3. 객체 공유 – 특정객체를 명시적으로 동기화하거나, 객체 내부에 적절한 동기화 기능을 내장시키자.

3.1 가시성 – 특정 변수의 값을 가져갈 때 다른스레드가 작성한 값을 가져갈수 있다는 보장도 없고, 심지어 값을 읽지 못할수도 있다.

메모리상의 공유된 변수를 여러 스레드에서 서로 공유할 수 있게 하려면 반드시 동기화 기능을 구현해야한다.

Ex3\_1\_NoVisibility

\*\* 동기화 되지않은 상황에서, 메모리상의 변수를 대상으로 작성해둔 코드가 반드시 이런순서로 동작할것이라고 단정할수 없다.

즉, 여러스레드에서 공동으로 사용하는 변수에는 항상 적절한 동기화 기법을 적용해야한다.

3.1.1 스테일데이터 – 동기화되지않은 프로그램을 실행하며 일어나는 변수데이터 오류현상. Ex3\_1\_NoVisibility

- 프로그램이 멈추지않아, 시스템 자원을 점유 가능성.

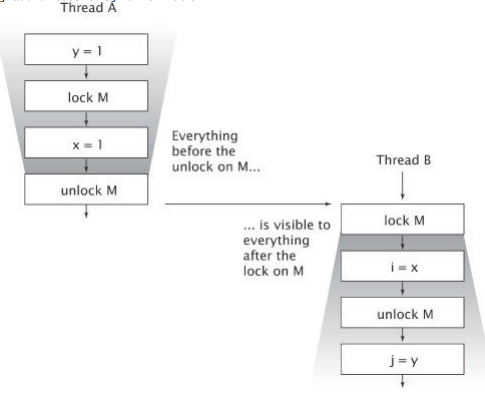
**Ex3\_2\_MutableInteger** - Get과 set의 각 스레드의 대상 객체가 다르다면.

**Ex3\_3\_SynchronizedInteger -** Get과 set의 각 스레드의 대상 객체를 동기화할 때(@GuardBy)

3.1.2 단일하지않은 64비트 연산 – long, double (64bits)데이터형태. 두번의 32비트 연산을 사용필요.

3.1.3 락과 가시성 – 내장된 락을 잘 사용해, 이전 스레드A가 해당 변수를 사용후 결과를 예측할수 잇는 상태에서

B스레드에서 계속 사용할수 있다.

쓰레드A = synchronized 블락.

쓰레드B = 같은 락을 사용하며, synchronized 블락

* 스레드 안정화.

\*\* 락은 상호 배제뿐만 아니라 정상적인 메모리 가시성을 확보하기 위해서도 사용한다. 변경 가능하면서 여러 스레드가 공유해 사용하는 변수를 각 스레드에서 각자 최신의 정상값으로 활용하려면, 동일한 락을 사용해 모두 동기화해야한다.

3.1.4 volatile(휘발성) 변수 – 컴파일러와 런타임 모두 ‘이 변수는 공유해 사용하고, 따라서 실행순서를 재배치해서는 안된다’고 이해한다.

Volatile변수는 synchronized나 락이 작용되지 않는다. 근데도 volatile변수와 synchronized블록에 진입한 변수와 비슷한 상태에 있다.

락 – 가시성과 연산의 단일성 보장. volatile변수 – 연산의 단일성은 보장못하고 가시성만 보장된다.

3.2 공개와 유출

Ex3\_5\_Secrets

특정 객체를 현재 코드의 스코프 범위 밖에서 사용할수 있도록 만들면 ‘공개상태’라고 한다.

Ex3\_6\_UnsafeStates

클래스 내부의 상태변수를 외부에 공개하거나, 객체가 안정적이지 않은 상태에서 의도적으로 공개하지 않았지만,

외부에서 사용할 수 있게 공개된 경우가 ‘유출상태’라고 한다.

Ex3\_7\_ThisEscape

객체를 공개하면 모든 변수 속성에 연결되어있는 객체 내부의 변수나 메소드를 불러오는 객체를 함께 공개된다.

3.2.1. 생성 메소드 안정성

Ex3\_7\_ThisEscape의 이슈를 방지하려면, 생성 메소드를 실행하는 도중에는 this 변수가 외부에 유출되지 않게 해야한다.

