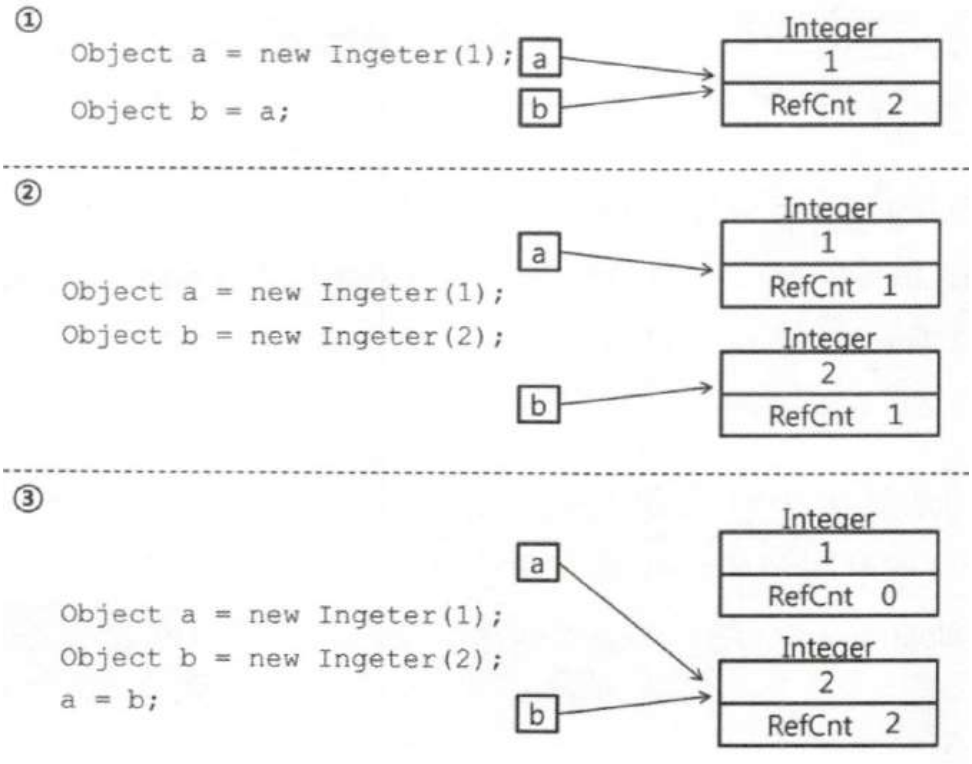
Garbage Collection의 기본 알고리즘

Reference Counting Algorithm

각 Object 마다 Reference Count를 관리하여 Reference Count가 0일 때 Garbage Collection을 수행한다.



장점

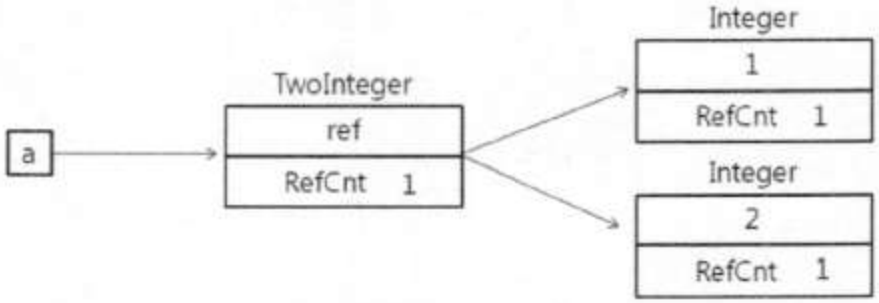
1. Garbage Object를 확인하는 작업이 간단하다.
2. Pause Time이 분산되어 실시간 작업에도 거의 영향을 주지 않는다.

단점

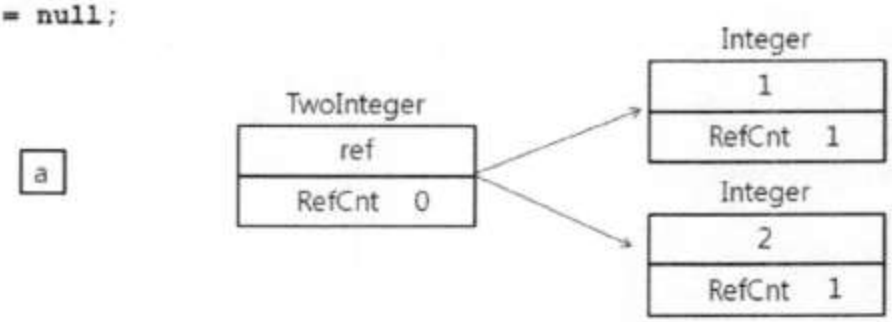
1. Reference의 변경이나 Garbage Collection의 결과에 따라, 각 Object 마다 Reference Count를 변경해 주어야 하기 때문에 관리 비용이 크다.
2. Garbage Collection이 연속적으로 일어날 때 문제가 발생 할 수 있다.

