Luke's Language로 파싱된 이펙티브 자바

신뢰성 스레드 프로그래밍

제작. 홍성혁

스레드 안전성 수준을 문서화하라

복습 - Synchronization

동기화(Synchronization)란 시스템이 조화를 이루어 작동하도록 이벤트를 조율하는 것을 말합니다. 예를 들어, 오케스트라의 지휘자(conductor)는 오케스트라가 *동기화(synchronized)되거나 일정한 시간(in time)에 맞춰 연주하도록 조율*합니다. 모든 부분이 동시에 작동하는 시스템은 동기적(synchronous) 또는 "동기화된(in sync)" 상태라고 하며, 그렇지 않은 경우를 *비동기적(asynchronous)*이라고 합니다.

스레드 안전성 수준을 문서화하라

복습 - Synchronization

동기화(Synchronization)란 시스템이 조화를 이루어 작동하도록 이벤트를 조율하는 것을 말합니다. 예를 들어, 오케스트라의 지휘자(conductor)는 오케스트라가 *동기화(synchronized)되거나 일정한 시간(in time)에 맞춰 연주하도록 조율*합니다. 모든 부분이 동시에 작동하는 시스템은 동기적(synchronous) 또는 "동기화된(in sync)" 상태라고 하며, 그렇지 않은 경우를 *비동기적(asynchronous)*이라고 합니다.

동기화(Synchronization)란
여러 프로세스(multiple process)의 결합(join up) 또는 handsake 할 시점을 조율(coordinating)하는 것
이를 통해,
- 합의에 도달하거나 (상태나 데이터에 대한 동기화)
- 특정 수행의 순차적 반영(commit)
하는 것을 목표로 한다.

스레드 안전성 수준을 문서화하라

복습 - Synchronization

동기화가 필요한 곳

1. 포크와 조인 Forks - Joins
  - 작업이 포크되면, 다수의 하위 작업으로 분할되고, 서비스
  - 모든 하위 작업이 끝날때까지 대기하고, 조인
  - *포크 - 대기* 과정에 동기화가 필요

2. 생산자 - 소비자 Producer - Consumer
  - 소비자 프로세스는 생산자 프로세스가 데이터를 만들때까지 대기
  - 소비자 프로세스는 생산자 프로세스에 종속
  - *생산자의 데이터 생성 - 소비자의 대기*에 동기화가 필요 (JS)

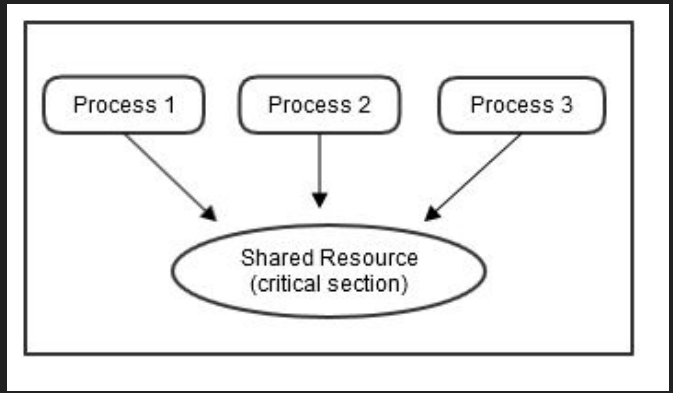
3. 배타적 자원 Exclusive use resources
  - 여러 프로세스가 리소스에 종속
  - 동시에 액세스하게 될 경우, 운영체제는 시점마다, 단 하나의 프로세스만 액세스 허용
  - *동시성 처리*를 이유로 동기화가 필요

