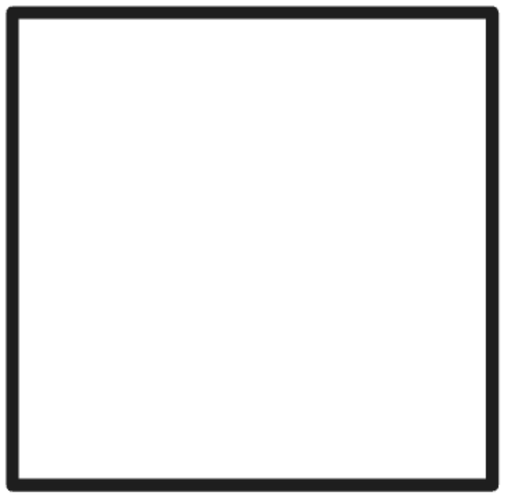
아이템 7. 다 쓴 객체 참조를 해제하라.

강한 참조 방식에서 —> 약한 참조 방식으로

2025.01.15

new 다람쥐();

new 다람쥐();



Heap

new 다람쥐();

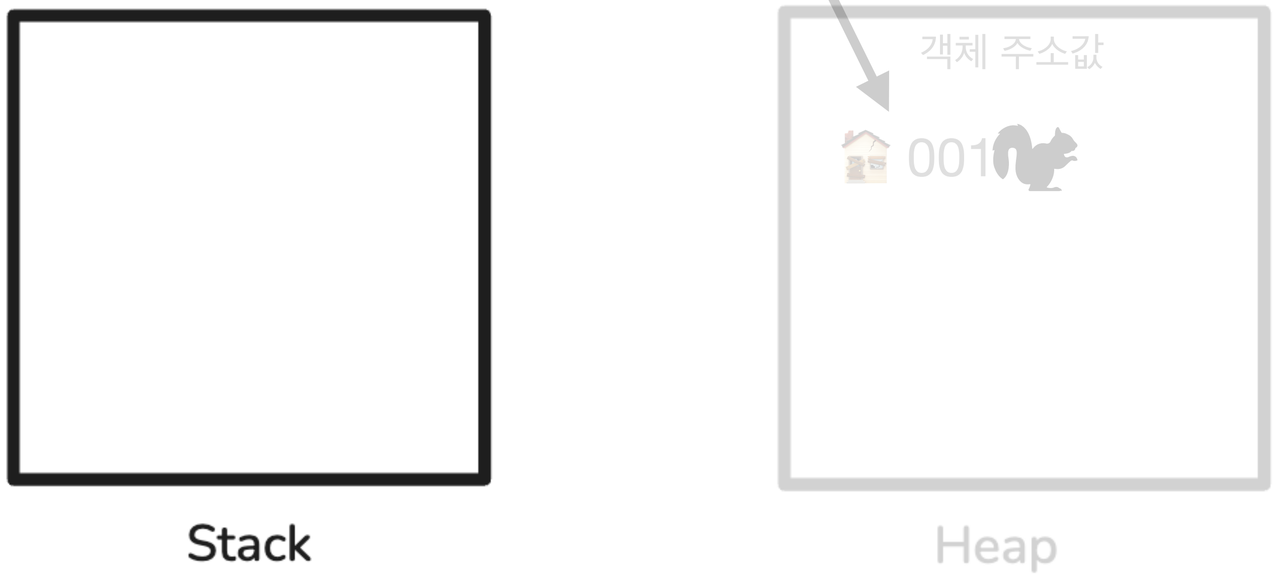
new 다람쥐();

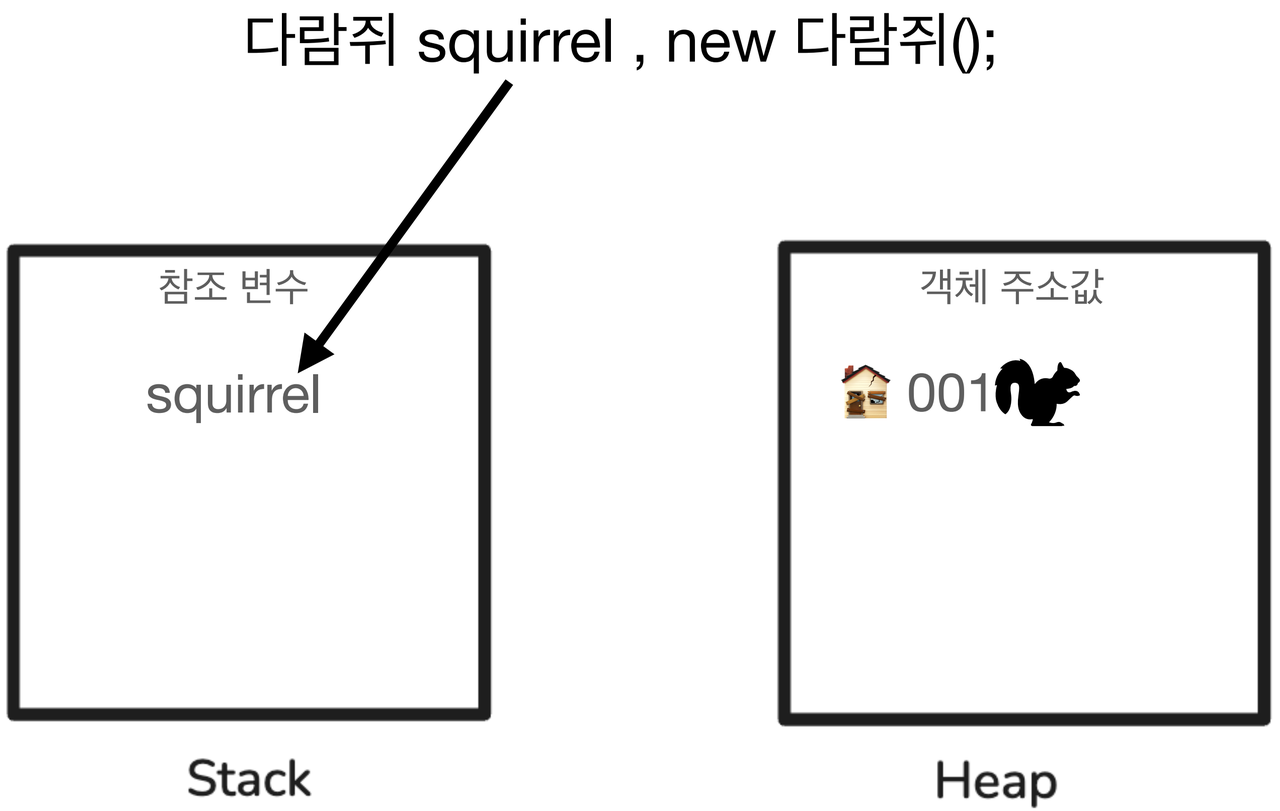
Heap

- JVM이 관리하는 프로그램 상에서 데이터를 저장하기 위해 런타임 시 동적으로 할당하여 사용하는 영역
- 참조형(Reference Type) 데이터 타입을 갖는 객체(인스턴스), 배열 등이 저장 되는 공간

다람쥐 squirrel , new 다람쥐();

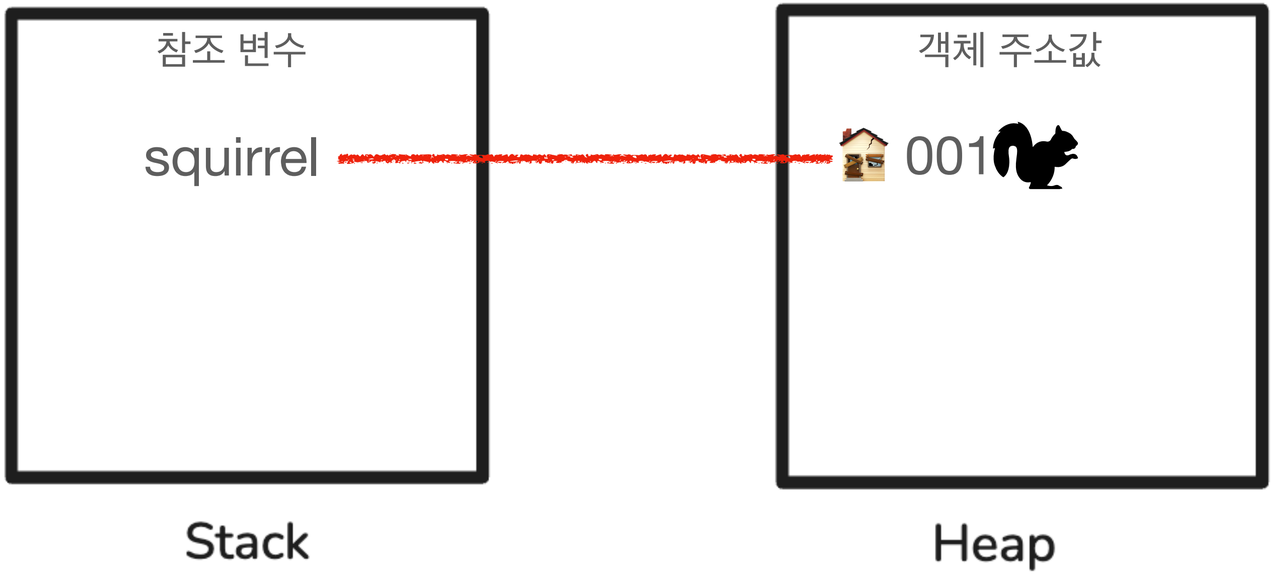
다람쥐 squirrel , new 다람쥐();



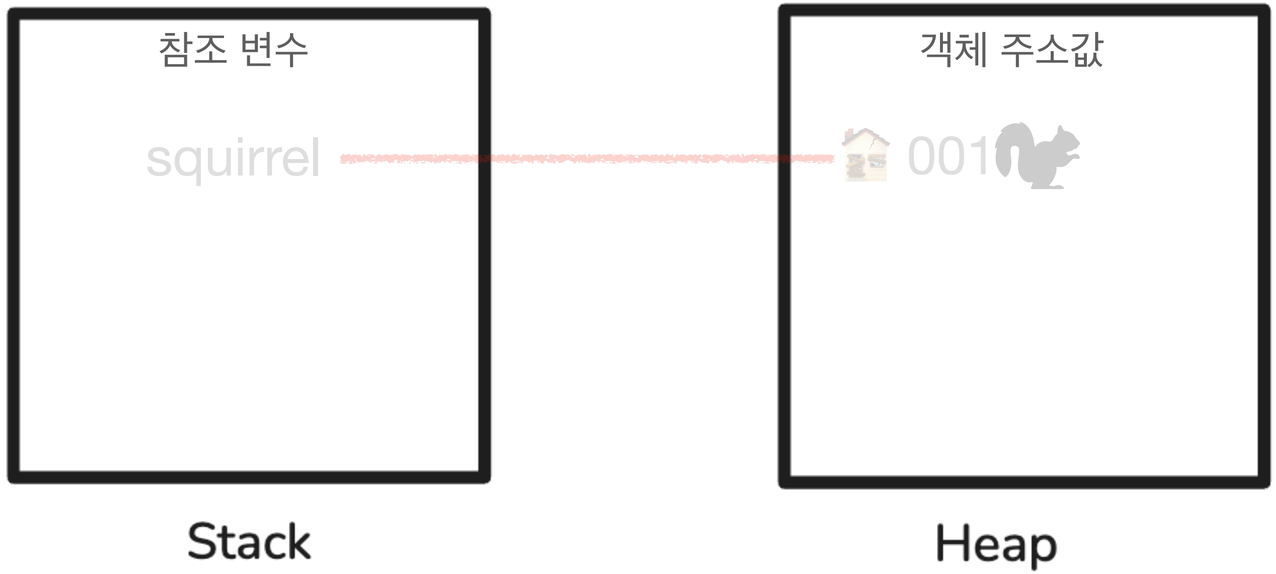

다람쥐 squirrel , new 다람쥐();
참조 변수
squirrel
객체 주소값
001
Stack
Heap

다람쥐 squirrel = new 다람쥐();

(객체를 할당할 때는 = 을 사용해요)

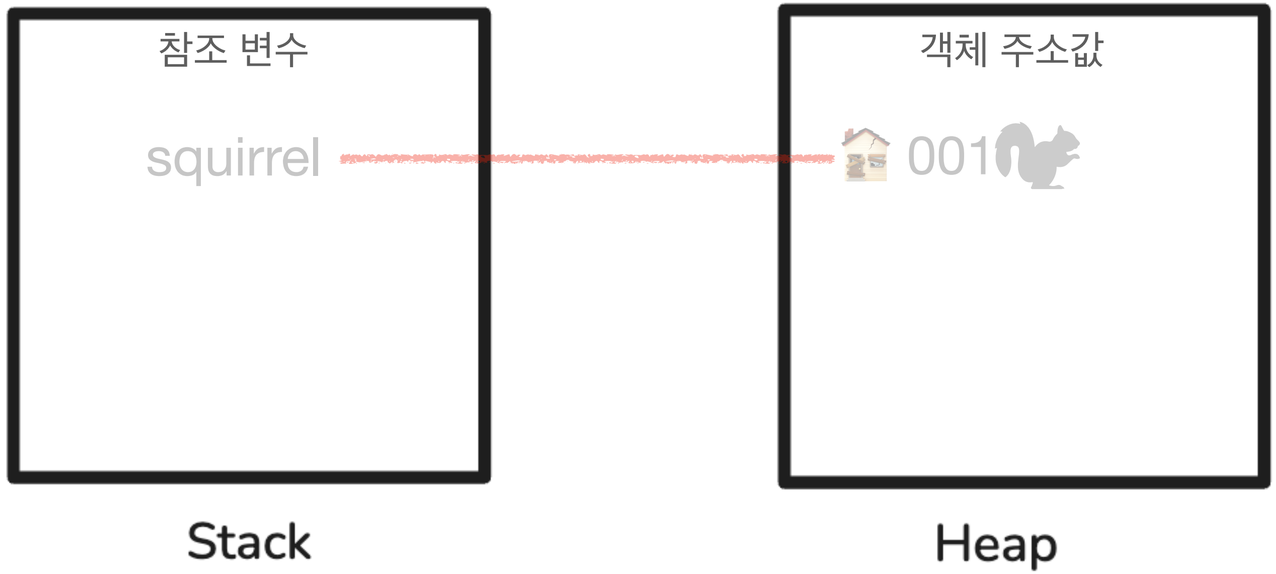


다람쥐 squirrel = new 다람쥐();



