Garbage Collection

Created	@Jan 27, 2021 10:22 PM
Property	
Property 1	
Tags	

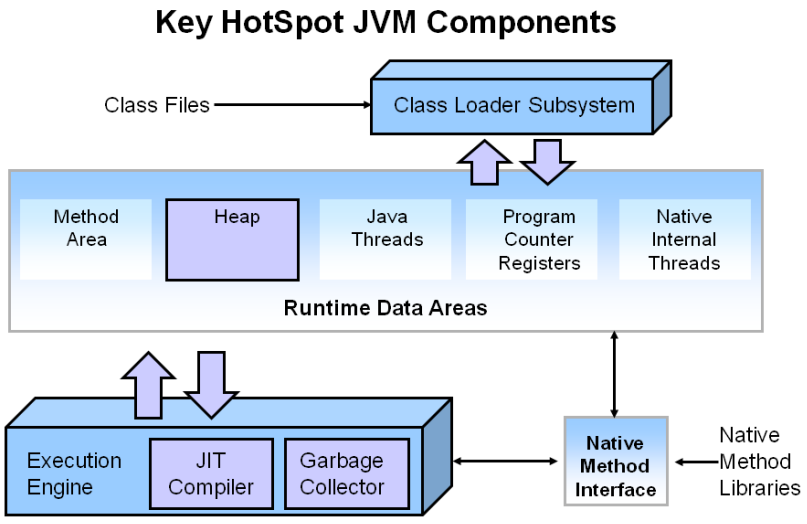
~~GC그거 그거 그냥 free나 delete를 프로그래머가 안쓰고 메모리 해제해주는거 아닌가요?(맞는데 그거 누가 모르냐고)~~

Garbage Collection(GC)

Summary(?)

- 크게 GC를 Stop-the-world와 Mark & Sweep이라고 기억하자.
  - Stop-the-world : GC 작업이 진행되는 동안 GC가 일어나는 쓰레드를 제외한 다른 쓰레드가 stop하는 것을 의미
  - Mark & Sweep : 자바 메모리 영역 중 stack의 오브젝트에서 heap에 도달가능한 오브젝트를 표시(mark)하고 그 외 도달 불가능한 오브젝트를 청소(sweep)하는 것을 의미

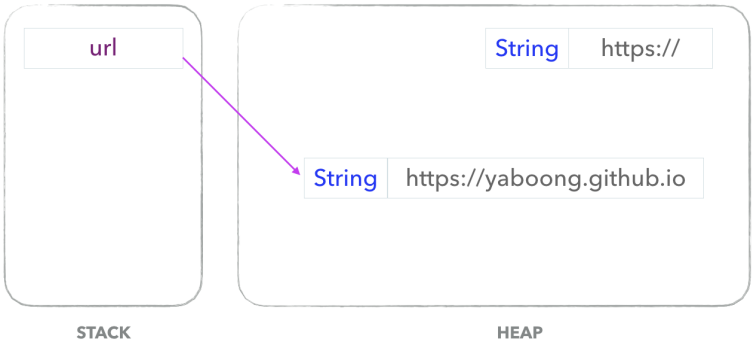
어디서 일어나는 걸까?



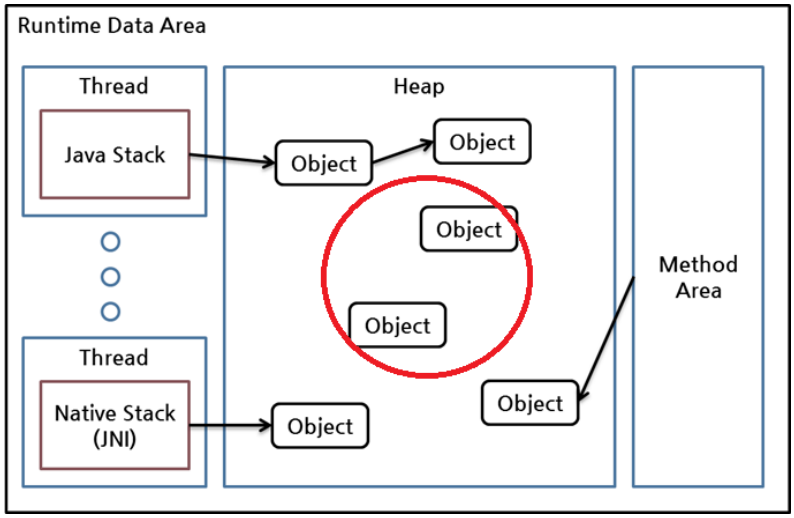
- C나 C++은 OS 단계의 메모리에 직접 접근해 메모리 할당/해제를 하지만 자바는 JVM을 통해 간접적으로 OS 메모리에 접근해 할당된 메모리로 프로그램을 동작한다.
- 거기서 GC는 OS에서 할당받은 JVM의 내부 요소 heap에서 있는 객체 데이터들 중 도달하지 않는 데이터들을 청소, 살아남은 데이터들을 메모리 한곳으로 모아주기도 한다.(compaction)
  → Java 5,6에서 사용되는 **Serial GC**
  (참고 : Serial GC는 데스크톱의 CPU 코어가 하나만 있을 때 사용하기 위해서 만든 방식이다. Serial GC를 사용하면 애플리케이션의 성능이 많이 떨어진다.)
- GC가 발생하는 동안 GC가 일어나는 쓰레드 외 쓰레드들은 멈추게 된다.(**Stop-the-world**)

