

1주차 과제

Study 할래?

스터디 내용

1. JVM이란 무엇인가
2. 컴파일 하는 방법
3. 실행하는 방법
4. 바이트코드란 무엇인가
5. JVM 구성 요소
6. JIT 컴파일러란 무엇이며 어떻게 동작하는지
7. JDK와 JRE의 차이

1-0. 사전지식

명령어가 H/W에 전달되려면 2진수로 저장되고 최종적으로는 전기 신호로 변환되어야 한다. 소프트웨어 초기에는 프로그램을 하드웨어에 적합한 2진수 형태의 기계어로 작성했다. 하지만 프로그램을 편리하게 작성하기 위해서는 사람이 이해하기 쉬운 문자 형태의 명령어가 필요했다. 이에 사람이 사용하는 언어에 가까운 형식을 갖춘 프로그램 언어가 여럿 등장하게 된다. 이처럼 사람이 이해하기 쉬운 프로그램 언어를 고급 언어라 하며 반대로 형태가 기계어에 가까운 프로그램을 저급언어라 한다. 고급 언어를 사용해서 명령어를 작성하면 사람은 편리하지만, 컴퓨터는 이해할 수 없는 형태이므로 반드시 변환 과정을 거쳐 기계어로 만들어야 한다. 이러한 변환 과정을 컴파일이라 한다. 컴파일을 통해 변환된 프로그램은 기계어를 통해 하드웨어에 전달되어 실행된다. 그리고 원하는 결과를 내게 된다.

1. JVM이란 무엇인가?

JVM : Java Virtual Machine / OS에 독립적이다. = Write Once Run Anywhere

JVM은 JAVA와 OS사이에서 중개자 역할을 수행한다. (기계어로 해석)

가장 중요한 메모리 관리, Garbage Collection을 수행한다.

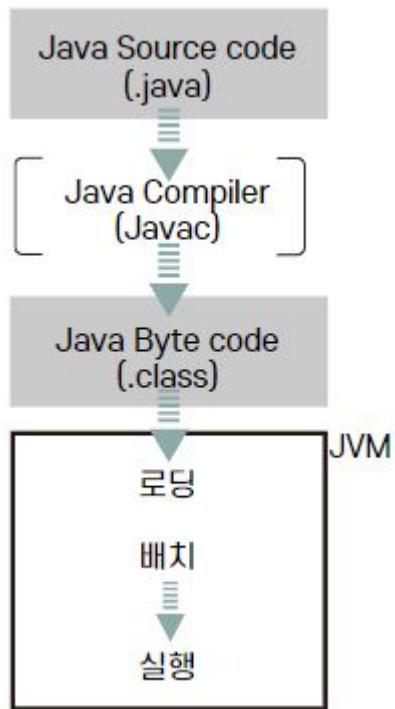
그리고 JVM은 스택기반의 가상머신이다. (스택기반으로 동작)

👉 왜? 알아야 하는가?

메모리는 한정되어 있고 효율적으로 사용해 최고의 성능을 내기 위함이다.

동일한 기능의 프로그램이더라도 메모리 관리에 따라 성능이 좌우된다.

2-0. 사전지식



1. 프로그램이 실행되면 JVM은 OS로부터
프로그램이 필요로하는 **메모리를 할당**받는다.
2. 자바 컴파일러가 소스코드를 읽어 **바이트코드로 변환**시킨다.
3. Class Loader를 통해 Class파일들을 JVM으로 **로딩**한다.
4. 로딩된 Class파일들은 Execution engine을 통해 **해석**된다.
5. 해석된 바이트코드는 Runtime Data Areas에 배치되어 **수행**한다.

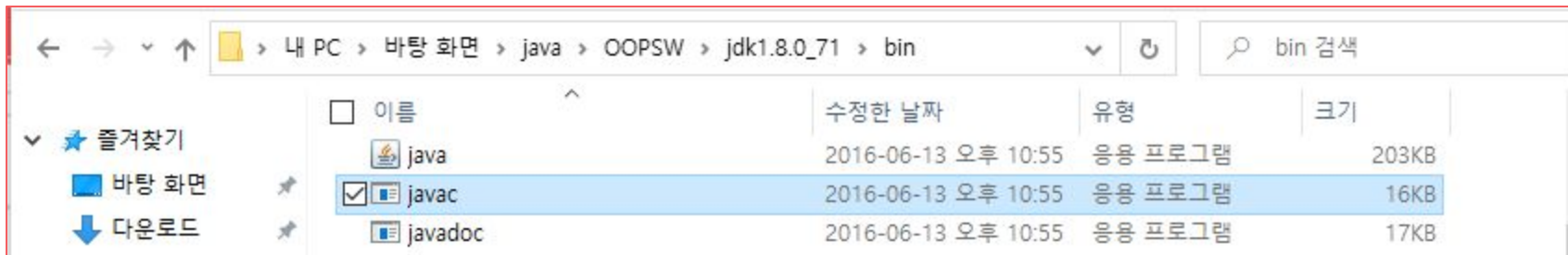
이러한 과정 속에서 JVM은 필요에 따라 Thread Synchronization 과 GC 관리작업을 수행한다.

2. Java 컴파일하는 방법

컴파일 한다 = .java 파일을 .class 파일(byte code)로 만든다.

JDK(Java Development Kit)를 설치하면 Javac라는 compiler가 포함되어 있다.

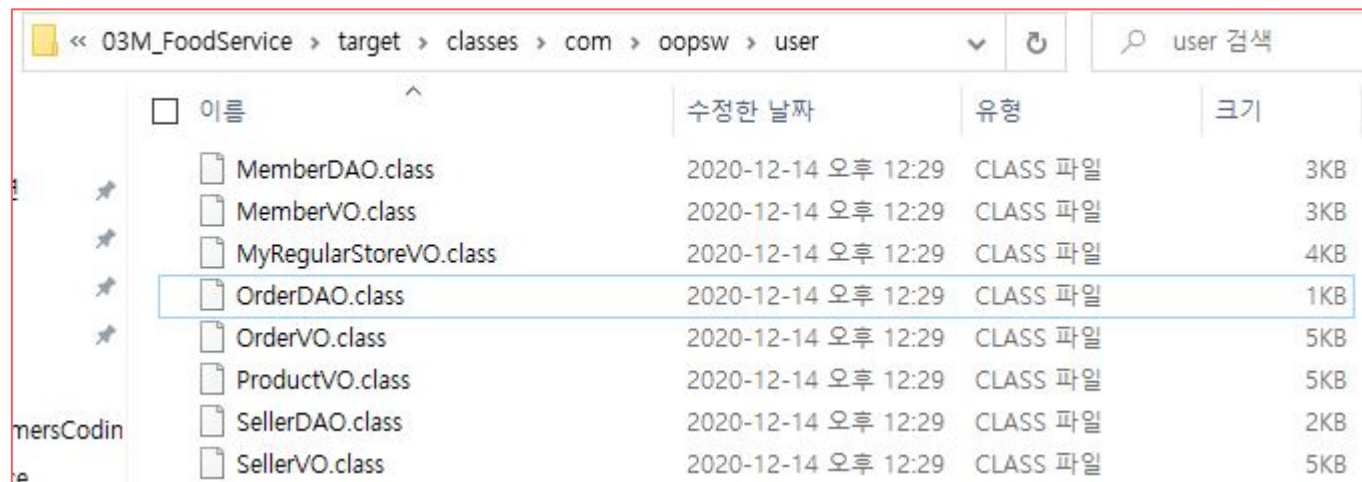
이 명령으로 .class 파일을 생성한다.



2-1. Java 컴파일하는 방법

우리는 컴파일을 직접하지 않고 주로 IDE를 사용해 대신한다.