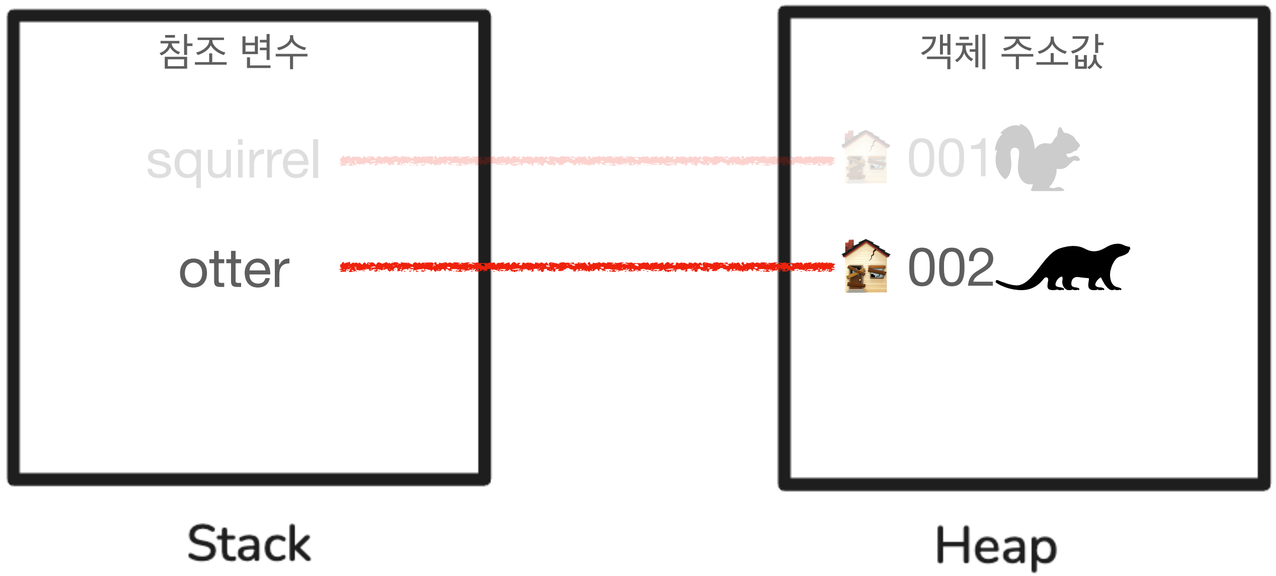
다람쥐 squirrel = new 다람쥐();

수달 otter = new 수달();



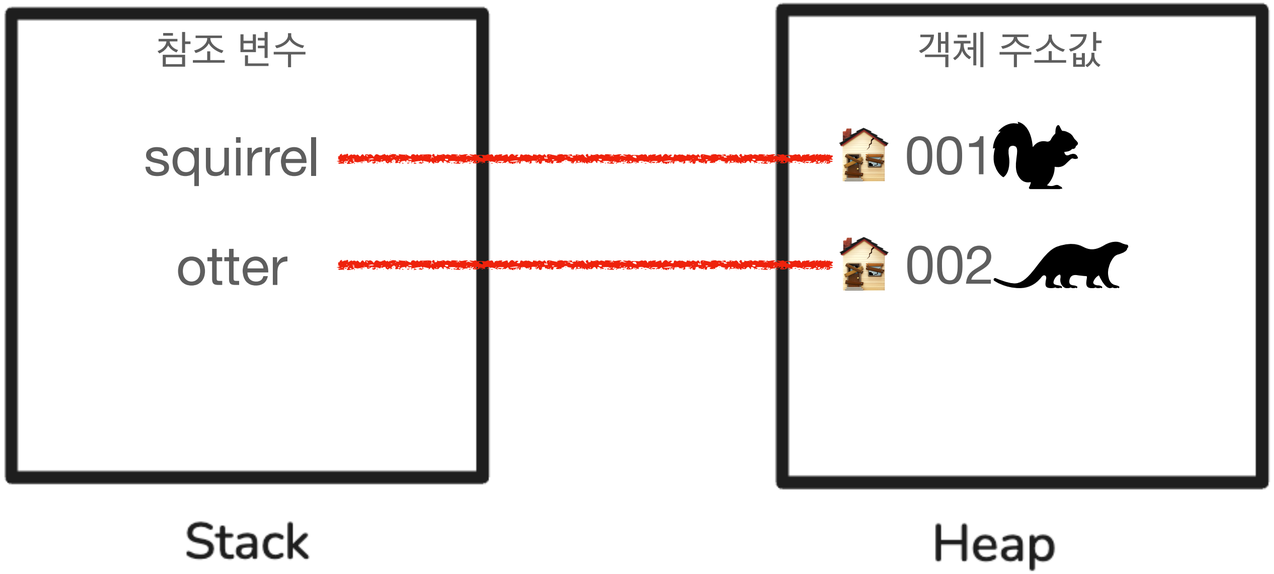
다람쥐 squirrel = new 다람쥐();

수달 otter = new 수달();



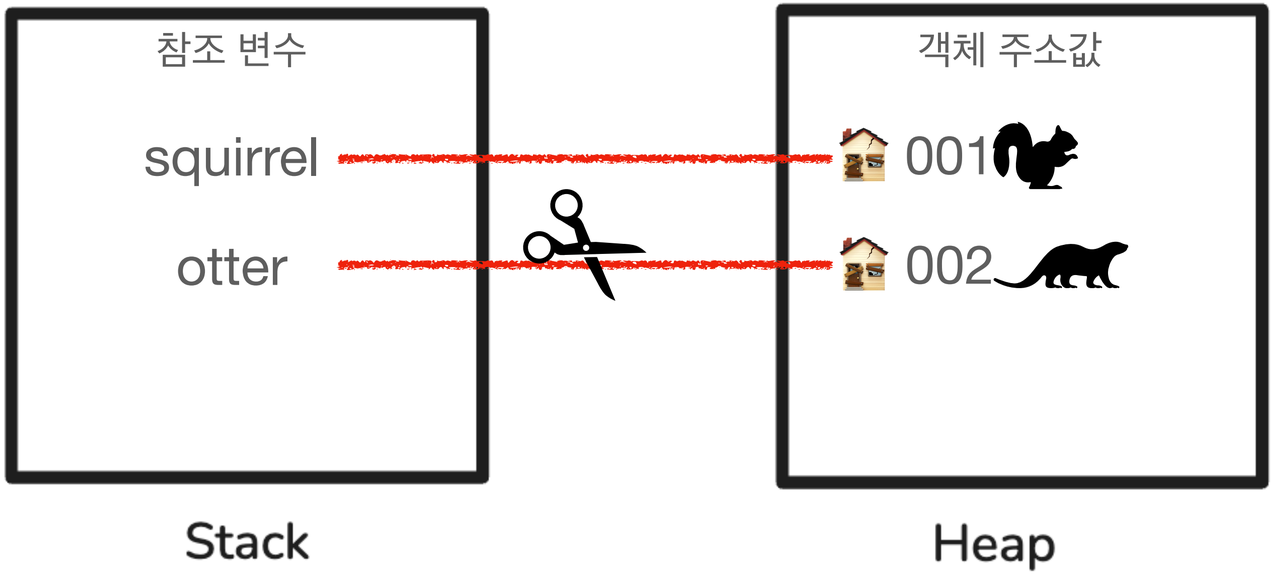
다람쥐 squirrel = new 다람쥐();

수달 otter = new 수달();



다람쥐 squirrel = new 다람쥐();

수달 otter = new 수달();



참조변수  인스턴스 주소

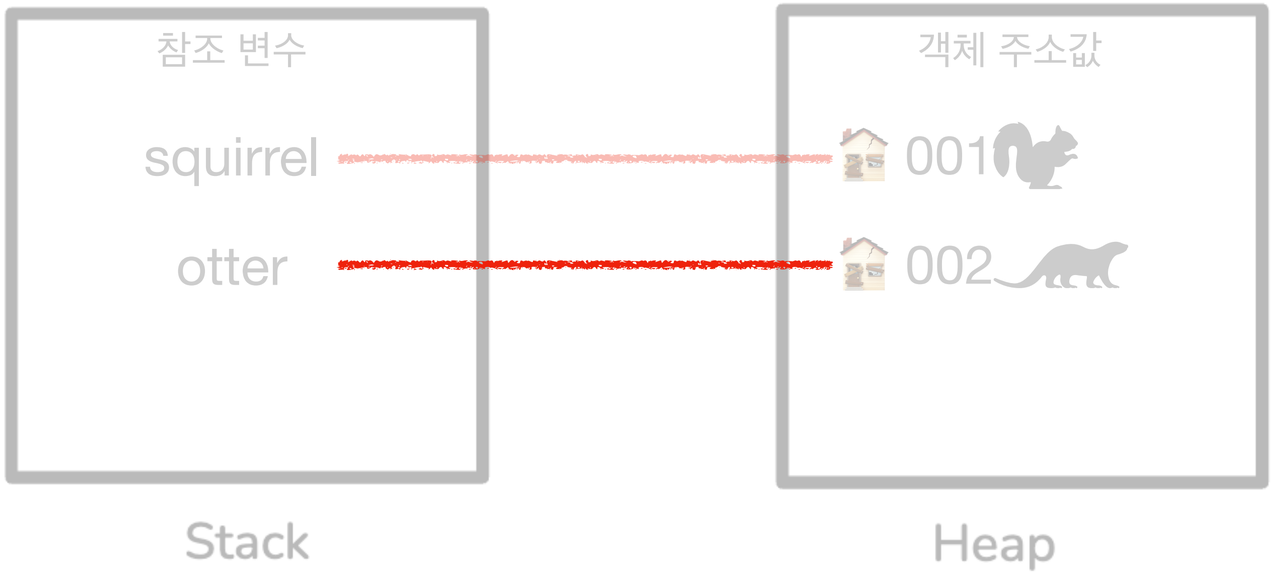
참조변수  인스턴스 주소

참조변수  인스턴스 주소

Null

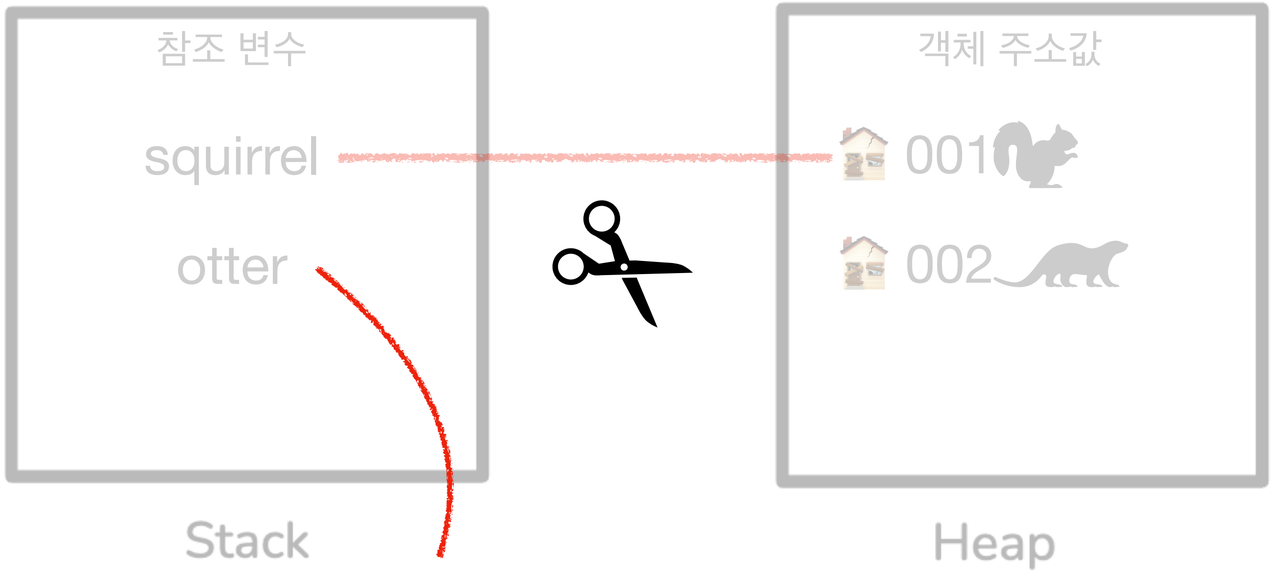
다람쥐 squirrel = new 다람쥐();

수달 otter = new 수달();



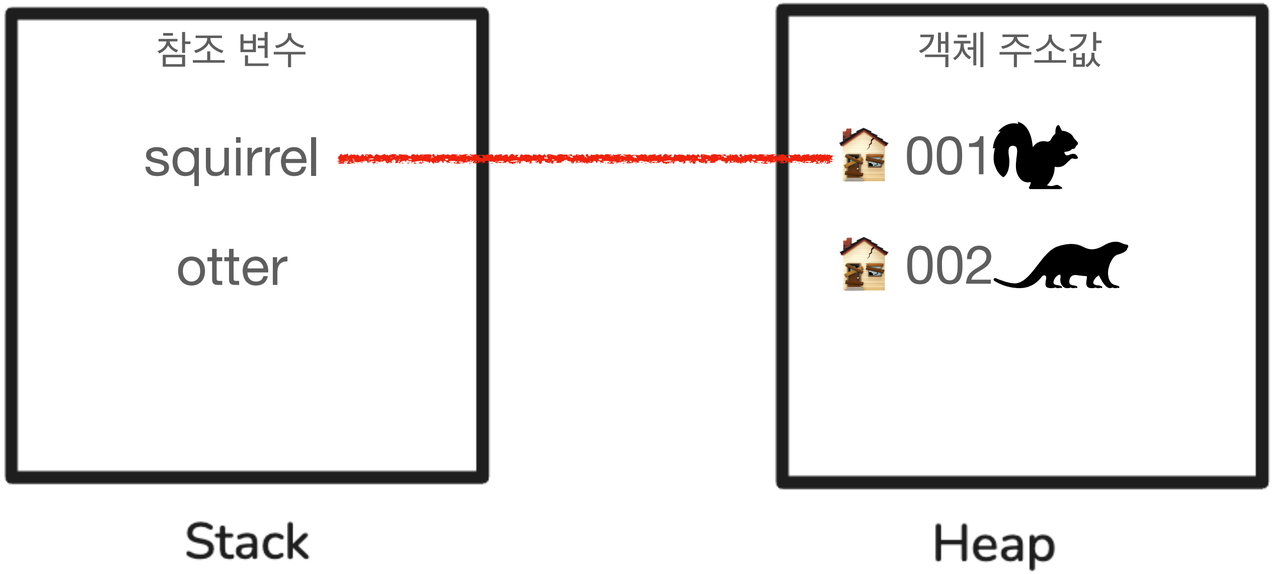
다람쥐 squirrel = new 다람쥐();

수달 otter = null;



다람쥐 squirrel = new 다람쥐();

수달 otter = null;



(휴식) 동물 코딩은 여기까지만~노트북 덮자

(휴식) 동물 코딩은 여기까지만~노트북 덮자

Heap에 남아있는 동물 인스턴스 친구들은? 

출처: 토이스토리4

동물 인스턴스를 청소하러온 GC

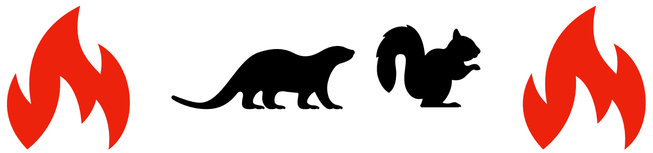
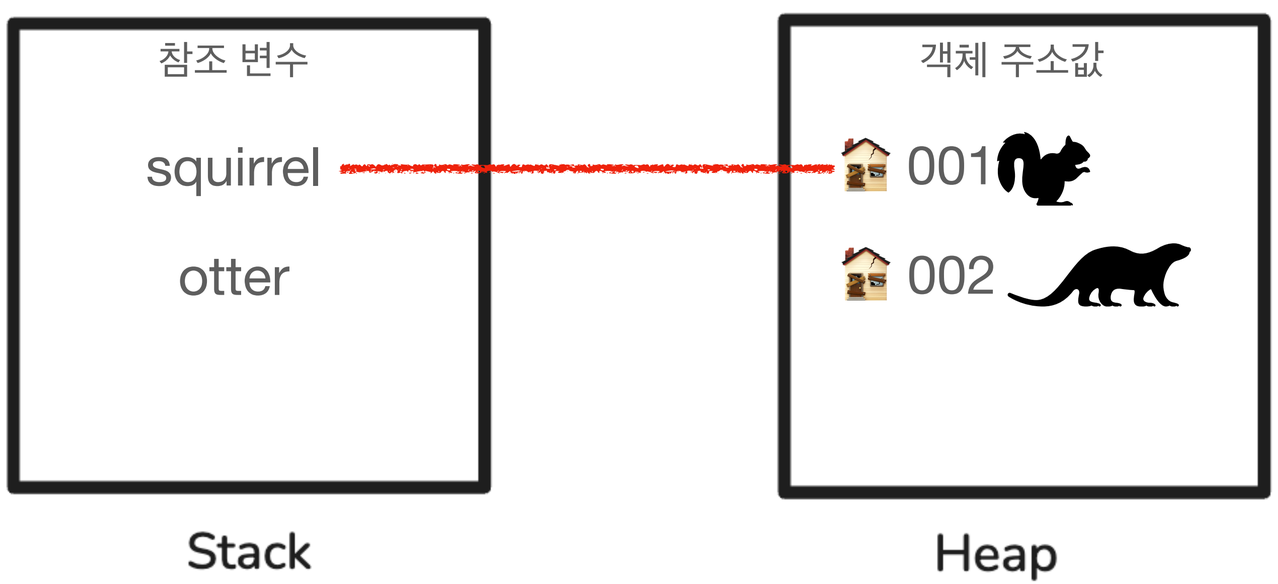


사진 출처 : LG전자

다람쥐 squirrel = new 다람쥐();

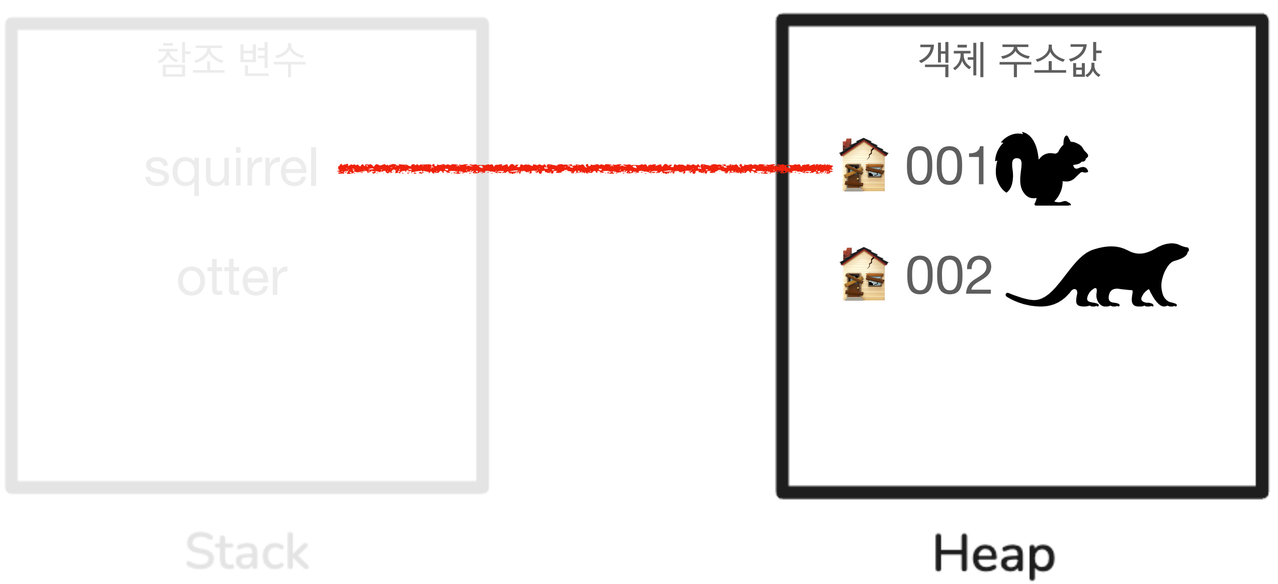
수달 otter = null;



다람쥐 squirrel = new 다람쥐();

수달 otter = null;

< 매번 Heap을 청소하러 오는 GC >
“참조없는 동물친구는 누구지!”



다람쥐 squirrel = new 다람쥐();

수달 otter = null;

< 매번 Heap을 청소하러 오는 GC >
“수달 나와!”

다람쥐 squirrel = new 다람쥐();

수달 otter = null;

다람쥐 squirrel = new 다람쥐();

수달 otter = null;

다람쥐는 여전히 참조되고 있구나!

