



❖ 자바의 탄생

1. 1995년 썬 마이크로시스템즈의 제임스 고슬링(James Gosling)과 다른 연구원들에 의해 개발된 **객체 지향적 프로그래밍 언어**
2. 처음에는 가전제품 내에 탑재되어 동작하는 프로그램을 위해 개발되었지만 현재 웹 어플리케이션 개발에 가장 많이 사용되는 언어

❖ 자바의 특징

1. 플랫폼 독립적
2. 네트워크와 분산처리 지원
3. 멀티 스레드 지원
4. 동적 로딩 가능
5. 가비지 컬렉터 : 자동 메모리 관리

◆ 컴퓨팅 플랫폼

위키백과, 우리 모두의 백과사전.

컴퓨팅 플랫폼(**영어**: computing platform)은 소프트웨어가 구동 가능한 하드웨어 아키텍처나 소프트웨어 프레임워크(응용 프로그램 프레임워크를 포함하는)의 종류를 설명하는 단어이다.

일반적으로 플랫폼은 컴퓨터의 아키텍처, **운영 체제**(OS), 프로그램 언어, 그리고 관련 런타임 라이브러리 또는 **GUI**를 포함한다.

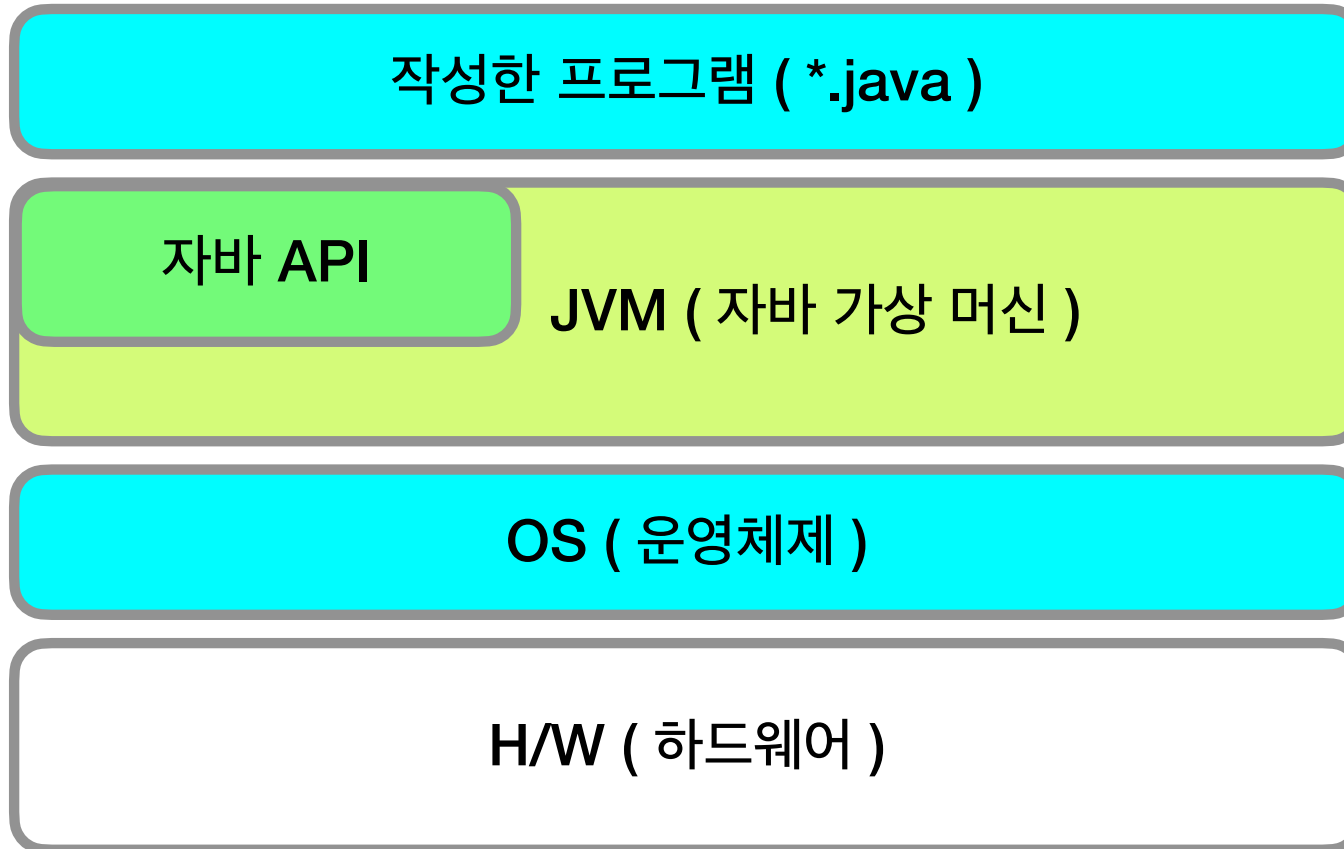
플랫폼은 소프트웨어 응용 프로그램들을 돌리는 데 쓰이는 하드웨어와 소프트웨어의 결합이다. 플랫폼은 하나의 운영 체제 또는 컴퓨터 아키텍처라고 단순히 말할 수 있으며 그 두 가지를 통칭해서 말할 수도 있다.