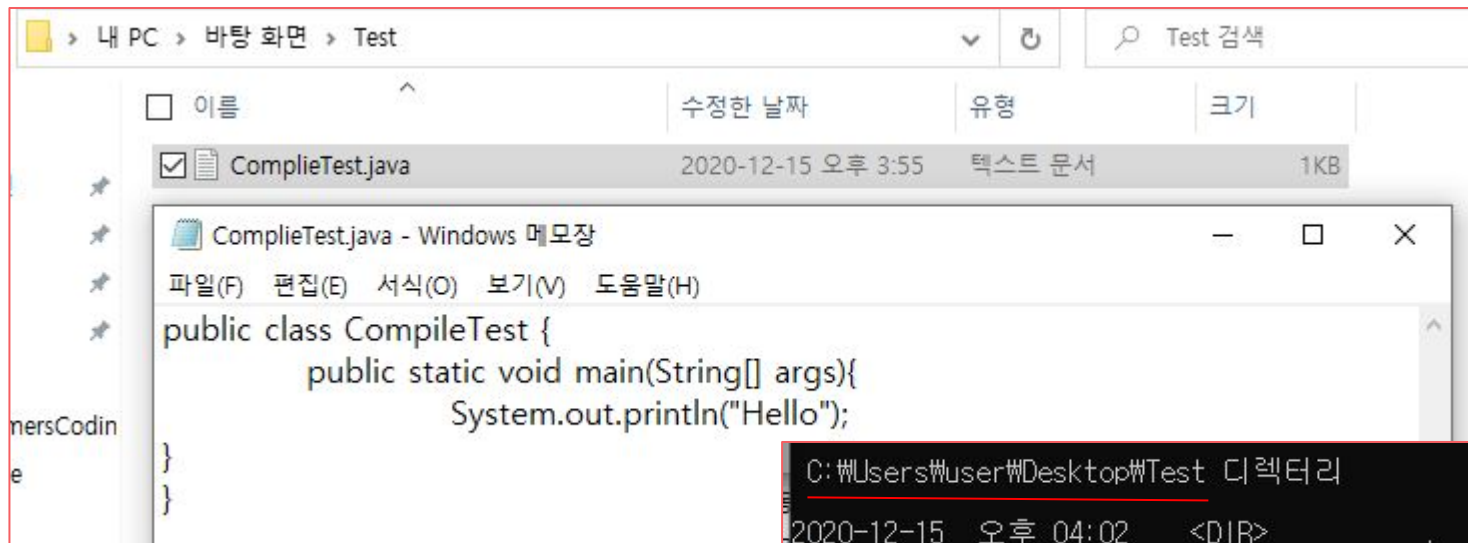
이제는 IDE를 사용하지 않고 명령프롬프트로 컴파일 하는 방법을 알아보자.



The screenshot shows a file explorer window with the path: << 03M_FoodService > target > classes > com > oopsw > user. A search bar on the right contains the text 'user 검색'. The main area displays a list of files with columns for '이름' (Name), '수정한 날짜' (Modified Date), '유형' (Type), and '크기' (Size). The file 'OrderDAO.class' is highlighted with a blue selection bar.

이름	수정한 날짜	유형	크기
MemberDAO.class	2020-12-14 오후 12:29	CLASS 파일	3KB
MemberVO.class	2020-12-14 오후 12:29	CLASS 파일	3KB
MyRegularStoreVO.class	2020-12-14 오후 12:29	CLASS 파일	4KB
OrderDAO.class	2020-12-14 오후 12:29	CLASS 파일	1KB
OrderVO.class	2020-12-14 오후 12:29	CLASS 파일	5KB
ProductVO.class	2020-12-14 오후 12:29	CLASS 파일	5KB
SellerDAO.class	2020-12-14 오후 12:29	CLASS 파일	2KB
SellerVO.class	2020-12-14 오후 12:29	CLASS 파일	5KB

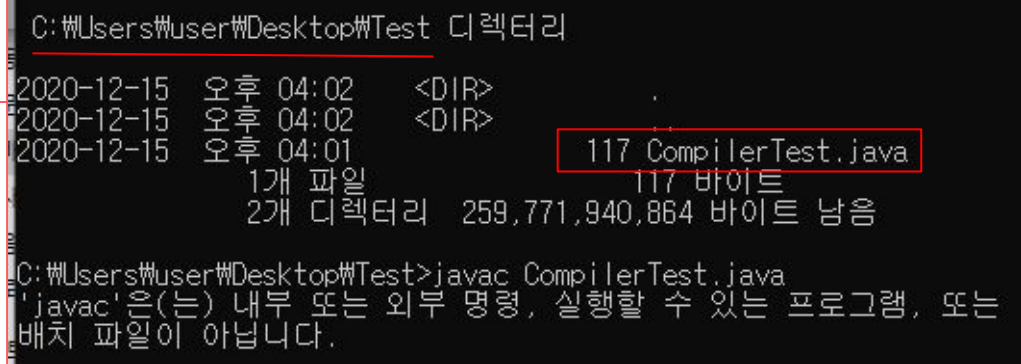
2-2. Java 컴파일하는 방법



밀줄의 경로에 **CompilerTest.java**
(java source code)를 저장해놓았다.
javac.exe를 실행시켰다.

이렇게 나온 이유는 환경변수 설정 때문. →

<https://blog.naver.com/hsm622/220979792342>



2-3. Java 컴파일하는 방법

환경변수 설정 후 `javac.exe`를 실행,
class 파일이 성공적으로 만들어졌다.

```
C:\Users\User\Desktop\Test>dir
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 6256-6182

C:\Users\User\Desktop\Test 디렉터리

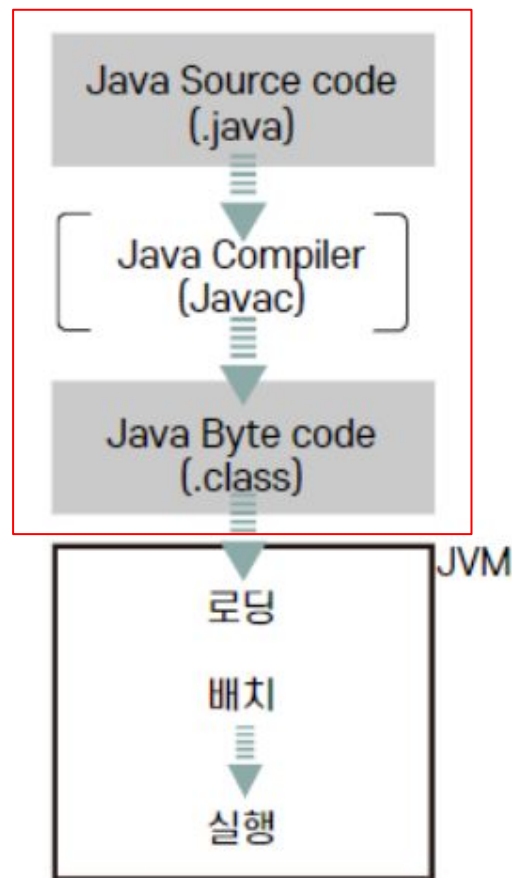
2020-12-15 오후 04:02 <DIR> .
2020-12-15 오후 04:02 <DIR> ..
2020-12-15 오후 04:01      117 CompilerTest.java
                  1개 파일              117 바이트
                  2개 디렉터리  259,979,407,360 바이트 남음

C:\Users\User\Desktop\Test>javac CompilerTest.java

C:\Users\User\Desktop\Test>dir
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 6256-6182

C:\Users\User\Desktop\Test 디렉터리

2020-12-15 오후 04:14 <DIR> .
2020-12-15 오후 04:14 <DIR> ..
2020-12-15 오후 04:14      423 CompilerTest.class
2020-12-15 오후 04:01      117 CompilerTest.java
                  2개 파일              540 바이트
                  2개 디렉터리  259,979,403,264 바이트 남음
```



3. Java 실행하는 방법

실행은 java 명령어로 한다.

```
C:\Users\User\Desktop\Test>java CompilerTest
Hello
```

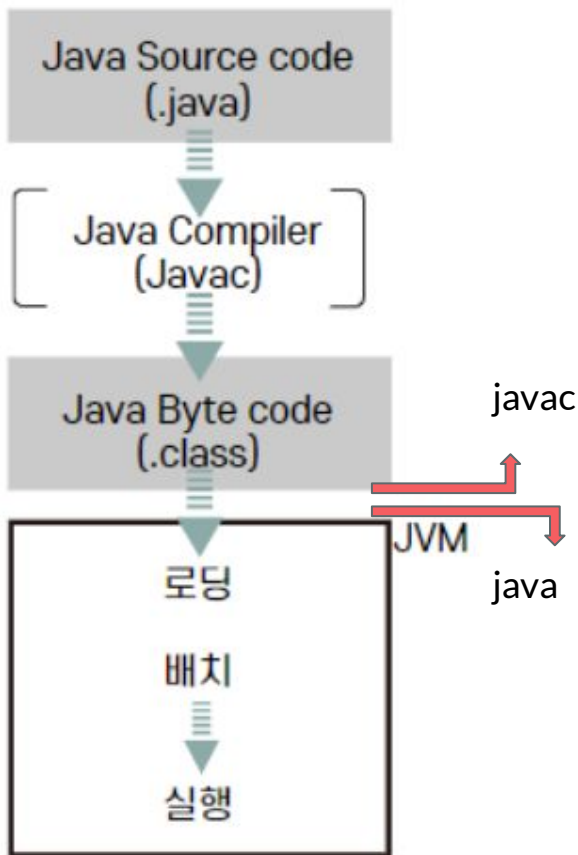
```
C:\Users\User\Desktop\Test 디렉터리
2020-12-15 오후 04:35 <DIR>
2020-12-15 오후 04:35 <DIR>
2020-12-15 오후 04:01      117 CompilerTest.java
                  1개 파일      117 바이트
                  2개 디렉터리 259,860,156,928 바이트 남음

C:\Users\User\Desktop\Test>java CompilerTest
오류: 기본 클래스 CompilerTest을(를) 찾거나 로드할 수 없습니다.
원인: java.lang.ClassNotFoundException: CompilerTest

C:\Users\User\Desktop\Test>javac CompilerTest.java

C:\Users\User\Desktop\Test>java CompilerTest
Hello
```

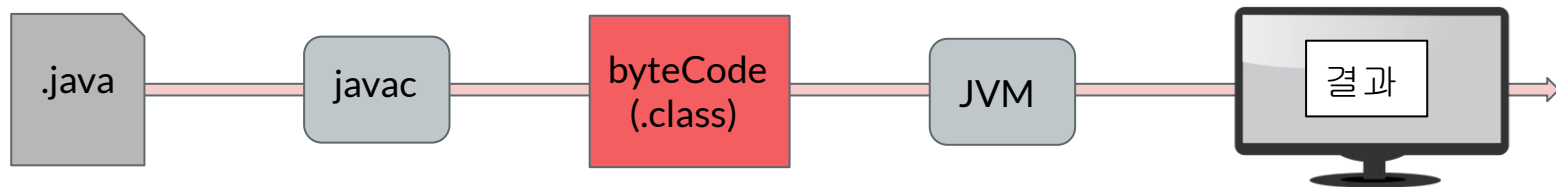
class 파일이 존재해야함



4. 바이트코드란 무엇인가?

Byte code = 특정 하드웨어가 아닌 VM에서 실행을 위한 이진 표현법(0과 1로 구성)
특정 하드웨어에 대한 의존성을 줄이고, 인터프리팅도 쉬운 결과물을 생성하려는 이유로 존재
실시간 번역기 또는 JIT (Just In Time)에 의해 기계어(바이너리 코드)로 번역된다.

- 1) CPU가 이해할 수 있는 언어가 바이너리 코드, VM이 이해하는 언어는 바이트 코드
- 2) 어떠한 개발, 실행환경에도 종속되지 않고 실행가능한 VM용 기계어
- 3) 고급언어로 작성된 소스코드를 VM이 이해가능한 중간코드로 컴파일한 결과
- 4) 실시간 번역기, JIT에 의해 바이너리 코드로 변환됨
- 5) CPU, VM 둘다 txt를 이해하지 못함 cf) JVM을 위한 바이트 코드 = 자바 바이트코드

