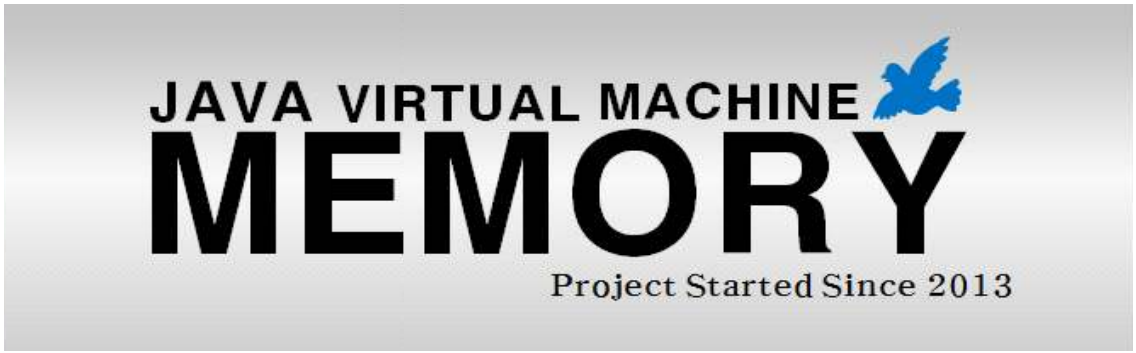


조 명	이정도면 괜찮조!	
주 제	JVM메모리 구조	
조 원	**설 미 라**	자료조사, 자료작성, PPT작성, 보고서작성. 발표. 조장.
	최 지 성	자료조사, 자료작성, PPT작성, 보고서작성. 발표.
	이 용 열	자료조사, 자료작성, PPT작성, 보고서작성.
	이 윤 경	자료조사, 자료작성, PPT작성, 보고서작성.
	이 수 은	자료조사, 자료작성, PPT작성, 보고서작성.
발 표 일	2013. 05. 07.	
비 고		