Reference Counting Algorithm의 문제 발생의 예 1.

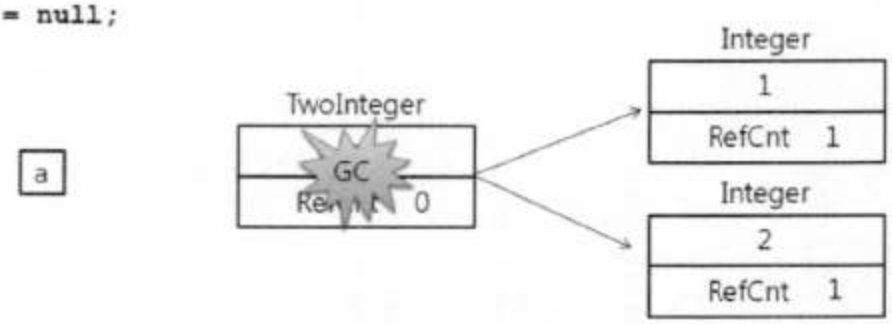
```
Object a = new TwoIngeter(new Integer(1), new Integer(2));
```




```
Object a = new TwoIngeter(new Integer(1), new Integer(2));

a = null;
```

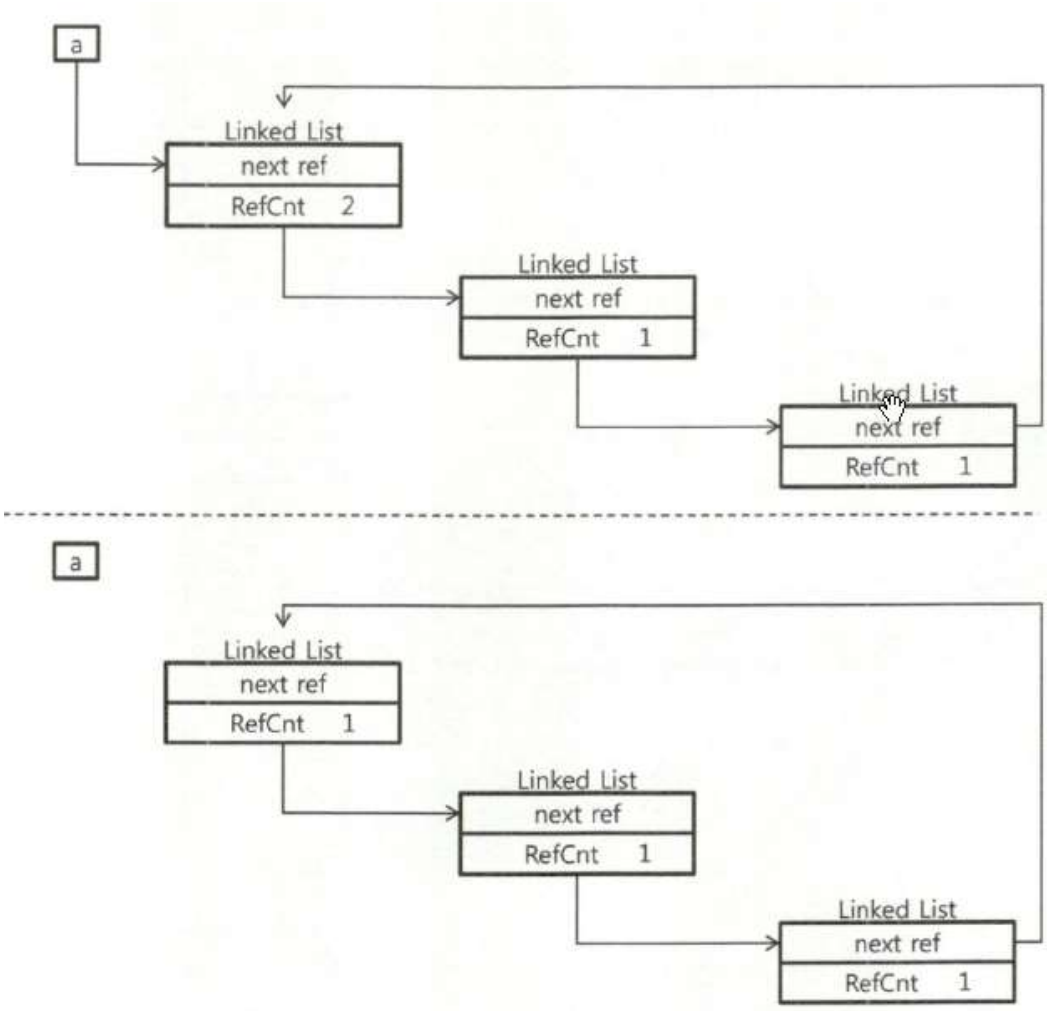



```
Object a = new TwoIngeter(new Integer(1), new Integer(2));

a = null;
```




```
Object a = new TwoIngeter(new Integer(1), new Integer(2));

a = null;
```