도달하지 못한 데이터?

```
// 코드로 예시
public class Main {
    public static void main(String[] args) {
        String url = "https://"; // 1
        url += "yaboong.github.io"; // 2
        System.out.println(url);
    }
}
```



- 1번 라인에서 String 객체인 `url` 에 `"https://"` 값이 할당됩니다.
- 2번 라인에서 기존 `url` 이 가지고 있던 `"https://"` 뒤에 합쳐질 것 같지만 `url`은 immutable 객체이기에 새로운 String 형태의 영역이 생성되어 거기에 합쳐진 결과값이 저장된다. 그리고 기존 `https://` 를 가리키던 `url` 이 새로 생성된 영역을 가리켜 기존 영역은 어떻게 해서도 찾아갈 수 없게 된다.



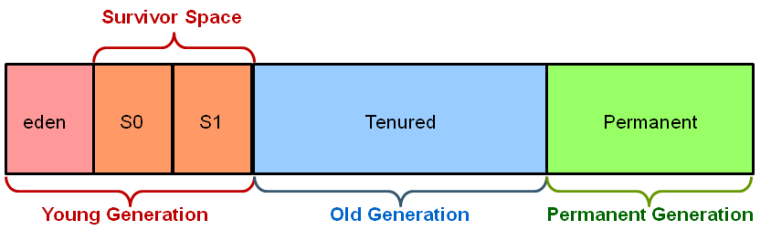
위 구조도(Oracle Hotspot VM 기준)로 추가설명하자면, 붉은 원안에 있는 객체처럼 객체를 지정하는 참고가 없을 경우 GC의 대상이 된다. 여기서 힙에 있는 객체들에 대한 참조는 4가지 종류 중 하나이다.

- 힙 내의 다른 객체에 의한 참조
- Java 메서드 실행 시에 사용하는 지역변수, 파라미터들에 의한 참조
- JNI(java native interface)에 의해 생성된 객체에 대한 참조
- 메서드 영역의 정적 변수에 의한 참조

도달 가능한 객체들도 등급(?)을 지정해 GC 대상으로 선정할 수 있도록 프로그래머가 코드로 개입할 수 있다. (Soft, Weak, PhantomReference 클래스)

JVM안의 메모리(Heap)구조

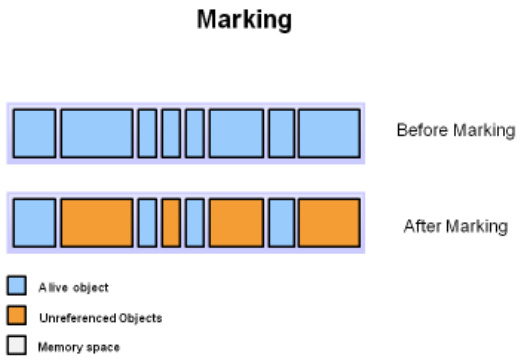
Hotspot Heap Structure



- **Young generation** : 새롭게 생성된 객체 대부분이 여기에 위치한다. 대부분의 객체가 금방 접근 불가능한 상태가 되기 때문에 해당 영역에서 GC가 발생하는데 이를 Minor GC라 한다.
- **Old generation** : Young generation 영역에서 살아남은 객체들이 여기로 복사된다. Young generation 영역보다 크게 할당 되고 GC가 적게 발생하는데 여기서 발생하는 GC를 Major GC라 한다.
- **Permanent generation** : Java application에 필요한 클래스 메타데이터를 저장하는 공간으로 Java8 기준으로 heap이 아닌 native memory로 관리된다.

