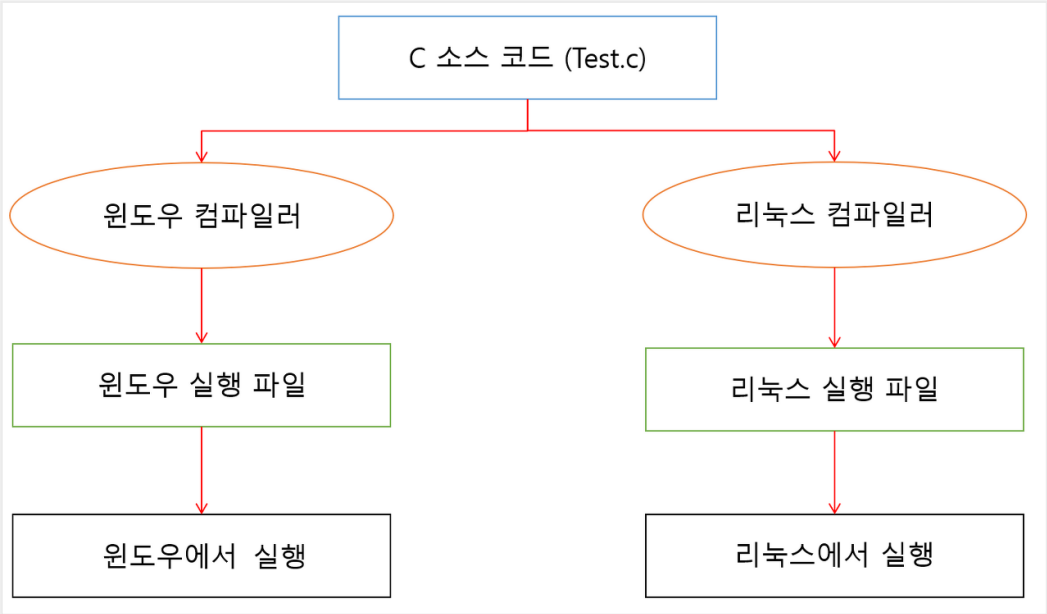
JVM 메모리 구조

❓ 개념
자바 가상머신 -> JVM(Java Virtual Machine)은 자바 프로그램의 실행환경을 만들어주는 소프트웨어입니다

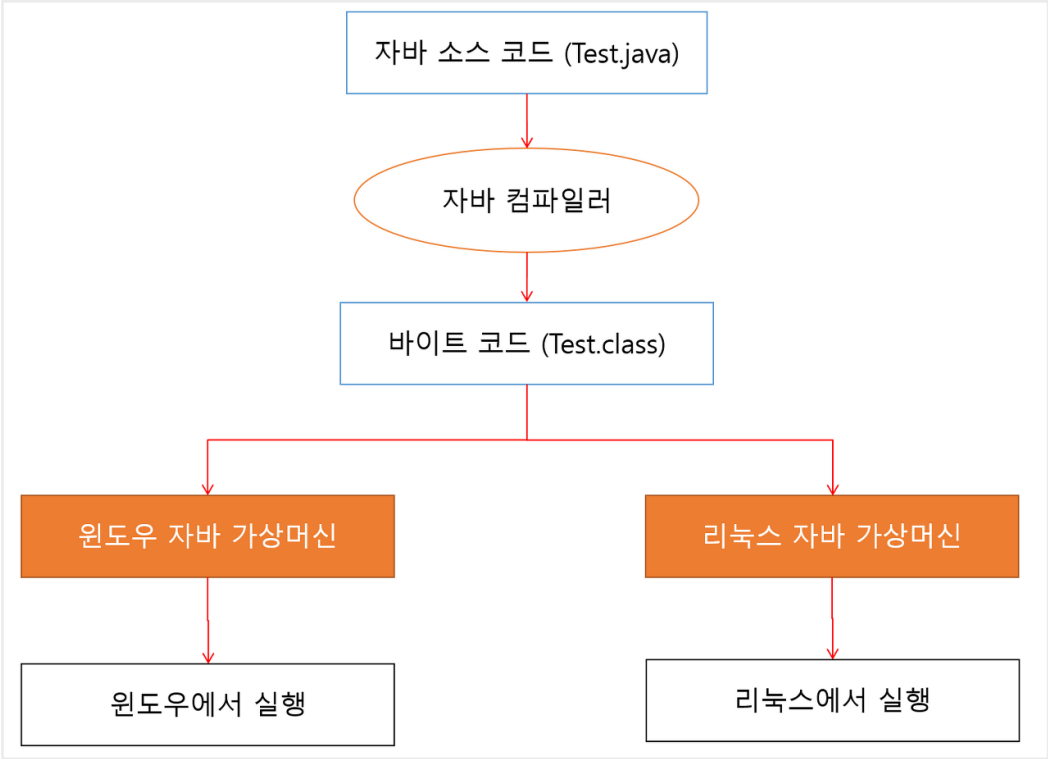
❓ 왜 사용하는지
자바가상머신을 사용하는 가장 큰 이유는 하나의 바이트코드 즉 (.class)파일로 모든 플랫폼에서 작동이 가능해서 사용합니다

