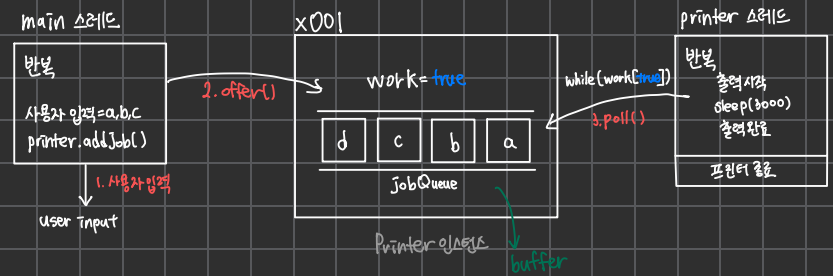
생산자 소비자 문제 - 멀티스레드 프로그래밍에서 등장하는 동시성 문제

= 한정된 버퍼 문제



- 생산자 (producer) : 데이터 생성 역할
  ex) 파일에서 데이터를 읽거나 네트워크에서 데이터를 받아오는 스레드
- 소비자 (consumer) : 생성된 데이터를 사용하는 역할
  ex) 데이터처리/저장 스레드
- 버퍼 (buffer) : 생산자가 생성한 데이터를 일시적으로 저장하는 공간
  한정된 크기, 생산자와 소비자는 버퍼를 통해 데이터를 주고 받음

문제 상황

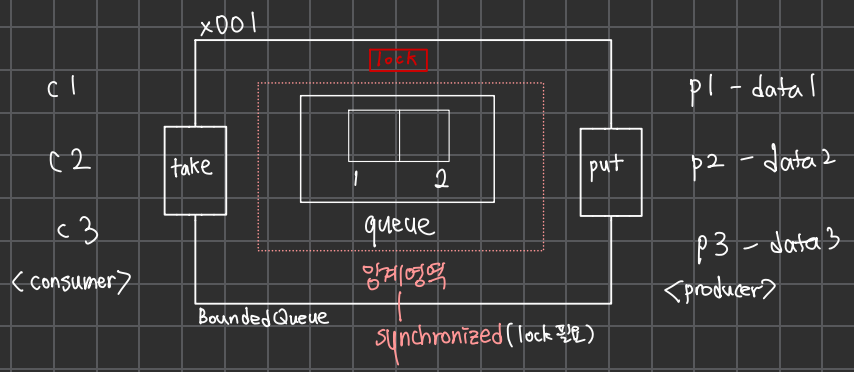
- 생산자가 너무 빠를 때 : 버퍼가 가득차서 더 이상 데이터를 넣을 수 없을때까지 데이터생성
  생산자는 버퍼가 비워질 때까지 대기
- 소비자가 너무빠를때 : 버퍼가 비어서 더 이상 소모할 데이터가 없을 때까지 소비자가 데이터 처리
  소비자는 버퍼에 데이터가 들어올때까지대기

생산자 소비자 문제 - 예제 1

- BoundedQueue : 버퍼 큐 인터페이스
  - void put(String data) : 버퍼에 데이터 보관 → 생산자 스레드에서 호출, 데이터생산
  - String take() : 버퍼에 보관된 값을 꺼내감 → 소비자 스레드에서 호출, 데이터 소비
- BoundedQueueV1 구현체
  - ArrayQueue : 버퍼 큐 ← 핵심공유자원
  - int max : 버퍼에 저장할 수 있는 최대크기
  - put(), take() ← synchronized

생산자먼저 (producerFirst)
- 생산자 스레드 3개 실행 → queue = [data1, data2] ※data3
- 소비자 스레드 3개 실행 → data1, data2, null 데이터가없으므로

소비자 먼저 (consumer)
- 소비자스레드 3개 실행 → null, null, null
- 생성자 스레드 3개 실행 → queue = [data1, data2] ※data3



producer 언러

- p1은 lock을 획득하여 put(data1) → lock 반납후 TERMINATED
- p2도 이어서 진행
- p3는 lock을 획득하여 put(data3)시도, 이미 꽉찼으로 waiting → lock반납후 TERMINATED