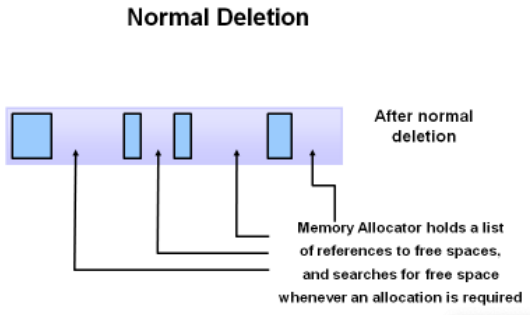
그래서 어떻게 GC는 동작?

Step 1: Marking



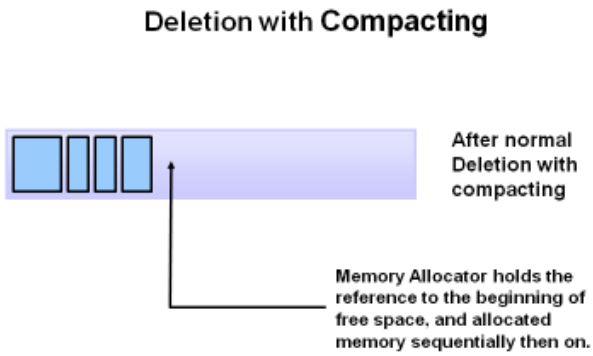
GC는 메모리(heap)에 존재하는 객체들을 확인하고, 그 중 도달하지 못한(참조할 수 없는) 객체가 무엇인지 마킹하는 절차를 진행한다. 여기서 주황색으로 표기된 객체가 도달하지 못한 객체이다.

Step 2: Normal Deletion



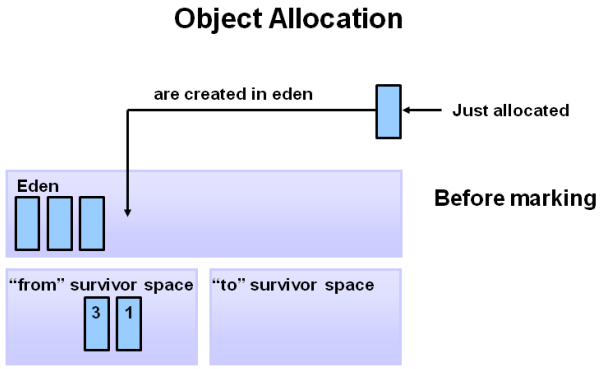
Step1에서 marking되지 않은 객체들(주황색)을 삭제한다.

Step 2a: Deletion with Compacting

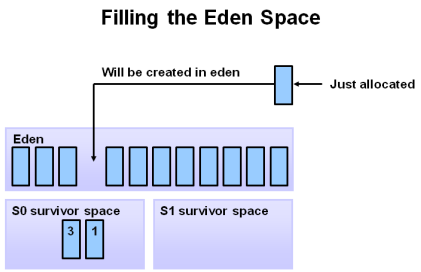


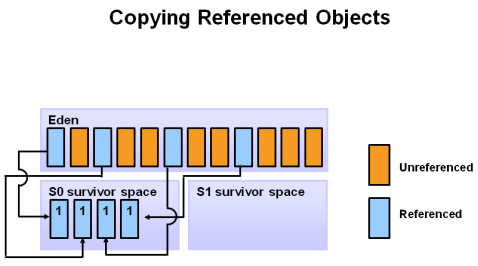
step2 진행하면서 메모리를 더욱 효과적으로 쓰기위해 객체들을 삭제함과 동시에 압축을 진행하기도 한다.

더 세부적으로 어떻게?(Java7 기준)

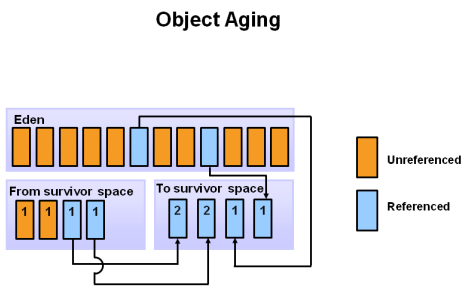


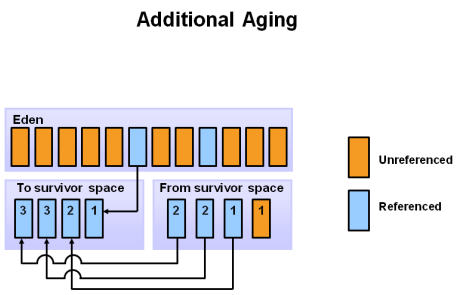
- 새로운 객체 생성시 Eden에 할당되고 suvivor space(from, to / 위 메모리 구조 그림에서는 s0, s1)은 empty 상태로 시작한다.