Reference Counting Algorithm의 문제 발생의 예 2.



Reference Count가 0이 되어야 Garbage Collection이 수행 되는데 순환 참조 구조에서는 Reference Count가 0이 되지 않을 수 있다. 이 경우 Memory Leak을 유발 할 수 있다.

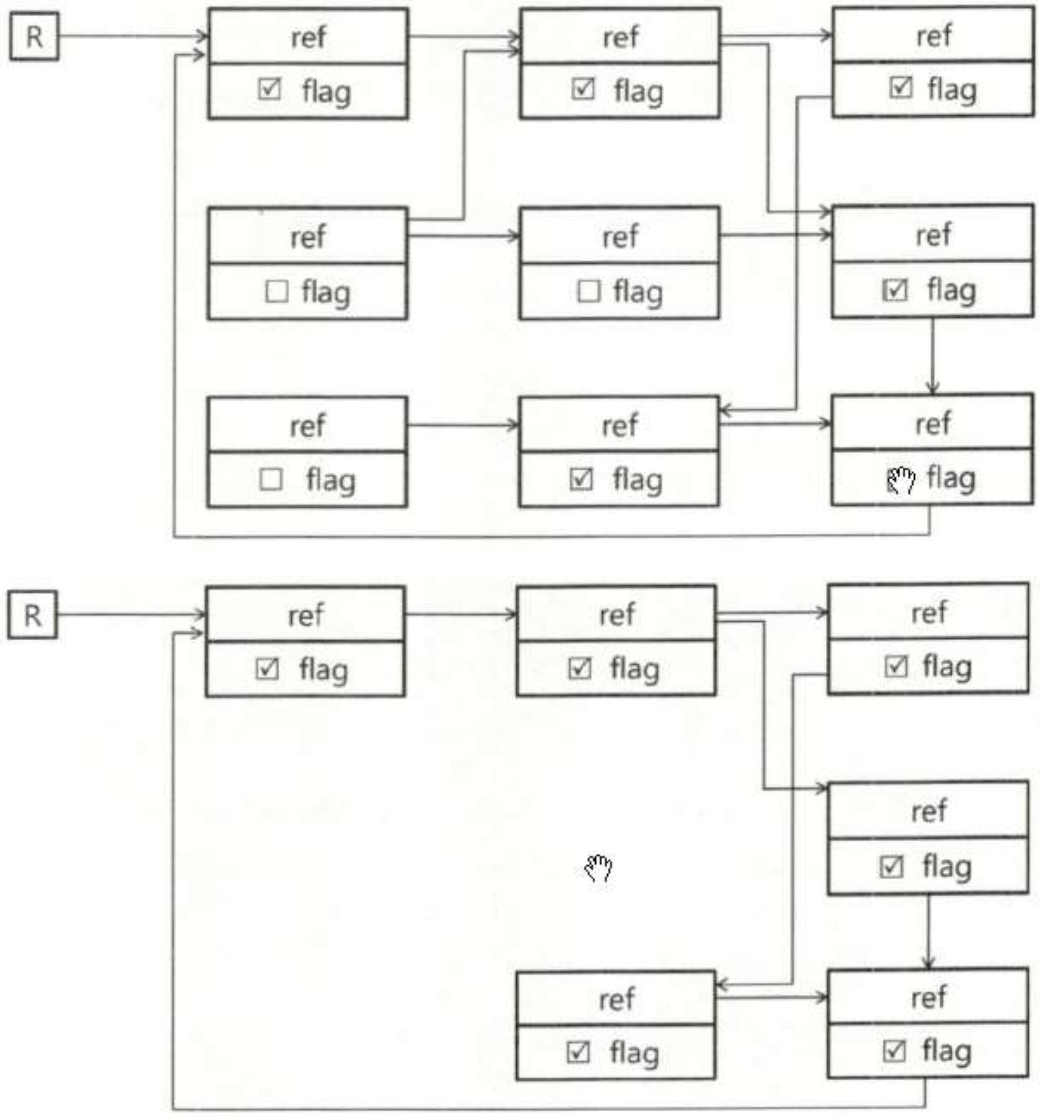
Mark-and-Sweep Algorithm

Tracing Algorithm 이라고도 불리운다.

Reference Counting Algorithm의 단점을 극복하기 위하여 Object 마다 Count를 하는 방식 대신, Root Set 에서 시작하는 Reference의 관계를 추적하는 방식을 사용한다. 이 방식은 Garbage를 찾을 때 효과적이다.

Mark Phase에서 Garbage Object를 구별해 내기 위해 Root Set에서 Reference 관계가 있는 Object에 Marking을 한다.

Sweep Phase에서 Marking이 없는 Object를 삭제한다.



장점

1. Reference 관계가 정확하게 파악된다.
2. Reference 변경시 부가 적업이 없으며 속도가 빠르다.

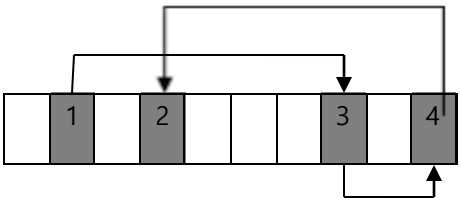
단점

1. Garbage Collection 과정 중에는 Heap의 사용이 제한된다. (Memory Corruption 방지) 이 때 프로그램 Suspend가 발생한다.
2. Fragmentation이 발생 할 수 있다.

Mark-and-Compaction Algorithm

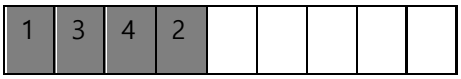
Flagmentation이 발생하는 단점을 보완하기 위한 알고리즘으로 Mark Phase 이후 Compaction Phase를 수행한다. Compaction Phase란 Live Object를 연속된 메모리 공간에 차곡차곡 적재하는 것을 의미하며 적제후 Sweep Phase를 포함한다. 일반적으로 Arbitrary(임의) 방식, Linear 방식, Sliding 방식 등이 있다.

[before Compaction]



[Arbitrary Compaction]

[Linear Compaction]



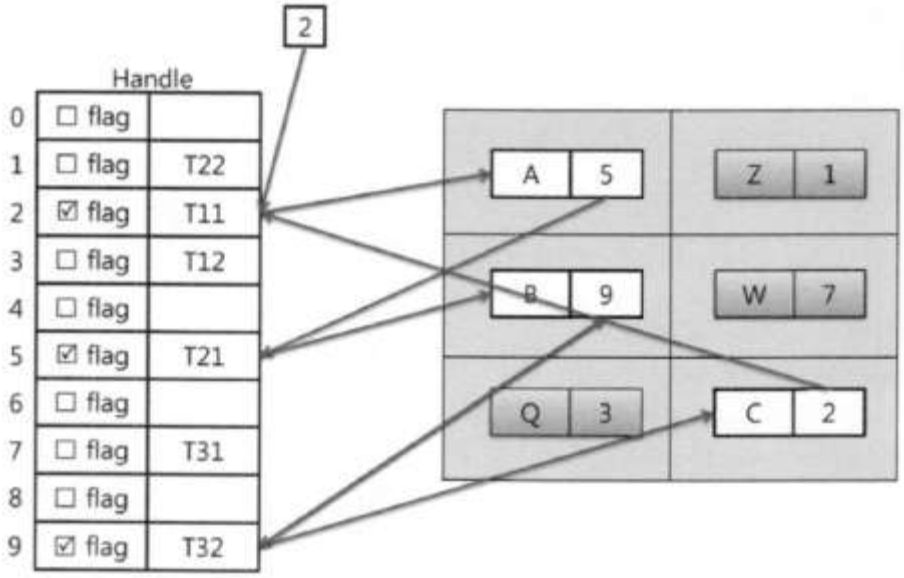
[Sliding Compaction]

