쓰레드와 쓰레드 안전성

프로세스 vs 쓰레드

프로세스

일반적으로 cpu에 의해 메모리에 올려져 실행중인 프로그램을 말하며, 자신만의 메모리 공간을 포함한 독립적인 실행 환경을 가지고 있다

쓰레드

프로세스안에서 실질적으로 작업을 실행하는 단위를 말하며, 자바에서는 JVM(Java Virtual Machine)에 의해 관리된다

프로세스에는 적어도 한개 이상의 스레드가 있으며, Main 스레드 하나로 시작하여 스레드를 추가 생성하게 되면 멀티 스레드 환경이 된다

이러한 스레드들은 프로세스의 리소스를 공유하기 때문에 효율적이긴 하지만 잠재적인 문제점에 노출될 수도 있다

동시성(concurrency) 이슈

여러 스레드가 동시에 하나의 자원을 공유하고 있기 때문에 같은 자원을 두고 **경쟁상태(raceCondition)** 같은 문제가 발생하는 것

스레드 안전성(Thread safe)

여러 스레드가 작동하는 환경에서도 문제 없이 동작하는 것을 스레드 안전하다고 말한다

즉, 동시성 이슈를 해결하고 일어나지 않는다면 `Thread safe` 하다

예시

카운트를 갖는 Count클래스

```
public class Count{
    private int count = 0;

    public void increase() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

다음 코드는 thread-safe 하지 않다

스레드A와 스레드B가 동시에 increase()메서드를 호출하여 count를 증가시켰을 때, A의 increase()로 count는 1이 되고, B의 increase()로 count는 2가 되는 것이 정상적인 동작

하지만 A와 B가 동시에 increase()메서드를 호출하게 된다면 count는 2가 아닌 1이 되는 문제가 발생한다

count를 증가시키는 로직이 **count변수 값 조회, count변수에 1을 더하고 저장** 두가지의 동작으로 이루어지기 때문

어떻게 해결하면 좋을까??

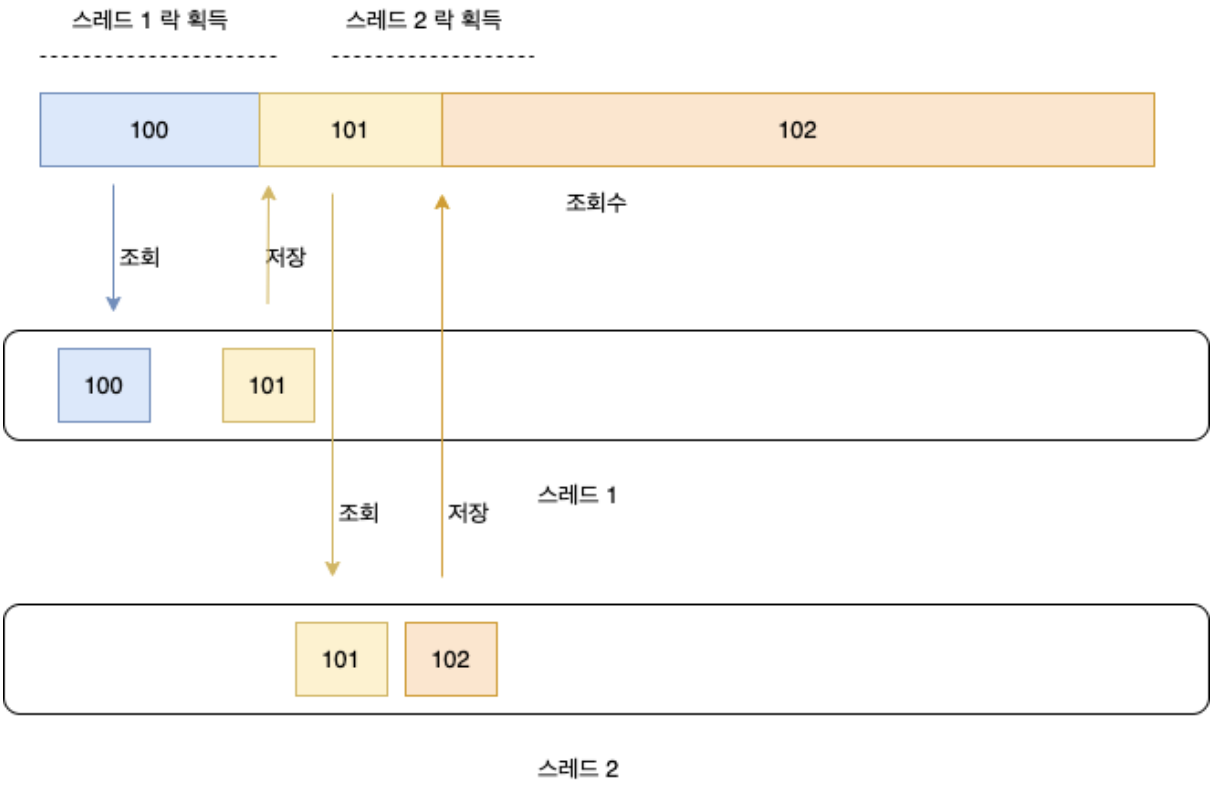
1. 암시적 Lock 사용

가장 간단하면서 쉬운 방법

Lock을 걸게 되면 스레드의 공유자원에 대한 독점을 허용하기 때문에 다른 스레드가 자원에 접근하지 못하고 대기하게 된다. 이는 여러 스레드가 공유자원에 동시에 접근하지 못하므로 동시성 문제를 막을 수 있지만, 하나의 스레드만 자원을 사용하므로 병렬성은 낮아지게 된다.

자바에서는 암시적 Lock은 같이 메서드에 synchronized키워드를 사용하여 구현할 수 있다. (변수도 가능하지만, 객체만 가능하다)