> 데이터를 버리지 않는 대안 : 큐에 빈공간이 생길 때까지 p3 스레드가 기다리는것
- 단순한 방법 : 생산자 스레드가 polling 방식으로 큐의 빈공간을 체크하는 법
빈공간이 없다면 sleep()을 짧게 사용해 대기후 깨어나서 다시 체크

> null 값을 받지 않는 대안 : 큐에 데이터가 생길 때까지 c3 스레드가 기다리는것
- 단순한방법 : 소비자 스레드가 polling 방식으로 큐에 데이터가 존재하는지 체크

But, data3가 앞서 버려졌기 때문에 c3가 데이터를 조회하는 시점에는 큐가 비어있어 null 값을 받게됨
⇒ 스레드가 대기할 수 있었다면 p3가 생산한 data3를 c3가 받을 수 있었을것.

Consumer 먼저

- C1은 lock을 얻어 큐에 데이터를 소비 시도 → null → lock 반환 → TERMINATED
- C2, C3도 이어서 모두 데이터를 받지 못하고 종료.
- P1은 lock을 획득하여 큐에 data1 추가 → lock반납 → TERMINATED
- P2도 마찬가지
- P3는 큐가 가득차 데이터를 삽입하지 못하고 종료

<문제점>

- producerFirst의 경우 : data3 우선, C3는 데이터를 받지못함
- consumerFirst의 경우 : C1, C2, C3 모두 null, P3의 data3 우선

- 버퍼가 가득찬 경우 : 생산자 입장에서 버퍼에 여유가 생길때까지 기다리면 되지만 기다리지 못하고 데이터를 버리게 됨
- 버퍼가 빈 경우 : 소비자 입장에서 버퍼에 데이터가 채워질때까지 기다리면 되는데 기다리지 못하고 null 값을 얻게 됨

↳ 해결 방안 : 스레드가 기다리면 됨!

put(data) - 큐가 가득 찼을 때 빈공간이 생길때까지 생산자 스레드 대기

```
while (queue.size() == max) { ... sleep(1000); } return queue.offer(data);
```

take() - 큐에 데이터가 추가될 때까지 소비자 스레드 대기

```
while (queue.isEmpty()) { ... sleep(1000); } return queue.poll();
```

하지만, 문제 발생

- 생산자 스레드 먼저 실행한 경우
  P1, P2 → 정상 종료, P3 → TIMED_WAITING
  C1, C2, C3 → BLOCKED

  P1, P2 : queue에 데이터 put → [data1, data2]
  P3 : lock 획득 → 큐저장실패 → 잠시 대기 (running → timed_waiting)
  ↳ lock을 가진 상태에서 큐에 반환하고 sleep 될때까지 대기
  C1, C2, C3 : lock이 없으므로 blocked (절대 반납할리가 없는 P3가 lock 반납할 때까지)

- 소비자 스레드 먼저 실행한 경우
  C1 → TIMED_WAITING
  C2, C3 → BLOCKED
  P1, P2, P3 → BLOCKED

  C1 : (조건이 만족되지 않아 lock 반납불가) lock 획득 → 큐에 데이터 X → 잠시 대기 (running → timed_waiting)
  C2, C3 : lock 획득불가 → C1이 락을 반납할때까지 blocked 상태로 무한 대기
  P1, P2, P3 : lock 획득불가 → C1이 락을 반납하려면 생산자 스레드에서 데이터를 추가해야함, 하지만 P1이 lock을 획득할 수없어 무한 대기하게 됨

락을 갖고 있는 스레드가 sleep()으로 감시 대기할때에는 아무일도 하지않음

⇒ 그동안에는 다른 스레드에게 락을 양보하자

그렇게 되면 락을 얻게된 생산자 스레드는 버퍼에 데이터를 채우고 락을 반납

그 뒤에 소비자 스레드에서 다시 락을 획득하고 버퍼에 있는 값을 가져가고 락을 반납

Object.wait(), Object.notify()로 락을 갖고 대기중인 스레드가 대기 동안 다른 스레드에게 락 양보 가능

Object의 wait, notify

- wait()
  - 현재 스레드가 가진 락을 반납하고 대기 (waiting)
  - 현재 스레드를 대기 상태로 전환 (현재 스레드가 synchronized 블록/메서드에서 락을 소유한 경우만 호출)
    호출한 스레드는 락을 반납하고 다른 스레드가 락을 획득할 수 있도록 함
    이렇게 대기 상태로 전환된 스레드는 다른 스레드가 notify() or notifyAll() 호출할 때까지 대기 유지