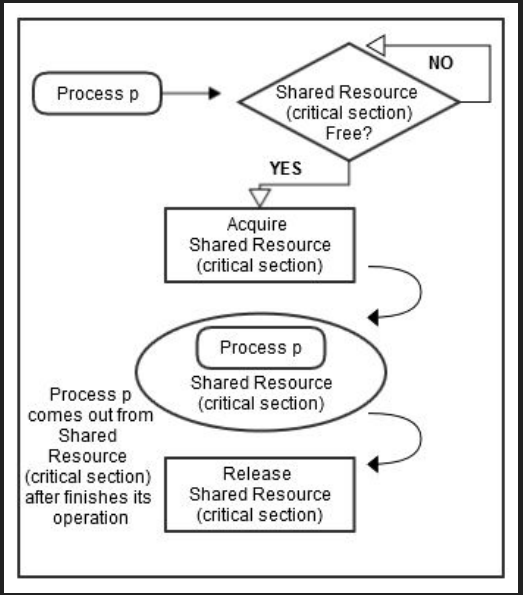
스레드 안전성 수준을 문서화하라

복습 - Synchronization

스레드 동기화는 두 개 이상의 동시 프로세스 또는 스레드가 **임계 영역**(critical section) 이라고 알려진 특정 프로그램 세그먼트를 동시에 실행하지 않도록 보장하는 메커니즘

한 스레드가 임계 구역 (프로그램의 직렬화된 세그먼트)을 실행하기 시작하면 다른 스레드는 첫 번째 스레드가 완료될 때까지 기다려야 합니다.





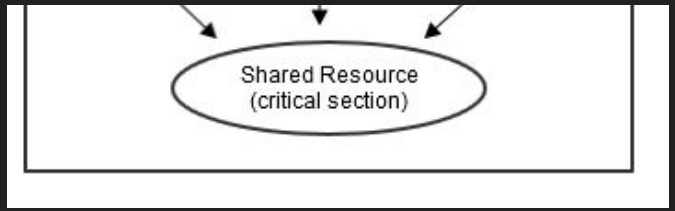
스레드 안전성 수준을 문서화하라

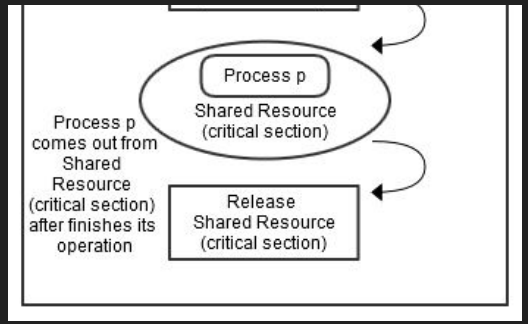
복습 - Synchronization

스레드 동기화는 두 개 이상의 동시 프로세스 또는 스레드가 **임계 영역**(critical section) 이라고 알려진 특정 프로그램 세그먼트를 동시에 실행하지 않도록 보장하는 메커니즘

한 스레드가 임계 구역 (프로그램의 직렬화된 세그먼트)을 실행하기 시작하면 다른 스레드는 첫 번째 스레드가 완료될 때까지 기다려야 합니다.

적절하게 적용되지 않으면
- 변수 값을 예측할 수 없거나,
- 프로세스 또는 스레드의 컨텍스트 전환 타이밍에 따라 달라질 수 있는 *경쟁 조건(Race Condition)* 이 발생할 수 있습니다 .





스레드 안전성 수준을 문서화하라

복습 - Synchronization

동기화에 고려해야할 부분

1. 데드락 Deadlock
- 프로세스가 다른 프로세스가 보유하고 있는 공유 리소스(중요 섹션)를 대기. 프로세스는 계속 기다림.

2. 기아 Starvation
- 다른 프로세스들로 인해, 프로세스가 임계 영역을 진입하지 못하고 계속 기다림.

3. 우선순위 역전 Priority Inversion
- 높은 우선 순위 프로세스가 임계영역에 있는데, 중간 순위 프로세스에 의해 중단되는 현상

4. 바쁜 대기 Busy Waiting
- 프로세스가 임계영역 진입을 위해 자주 폴링한다.
- 빈번한 폴링으로인해, 다른 프로세스의 처리 시간을 뺏는다.

스레드 안전성 수준을 문서화하라

복습 - Synchronization

동기화에 고려해야할 부분

1. 데드락 Deadlock
- 프로세스가 다른 프로세스가 보유하고 있는 공유 리소스(중요 섹션)를 대기. 프로세스는 계속 기다림.

2. 기아 Starvation
- 다른 프로세스들로 인해, 프로세스가 임계 영역을 진입하지 못하고 계속 기다림.