아래의 경우는 c언어의 경우입니다.



컴파일 플랫폼과 타겟플랫폼이 다르면 프로그램 실행이 불가능하다는 단점이 있습니다.

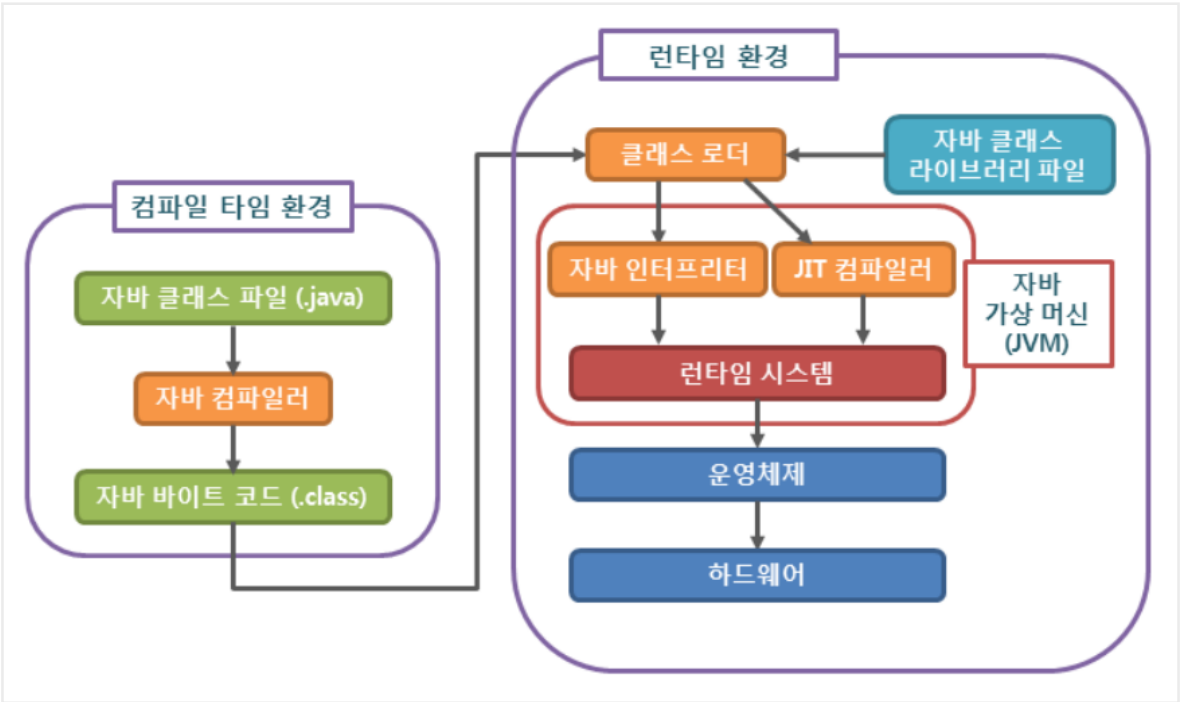
하지만, 자바의 경우



독립적인 자바 컴파일러를 사용하여 타겟플랫폼이 원하는 컴파일 플랫폼으로 가상머신을 생성해줍니다.

