Avoid finalizers and cleaners

개요

finalizer는 예측할 수 없고, 종종 위험하고 일반적으로 필요없습니다.

finalizer는 협소하게 사용가능한 범위가 있긴 합니다만,,, 그래도 쓰지 마세요!

Java 9에서는 finalizer는 deprecated되었고 그 대안으로 Cleaner가 나왔습니다.

finalizer보다는 덜 위험하지만, 여전히 예측할 수 없고 느리고 필요없습니다!!

C++ 프로그래머 주의사항

C++프로그래머는 finalzier와 cleaner를 C++의 destructors와 동일시하지 않길 바랍니다. destructors는 자원을 회수하는 일반적인 방법입니다. constructor에 반대되죠.

하지만 Java에서는 자동으로 garbage collector가 회수해줍니다. 차라리 자바에서는 try-with-resource나 try-finally block이 destructor개념에 가깝다고 볼 수 있네요.

Finalizer와 Cleaner를 사용하지 말아야할 이유들

1. 정확한 타이밍에 실행되리라 보장해주지 않습니다.

object 참조가 불가능한 시점과 finalizer 또는 cleaner가 실행되는 시점은 차이가 있으며 제멋대로 입니다.

이 뜻은 시간이 중요한 프로그램에서는 절때 쓰지 말라는 것이죠.

예를 들어서 파일을 닫는 작업을 finalizer나 cleaner에게 맡긴다면 많은 폴더들이 open된 채 clsose되지 않아 결국 파일을 열 수 없게 될 것 입니다.

Garbage Collection에 종속적인 시점

finalizer와 cleaner의 실행 시점은 garbage collection의 알고리즘에 달려 있습니다. 이 알고리즘은 구현에 따라 매우 다양해지죠. 따라서 실행시점 또한 프로그램마다 다양해질 수 밖에 없습니다.

그래서 테스트 환경에서는 잘 작동했어도 배포됐을 때 에러날 가능성을 가지게 됩니다.

느린 finalization의 추가적인 영향

느린 finalization은 실전에서도 문제가 됩니다. finalization을 적용한다면 임의적으로 인스턴스 회수가 지연될 수 있습니다.

실제 GUI App에서 이로 인해 OutOfMemoryError가 발생한 적이 있습니다. 프로그램이 죽었을 당시 수천개의 finalizer queue가 실행되길 기다리고 있었습니다.

finalizer thread는 우선순위가 낮기 때문에 이러한 현상이 발생했었죠.

java specification에서 또한 collection이 언제 발생할 지 명시해주지도 않기 때문에 최선의 방법은 결국 사용하지 않는 것입니다.

그렇다면 cleaner는?

class가 cleaner thread를 통제하고 있다는 점에서 좀 더 낫습니다만, cleaner는 여전히 백그라운드에서 GC에 의해 컨트롤 되므로 즉시 cleaning작업할 것은 보장하지 않습니다.

최악의 경우, finalizer와 cleaner는 실행되지 않고 object가 제거 될 수도 있습니다.

System.gc, System.runFinalization의 사용

이 두 메서드를 사용하면 되지 않느냐!라고 말할 수 있습니다. 하지만 이 둘은 실행 가능성을 높혀줄 뿐, 여전히 보장해주진 않습니다. 이 둘은 쓰레기라 deprecated되었습니다.

2. 성능에 심각한 악영향을 주기도 합니다.

try-with-resource로 해제하는 것보다 finalizer가 50배가까이 느리며, cleaners는 5배가까이 느립니다.

3. Finalizer는 보안에 심각한 악영향을 끼치기도 합니다.

constructor 또는 직렬화 과정에서 예외가 발생한다면, 악의적인 finalizer의 서브클래스가 부분 생성된 constructor에서 실행되게 됩니다.

이 finalizer는 object의 static field값을 참조함으로써 GC되는 것을 막습니다.

또한 이렇게 이런 객체가 생성되면 실행되면 안 될 메서드를 실행하는것도 가능하게 되죠.

생성하는 것을 막는 것은 간단한 일이지만, finallizer를 막는 것은 간단한 일이 아니죠. 따라서 아예 finalizer의 서브클래스를 생성할 수 없도록 finalizer를 final로 선언해주는 것은 좋은 방법입니다.

할당 해제를 위한 finalizer와 cleaner의 대안책

바로 AutoCloseable을 구현해주고, 다 쓰고나면 close() 메서드를 실행해주면 됩니다. 이는 일반적으로 예외가 발생해도 제대로 종료될 수 있도록 try-with-resources를 사용하죠.

한편으로 인스턴스가 자신이 닫혔는지 추적하는 좋은 방법으로는 객체의 유효성을 필드에 기록해둡니다. clone() 메서드에서 필드를 닫혔다고 기록한 뒤, 다른 메서드가 실행됬을 때 이 필드를 체크합니다. 만약에 닫혀있다고 기록되어 있다면 IllegalStateException을 던지도록 합니다.

finalizer와 cleaner의 사용가능성

1. 안전망

close()를 혹시나, 사용자가 작성하지 않았을 때 안전망(대비책)으로 작성할 수 있습니다. (안쓰는 것보단 나으니까요.)

2. native peer

네이티브 피어란 자바 객체가 네이티브 메서드를 통해 실행시키는 네이티브 객체를 의미합니다.

네이티브 피어는 자바 객체가 아니여서 GC가 알 수 없으므로 자바 객체가 회수될 때 네이티브 객체는 회수 되지 않을 수 있습니다.

따라서 이때 사용할 순 있긴 합니다. 그런데 이것도 네이티브 피어가 심각한 자원을 차지하고 있을 때 적용가능하빈다.

즉시 회수하고 싶다면 close()를 이용하세요.

Cleaner는 사용하기에 좀 까다롭습니다.

₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩

₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩

미작성

₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩₩