1. JVM이란 무엇인가

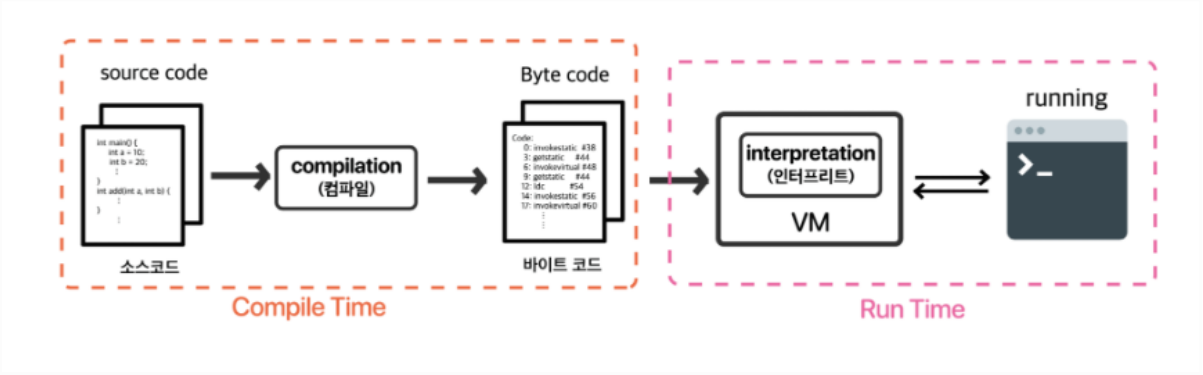
JVM (= Java Virtual Machine)

Java 등장 배경을 보면 JVM 의 등장 배경을 알 수 있다.
가전제품에 사용할 SW를 개발하기 시작했고, 가전제품은 매우 다양한 플랫폼을 가지고 있기 때문에 플랫폼에 독립적인 언어가 필요. -> 그렇게 개발한 게 Java.

"OS에 종속시키지 않고, 그 위에서 Java를 실행시킬 가상 머신을 만들자!"

그렇게 만든 가상 머신이 바로 JVM이다.

동작 과정을 그림으로 보면 다음과 같다.



이미지 출처: 프로그래밍 언어와 빌드 과정 [Build Process]

JVM 의 역할이 인터프리트 언어의 인터프리터인데, 인터프리트 언어가 이러한 방식으로 각 플랫폼에 지원하는 인터프리터만 있다면 실행 가능하기 때문에 플랫폼에 독립적인 것.

*

자바는 같은 코드를 사용하여 다른 CPU에서 실행되도록 하기 위해 직접 CPU의 기계어 코드를 생성해서는 안 된다. 대신,

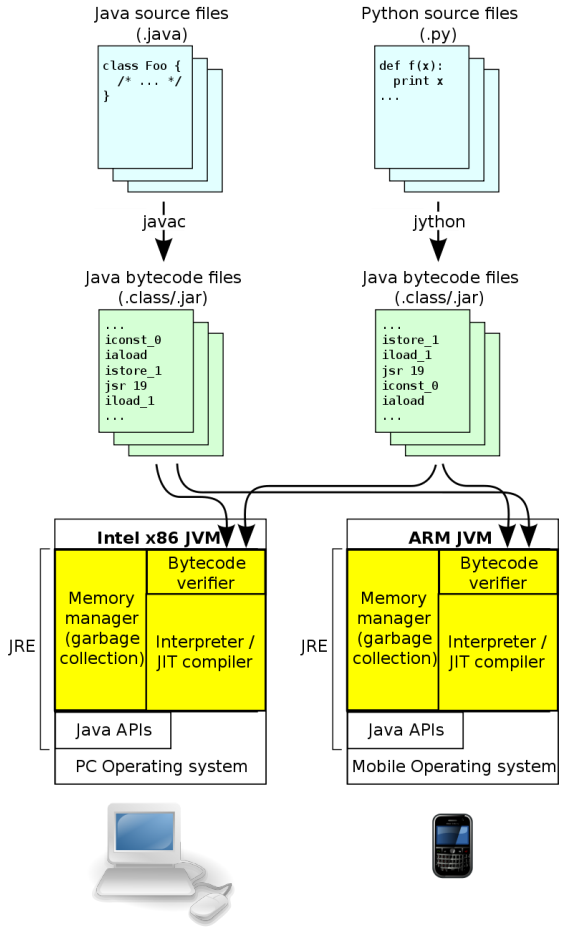
1. 자바는 .java 파일을 컴파일하여 .class 형식의 바이트코드(Java bytecode)라는 것을 생성한다. 이 과정은 JVM에서 이루어지는 것이 아니다!!