3.3 스레드 한정 – 특정 객체를 단일 스레드에서만 활용하는 기법. 스윙 이벤트 처리와 JDBC connection 객체를 풀링해 사용하는 방법 두가지.

단, 언어 자체와 기본 라이브러리 ThreadLocal 클래스 등으로의 스레드 한정기법에선 객체유출을 주의해야함.

3.3.1 주먹구구식 – 특정 모듈의 기능을 단일 스레드로 동작하게 구현, 오류 최소화.

3.3.2 스택한정 – Ex3\_9\_Animals

특정 객체를 로컬변수를 통해서만 사용할수 있는 특별한 경우의 스레드 한정기법.

로컬변수로 변경상태관리 용이, 암묵적으로 특정 스레드에 한정으로 실행중.

3.3.3 ThreadLocal – 스레드 내부의 값과 값을 갖고있는 객체를 연결해 스레드 한정 기법을 적용할수 있게 도와주는 형식적인 방법.

Get, set 메서드로 스레드마다 다른값을 사용하게 관리한다. JDBC의 connection 인스턴스를 전역변수에 넣지않고, ThreadLocal로 각 스레드마다의 연결 객체를 갖게하는 방법이다.

ThreadLocal 클래스는 앱 프레임웤을 구현할 때, 종종 사용된다. 예를 들어, J2EE컨테이너는 EJB를 사용하는 동안 해당스레드와 트랜잭션 컨텍스트를 연결해 관리한다. 스레드 단위로 트랜잭션 컨텍스트를 관리하고자 할때는, static으로 선언된 ThreadLocal 변수에 트랜잭션 컨텍스트를 넣으면 편리하다. 프레임 웤에서 현재 진행중인 트랜잭션을 확인하고 싶다면, 보관되어있는 ThreadLocal 클래스에서 찾으면 된다.

3.4 불변성 – 처음 생성후, 값이 전혀 바뀌지않는 객체.

조건) 객체 생성 후, 객체 상태를 변경할수 없다.

내부의 모든 변수는 final로 설정되어 있어야한다. \*굳이 final 변수가 아니라도 불변객체는 가능하다. ex) Java.lang.String

적잘한 방법으로 생성되야 한다, 예컨대, this 변수에 대한 참조가 외부로 유출되지 말아야한다.

3.4.1 final 변수 – 초기화 안전성

변수의 final화함으로 해당 변수에 어떤값이 들어갈수 있는지에 대해 고려해야할 범위가 줄어든다. 외부에서 사용할일 없는 변수는 private, 나중에 변경할일 없다고 판단되는 변수는 final화 하는것이 현명하다.

3.4.2 불볍객체를 공개할떄, volatile 사용.

Ex3\_12\_OneValueCache

참고. Ex2\_2\_UnsafeCountingFactorizer에선 AtomicReference 두개로 필요한 두개 변수를 단일 연산으로 읽고, 쓰는것이 어려워 스레드 안정성에 침해됬다.

3\_12의 예제와 같이, 여러 개의 값을 한데 묶는 불변 클래스를 만들어 사용하는 편이 안정성에 기여된다.

서로 연관된 여러개의 변수값을 서로 읽고 쓰는과정에 경쟁조건이 발생할수 있는데, 불변객체에서 해당 변수를 모아두면 경쟁조건을 방지할수있다.

Ex3\_13\_VolatileCachedFactorizer

Ex3\_12\_OneValueCache 클래스의 입력값과 결과를 캐시한다. 스레드가 volatile로 선언된 cache변수에 새로 생성한 Ex3\_12\_OneValueCache 인스턴스를 설정하면, 다른 스레드에서도 cache변수에 설정된 새로운 값을 즉시 사용가능하다.

3.5 안전공개

Ex3\_14\_StuffIntoPublic // 동기화전 객체 공개는 좋지않은 코드이다.

3.5.1 적절하지 않은 공개방법: 정상적인 객체도 문제를 일으킨다.

Ex3\_15\_Holder

객체를 공개하는 스레드가 아닌 다른 스레드에서 assertSanity메서드를 호출하면서 assertionError가 발생할 수 있다.

문제의 원인은 Holder 객체를 안전하지 않은 방법으로 공개 했기때문인데, n변수를 final로 지정하게되면 문제요인은 사라진다.

동기화 하지않은 공개방법의 문제점: 1. 스테일 상태로, 변수에 값을 지정해도 참조가 제대로 들어가지 않은경우.

