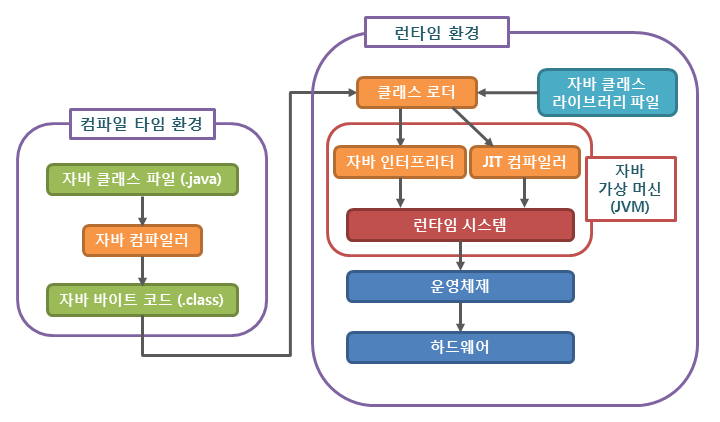
**1. Java의 byte code가 실행되는 Process를 설명하시오.**

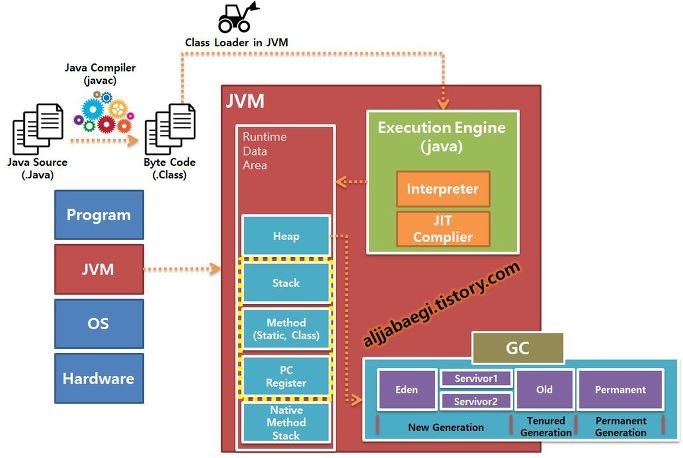


우선 개발 툴을 사용해 .java 파일을 생성한다. 그리고 Build 라는 작업을 하게 되면 Java Complier의 javac 라는 명령어를 사용해 .class 파일을 생성하게 된다. 이것이 아직 컴퓨터는 읽을 수 없는 자바 바이트코드(반 기계어)이다.

이렇게 생성된 자바 바이트 코드(.class)는 클래스 로더에 의해서 JVM내로 로드 되고, 실행엔진(Interpreter, JIT Compiler)에 의해 기계어로 해석되어 메모리 상(Runtime Data Area)에 배치되게 된다. Interpreter는 바이트 코드를 한 줄씩 읽기 때문에 실행이 느린 단점이 있어 이를 보완하기 위해 나온 것이 JIT(Just In Time) Compiler 이다.

인터프리터 방식으로 실행을 하다가 적절한 시점에 바이트 코드 전체를 컴파일 하고, 더 이상 인터프리팅 하지 않고 해당 코드를 직접 실행 하는 것이다. JIT Compiler에 의해 해석된 코드는 캐시에 보관하기 때문에 한 번 컴파일 된 후에는 빠르게 수행한다는 장점이 있다. 하지만 인터프리팅 방식보다는 훨씬 오래 걸리므로 한번만 실행하면 되는 코드는 인터프리팅 하는 것이 유리하다.

**2. JVM 내 메모리 구조에 대해 설명하시오.**



**Stack Area**

클래스 내의 메소드에서 사용되는 정보들이 저장되는 공간이다. 매개변수, 지역변수, 리턴 값 등이 저장되며 LIFO(Last In First Out) 방식으로 메소드 실행 시 저장되었다가 실행이 완료되면 제거된다. 임시 저장공간으로 생각하시면 됩니다.