Index

1. 메모리 구조 인식의 중요성

1) 메모리의 정의

- 프로그램을 실행하기 위한 데이터 및 명령어를 저장하는 공간

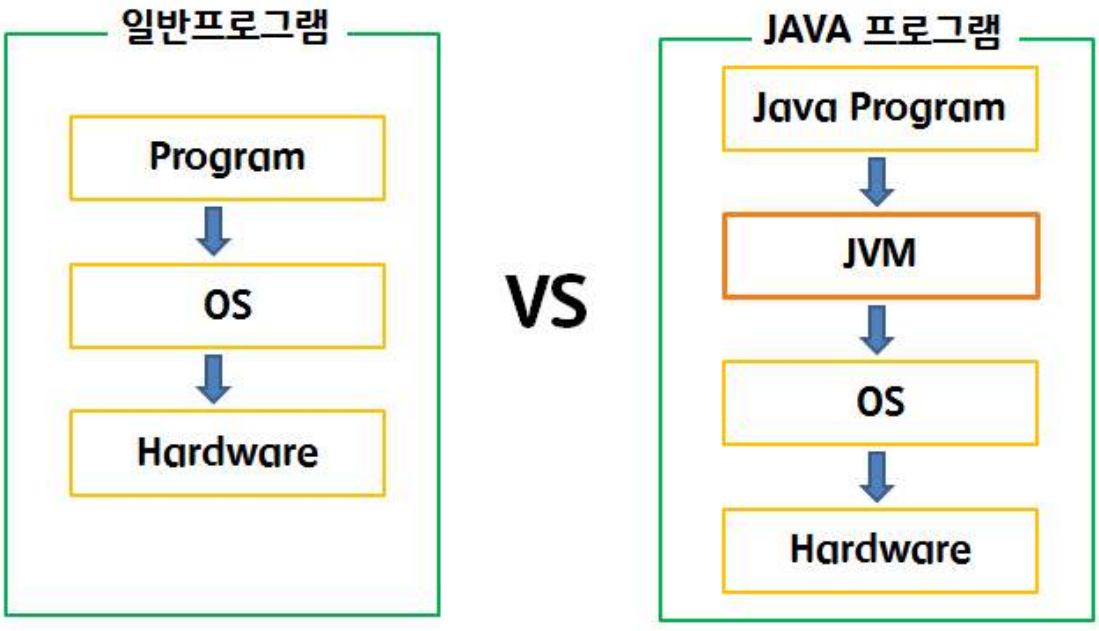
2) 메모리구조를 공부하는 이유

- 같은 기능의 프로그램이더라도 메모리 관리에 따라 성능이 좌우됨
- 메모리 관리가 되지 않은 경우 속도저하 현상이나 튕김 현상 등이 일어날 수 있음
- 한정된 메모리를 효율적으로 사용하여 최고의 성능을 내기 위함

2. JAVA 프로그램 실행구조

1) 일반 프로그램 과 JAVA프로그램

- 일반 프로그램 : OS에서 실행
- JAVA 프로그램 : JVM에서 실행되고, JVM은 OS에 종속적이다.



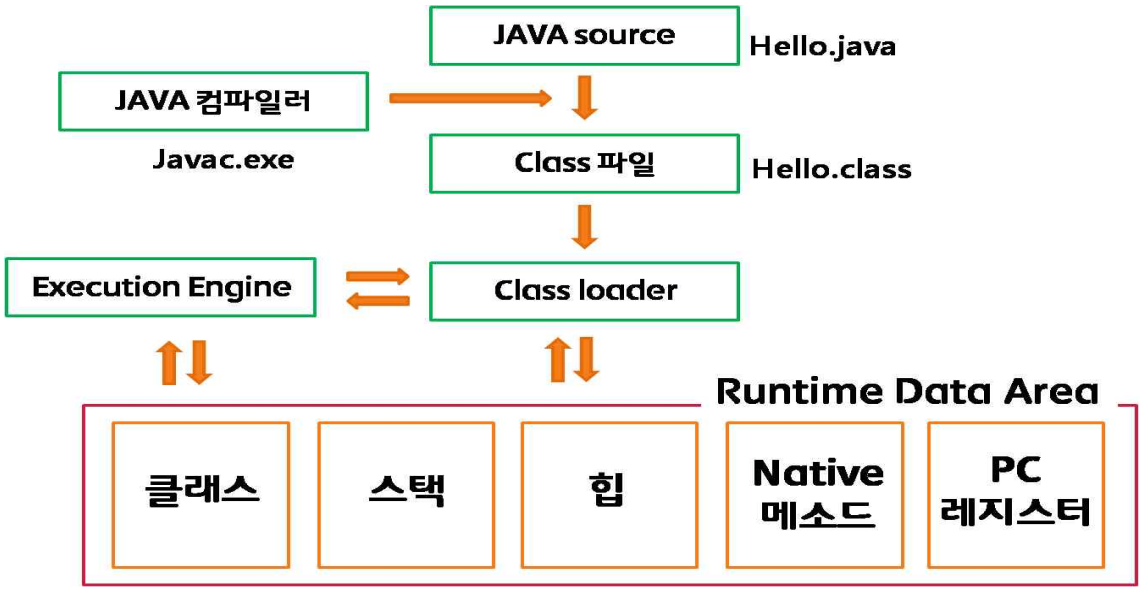
2) JVM이란?

- Java Virtual Machine
- JAVA와 OS 사이에서 중계자 역할
- JAVA가 OS에 구애받지 않고 재사용을 가능하게 해 줌
- 메모리 관리 기능(Garbage Collection)

3. JVM 메모리 구조와 과정

1) JVM 메모리 구조

- JAVA Source : 사용자가 작성한 JAVA 코드
- JAVA Compiler : JAVA 코드를 Byte Code로 변환시켜주는 기능
- Class Loader : Class파일을 메모리(Runtime Data Area)에 적재하는 기능
- Execution Engine : Byte Code를 실행 가능하게 해석해주는 기능
- Runtime Data Area : 프로그램을 수행하기 위해 OS에서 할당 받은 메모리 공간