3. 우선순위 역전 Priority Inversion
- 높은 우선 순위 프로세스가 임계영역에 있는데, 중간 ㅊ

4. 바쁜 대기 Busy Waiting
- 프로세스가 임계영역 진입을 위해 자주 폴링한다.
- 빈번한 폴링으로인해, 다른 프로세스의 처리 시간을 뺏

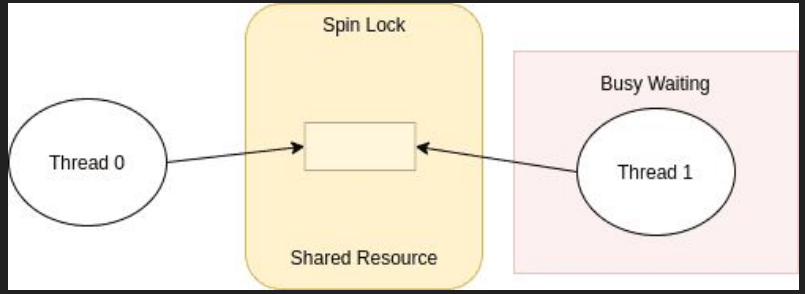
- 프로세스 H, M, L이 있다.
- H와 L은 자원 R(임계 영역)에 의존적이다.
- L이 실행중에 M이 L을 선점한다(preempt, CPU 선점)
- H는 L로부터 R을 반환 받아야하지만, L은 R을 반환할 수 없다.
- 결론적으로, H는 M으로 인해 자원을 반환받을 수 없다.