**Method (Class, Static) Area**

클래스와 메소드, 멤버(클래스, 인스턴스) 변수와 상수(final) 정보 등이 저장되는 공간이다.

**PC Register Area**

쓰레드마다 하나씩 생성, JVM 명령의 주소 값이 저장되는 공간이다.

**Native Method Stack Area**

자바 외 다른 언어의 호출을 위해 할당되는 영역이다. 자바에서 C/C++의 메소드를 호출할 때 사용하는 Stack 영역이라고 생각하면 된다.

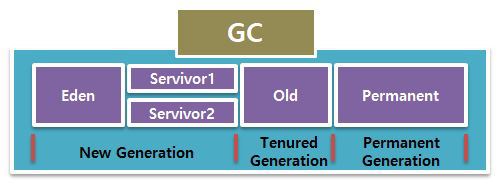
예를 들어 getList, insertList, updateList, deleteList CRUD 메소드가 있는 ListController class가 있을 때 이 클래스와 메소드의 정보는 실행엔진에 의해 Method 영역에 올라간다. 클래스의 메소드 호출이 발생하면 Method 영역의 정보를 읽어 해당 메소드의 매개변수, 지역변수 리턴값 등이 Stack 영역에 올려 처리하게 되고, 메소드의 실행이 끝나면 Stack 영역에서는 자동으로 제거된다.

만약 메소드 내에 New 명령어로 생성한 인스턴스나 배열이 있을 경우, 해당 값은 Heap 영역에 저장되고 Stack 영역에서는 이 Heap 영역의 값을 참조 할 수 있는 메모리 주소 값만 저장된다.

배열을 System.out.println(); 하게 되면 메모리 주소 값이 출력되는 이유이다.

**Heap Area**

Heap 영역은 아래와 같이 크게 세 영역으로 나뉜다. (자바8부터는 Permanent 영역 대신에 Java Heap 영역이 아니라 Metaspace로 호출되는 네이티브 영역에 저장된다.)

****

New 명령어를 통해 생성한 인스턴스와 배열 등의 참조형 변수정보가 저장되는 공간이다. 물론 Method Area에 올라온 클래스들만 생성이 가능하고, GC의 대상이 된다.

**- Garbage Collector**

Heap 영역의 메모리를 관리하고, reachability라는 개념을 사용해 참조되지 않는 객체들의 메모리를 회수하는 역할을 한다. 자바에서는 메모리를 명시적으로 지정하여 해제하지 않기 때문에 GC의 역할이 중요하다.

**- Minor, Major GC**

Heap 영역에 객체가 생성되면 최초로 Eden 영역에 할당된다. 그리고 이 영역에 데이터가 어느 정도 쌓이게 되면 참조 정도에 따라 Servivor1, Servivor2 중 빈 공간으로 이동되거나 회수된다. New Generation(Eden+Servivor1,2) 영역이 차게 되면 또 참조 정도에 따라 Old영역으로 이동 되게 되거나 회수된다. 이렇게 New Generation과 Tenured Generation 에서의 GC를 Minor GC 라고 한다.

또한 Old영역에 할당된 메모리가 허용치를 넘게 되면, Old 영역에 있는 모든 객체들을 검사하여 참조되지 않는 객체들을 한꺼번에 삭제하는 GC가 실행된다. 시간이 오래 걸리는 작업이기 때문에 이 때 GC를 실행하는 쓰레드를 제외한 모든 쓰레드는 작업을 멈추게 된다. 이를 'Stop-the-World' 라고 하고, 이렇게 'Stop-the-World'가 발생하고 Old영역의 메모리를 회수하는 GC를 Major GC라고 한다. 앞에서 말한 것과 같이 Major GC가 실행되면 이것이 종료될 때까지 다른 모든 쓰레드가 멈추기 때문에 성능에 영향을 끼칠 수 밖에 없다.