2) Runtime Data Area

① Class Area

- Method Area, Code Area, Static Area 로 불리어짐

i) Field Information : **멤버변수의** 이름, 데이터 타입, 접근 제어자에 대한 정보

ii) Method Information : **메서드**의 이름, 리턴타입, 매개변수, 접근제어자에 대한 정보

iii) Type Information :
- Type의 속성이 Class인지 Interface인지의 여부 저장
- Type의 전체이름(패키지명+클래스명)
- Type의 Super Class의 전체이름
(단, Type이 Interface이거나 Object Class인 경우 제외)
- 접근 제어자 및 연관된 interface의 전체 리스트 저장

iv) 상수 풀(Constant Pool)

- Type에서 사용된 상수를 저장하는 곳(중복이 있을 시 기존의 상수 사용)
- 문자 상수, 타입, 필드, Method의 symbolic reference(객체 이름으로 참조하는 것)도 상수 풀에 저장

v) Class Variable
- Static 변수라고도 불림
- 모든 객체가 공유 할 수 있고, 객체 생성 없이 접근 가능

vi) Class 사용 이전에 메모리 할당
- final class 변수의 경우(상수로 치환되어) 상수 풀에 값 복사

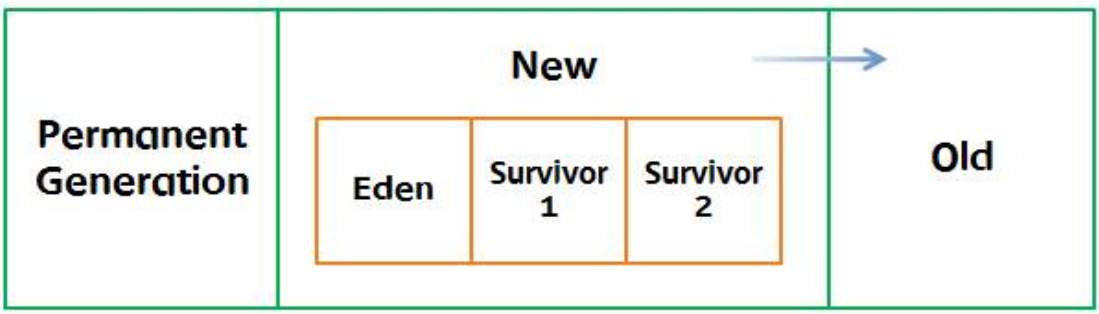
② Stack Area
- Last In First Out (LIFO)
- 메서드 호출 시마다 각각의 스택프레임이 생성
- 호출된 메서드의 매개변수, 지역변수, 리턴 값 및 연산 시 일어나는 값들을 임시로 저장
- 메서드 수행이 끝나면 프레임별로 삭제

③ Heap Area
- new 연산자로 생성된 객체와 배열을 저장하는 공간
- 클래스 영역에 로드된 클래스만 생성가능
- Garbage Collector를 통해 메모리 반환

i) Permanent Generation
- 생성된 객체들의 정보의 주소 값이 저장된 공간

ii) New Area
- Eden : 객체들이 최초로 생성되는 공간
- Survivor : Eden에서 참조되는 객체들이 저장되는 공간

iii) Old Area : New Area에서 일정시간이상 참조되고 있는 객체들이 저장되는 공간



④ Native method stack area
- 자바 외의 다른 언어에서 제공되는 메서드들이 저장되는 공간

⑤ PC Register
- Thread가 생성 될 때마다 생성되는 공간
- Thread가 어떤 부분을 어떤 명령으로 실행할 지에 대한 기록
- 현재 실행되는 부분의 명령과 주소를 저장

4. Garbage Collection

- 참조되지 않은 객체들을 탐색 후 삭제
- 삭제된 객체의 메모리를 반환
- Heap 메모리의 재사용

1) Minor Garbage Collection

i) New 영역에서 일어나는 Garbage Collection
ii) Eden영역에 객체가 가득 차게 되면 첫 번째 Garbage Collection 발생
iii) Survivor1 영역에 값 복사
iv) Survivor1 영역을 제외한 나머지 영역의 객체들을 삭제
v) Eden영역과 Survivor1영역의 메모리가 기준치 이상일 경우, Eden 영역에 생성된 객체와 Survivor1영역에 있는 객체 중 참조되고 있는 객체가 있는지 검사
vi) 참조되고 있는 객체를 Survivor2 영역에 복사
vii) Surviver2 영역을 제외한 영역의 객체들을 삭제
viii) 일정시간이상 참조되고 있는 객체들을 Old영역으로 이동
ix) 반복