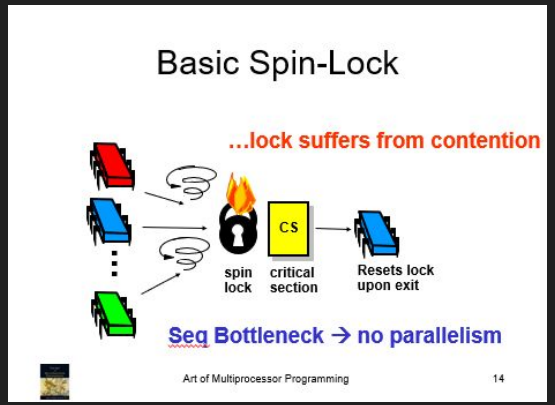
스레드 안전성 수준을 문서화하라

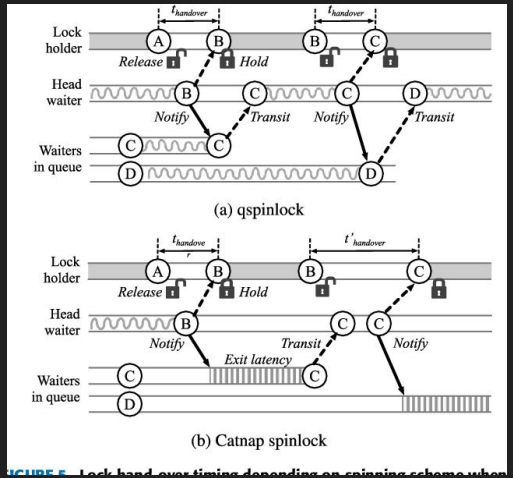
복습 - Synchronization

동기화 구현

1. Spinlock
   - poll
2. Barrier
   - (Wbarrier)i = f((Tbarrier)i, (Rthread)i)
3. Semaphore
   - wait
   - Mutex





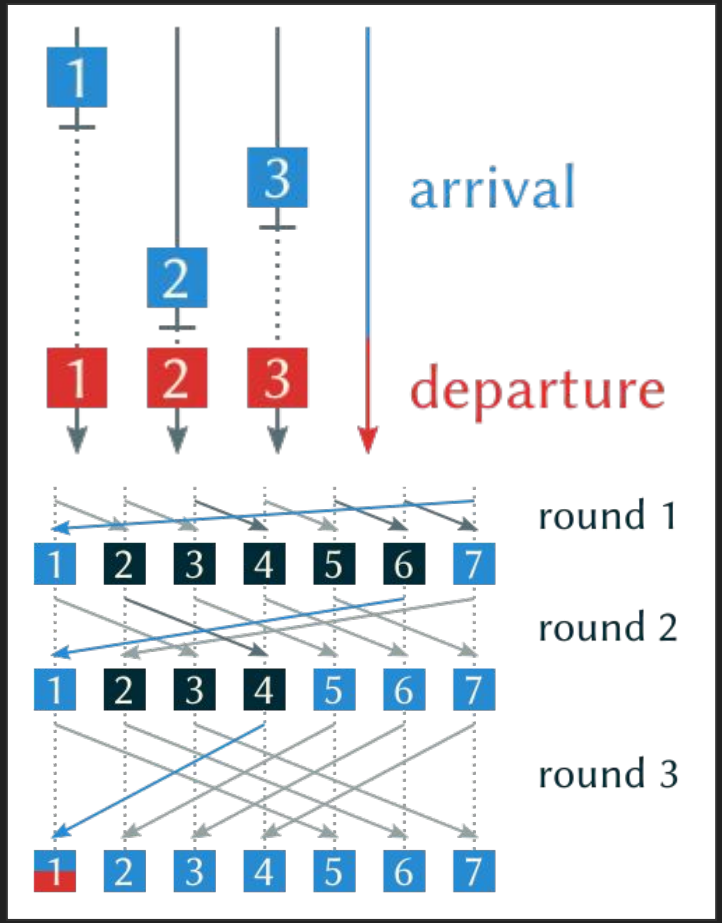


스레드 안전성 수준을 문서화하라

복습 - Synchronization

동기화 구현

1. Spinlock
   - poll
2. Barrier
   - (Wbarrier)i = f((Tbarrier)i, (Rthread)i)
3. Semaphore
   - wait
   - Mutex

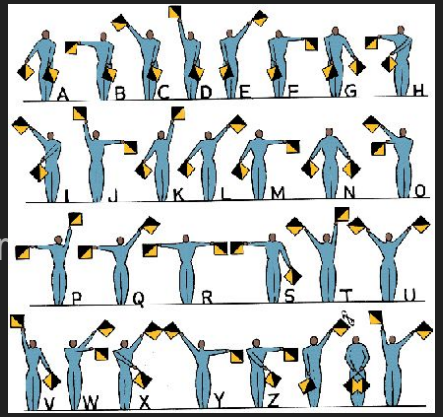


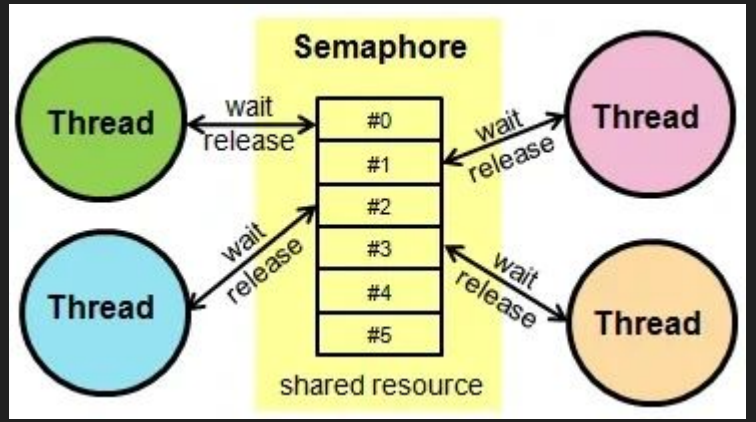
스레드 안전성 수준을 문서화하라

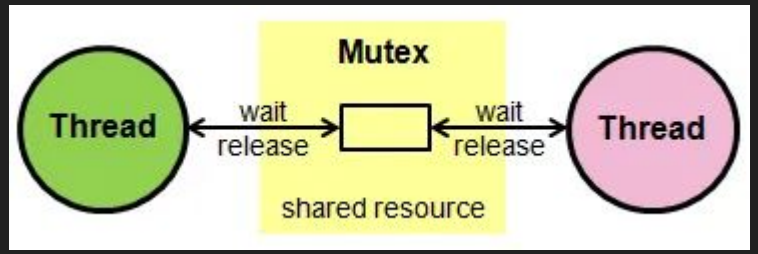
복습 - Synchron

동기화 구현

1. Spinlock
   - poll
2. Barrier
   - (Wbarrier)i = f((Tbarrier
3. Semaphore
   - wait
   - Mutex







스레드 안전성 수준을 문서화하라

복습 - Synchronization

Spin Lock vs Mutex

Features	Spinlock	Mutex
Definition	It is a type of lock that causes a thread attempting to obtain it to check for its availability while waiting in a loop continuously.	It is a program object designed to allow different processes may take turns sharing the same resource.
Sleep	The process may not sleep while waiting for the lock.	The process may sleep while waiting for the lock.
Context Switching	Spinlock makes no use of context switching.	It involves context switching.
Holding Time	It temporarily prevents a thread from moving.	It may block a thread for an extended amount of time.
Usage	It is useful for limited critical sections; else, it wastes the CPU cycles.	It is useful for crucial extended areas where frequent context switching would add overhead.
Preemption	It doesn't support preemption.	It supports preemption.