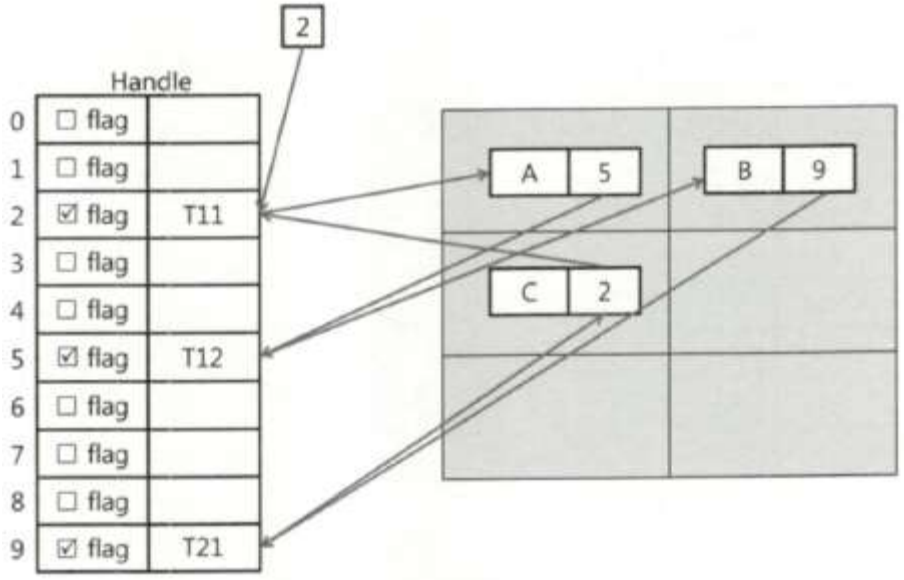
가장 효율적인 방식은 Sliding 방식으로 알려져 있다. Linear 방식은 Reference 순서를 따라가기 위한 Overhead가 발생하기 때문이다. 또한 Object 탐색은 순차적 Access가 아닌 포인터를 기반으로한 Random Access를 수행하기 때문에 인접해 있다고 해서 장점이 되지 못한다.

Mark-and-Compaction Algorithm은 Compaction을 원활하게 하기 위한 Handle과 같은 자료구조를 사용할 수도 있다. Handle에는 객체의 논리적인 주소와 실제 주소가 들어가 있다.

6개의 Heap으로 구성되어 왼쪽 위부터 오른쪽 하단까지 T11,T12,T21,T22,T31,T32의 순서로 주소가 되어 있다고 가정하자. Root Set은 Handle의 2번 주소를 가리키며 A, B, C는 Reference 관계를 갖는다. Q, W, Z도 상호 Reference 관계를 갖지만 Root Set에서 Reference 하지 않고 있다.

Mark Phase 에서는 Live Object인 A, B, C의 Handle에 Marking 하게 된다.