2) Major Garbage Collection (Full Garbage Collection)

i) Old영역에 있는 모든 객체들을 검사
ii) 참조되지 않은 객체들을 한꺼번에 삭제
iii) Minor Garbage Collection에 비해 시간이 오래 걸리고 실행 중 프로세스가 정지

5. 소스예제 1

```
1  public class A
2  {
3      static int x = 5;
4      public static void main(String[] args)
5      {
6          int a = 10;
7          int b = 20;
8          int c = 0;      // 1번
9          c = add(a,b);
10         System.out.println("int c = "+c);
11         System.out.println("static int x = "+x);
12     }
13     public static int add(int a,int b)
14     {
15         x++;        // 2번
16         return a+b;
17     }
18 }
```

Method 영역	Stack 영역	Heap 영역

6. 소스예제 2

```
1  public class B
2  {
3      public static void main(String[] args)
4      {
5          int a = 5;
6          int b = 10;
7          int result = 0;      // 1번
8
9          C c = new C();       // 2번
10
11         result = c.cMethod(a, b);   // 3번
12
13         System.out.println(result);
14
15
16     }
17 }
```

```
1  public class C
2  {
3      int x = 0;
4      public int cMethod(int a, int b)
5      {
6          x = a+b;
7          return x;
8      }
9  }
```

Method 영역	Stack 영역	Heap 영역

7. 소스예제 3

```
1  public class D {
2      public static void main(String[] args)
3      {
4          int a[] = new int[] {1,2,3};
5          a = new int[] {5,6,7,8};
6          System.out.println(a[3]);    //1번
7
8          int b[][] = new int[][] {{1,2,3},{4,5},{6,7,8,9}};
9          System.out.println(b[2][3]);    //2번
10
11         String s1 = "Hi Java";
12         System.out.println(s1);
13         String s2[] = new String[] {"Hi","Java"};
14         System.out.println(s2[1]);     //3번
15     }
16 }
```

Method 영역	Stack 영역	Heap 영역

8. 참고문헌

class loader	http://hidka.tistory.com/18
GC	http://sirius.pe.kr/v5/tt?page=2&TSSESSION=11ebac6256a65e1b53499ee647d269da
GC	http://novathin.kr/m/14
GC	http://blog.naver.com/wwgenii?Redirect=Log&logNo=60510003
Heap	http://sanggu.blogspot.kr/2011/10/java-heap.html
JVM	http://hidka.tistory.com/entry/The-Java-Language-and-JVM
JVM구조	http://performeister.tistory.com/category/Performance
JVM구조	http://helloworld.naver.com/helloworld/1230
JVM구조	http://force44.blog.me/130095188664
JVM구조	http://cafe.naver.com/fordeveloper
JVM구조	http://blog.naver.com/zzooki/90034976951
JVM구조	http://blog.naver.com/pslovejin?Redirect=Log&logNo=50045585277
JVM구조	http://abacus.tistory.com/230
JVM구조	http://stophyun.tistory.com/37
JVM구조	http://hidka.tistory.com/entry/The-Java-Virtual-MachineJVM
memory 구조	http://blog.naver.com/0bloodwind0/20127908176
Runtime area	http://blog.naver.com/kmk1030?Redirect=Log&logNo=150141683660
Runtime area	http://blog.naver.com/2000yujin?Redirect=Log&logNo=130156226754
Runtime area	http://hidka.tistory.com/entry/The-Java-Application-Interface
Runtime area	http://gggura.egloos.com/3685556
Thread	http://www.jabook.com
Thread	http://blog.naver.com/myca11?Redirect=Log&logNo=80130599386
JVM실행구조	http://blog.naver.com/web4click?Redirect=Log&logNo=110159692556
JVM실행구조	http://cafe.naver.com/jjdev