- notify()
  - 대기 중인 스레드 하나를 깨운다.
  - 깨운 스레드는 락을 다시 획득할 기회를 얻음

- notifyAll()
  - 대기중인 모든 스레드를 깨움
  - 모든 대기중인 스레드가 락을 획득할 수 있는 기회를 얻음

synchronized 블록/메서드 에서 호출되어야 함

```
put(data)
  while(queue.size()==max){
    try{
      wait(); → 락반납후 대기
    } catch(InterruptedException){
      throw new RuntimeException();
    }
  }
  queue.offer(data);
  notify(); → 저장된 데이터가 있으니 가져가라고 알림
```

```
take()
  while(queue.isEmpty()){
    try{
      wait(); → 락을 반납하고 대기
    } catch(InterruptedException){
      throw ...
    }
  }
  String data = queue.poll();
  notify(); → 큐에 여유공간이 생겨 대기중인 스레드를 깨워
  return data; 데이터를 생산하라고 알림
```

- wait(), notify()를 적용한 결과

- 생산자 먼저 실행

P1, P2 → [data1, data2]

P3 → 큐가 가득차 대기 (lock은 양보) wait()

P1, P2는 terminated, P3는 waiting

C1 → data1 소비 → notify() 호출

P3 데이터저장 queue=[data2, data3]

C2, C3 → data2, data3 소비

- 소비자 먼저 실행

C1, C2, C3 → 소비할 데이터가 없어 대기, 모두 waiting

P1 → 데이터추가 queue=[data1] → notify() 호출

C1 → data1 소비 → notify() 호출

대기중인 스레드가 여러개 이므로 어떤 소비자 스레드가 깨워질지는 모름

C2가 깨어남 (대기집합에는 소비자 스레드만 존재함)

→ 큐에 데이터가 없어 wait() 호출후 다시 대기 집합으로 추가됨

P2 → data2 저장 → notify() 호출

C3 → 신호를 받고 깨어남, 락획득을 위해 임계영역 안에서 Blocked 상태 → 락획득후 runnable 상태 → notify() 호출

대기집합에는 소비자 스레드 C2 밖에 없기에 C2가 깨어남

C2는 Blocked

C3 → data2 소비 → 락반납

C2 → 락 획득 시도, 데이터가 없어 다시 대기

P3 → data3 생산 → notify() 호출

C2 → data3 소비

소비자 스레드가 생산자, 소비자 스레드를 선택해 깨울수 있다면 같은 소비자인 C2를 깨워 비효율적인 진행을 막을 수 있을 것 하지만, notify()로는 불가능

* 모든 객체는 monitor lock과 wait set을 가짐

대기 상태의 스레드 관리

대기 집합에 있는 스레드가 notify() 신호를 받고바로 작동하지 않음

깨어난 스레드는 "임계영역" 안에 있다.

즉, P3는 대기 집합에선 제외되나 임계영역안에 있어 락을 획득하기 위해 blocked 상태로 대기 (waiting → blocked)

획득 했다면 wait() 이후의 코드를 이어서 수행함

wait와 notify의 한계

· notify는 임의의 스레드를 깨움

⇒ 같은 종류의 스레드를 깨울 때 비효율 발생

Thread Starvation 발생문제 - 대기상태의 스레드가 실행순서를 계속 얻지못해 실행되지 않는 상황

⇒ 최악의 경우, 스레드 대기집합 = [C1, C2, C3, C4, C5, P1]일 때

C1 ~ C5만 계속 반복해서 깨어날 수 있음

해결방법

notifyAll() - 대기 집합의 모든 스레드를 다깨움

C1 ~ C5, P1는 모두 깨어남 → 락 획득을 해야함

락을 획득한 하나의 스레드를 제외, BLOCKED 상태가됨 → 대기집합행

기아 상태는 막을 순 있어도 여전히 비효율적