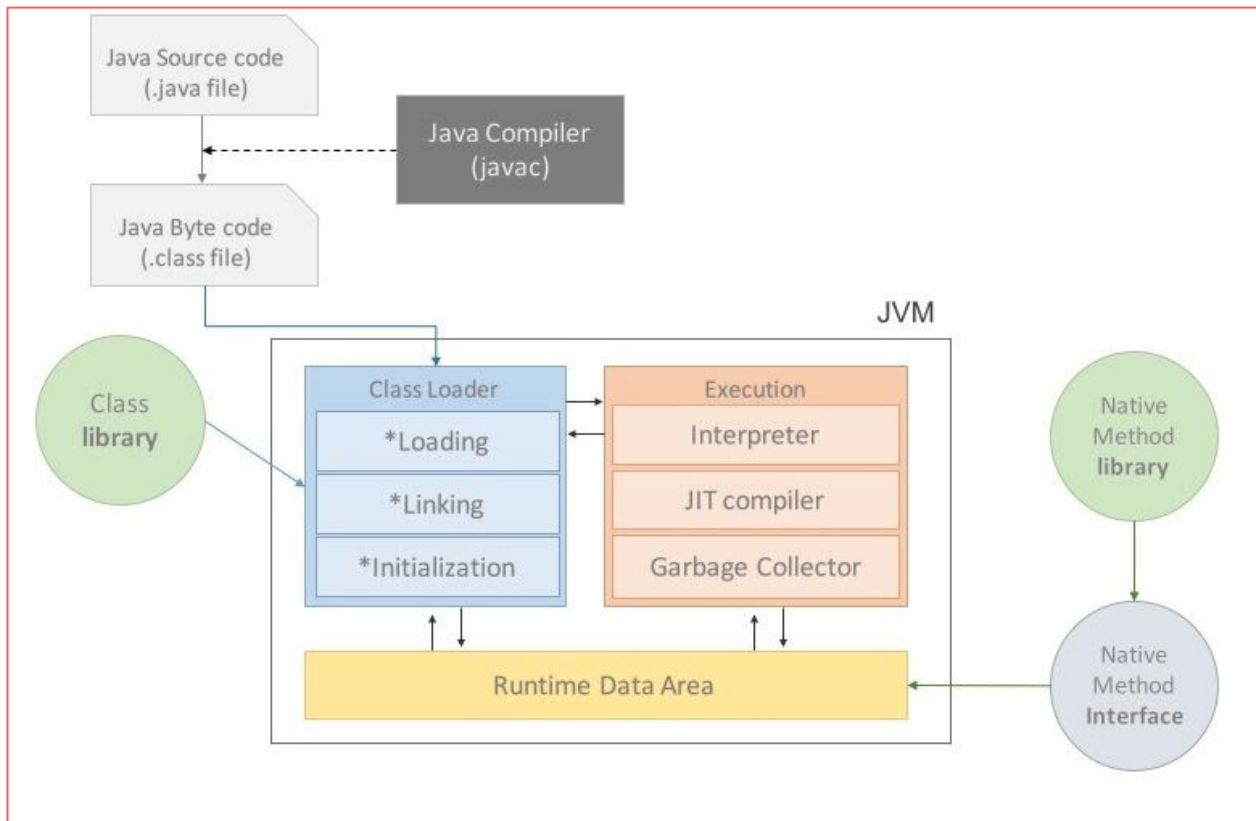


5. JVM구성요소

JVM의 흐름이다.

JIT컴파일러 알아보기전
몇 가지 JVM구성을 보자.

1. Class Loader
2. Execution Engine
3. Interpreter
4. JIT
5. Garbage collector
6. Runtime Data Area



출처 :

<https://asfirstalways.tistory.com/158>

5-1. JVM구성요소

1. Class Loader (클래스 로더)

abstract class로써 `bytecode`를 읽어 들어서 `class`객체를 생성하는 역할.

프로그래머가 `SampleTest aa = new SampleTest();` 라는 코드를 처음 실행하면 JVM은 `SampleTest Class`를 `Loader`를 통해서 `ByteCode`로 최초 메모리에 Load.

Linking, initialize 등 세부내용 참조: <https://keichee.tistory.com/105>

2. Execution Engine (실행 엔진)

`Class Loader`에 의해 로드된 `.class`들은 `Runtime Data Area`의 `Method Area`에 배치되는데, JVM은 `Method Area`의 바이트 코드를 실행엔진에게 제공하여, `.class`에 정의된 내용대로 바이트 코드를 실행시킨다. Load 된 바이트코드를 실행하는 `Runtime Module`이 `Execution Engine` 이다.

5-2. JVM구성요소

실행엔진은 바이트코드를 명령어 단위로 읽어서 실행하는데, 두 가지 방식을 혼합하여 사용한다 (Interpreter / JIT or Dynamic Translation(동적번역))

3. Interpreter

바이트코드를 한 줄씩 해석, 실행하는 방식이다. 초기 방식으로 속도가 느리다.

4. JIT(Just In Time) 컴파일 or Dynamic Translation (동적번역)

인터프리터의 단점때문에 나온것이 JIT 방식이다. 바이트 코드를 JIT 컴파일러를 이용해 실제 실행하는 시점에 각 OS에 맞는 Native Code로 변환하여 속도를 개선했다. 하지만 Native Code로 변환하는데에도 비용이 소모되므로, 전부를 JIT 컴파일러 방식으로 실행하지 않고 Interpreter 방식으로 사용하다 일정기준이 넘어가면 JIT방식으로 명령어를 실행한다.