```
public class Count {
    private int  count = 0;

    public synchronized void increase() { //synchronized 추가
        count++;
    }

    public synchronized int getCount() { //synchronized 추가
        return count;
    }
}
```



2. 명시적 Lock 사용

명시적 Lock은 synchronized키워드를 사용하는 암시적 Lock과는 달리 Lock 인터페이스를 이용한다. Lock객체의 lock()메서드를 호출하여 잠그고, unlock()메서드를 호출하여 잠금을 해제한다.

```
public class Count {
    private int  count = 0;
    private Lock lock = new ReentrantLock();

    public void increase() {
        lock.lock(); //잠금
        try {
            count++;
        } finally {
            lock.unlock(); //잠금 해제
        }
```

```
    }

    public int getCount() {
        lock.lock(); //잠금
        try{
            return count;
        }finally{
            lock.unlock(); //잠금 해제
        }
    }
}
```

3. volatile을 사용

volatile 키워드를 사용하면 변수가 항상 메인 메모리에서 읽고 쓰이도록 보장된다.

이렇게 하면 변수에 대한 변경 사항이 다른 스레드에 즉시 반영되므로, 스레드 간의 동시성 문제를 해결할 수 있다.

```
public class Count {
    private volatile int count; //volatile 추가

    public void increase() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

volatile 키워드를 변수에 붙여주면, 해당 변수는 캐시에 저장되지 않고 메인 메모리에 항상 저장이 되는데, 바로 이 점을 이용해서 캐시 사용으로 인한 데이터 불일치를 막을 수 있다.

4. Concurrent 패키지 사용

자바에서 제공하는 Concurrent패키지는 동시성 문제를 해결하기 위한 다양한 클래스와 인터페이스를 제공한다.

```
public class Count {
    private ConcurrentHashMap<String, Integer> map = new Conc

        public void increase() {
            Integer currentValue = map.get("count");
            if (currentValue == null) {//"count"라는 키가 Map에
                map.put("count", 1);
            } else { //존재하면
                map.put("count", currentValue + 1);
            }
        }

    public int getCount() {
        return map.get("count");
    }
}
```

위와 같이 ConcurrentHashMap을 사용하면 내부적으로 스레드 간의 안전한 데이터 공유를 보장하기 때문에 동시성 문제가 발생하지 않는다.

5. 불변 객체 사용

불변 객체는 상태가 변경될 수 없는 객체이기 때문에 Thread-safe하다.

아래 코드에서 보이는 것처럼 final 키워드로 클래스와 변수를 선언하여 변경할 수 없도록 만들었기 때문에 count 변수는 반드시 생성자를 통해서 초기화한다. increase()에서는 Count 객체를 생성하면서, count를 증가시킨다.

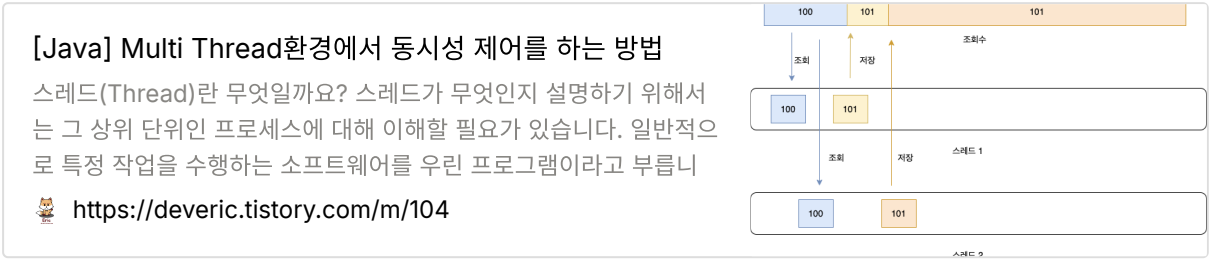
```
public final class Count {
    private final int count;

    public Count(int count) {
        this.count = count;
    }
    public Count increase(){
```

```
        return new Count(count + 1); //새로운 Count 객체 생성
    }
    public int getCount(){
        return count;
    }
}
```

이렇게 불변 객체를 사용하면 동시성 문제를 해결할 수 있지만, 객체를 생성하는 비용이 굉장히 크기 때문에 성능상 문제가 발생할 수 있다. 따라서 불변 객체를 사용하는 경우 객체를 재사용하는 방법이 필요할 것이다.

동시성 문제를 의식하며 코드를 작성해보자!!

참고 링크



https://kadosholy.tistory.com/121