스레드 안전성 수준을 문서화하라

Item 82 - 스레드 신뢰성

API 제공시, 스레드 안전성 수준을 명시해야한다.

스레드 안정성 수준

1. immutable
2. unconditionally thread-safe
3. conditionally thread-safe
4. not thread-safe
5. thread-hostile

스레드 안전성 수준을 문서화하라

Item 82 - 스레드 신뢰성

스레드 안정성 수준

1. immutable
   - 불변
   - String, Long, BigInteger
2. unconditionally thread-safe
   - 내부에 동기화가 잘 구현되어 있어, 외부 동기화 여부와 상관 없이 무조건 스레드에 안전
   - AtomicLong, ConcurrentHashmap
3. conditionally thread-safe
   - 일부 메서드의 동시 사용시 외부 동기화를 필요로 함. 문서화 필수
   - Collection.synchronized (컬렉션이 반환하는 반복자가 동기화를 필요로 함)

스레드 안전성 수준을 문서화하라

Item 82 - 스레드 신뢰성

스레드 안정성 수준

1. immutable
   - 불변
   - String, Long, BigInteger
2. unconditionally thread-safe
   - 내부에 동기화가 잘 구현되어 있어, 외부 동기화
   - AtomicLong, ConcurrentHashmap
3. conditionally thread-safe
   - 일부 메서드의 동시 사용시 외부 동기화를 필요로 함. 문서화 필수
   - Collection.synchronized (컬렉션이 반환하는 반복자가 동기화를 필요로 함)

synchronizedMap이 반환한 맵의 컬렉션 뷰를 순회하려면 반드시 그 맵을 락으로 사용해

수동으로 동기화하라.

```
Map<K, V> m = Collections.synchronizedMap(new HashMap<>());
Set<K> s = m.keySet(); // 동기화 블록 밖에 있어도 된다.
   ...
synchronized(m) { // s가 아닌 m을 사용해 동기화해야 한다!
   for (K key : s)
      key.f();
}
```

이대로 따르지 않으면 동작을 예측할 수 없다.

스레드 안전성 수준을 문서화하라

Item 82 - 스레드 신뢰성

스레드 안정성 수준

4. not thread-safe
   - 외부 동기화 메커니즘을 필요로 함
   - ArrayList, HashMap

5. thread-hostile
   - 외부 동기화로 감싸도 멀티스레드 환경에서 안전하지 않음. 동시성을 고려하지 않은 사례

지연 초기화는 신중히 사용하라

Item 83 - Lazy Initialization

지연 초기화란?
필드의 초기화 시점을 그 값이 처음 필요할 때까지 늦추는 기법

코드 83-1 인스턴스 필드를 초기화하는 일반적인 방법

```
private final FieldType field = computeFieldValue();
```

지연 초기화는 신중히 사용하라

Item 83 - Lazy Initialization

지연 초기화 vs 일반 초기화
대부분의 상황에서 일반 초기화가 지연 초기화보다 낫다.
해당 필드를 초기화하는 비용이 클 때, 쓸 것

지연 초기화는 신중히 사용하라

Item 83 - Lazy Initialization

초기화 순환성을 깨뜨릴 것 같으면 synchronized 사용

코드 83-2 인스턴스 필드의 지연 초기화 - synchronized 접근자 방식

```
private FieldType field;

private synchronized FieldType getField() {
    if (field == null)
        field = computeFieldValue();
    return field;
}
```

지연 초기화는 신중히 사용하라

Item 83 - Lazy Initialization

성능으로 정적 필드를 지연 초기화한다면,
지연 초기화 홀더 클래스(lazy initialization holder class) 관용구 사용

코드 83-3 정적 필드용 지연 초기화 홀더 클래스 관용구

```
private static class FieldHolder {
    static final FieldType field = computeFieldValue();
}

private static FieldType getField() { return FieldHolder.field; }
```