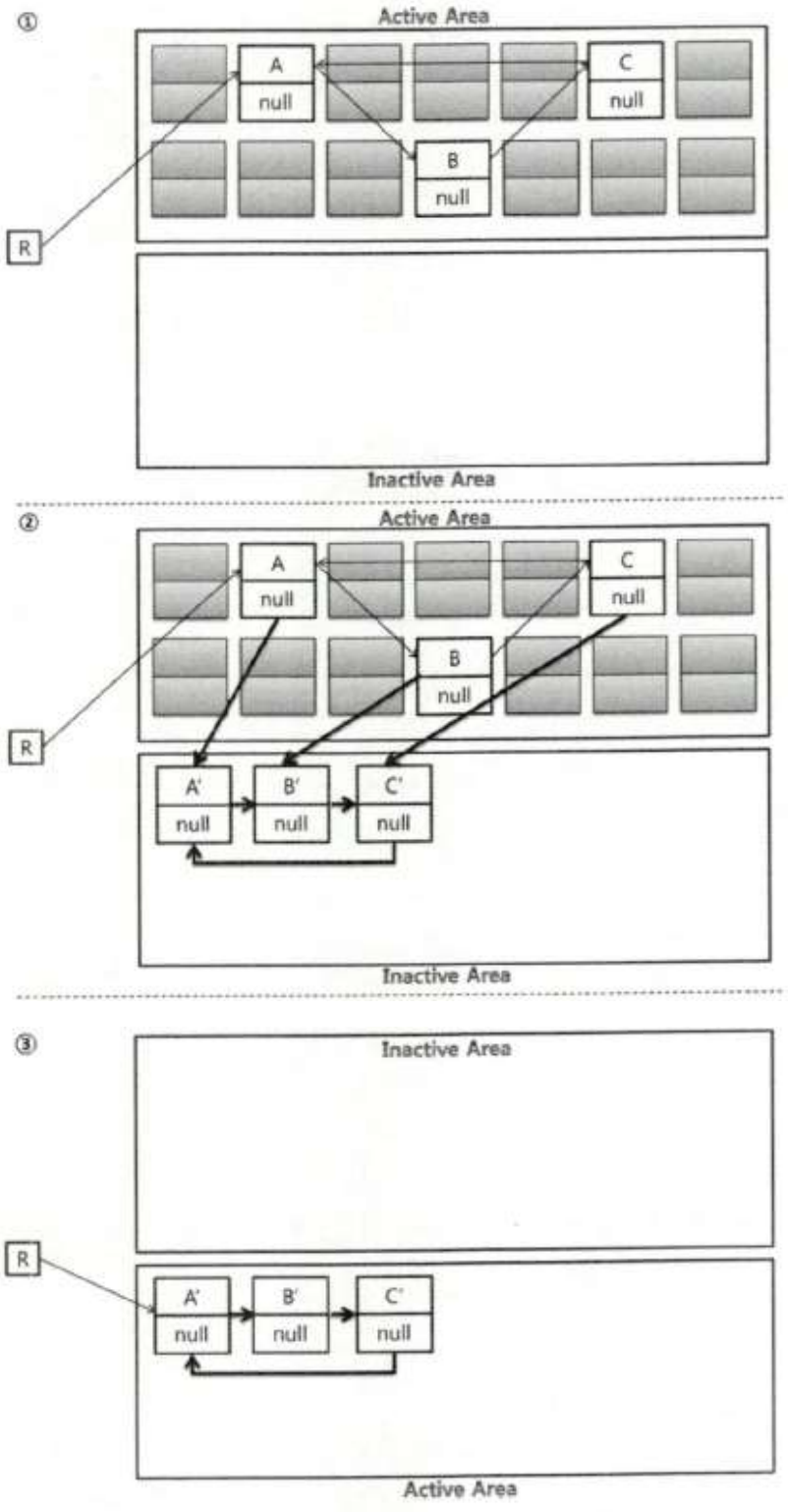
Compaction Phase에서는 Marking 된 정보를 바탕으로 Garbage Object를 Sweep 한 후 Heap의 한 쪽 방향으로 Live Object를 이동 시킨다. 위 그림의 이동하는 방법은 Sliding Compacion의 방법을 사용하였다. 그 이후 이동한 새로운 주소로 모든 Reference를 변경하는 작업을 수행한다.

장점 : Fragmentation의 방지에 중점을 둔 Garbage Collection 기법. 메모리 공간의 효율성이 좋다.

단점 : Compaction 작업 이후 모든 Reference를 업데이트 하는 작업은, 경우에 따라서 모든 Object를 Access 하는 등의 부가적인 Overhead를 수반할 수 있으며 Mark Phase와 Compaction Phase 모두 Suspend 현상이 발생한다.

Copying Algorithm

Fragmentation의 문제를 해결하기 위해 제시된 또 다른 방법으로 Heap을 Active 영역과 Inactive 영역으로 나누어 사용한다. Active 영역에 Object를 할당 하고 Active 영역이 꽉 차게 되어 더 이상 Allocation이 불가능하게 되면 Garbage Collection이 수행된다.



