

1. Race Condition

Race Condition이란?

Race Condition이란 한국어로 경쟁 상태를 의미하며 공유 자원에 대해 여러 개의 프로세스 또는 스레드가 동시에 접근을 시도할 때 접근의 타이밍이나 순서 등이 결과값에 영향을 줄 수 있는 상태를 말한다. 즉, Race Condition이 발생할 경우 작업은 일관성이 깨질 수 있기 때문에 이를 방지하는 다양한 기법들을 사용해야 한다.

보통 프로그램에서 경쟁 조건을 일으킬 수 있는 구간 즉, 공유 자원에 동시에 접근하는 구간을 Critical Section이라고 한다.

Race Condition을 막는 방법은 크게 2가지가 있다.

1. Shared Memory를 없애고 Process 또는 Thread 간에 직접 Communication 하는 방법
 1. 확실한 방법이지만 Overhead가 발생한다는 단점이 있다.
2. Thread 또는 Process 간에 Synchronization 하는 방법

Process, Thread 간에 동기화를 유지하는 기법들

스레드간의 동기화를 위해선 소프트웨어적인 해결 방법인 Peterson을 사용할 수 있고, 하드웨어적인 해결 방법인 lock을 사용하는 방법이 있다.

lock을 그냥 사용하기엔 너무 low 레벨이기 때문에 이를 추상화시킨 방법으로는 mutex lock과 Semaphore, Monitor 등이 있다.

- mutex lock과 Semaphore는 lock을 사용한다는 점은 동일하지만 lock을 얻는 주체와 해제하는 주체가 동일한지의 여부가 가장 큰 차이이다.
- Monitor는 상호 배제를 유지하기 위해 사용되어지는 보다 추상화된 데이터 타입(ADT)이다. 아마 이 형태도 Lock을 사용하지 않을까 싶다.

2. Java의 스레드를 동기화 기법

- 객체
 - 필드 : volatile(스레드간에 동기화를 시키는 자료구조), => AtomicInteger

```
class SynchronizedThread {
    private volatile int number

    public int get() {
        return number;
    }

    public void increment() {
        number++;
    }
}
```

- 메소드 : synchroized, synchronized Block, static synchronized => Vector

```
class synchronizedThread {
    private int number;

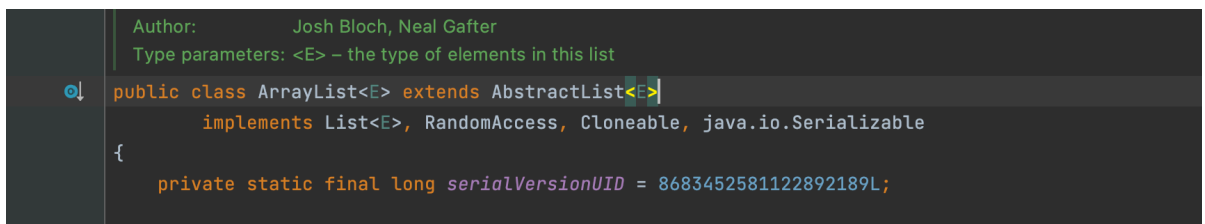
    public synchronized int get() {
        return number;
    }

    public synchronized void increment() {
        return number++;
    }
}
```

- 메서드 접근시에 시킬래?, 필드 접근시에 시킬래?
- synchronized Block, synchronized Method를 사용해서 Thread간의 동기화를 유지.
- 특정 자료구조를 사용하는 경우라면 synchronized Block 대신 자체적으로 동기화 되어지는 자료구조를 사용.
 - ex) ArrayList 대신 Vector 사용

ArrayList

- 명세는 동일



```
Author: Josh Bloch, Neal Gafter
Type parameters: <E> - the type of elements in this list

public class ArrayList<E> extends AbstractList<E>
    implements List<E>, RandomAccess, Cloneable, java.io.Serializable
{
    private static final long serialVersionUID = 8683452581122892189L;
```

- but 메소드 레벨을 보면 차이를 볼 수 있음

```

/**
 * Appends the specified element to the end of this list.
 *
 * @param e element to be appended to this list
 * @return {@code true} (as specified by {@link Collection#add})
 */
public boolean add(E e) {
    modCount++;
    add(e, elementData, size);
    return true;
}

```

Vector

- 명세는 동일

```

Author: Lee Boynton, Jonathan Payne
Type parameters: <E> – Type of component elements

public class Vector<E>
    extends AbstractList<E>
    implements List<E>, RandomAccess, Cloneable, java.io.Serializable
{
    The array buffer into which the components of the vector are stored. The capacity of the vector
    the length of this array buffer, and is at least large enough to contain all the vector's elements.
}

```

- but 메소드 레벨을 보면 차이를 볼 수 있음

```

*
* @param e element to be appended to this Vector
* @return {@code true} (as specified by {@link Collection#add})
* @since 1.2
*/
public synchronized boolean add(E e) {
    modCount++;
    add(e, elementData, elementCount);
    return true;
}

```

- static synchronization method