지연 초기화는 신중히 사용하라

Item 83 - Lazy Initialization

성능 때문에 인스턴트 필드를 지연 초기화해야 한다면,
이중검사(double-check) 관용구를 사용

- 필드 초기화에 동기화를 위해 volatile 필요

코드 83-4 인스턴스 필드 지연 초기화용 이중검사 관용구

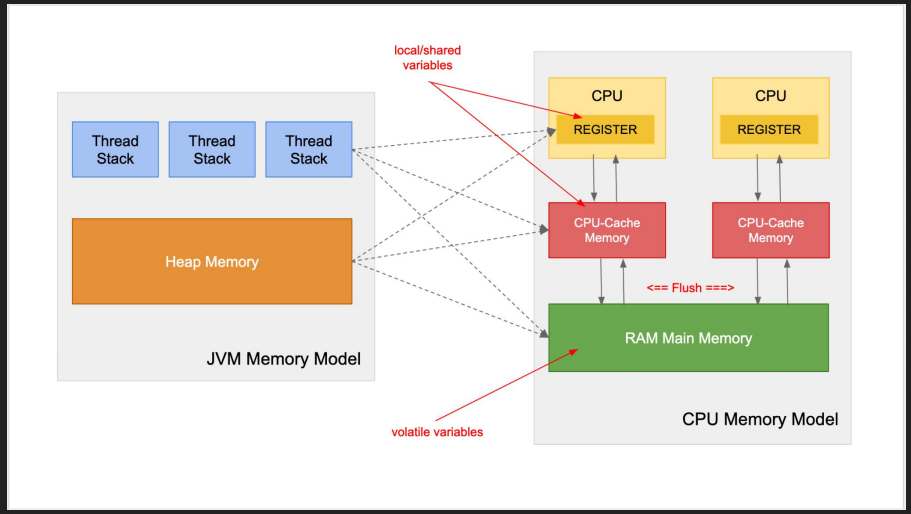
```
private volatile FieldType field;

private FieldType getField() {
    FieldType result = field;
    if (result != null) { // 첫 번째 검사 (락 사용 안 함)
        return result;

    synchronized(this) {
        if (field == null) // 두 번째 검사 (락 사용)
            field = computeFieldValue();
        return field;
    }
}
```

프로그램의 동작을 스레드 스케줄러에 기대지 말라

Item 84 - 스레드 스케줄러에 종속되지 않기

스레드 스케줄러란?
스레드에 CPU를 할당하는 것

자바 스레드는 요즘엔 가상 스레드도 있긴 하지만,
기본적으로 운영체제 스레드에 일대일 래핑된 스레드
즉, 운영체제가 관리한다는 것



프로그램의 동작을 스레드 스케줄러에 기대지 말라

Item 84 - 스레드 스케줄러에 종속되지 않기

CPU 자원을 할당받아 빠르게 처리되는 코드를 짠다면,
다른 플랫폼에서 다르게 작동될 수 있다.
JVM마다, 운영체제마다

프로그램의 동작을 스레드 스케줄러에 기대지 말라

Item 84 - 스레드 스케줄러에 종속되지 않기

스레드는 당장 처리해야 할 작업이 없다면 실행돼서는 안 된다.

바쁜 대기(Busy waiting) 상태를 만들지 말라
프로세서에 큰 부담을 주어, 다른 유용한 작
실행의 기회를 박탈한다.

Thread.yield 써서 해결하려고도 하지 말자

코드 84-1 끔찍한 CountDownLatch 구현 - 바쁜 대기 버전!

```
public class SlowCountDownLatch {
    private int count;

    public SlowCountDownLatch(int count) {
        if (count < 0)
            throw new IllegalArgumentException(count + " < 0");
        this.count = count;
    }

    public void await() {
        while (true) {
            synchronized(this) {
                if (count == 0)
                    return;
            }
        }
    }

    public synchronized void countDown() {
        if (count != 0)
            count--;
    }
}
```

프로그램의 동작을 스레드 스케줄러에 기대지 말라

Item 84 - 스레드 스케줄러에 종속되지 않기

특정 스레드가 CPU를 충분히 얻지 못하더라도,

Thread.yield 써서 해결하려고도 하지 말자
테스트할 수단도 없다.

우선순위 조절하는 방법도 쓰지 말자

앱에 문제가 있다면, 근본적인 문제를 찾아 고치자

E.O.D.