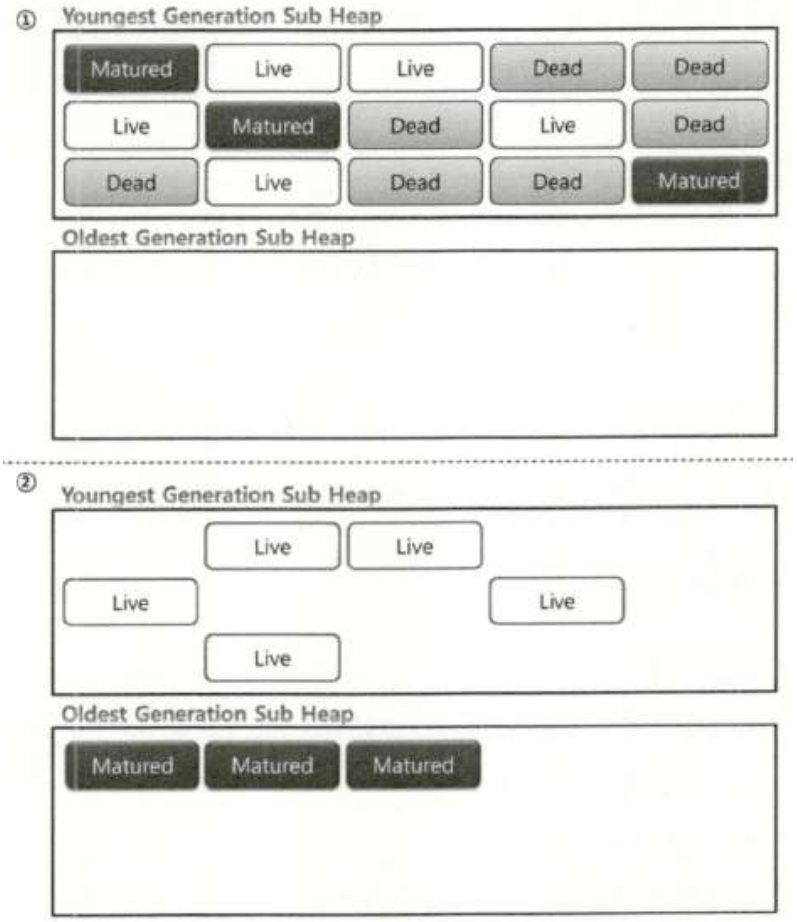
장점 : Fragmentation 방지에 효과적

단점 : 전체 Heap의 절반 정도를 Inactive Area로 사용하므로 활용도가 50%로 떨어지며 Suspend 현상과 Copy에 대한 Overhead가 발생한다.

Generational Algorithm

Hotspot JVM이나 IBM JVM에서 사용하는 대표적인 Heap인 Generational Heap에서 사용하는 Garbage Collection Algorithm.

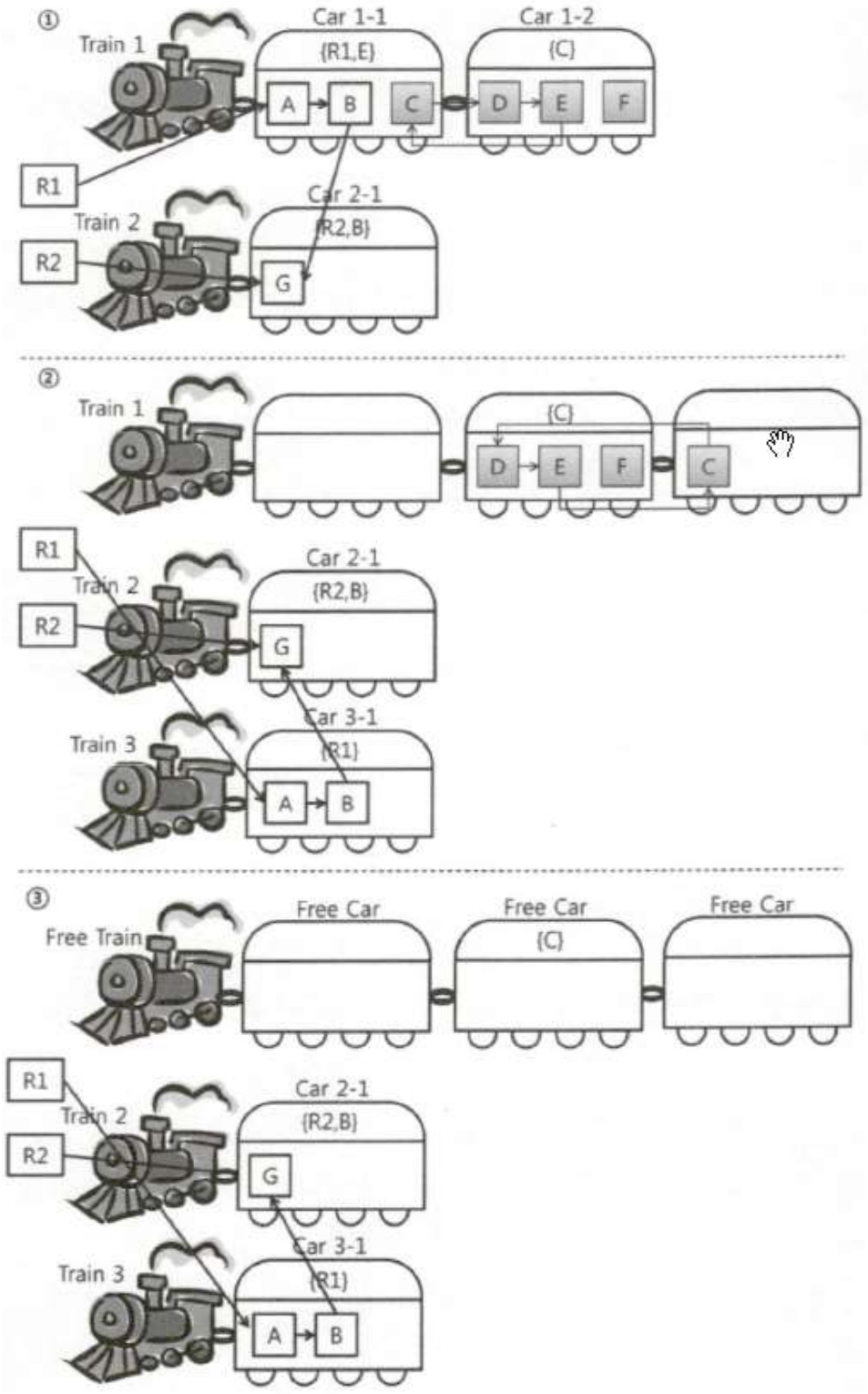
대다수의 Object는 생성된지 얼마 되지 않아 Garbage가 되는 짧은 수명을 가진다. 때문에 수명이 긴 소수의 Long Lived Object의 경우 Inactive와 Active를 여러번 번갈아가며 Copy 작업을 할 때 Overhead가 발생한하게 된다. 이를 개선하기 위해 Generational Algorithm은 Heap을 Age 별로 몇 개의 Sub Heap으로 나눈다. 그리고 Marking시 Age를 카운트 하여 Age가 일정 수치를 넘게 되면 Long Lived Object로 판단하고 Matured Object가 되어 Promotion을 하고 Garbage Object는 Sweep 한다.