2. 이것을 자바 가상 머신(JVM, Java virtual machine)이 해석을 하여 실행한다.
   자바 가상 머신이 인터프리터가 되어 코드 해석 방식의 실행을 함으로써, 같은 바이트코드를 가지고 여러 가지의 CPU에서 실행이 가능해진다.

즉, Java는 OS에는 종속적이지 않지만, JVM이 OS에 종속적이다!

---

2. 바이트코드란 무엇인가



쉽게 생각하면 영어 같은 존재.

영어는 현재 세계 공용어이기 때문에 프랑스어, 힌두어 등등 다른 국가의 언어로 번역된 것을 1차적으로 영어로 번역을 해둔다. 그런 다음 영어로 번역된 것을 한국어로 번역하면 번역 가능한 사람이 많은 만큼 쉽고 빠르게 번역할 수 있다.

즉, 영어는 어디서든 대부분 쓰기 때문에 조금은 번거롭더라도 중간 번역과정 을 한 번 거쳐두면 그 다음부터는 재사용을 하기도 쉽고 다른 언어로 번역하기도 쉽다는 장점을 살린 것처럼 프로그래밍에서도 이를 적용한 방식이 바로 하이브리드 방식이고 영어가 곧, 바이트코드가 된다.

해당 가상머신 전용 기계어라고 볼 수 있다.

이미지 출처: 자바 (프로그래밍 언어) - 위키백과, 우리 모두의 백과사전

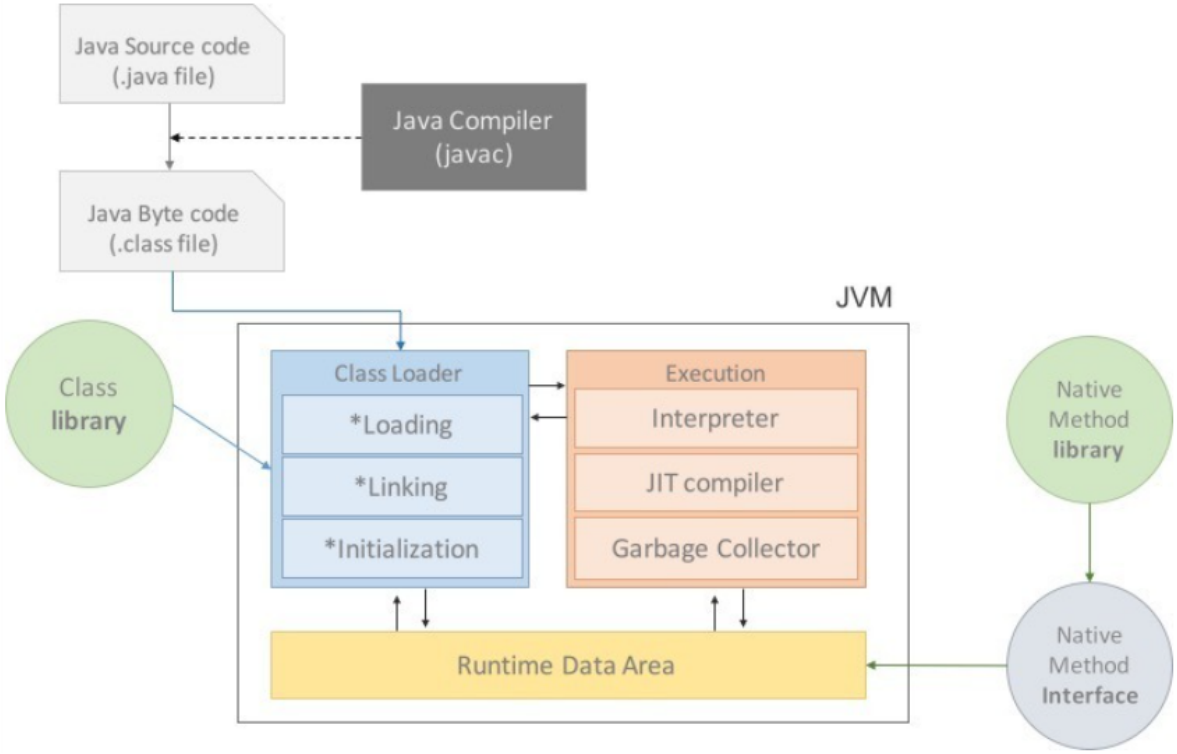
바이트코드(Bytecode, portable code, p-code)는 특정 하드웨어가 아닌 가상 컴퓨터에서 돌아가는 실행 프로그램을 위한 이진 표현법.