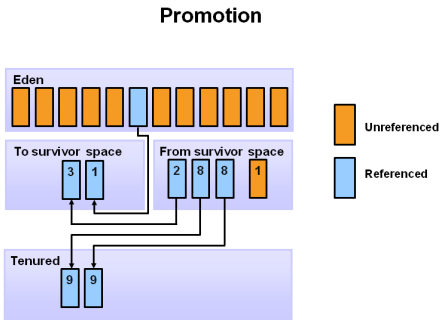


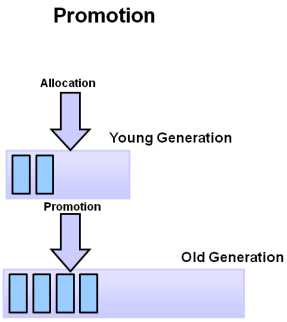
- 객체들을 생성하다가 Eden 영역이 꽉 찼을 경우 Minor GC가 동작. 거기서 도달하지 못한(우측 그림의 주황색) 객체들은 청소되고 남은 객체들은 S0로 이동한다.





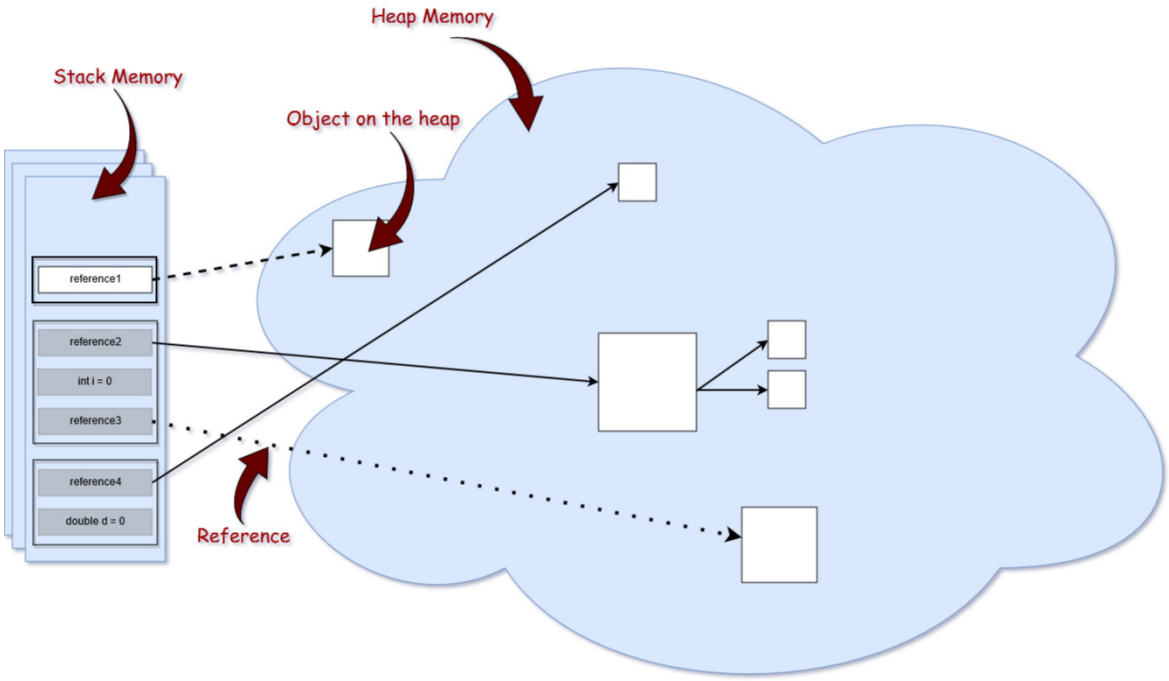
- 다음 Minor GC가 발생할 때 Eden에서 살아남은 객체들은 바로 이전 Minor GC로 살아남은 객체들이 이동한 survivor 영역이 아닌 나머지 영역으로 이동하게 된다.(ex. 이전 결과가 s0으로 갔다면 해당 결과는 s1으로) 그 뿐만 아니라 바로 이전 Minor GC로 살아남은 객체들 또한 Minor GC를 통해 도달하지 못하는 객체를 청소하고 살아남은 객체를 다른 suvivor 영역으로 이동시켜준다. (단. 여기서 survivor 영역간의 이동시 객체에 age가 추가된다)