다람쥐 squirrel = new 다람쥐();

수달 otter = null;

다람쥐 squirrel = new 다람쥐();

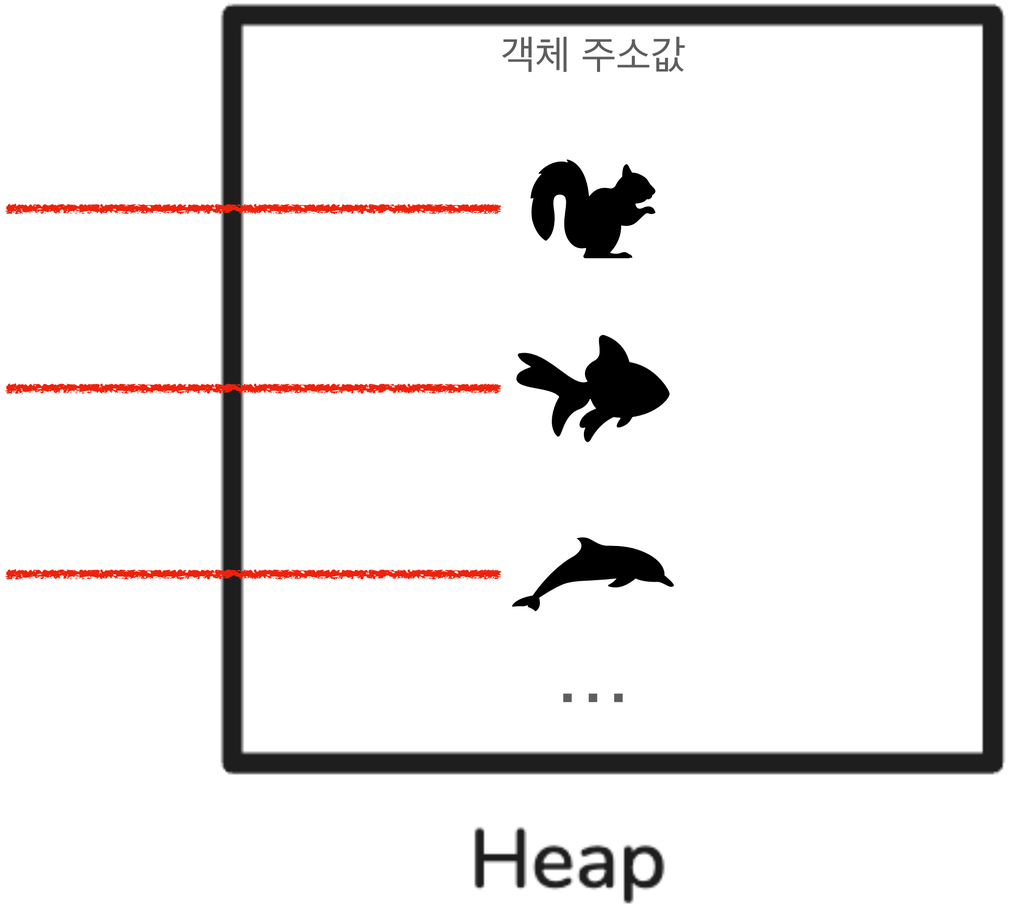
수달 otter = null;

다람쥐는 여전히 참조되고 있구나!
(2트)

다람쥐 squirrel = new 다람쥐();

수달 otter = null;

만약 참조된 친구들이 꽉 차 있다면 힙은 어떻게 되는걸까...



OutOfMemoryError

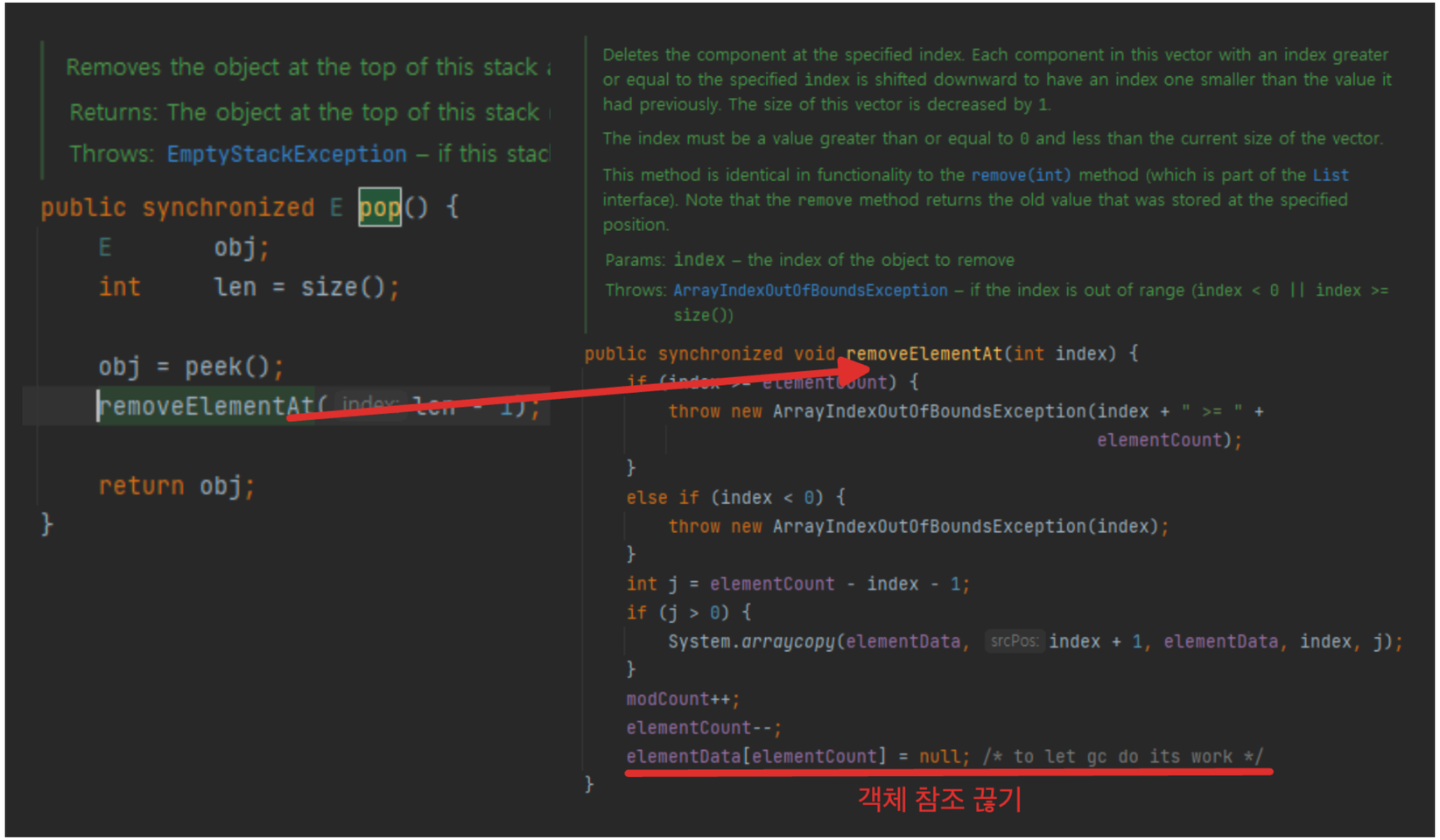
OutOfMemoryError를 예방하는 방법

OutOfMemoryError를 예방하는 방법

사용하지 않는 인스턴스들은
직접 null 할당하기 !

OutOfMemoryError를 예방하는 방법

사용하지 않는 인스턴스들은
직접 null 할당하기 !



Stack.class의 pop() 메서드

자바에서 ArrayList와 같은 Collection들은 모두 요소 제거 작업을 할 때, 명시적으로 null 처리를 해 주며 객체 참조에 대한 해제를 하고 있다.

출처 : https://olrlobt.tistory.com/74

GC가 알아서 처리하도록
GC에 코드를 개입시켜볼까?