덧붙이자면,
“ Java는 플랫폼에 종속적이지 않지만, Jvm은 플랫폼에 종속적” 라고 표현이 가능할 것 같습니다.

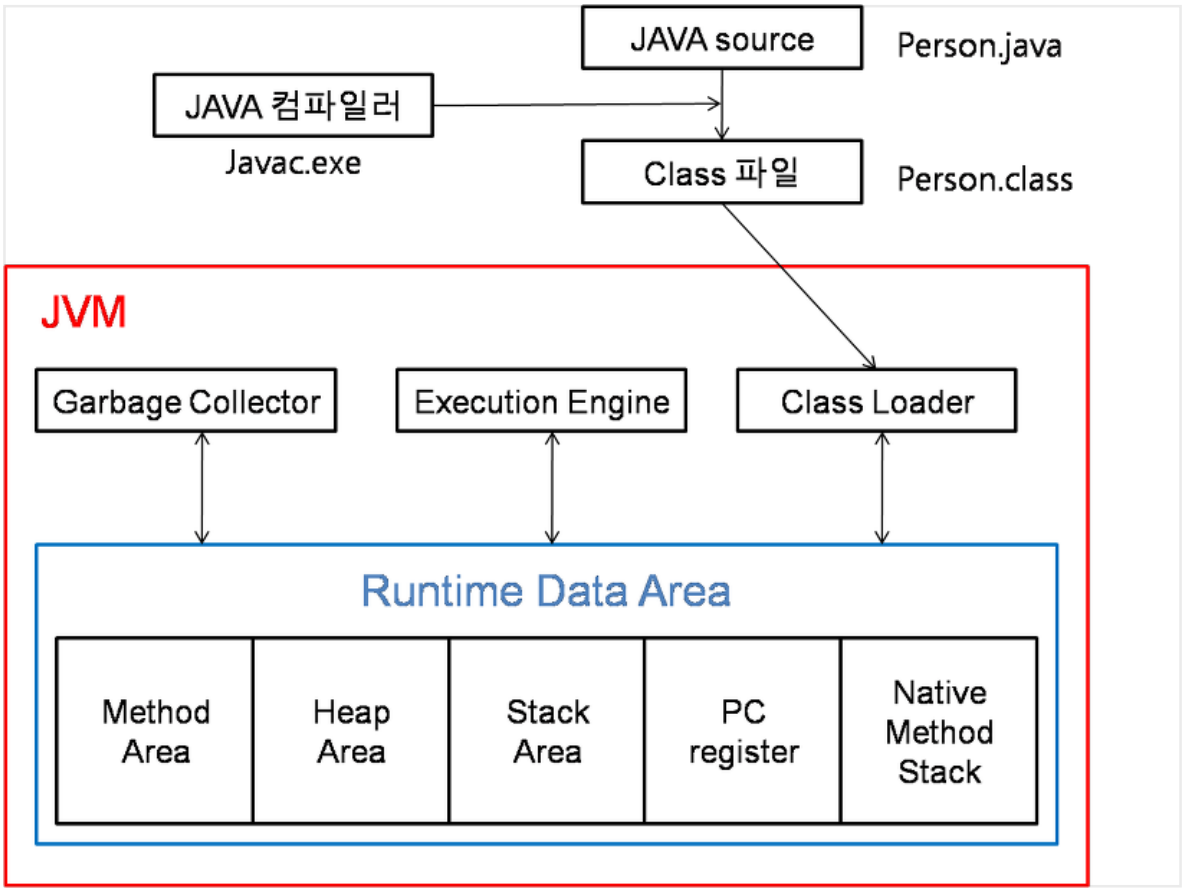
❓ 자바 프로그램의 실행과정



우리가 자바로 .java 코드를 작성하고 터미널에 있는 자바 컴파일러인 javac에

컴파일 명령을 내리면 .class 파일이 만들어집니다. 이후 이 바이트 코드는 클래스 로더를 통해 JVM Runtime Data Area로 로딩되고 로딩된 .class 바이트 코드를 실행할 컴퓨터에 깔린 JVM에 가져다주면 그 컴퓨터가 이 프로그램을 실행할 때 이 JVM이 그때그때 기계어로 해석합니다.