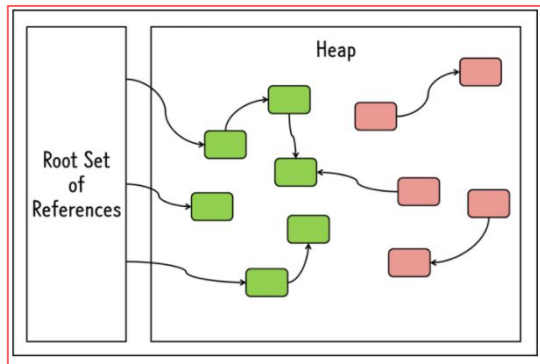
또한 매번 해석하지 않고 컴파일하며 해당 코드를캐싱해 이후에는 바뀐부분만 컴파일 하고 나머지는 캐싱된 코드를 사용한다.

5-3. JVM구성요소

5. Garbage Collector

GC는 Heap 메모리 영역에 생성된 객체들 중 참조되지 않은 객체들을 제거하는 역할을 한다. GC의 동작시간은 일정하게 정해져 있지 않기때문에 언제 정리할지는 알 수 없다. 즉 참조가 없어지자마자 작동하는 것이 아니라는 것이다. 또한 GC를 수행하는 동안 GC Thread를 제외한 모든 Thread는 일시정지가 된다. 특히, Full GC가 일어나는 수초간 모든 Thread가 정지한다면 심각한 장애로 이어질 수 있다.

출처 : <https://madplay.github.io/post/java-garbage-collection-and-java-reference>



5-4. JVM구성요소

6. Runtime Data Area

프로그램을 수행하기 위해 OS에서 할당받은 메모리 공간
각각의 목적에 따라 5개의 영역으로 나뉜다.

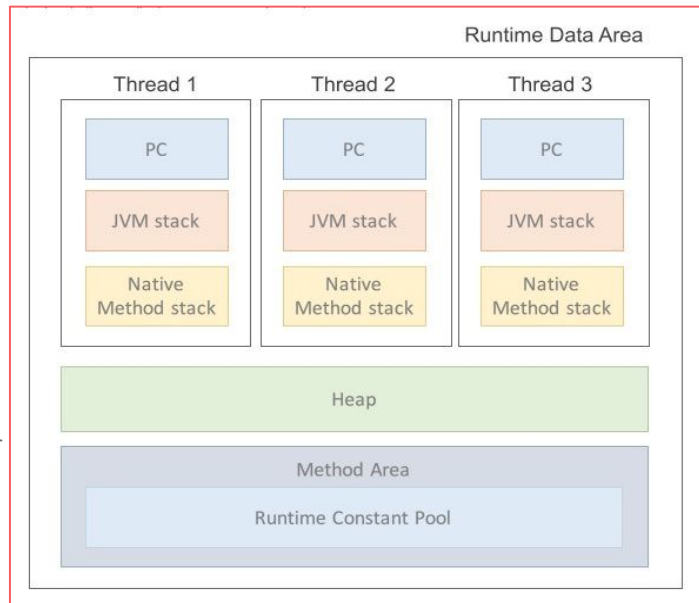
PC Register - Thread 생성될 때 마다 생기는 공간, Thread
가 어떤 명령어를 실행하게 될지를 기록

JVM stacks - Thread 제어를 위해 사용되는 메모리 영역

Native Method Stacks - 자바 이외의 이기종 언어에서 제공
되는 method의 정보가 저장되는 공간.