Java™
1.0


Java™
1.2

JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

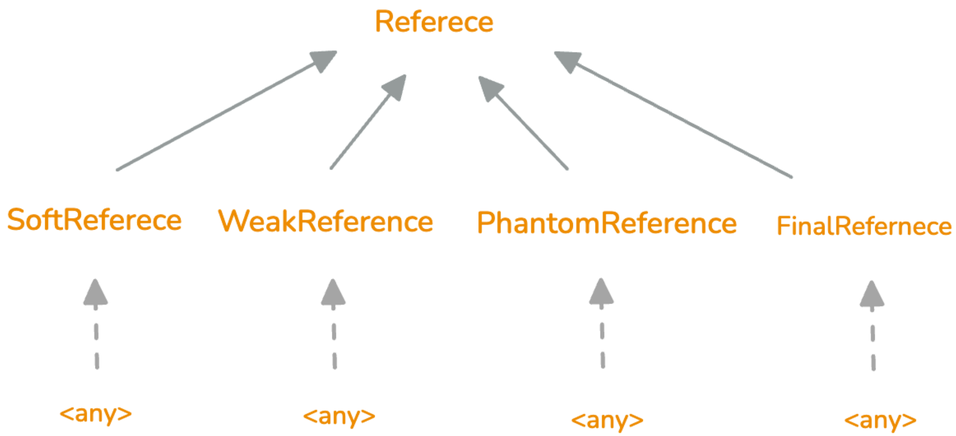
java.lang.ref 패키지 등장

```
package java.lang.ref;

import ...

Abstract base class for reference objects. This class defines the operations common to all reference objects. Because reference
objects are implemented in close cooperation with the garbage collector, this class may not be subclassed directly.
Since:          1.2
Author:         Mark Reinhold
Type parameters: <T> – the type of the referent
sealedGraph

public abstract sealed class Reference<T>
    permits PhantomReference, SoftReference, WeakReference, FinalReference {
```



Reference<T> 클래스는 자바에서 '참조 객체(Reference Object)'를 나타내는 추상 클래스입니다.
이때 "sealed class" 키워드를 통해 상속을 제한하고 있으며,
permits 구문에 명시된 PhantomReference, SoftReference, WeakReference, FinalReference 네 클래스만이 Reference<T>를 상속(extends)할 수 있습니다.

참조 개체는 GC와 긴밀하게 협력하여 구현되므로, 이클래스를 직접 하위 클래스화할 수 없습니다.

(참고 : https://download.java.net/java/early_access/jdk24/docs/api/java.base/java/lang/ref/Reference.html

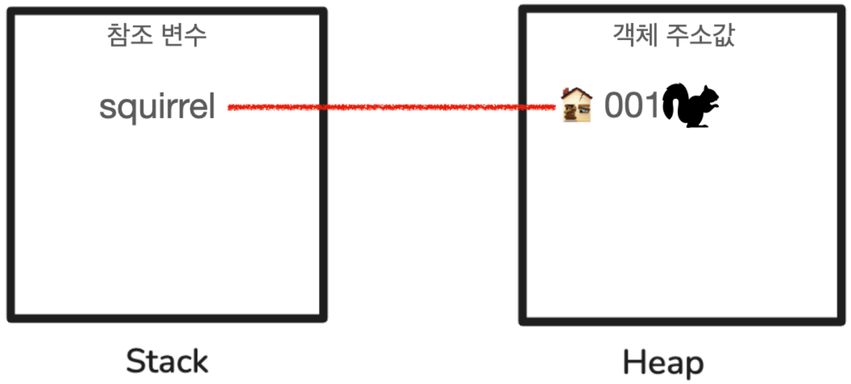
JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

객체를 할당하는 = 방식은 흔하게 쓰는 일반 참조 방식이고 Strong Reference(강한참조)라고 일컫습니다.

다람쥐 squirrel = new 다람쥐();

(객체를 할당할 때는 = 을 사용해요)



JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

객체를 할당하는 = 방식은 흔하게 쓰는 일반 참조 방식이고 Strong Reference(강한참조)라고 일컫습니다.

다람쥐 squirrel = new 다람쥐();

다람쥐는 여전히 참조되고 있구나!

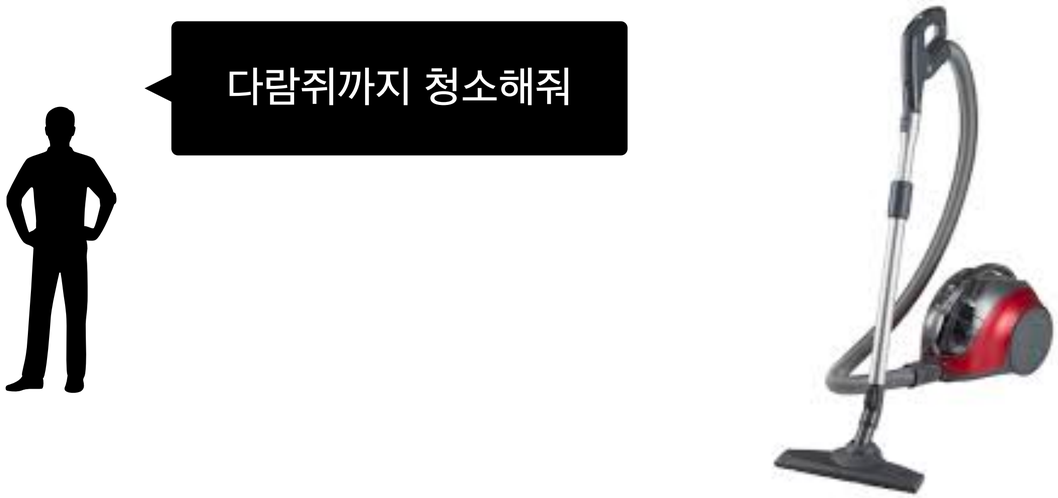
JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

객체를 할당하는 = 방식은 흔하게 쓰는 일반 참조 방식이고 Strong Reference(강한참조)라고 일컫습니다.

다람쥐 squirrel = new 다람쥐();

다람쥐는 여전히 참조되고 있구나!



JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

객체를 할당하는 = 방식은 흔하게 쓰는 일반 참조 방식이고 Strong Reference(강한참조)라고 일컫습니다.

다람쥐 squirrel = new 다람쥐();

다람쥐는 여전히 참조되고 있구나!

JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

객체를 할당하는 = 방식은 개발자들이 흔하게 사용하는 방식이고 Strong Reference(강한참조)라고 일컫습니다.

GC의 대상이라는 표시를 남기는 StrongReference,WeakReference,PhantomReference 방식은
메모리 누수 방지를 위해 특별히 사용하는 방식이고 WeakReference(약한참조)라고 일컫습니다.

JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

객체를 할당하는 = 방식은 개발자들이 흔하게 사용하는 방식이고 Strong Reference(강한참조)라고 일컫습니다.

GC의 대상이라는 표시를 남기는 StrongReference,WeakReference,PhantomReference 방식은
메모리 누수 방지를 위해 특별히 사용하는 방식이고 WeakReference(약한참조)라고 일컫습니다.

다람쥐 squirrel = new 다람쥐();

JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

객체를 할당하는 = 방식은 개발자들이 흔하게 사용하는 방식이고 Strong Reference(강한참조)라고 일컫습니다.