JVM 메모리 구조



Garbage Collector

통칭 GC라고 부르는데 메모리를 사용하고 더이상 사용되지 않는 메모리들을 처리하는 역할을 수행합니다.
보통 new 생성자로 생성된 객체가 더이상 참조되지 않는다면 사용되지않는 통칭 Garbage가 되고 어느 순간에 GC에 의해 처리 됩니다.
주의 해야 할것은 GC가 움직일 때는 GC이외에 쓰레드는 전부 일시 정지합니다.
따라서 최대한 Garbage를 줄이는 것이 좋은 프로그램이라고 할수 있습니다.

Execution Engine

위에서 설명드린 바이트코드(.class)파일을 기계어로 변경하는 작업을 하는 공간입니다.
보통 인터프리터 방식으로 변환하게 되는데 인터프리터 방식 특성상 한줄한줄 해석해야 하기 때문에 실행속도가 매우 느립니다.

그래서 이 단점을 보완한 것이 JIT(Just In Time) 입니다. JIT는 native code로 변환시켜 실행속도를 개선하였지만 비용이 들기 때문에,
인터프리터 방식으로 컴파일 하다가 일정 수치를 넘어간다면 JIT 컴파일러를 사용하게 됩니다.

❓ 바이트 코드를 읽는 방식

Jvm은 바이트코드를 명령어 단위로 해석합니다.
인터프리터 방식과 JIT방식으로 두가지 방식을 혼합하여 해석하게 됩니다.
인터프리터 방식은 바이트 코드를 한줄씩 해석하는 방식이어서 실행속도가 매우 느리다는 단점이 있습니다.
이를 보완하기 위하여 만들어 진것이 JIT입니다.
바이트코드를 JIT 컴파일러를 이용하여 프로그램을 실제 실행하는 시점에 각 OS에 맞는 native code로 변환시켜서 실행속도를 개선 하였습니다.
하지만 JIT 컴파일러가 native code로 변환시키는 것도 비용이 들기 때문에 우선, 인터프리터 방식으로 컴파일 하다가 일정 수치가 넘어가게 되면 JIT 컴파일러로 해석하게 됩니다.

❓ JIT 란?

JIT의 약어는 just in time 입니다.
JIT가 자바인터프리터 방식을 개선하기 위하여 만들어진 것은 위에 설명 했고, 또 JIT를 설명하는 다른 방법은 JIT는 자바인터프리터와 다르게 같은 코드를 매번 해석하지 않고 실행할때 컴파일하면서 해당 코드를 캐싱 해버립니다. 이후에는 변경된 부분만 컴파일 하고 나머지는 캐싱된 코드를 사용합니다. 이렇듯 JIT 컴파일러는 운영체제에 맞는 코드를 변환시켜서 실행하기 때문에 기존에 사용하던 자바인터프리터 방식의 10배~20배 정도 성능 차이를 보입니다.

Runtime Data Area

Runtime Data Area는 애플리케이션이 실행될 때 사용되는 메모리를 적재하는 영역입니다.
1.Method 영역
- Method 영역에서는 static 변수, method, interface, class 등이 생성되는 영역입니다.

2.Heap 영역
- Heap 영역에서는 new 생성자로 생성된 객체가 생성되는 영역입니다. 또한 GC에 의해 참조되지 않은 메모리를 확인하고 처리하는 영역입니다.

3.Stack 영역
- 연산에 필요한 지역변수, 리턴값, 파라미터 등이 생성되는 영역입니다.

4.PCregister
- 쓰레드가 생성될때 생성되며 쓰레드에 각각 존재합니다. 쓰레드가 어느 부분에서 어떤 명령을 실행해야 할지에 대한 기록을 하는 부분으로 수행중인

JVM명령의 주소를 갖습니다.

5.native method stack

- 자바 이외에 언어로 작성된 네이티브 코드를 위한 메모리 영역입니다.