--------------------------------------------------------

1부. 기본원리

챕터2. 쓰레드 안정성

--------------------------------------------------------

안전한 코드 - 근본적으로, 상태, 특히 공유되고 변경할수 있는 상태에 대한 접근관리.

스레드가 하나이상 상태변수에 접근하고, 변수값을 쓰게되면 관련 모든 스레드가 동기화synchronized를 통해 조율해야함.

\*\*스레드 클래스 설계시 바람직한 객체 지향기법이 필수다. 캡슐화, 불변객체와 조건명시 해야함.

2.1 쓰레드 안정성

\*\*상태없는 객체(선언한 변수가 없고, 다른 클래스의 변수 참조하지않음)는 항상 스레드안전.

Ex2\_1\_StatelessFactorizer.

2.2 단일연산

한개의 상태를 추가. count++

Ex2\_2\_UnsafeCountingFactorizer

2.2.1 경쟁조건 - 점검 후 행동. 사실확인(데이터 존재여부) 후 행동(데이터 생성).

2.2.2 늦은초기화lazy initialization시 경쟁조건 - 늦은 초기화면 인스턴스 생성, 리턴.

Ex2\_3\_LazyInitRace

2.2.3 복합동작 - 점검 후 행동에 읽고 수정하고 쓰기 같은 일련의 동작. count++같이.

2.3 락

더 많은 상태관리.

공유된 상태에 배타적으로 접근할수 있도록 보장하는 규칙을 만들떄 유용.

필수정보 - 마지막 입력값, 입력값에 대한 결과값.

Ex2\_5\_UnsafeCachingFactorizer

\*\* 상태를 일관성을 유지하려면, 관련 변수들을 하나의 단일 연산으로 갱신해야한다.

2.3.1 암묵적인 락(모니터 락) - 자바에 내장된 락.

synchronized(lock){ // static 클래스.

// lock 으로 보호된 공유상태에 접근하거나 해당상태를 수정한다. }

Ex2\_6\_SynchronizedFactorizer

2.3.2 재진입성 - 확보요청단위가 아닌 스레드 단위로 락을 얻는다는것. 데드락 응급처치.

구현시, 획보 횟수와 확보한 스레드 루트를 알아야함.

Ex2\_7\_LoggingWidget

2.4 락으로 상태보호

복합동작에 synchronized블록으로 감쌀뿐 아니라, 특변변수에 접근조율로 동기화할때

변수에 접근하는 모든 부분을 동기화해야한다.

흔한 실수 - 공유변수에 값을 쓸때만 동기화 필요? NO!

\*\*여러 스레드에서 접근할수 있고 변경가능한 모든 변수를 대상으로

해당변수에 접근할때는 항상 동일한 락을 먼저 확보한 상태여야 한다.

이 경우, 해당변수는 확보된 락에 의해 보호된다고 말한다. @GuardedBy

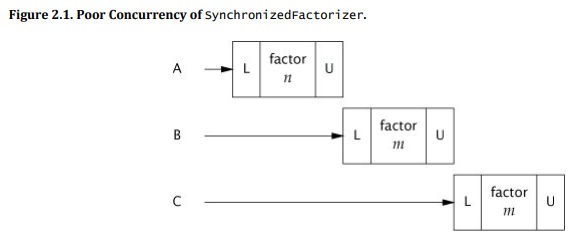
\*\*모든 변경할수있는 공유변수는 정확하게 하나의 락으로 보호해야한다. 락보호에 표기해야, 유지보수에도 용이함.

여러변수에 대한 불변조건이 있으면, 해당변수들은 모두 같은 락으로 보호해야한다. Ex2\_6\_SynchronizedFactorizer

무차변적 synchronized 메서드는 과도한 동기화나 자원부족이 생길우려 -> 활동성, 성능문제.

2.5 활동성과 성능

Ex2\_6\_SynchronizedFactorizer와 같이 service 전체에 synchronized하면 병렬처리능력이 떨어짐(Fig2.1)



요점: synchronized의 단일연산 블록을 소규모화 + 스레드 안정성 유지 -> 동시성 향상.

Synchronized의 기준은 복합동작을 수행하는 동안 락을 잡기, 오래걸릴 가능성이 있는 인수분해 작업을 시작하기 전에 락을 잡기.

\*\* 복잡하고 오래걸리는 계산작업, 네트워크작업, 사용자 입출력작업과 같이 빨리 끝나지 않을 수 있는 작업을 하는 부분에선 가능한 락을 잡지 말아라.