GC의 대상이라는 표시를 남기는 StrongReference,WeakReference,PhantomReference 방식은
메모리 누수 방지를 위해 특별히 사용하는 방식이고 WeakReference(약한참조)라고 일컫습니다.

다람쥐 squirrel = new 다람쥐();

SoftReference<다람쥐> softRef = new SoftReference<>(squirrel);

다람쥐는 특별히 약하게 참조되고 있구나!

JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

객체를 할당하는 = 방식은 개발자들이 흔하게 사용하는 방식이고 Strong Reference(강한참조)라고 일컫습니다.

GC의 대상이라는 표시를 남기는 StrongReference,WeakReference,PhantomReference 방식은
메모리 누수 방지를 위해 특별히 사용하는 방식이고 WeakReference(약한참조)라고 일컫습니다.

다람쥐 squirrel = new 다람쥐();

SoftReference<다람쥐> softRef = new SoftReference<>(squirrel);

WeakReference<다람쥐> weakRef = new WeakReference<>(squirrel);

다람쥐는 특별히 약하게 참조되고 있구나!

JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

객체를 할당하는 = 방식은 개발자들이 흔하게 사용하는 방식이고 Strong Reference(강한참조)라고 일컫습니다.

GC의 대상이라는 표시를 남기는 StrongReference,WeakReference,PhantomReference 방식은
메모리 누수 방지를 위해 특별히 사용하는 방식이고 WeakReference(약한참조)라고 일컫습니다.

```
다람쥐 squirrel = new 다람쥐();
SoftReference<다람쥐> softRef = new SoftReference<>(squirrel);
WeakReference<다람쥐> weakRef = new WeakReference<>(squirrel);
```

다람쥐는 특별히 약하게 참조되고 있구나!

그외 팬텀 래퍼런스가 있고,
파이널 래퍼런스가 있는데, 파이널 래퍼런스는 일반개발자가 직접 다루진 않고 jvm 스펙에 가깝습니다.

JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

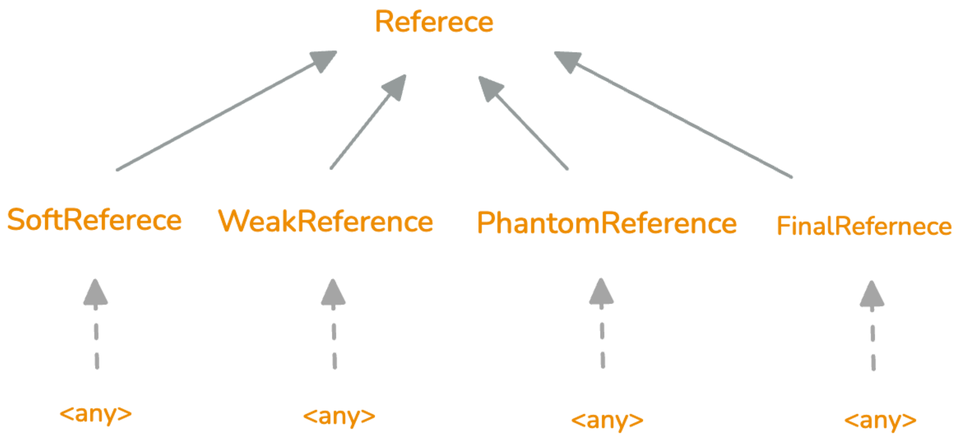
java.lang.ref 패키지 등장

```
package java.lang.ref;

import ...

Abstract base class for reference objects. This class defines the operations common to all reference objects. Because reference
objects are implemented in close cooperation with the garbage collector, this class may not be subclassed directly.
Since:          1.2
Author:         Mark Reinhold
Type parameters: <T> – the type of the referent
sealedGraph

public abstract sealed class Reference<T>
    permits PhantomReference, SoftReference, WeakReference, FinalReference {
```



: Reference<T> 클래스의 구체적인 특징, 용도를 알아볼 시간이 드디어 온 것 같아요.

(참고 : https://download.java.net/java/early_access/jdk24/docs/api/java.base/java/lang/ref/Reference.html

JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

강도가 강한것부터 약한 것까지
내림차순

특징

용도

JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

강도가 강한것부터 약한 것까지 내림차순	특징	용도
SoftReference	메모리 부족하다면 -> GC에 의해 수거됨 메모리 충분하다면 -> 강한 참조 유지로 수거 안됨	캐싱

본문은 ChatGpt o1의 내용을 포함하고 있습니다.

JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

강도가 강한것부터 약한 것까지 내림차순	특징	용도
SoftReference	메모리 부족하다면 -> GC에 의해 수거됨 메모리 충분하다면 -> 강한 참조 유지로 수거 안됨	캐싱
WeakReference	GC가 해당 객체를 참조하는 Strong Reference(강한 참조)가 없을 경우 즉시 GC 대상이 됨	Map의 캐시 엔트리나, 리스너, 콜백 객체 관리

본문은 ChatGpt o1의 내용을 포함하고 있습니다.

JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

강도가 강한것부터 약한 것까지 내림차순	특징	용도
SoftReference	메모리 부족하다면 -> GC에 의해 수거됨 메모리 충분하다면 -> 강한 참조 유지로 수거 안됨	캐싱
WeakReference	GC가 해당 객체를 참조하는 Strong Reference(강한 참조)가 없을 경우 즉시 GC 대상이 됨	Map의 캐시 엔트리나, 리스너, 콜백 객체 관리
PhantomReference	강한 참조가 없어졌더라도, 수집을 바로 안함 객체가 `finalize` 되더라도 메모리에서 완전히 제거되기 전에 참조가 가능 - `finalize()` 호출 후, 실제 메모리 해제 전에 정리 작업을 수행	객체가 GC로 수거된 이후에 어떤 처리를 하고 싶을 때 (예 : 직접적인 celan-up 작업)

참고 : https://shout-to-my-mae.tistory.com/464

본문은 ChatGpt o1의 내용을 포함하고 있습니다.

JDK 1.2부터 개발자가 GC에 제한적으로 관여할 수 있게끔,

java.lang.ref 패키지 등장

강도가 강한것부터 약한 것까지 내림차순	특징	용도
SoftReference	메모리 부족하다면 -> GC에 의해 수거됨 메모리 충분하다면 -> 강한 참조 유지로 수거 안됨	캐싱
WeakReference	GC가 해당 객체를 참조하는 Strong Reference(강한 참조)가 없을 경우 즉시 GC 대상이 됨	Map의 캐시 엔트리나, 리스너, 콜백 객체 관리
PhantomReference	강한 참조가 없어졌더라도, 수집을 바로 안함 객체가 `finalize` 되더라도 메모리에서 완전히 제거되기 전에 참조가 가능 - `finalize()` 호출 후, 실제 메모리 해제 전에 정리 작업을 수행	객체가 GC로 수거된 이후에 어떤 처리를 하고 싶을 때 (예 : 직접적인 celan-up 작업)
FinalReference	JVM 내부적으로 finialize() 관련 처리를 위해 사용되는 참조 형태 일반 개발자가 직접 다루기보다는 JVM 구현 세부 사항에 가까움	

참고 : https://shout-to-my-mae.tistory.com/464

본문은 ChatGpt o1의 내용을 포함하고 있습니다.

아이템 7. 다 쓴 객체 참조를 해제하라.

강한 참조 방식에서 —> 약한 참조 방식으로

끝.

2025.01.15