- Young generation 영역에서 계속 살아남아 특정 값이상의 age값을 갖은 객체들은 Old generation(위 그림의 Tenured 영역)으로 이동하는 이를 Promotion이라 한다.
- Minor GC가 계속 되면서 Old Generation영역도 가득차게 되면 Major GC가 발생하게 된다.

Stack / Heap?



1. **Stack**
    - Heap 영역에 생성된 Object 타입의 데이터의 참조값이 할당된다.
    - 원시타입의 데이터가 값과 함께 할당된다.
    - 지역변수들은 scope 에 따른 visibility 를 가진다.
    - 각 Thread 는 자신만의 stack 을 가진다.
2. **heap**
    - Heap 영역에는 주로 긴 생명주기를 가지는 데이터들이 저장된다. (대부분의 오브젝트는 크기가 크고, 서로 다른 코드블럭에서 공유되는 경우가 많다)
    - 애플리케이션의 모든 메모리 중 stack 에 있는 데이터를 제외한 부분이라고 보면 된다.
    - 모든 Object 타입(Integer, String, ArrayList, ...)은 heap 영역에 생성된다.
    - 몇개의 스레드가 존재하든 상관없이 단 하나의 heap 영역만 존재한다.
    - Heap 영역에 있는 오브젝트들을 가리키는 레퍼런스 변수가 stack 에 올라가게 된다.

☠️ system.gc()?

- Java는 코드상에서 메모리를 명시적으로 지정해 해제하지는 않는다.
- But, 명시적으로 해제하려고 객체를 null로 지정하던가 system.gc()를 호출하려고 하는데 system.gc()는 시스템 성능에 큰 영향을 미친다.

- system.gc()를 타고 내려가보면 native함수로 선언되어 있어 java 외 언어(아마도 C이지 않을까)로 개발되었기에 불러와서 실행하는데 시간이 오래 걸려서 시스템 성능을 저하할 것으로 추측된다.

☠️ GC의 위험성

- 어떤 메모리를 해제할지 결정하는 데 **비용**이 든다.
- 객체가 필요 없어지는 시점을 프로그래머가 미리 알고 있는 경우에도 GC 알고리즘이 메모리 해제 시점을 추적해야 하므로 이 작업은 **오버헤드**가 발생한다.
- GC가 일어나는 **타이밍**이나 **점유 시간**을 예측하기 어렵다.

  → 즉 할당된 메모리가 해제되는 시점을 알 수 없다. 그러므로 프로그램이 **예측 불가능**하게 일시적으로 **정지**할 수 있다. 이런 특성은 특히 **실시간 시스템**에는 적합하지 않다.

❓ 궁금한 점

- 정확하게 왜 모든 쓰레드를 멈추는가?
  → GC과정에서 새로운 오브젝트 할당/생성을 막기 위해서인가?
- JIT compiler도 성능에 영향을 준다고 하던데 어떤 식으로 관여하는가?
- Soft, Weak, Phantom Reference 클래스는 어디서 쓰는건가?
  → 도달할 수 있는 객체들 중 등급을 나눠서 GC 대상 판별를 코드로 작성할 수 있다.
- 대부분의 GC는 포인터 추적방식을 사용한다? → 어떤 포인터 추적 방식들이 있는가

⏭️ 다음에 하면 좋은 주제 / 보충하고 싶은 내용

- 채팅 프로그램 제작
- GC 대상으로 어떻게 지정할 수 있나?(Soft, Weak, Phantom Reference 클래스 사용방법?)
- GC의 종류와 성능 비교

※ 참고 자료

- https://yaboong.github.io/java/2018/06/09/java-garbage-collection/
- https://d2.naver.com/helloworld/1329
- https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html
- https://johngrib.github.io/wiki/jvm-memory/
- https://jeong-pro.tistory.com/148

- https://blog.sessionstack.com/how-javascript-works-memory-management-how-to-handle-4-common-memory-leaks-3f28b94cfbec
- https://ko.wikipedia.org/wiki/쓰레기_수집_(컴퓨터_과학)
- https://d2.naver.com/helloworld/329631