Fragmentation이나 메모리의 활용이 좋으며 Copy Overhead가 짧아지며, Youngest Generational

Heap 에서는 Mark-Sweep Algorithm을 Promotion과정은 Copying Algorithm을 적용하는 등 각 Sub Heap에서 각각 적절한 Algorithm을 적용할 수 있다.

Train Algorithm



Garbage Collection은 메모리를 재사용하기 위한 것이다. 이를 위한 Tracing Algorithm이 등장한 이후, Garbage Collection을 수행할 때 프로그램에 Suspend 현상이 나타나는 것은 감수할 수 밖에 없는 일이었다. 그러나 Web Application Server의 짧은 트랜잭션을 처리하는 시스템의 경우 불규칙적인 Suspend 현상은 비즈니스에도 악영향을 끼친다.

Train algorithm은 Garbage Collection 중의 Suspend 발생시, 전체 Heap이 Suspend가 발생하는 것을 피하고 최소의 Memory Block만 Suspend를 발생시킨다. 이를 위해 Heap을 작은 Memory Block으로 나누어 Single Block 단위로 Mark Phase와 Copy Phase로 구성된 Garbage Collection을 수행한다. 따라서 Suspend를 분산시켜 전체적인 Pause Time이 줄어드는 효과가 발생한다. 이러한 특징 때문에 Incremental Algorithm이라고도 한다.

Train Algorithm 에서는 Heap을 Train으로 구성한다. 여기서 Train은 Car(객차)라고 불리는 Fixed Size의 Memory Block을 묶어 놓은 체인이다. Train은 필요에 따라 Car또는 Train을 추가할 수 있다.

그리고 각 Memory Block은 Car 외부에서의 참조를 기억하는 Remember Set 이라는 장치가 있다. Block 체인 내부는 빠르게 검색하고 Block 체인을 벗어났을 경우 Remember Set에 남겨 빠른 참조가 가능하게 한다.

Mark를 통한 Reachable Object가 아닌 Garbage Objec들이 순환참조일 때 새로운 객차가 아닌 새로운 트레인을 추가하고 Copying과 Sweep을 수행한다.

장점 : Pause Time을 분산하여 장시간의 Suspend를 피할 수 있다.

단점 : 개별 Suspend 시간의 합이 다른 Algorithm의 Suspend에 비해 클 수 있다.

Adaptive Algorithm

JVM은 Heap의 현재 상황을 모니터링하여 적절한 Algorithm을 선택하고 Heap Sizing을 자동화하는 등 Collection 방법을 적용한다. 이를 Adaptive Algorithm이라고 하며 Hotspot JVM의 Ergonomics기능, Adaptive Option이나 IBM JVM의 Tilting 기능 등으로 구현되어 있다.