Heap - 사용자가 관리하는 인스턴스가 생성되는 공간

Method Area - 프로그램 실행중 클래스가 사용되면 JVM은
해당 클래스 파일을 읽어서 분석해 Method Area에 저장



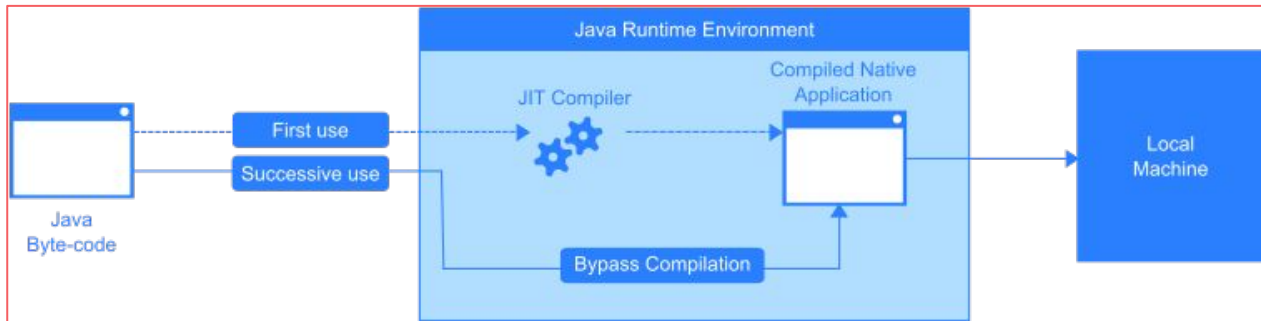
출처 : <https://asfirstalways.tistory.com/158>

<https://www.holaxprogramming.com/2013/07/16/java-jvm-runtime-data-area/>

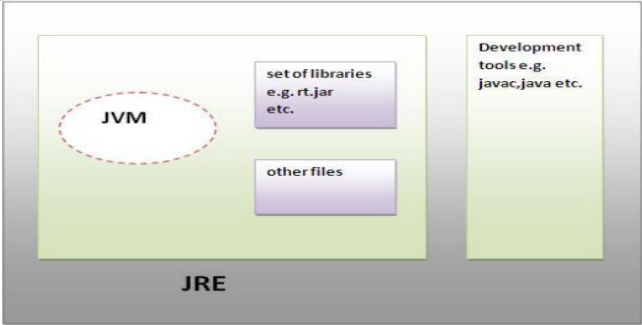
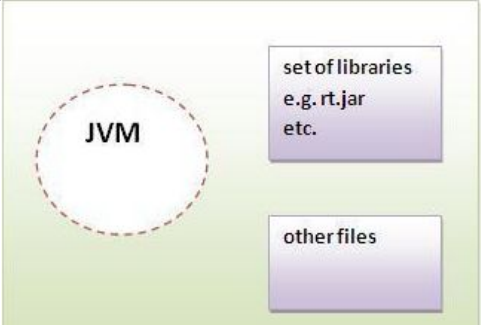
6. JIT 컴파일러란?

JIT 컴파일러가 컴파일 하는 과정은 **Interpreter**보다 훨씬 오래 걸리기 때문에 **JVM**은 내부적으로 해당 메서드의 호출 빈도를 체크해 일정 기준이 넘었을 때만 **JIT**를 통해 네이티브 코드를 생성한다. 이때 바로 만드는 것이 아니라 안에서 **IR(Intermediate Representation)**로 변환하여 최적화를 수행하고 그 다음에 변환한다. **JIT**를 사용하면 반복적으로 수행되는 코드는 매우 빠른 성능을 보인다는 장점이 있지만 반대로 처음에 시작할 때에는 변환 단계를 거쳐야 하므로 성능이 느리다는 단점이 있다.

하지만 최근들어 **CPU** 성능이 많이 좋아졌고, **JDK**의 성능 개선도 많이 이루어졌기 때문에 이 단점도 많이 개선되었다.



7. JDK와 JRE의 차이

JDK = Java Development Kit	JRE = Java Runtime Environment
개발을 위해 필요한 도구(javac, java등)	JRE는 JVM이 JAVA를 동작시킬 때 필요한 lib와 기타 파일들을 가지고 있다.
JDK를 설치하면 JRE도 같이 설치된다.	JRE는 JVM의 실행 환경을 구현했다.
JDK = JRE + @	JAVA 실행을 위해 필수, 프로그래밍은 JDK가 필요
 <p>The diagram illustrates the JDK structure. It is represented as a large container labeled 'JDK' at the bottom. Inside this container, there is a sub-container labeled 'JRE' on the left and a separate box labeled 'Development tools e.g. javac, java etc.' on the right. The 'JRE' container itself holds a dashed oval labeled 'JVM', a box labeled 'set of libraries e.g. rt.jar etc.', and another box labeled 'other files'.</p>	 <p>The diagram illustrates the JRE structure. It is represented as a single large container labeled 'JRE' at the bottom. Inside this container, there is a dashed oval labeled 'JVM', a box labeled 'set of libraries e.g. rt.jar etc.', and another box labeled 'other files'.</p>