가상머신(Virtual Machine)이 이해할 수 있는 중간언어(intermediate language)라고 생각하면 된다.

예시 사진을 보면 python 으로 짠 코드도 jython 이라는 걸 통해 java bytecode 로 변환되어, JVM에서 실행시킬 수 있음을 볼 수 있다.

---

3. JVM 구성 요소

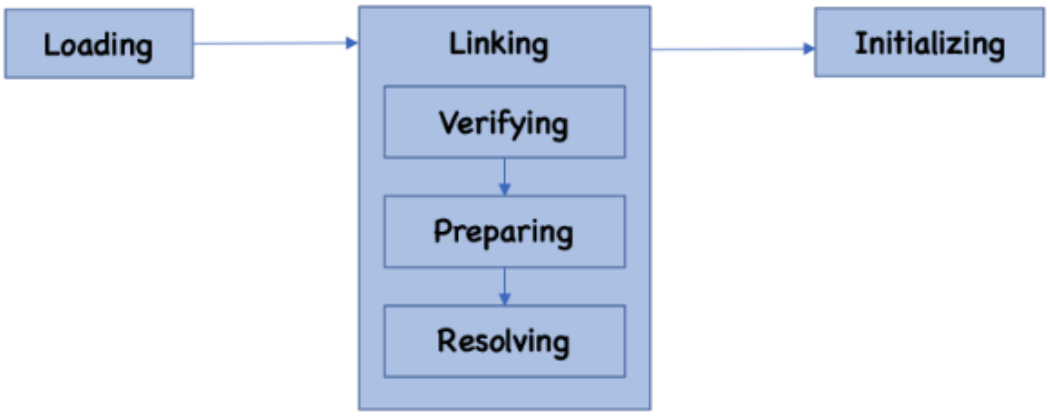


이미지 출처: https://jaehoney.tistory.com/173

JVM은 크게 4가지로 구성된다.

1) 클래스 로더

: 클래스들을 동적으로 메모리에 로딩하는 역할을 담당한다.



이미지 출처: https://hbase.tistory.com/174

Loading	- .class 파일 읽기 (+ .class 파일이 JVM 스펙이 맞는지 확인/Java 버전 확인) - 바이너리 데이터 만들기 - 메소드 영역에 저장
Linking	Linking 은 3단계로 이루어져있다. **1) Verify** : Loading 에서 만든 바이너리 데이터가 유효한 것인지 확인. 이 과정은 다소 오버헤드가 발생하기 때문에 신뢰할 수 있는 파일이라면! 성능 향상을 위해 진행하지 않을 수도 있음! (-Xverify:none 옵션 사용) **2) Prepare** : 클래스의 static 변수와 기본값에 필요한 메모리 공간 준비. **3) Resolution** : 런타임 상수 풀에 있는 심볼릭 참조를 실제 메모리 주소 값으로 교체해준다. ‘ 추상적인 기호를 구체적인 값으로 동적으로 결정하는 과정!’ 이라는데 …. 쉽게(?) 보면! JVM 명령인 anewarray, checkcast, new, putfield, putstatic 등은 런타임 상수 풀에 있는 심볼릭 참조를 사용하기 때문에 이런 명령어들을 실행하려면 먼저 심볼릭 참조를 해석해야한다! 그 과정임! 해석 단계는 필요에 따라 생략할 수도 있다!

Initializing	- Linking 의 Prepare 단계에서 확보한 메모리 영역에 static 값 할당! - SuperClass 초기화 + 해당 클래스 초기화 진행

클래스 로더를 통해 읽어들여 자바 API와 함께 실행하는 파트가 Execution 이다.

+ 런타임 상수 풀

> 런타임 상수 풀이란,
>
> OS가 JVM에 할당해준 메모리 영역(Runtime Data Area) 은
> 모든 쓰레드가 공유하는 메모리 영역과 쓰레드 별로 생성되는 영역이 있는데, 그 중 쓰레드가 공유하는 메모리 영역이 전역 변수와 정적 멤버 변수를 저장한다.
>
> 그 중 **Type**의 모든 상수 정보를 갖고 있는 데이터를 의미한다.

2) 자바 인터프리터

: 자바 컴파일러에 의해 변환된 바이트 코드를 읽고 한 줄씩 기계어로 해석하는 역할을 하는 것이 자바 인터프리터.
원래 JVM에서 인터프리터 방식만 사용하다가 성능(...)상의 문제로 JIT 컴파일러를 추가. 현재는 컴파일과 인터프리터 방식 병행하여 사용한다.