2. 변수에서 정상적인 참조값을 가져가지만, 클래스입장에서 스테일상태에 빠진경우.

3.5.2 불변객체와 초기화 안정성

불볍 객체를 사용하면, 객체의 참조를 외부에 공개할 때 추가적인 동기화 방법을 사용하지않았다고 해도, 항상 안전하게 올바른 참조값을 사용할수 있다.

\*\*불변객체는 별다른 동기화 방법을 적용하지않아도, 어느 스레드에서든 안전하게 사용가능하다. 불변객체를 공개하는 부분에 동기화 처리를 하지않아도 문제없다.

3.5.3 안전한 공개방법의 특성

안전한 공개 = 공개 스레드와 변수사용 스레드는 모두 동기화가 되야한다.

|  |
| --- |
| 객체를 안전하게 공개하려면 해당 객체에 대한 참조와 객체 내부의 상태를 외부의 스레드에게 동시에 볼 수 있어야한다.   * 객체에 대한 참조를 static 메서드에서 초기화 시킨다. * 객체에 대한 참조를 volatile 변수 또는 AtomicReference 클래스에 보관한다 * 객체에 대한 참조를 올바르게 생성된 클래스 내부의 final 변수에 보관한다. * 락을 사용해 올바르게 막혀 있는 변수에 객체에 대한 참조를 보관한다. |

Ex) Vector 객체와 synchronizedList 메서드를 사용한 동기화된 스레드 안전한 컬렉션으로 객체 보관 - 마지막 방법.

스레드A가 객체X를 컨렉션에 보관, 스레드B가 객체X를 읽는다면.. 스레드A와 B에 동기화를 안해도, 자바 제공 스레드 안전한 컬렉션은 API에 동기화 기능이 있다.

해싱 Hash Table, concurrent Map, synchronizedMap은 K,V모두 스레드 안전하다.

객체 Vector, CopyOnwriteArrayList, CopyOnWriteArraySet, synchronizedList, synchronizedSet 메서드로만든 컬렉션은 스레드 안전하다.

BlockingQueue, ConcurrentLinkedQueue 컬렉션 객체도 스레드 안전하다.

static변수선언하며 new연산자로 생성 메소드를 실행해 객체 생성하는 방법이 가장 안전한 객체 공객방법이다

public static Holder holder = new Holder(42);

3.5.4 결과적으로 불변인 객체

Date 클래스는 불변객체가 아니므로, 여러스레드에서 공유해 사용하려면 락을 걸어야한다.

로그인 시각으로 매핑하면, public Map<String, Date> lastLogin = Collections.synchronizedMap(new HashMap<String, Date>());

Date 인스턴스 값이 ㄷ안ㄱ바뀌면 synchronizedMap은 자동 동기화가 적용된다.

3.5.5 가변객체

가변객체로 사용할 때, 공개하는 부분과 가변객체를 사용하는 부분을 동기화 해야한다.

|  |
| --- |
| 가변성에따라 객체를 공개할 때, 필요한점은   * 불변객체는 어떤 방법으로 공개해도 아무 문제없다 * 결과적으로 불변인 객체는 안전하게 공개 해야한다 * 가변객체는 안전하게 공개해야하고, 스레드에 안전하게 만들거나 락으로 동기화 시켜야한다. |

3.5.6 객체를 안전하게 공유하기

객체의 참조 용도에 따라 참고해야할 사항이 잇다.

여러 스레드를 동시에 사용하는 병렬 프로그램에서 객체를 공유해 사용하고자 할 때 가장 많이 사용되는 몇가지 원칙은…

* 스레드한정: 스레드에 한정된 객체는 완전하게 해당 스레드 내부에 존재하면서 그 스레드에서만 호출해 사용할수 있다.
* 읽기 전용 객체공유: 읽기전용 객체는 공유하지않아도, 여러 스레드에서 사용가능하다(객체 변경불가).
* 스레드 안전한 객체공유: 객체 내부적으로 필수적인 동기화 기능이 만들어져있어서, 외부동기화가 필요없다. 여러 스레드에서 사용가능.
* 동기화 방법 적용: 특정 객체에 동기화 방법을 적용해두면, 지정한 락을 획득하기전엔 해당 객체를 사용할수 없다. 객체 내부에서 사용하는 객체나 공개된 객체 가운데 특정락을 확보해야 사용하는 동기화 방법이 적용된것이라 보면된다.