3) JIT 컴파일러

: Just-In-Time 의 약자로, 인터프리터 방식의 단점을 보완하기 위해 도입되었다.
JIT 컴파일러는 실행 시점에 인터프리터 방식으로 기계어 코드를 생성할 때 자주 사용되는 메소드의 경우 컴파일하고 기계어를 캐싱 한다.

+ 자주의 기준은**?**

> 컴파일 임계치(Compile Threshold)
> : JIT 컴파일러 내부에서 메서드가 호출될 때마다 호출 횟수를 카운팅!
> -> 특정 수치를 초과할 때 캐싱

4) 가비지 컬렉터

: 메모리 관리를 자동으로 해준다.

동작 과정은 다음과 같다.

1. Stop The World
   : 가비지 컬렉션을 실행하기 위해 JVM이 애플리케이션의 실행을 멈춘다.
   가비지 컬렉션을 실행하는 쓰레드를 제외한 모든 쓰레드들의 작업이 중단된다.
2. Mark and Sweep
   : 스택의 모든 변수 또는 객체를 스캔하면서, 사용되고 있는 메모리를 식별하고 Mark 한다. 이후 Mark가 되지 않은 객체들을 메모리에서 제거(Sweep)한다.

---

4. 컴파일 / 실행 하는 방법

바이트코드에 나왔던 이미지를 보면 알 수 있듯이,

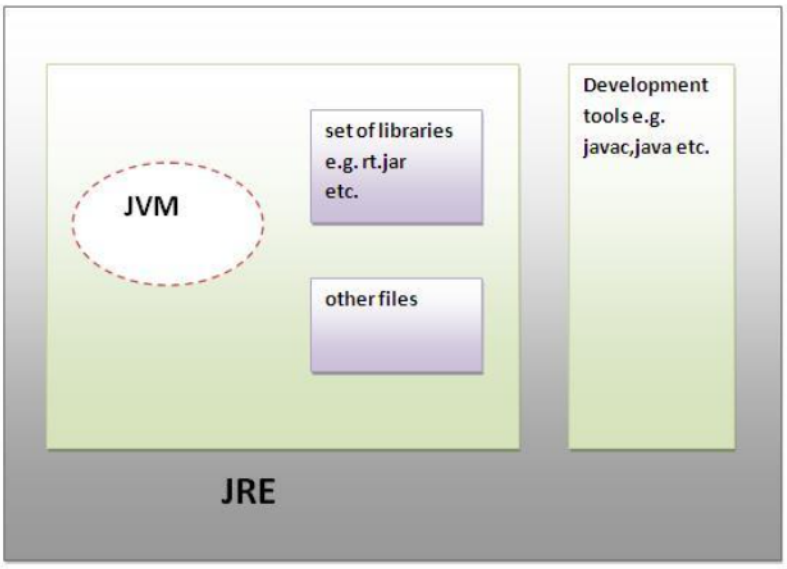
1. Java Compiler 인 `javac.exe` 파일을 사용하여 컴파일 하기 때문에 **javac** 명령어 통해서 `.java` 파일을 `.class` 파일로 컴파일 해준다..

2. `.class` 파일로 컴파일 된 bytecode 를 java 명령어를 통해서 실행시킨다.

---

5. JDK와 JRE의 차이

JDK 와 JRE 의 차이는 아래 그림을 통해 한 눈에 확인할 수 있다.



JDK

이미지 출처: https://wikidocs.net/257

JVM ⊂ JRE ⊂ JDK

- **JDK**

: Java Development Kit 의 약자로, JRE 와 자바 컴파일러(javac) , 개발을 위해 필요한 도구(java 등) 을 모두 포함한다.

- **JRE**

: Java Runtime Environment 의 약자로, 자바 프로그램을 동작 시킬 때, 필요한 라이브러리 파일들과 기타 파일들을 갖고 있다. JVM 의 실행 환경을 구현했다고 보면 된다!

출처

- (JVM이란? 개념 및 구조 JDK, JRE, JIT, 가비지 콜렉터...)
- JVM 구성 요소와 역할 정리! (+ JIT 컴파일러, 인터프리터, 클래스 로더, 가비지 컬렉터란 무엇인가?)
- 프로그래밍 언어와 빌드 과정 [Build Process]

- Java API: https://docs.oracle.com/en/java/javase/19/docs/api/index.html

추가로 더 보면 좋을 것 같은 링크!

- 스택 기반 VM과 레지스터 기반 VM: JVM이 스택 기반 VM을 선택한 이유
- [Java] 자바 클래스로더(Class Loader) : 클래스 로더의 구조, 원칙 등 더 자세한 내용을 볼 수 있다!
- 클래스로더 1. 동적인 클래스 로딩과 클래스로더 : 정적 클래스 로더 vs 동적 클래스 로더