대중에게 가장 친근한 플랫폼은 x86 아키텍처에서 수행되는 **마이크로소프트 윈도우**다. 잘 알려진 다른 데스크톱 컴퓨터 플랫폼들은 **리눅스**와 **OS X**을 포함한다. 그러나 휴대 전화와 같은 많은 장치들은 효과적으로 컴퓨터 플랫폼이라고도 하지만 보통 그렇게 불리진 않는다.

응용 소프트웨어는 플랫폼에 특화된 하드웨어나 운영 체제, 아니면 **가상 머신**의 기능들에 맞추기 위해 프로그래밍된다. **자바 플랫폼**은 가상 기기 플랫폼으로 여러 운영 체제와 하드웨어에서 실행되며 소프트웨어가 만들어지는 일반적인 플랫폼의 한 종류이다.

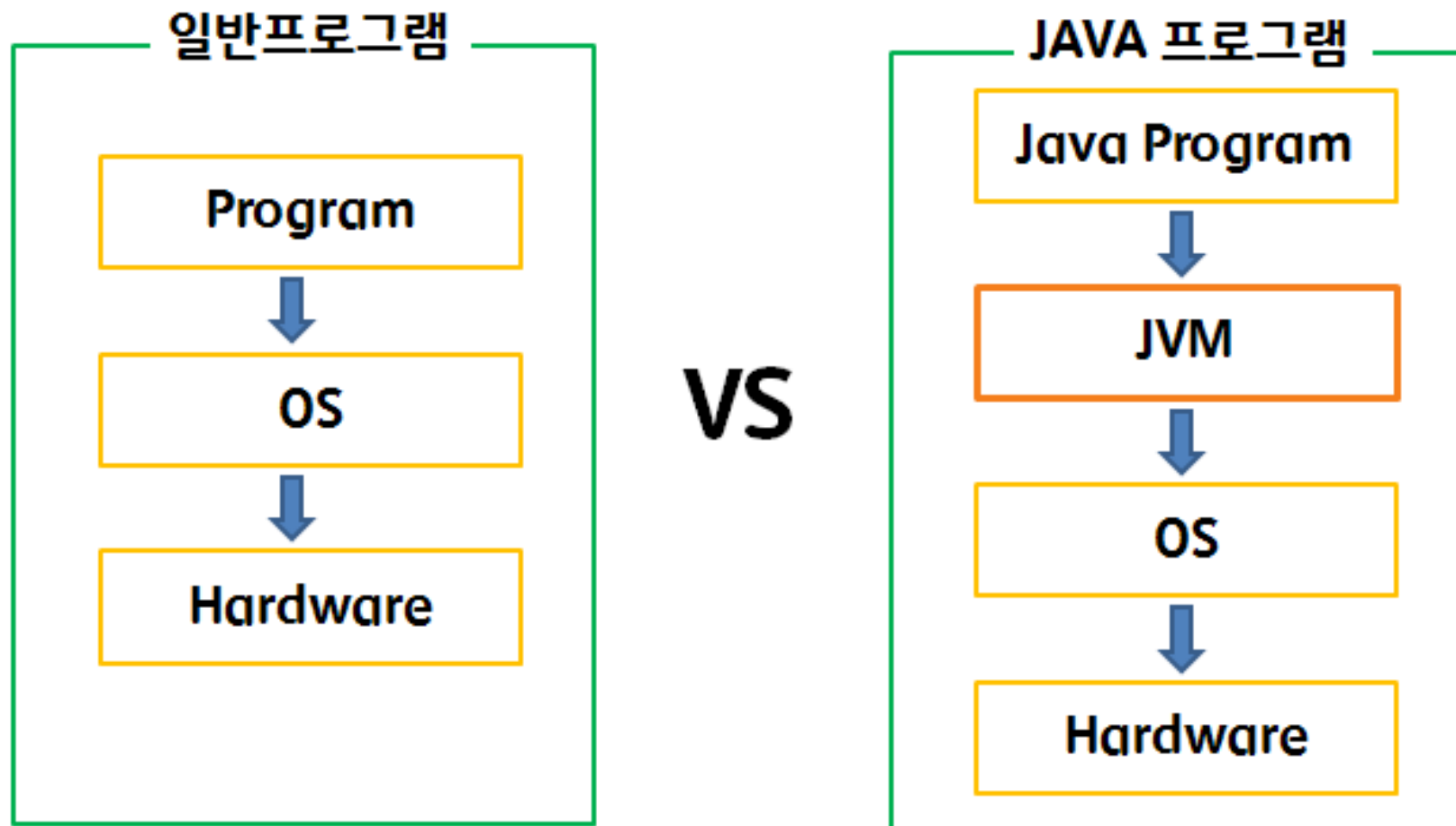
플랫폼은 소프트웨어 개발 중에서도 핵심적이고도 기술적으로 어려운 부분이다. 플랫폼을 간략히 정의 해 보면 소프트웨어를 실행할 수 있는 기반으로 볼 수 있다. 또한 플랫폼은 이를 이용하는 소프트웨어 개발자에게는 다른 어떤 플랫폼 위에서 자신의 로직 코드가 돌아가건 동일하게 작동할 수 있도록 약속 하는 하나의 계약이기도 하다. 로직 코드란 바이트 코드, 소스 코드 그리고 기계 코드도 될 수 있다. 이를 통해 프로그램의 실행이 특정 운영 체제에 제한을 받지 않을 수 있다. 이는 언어 독립적으로, 기계들을 쉽게 교체할 수 있게 한다.

❖ 자바의 플랫폼

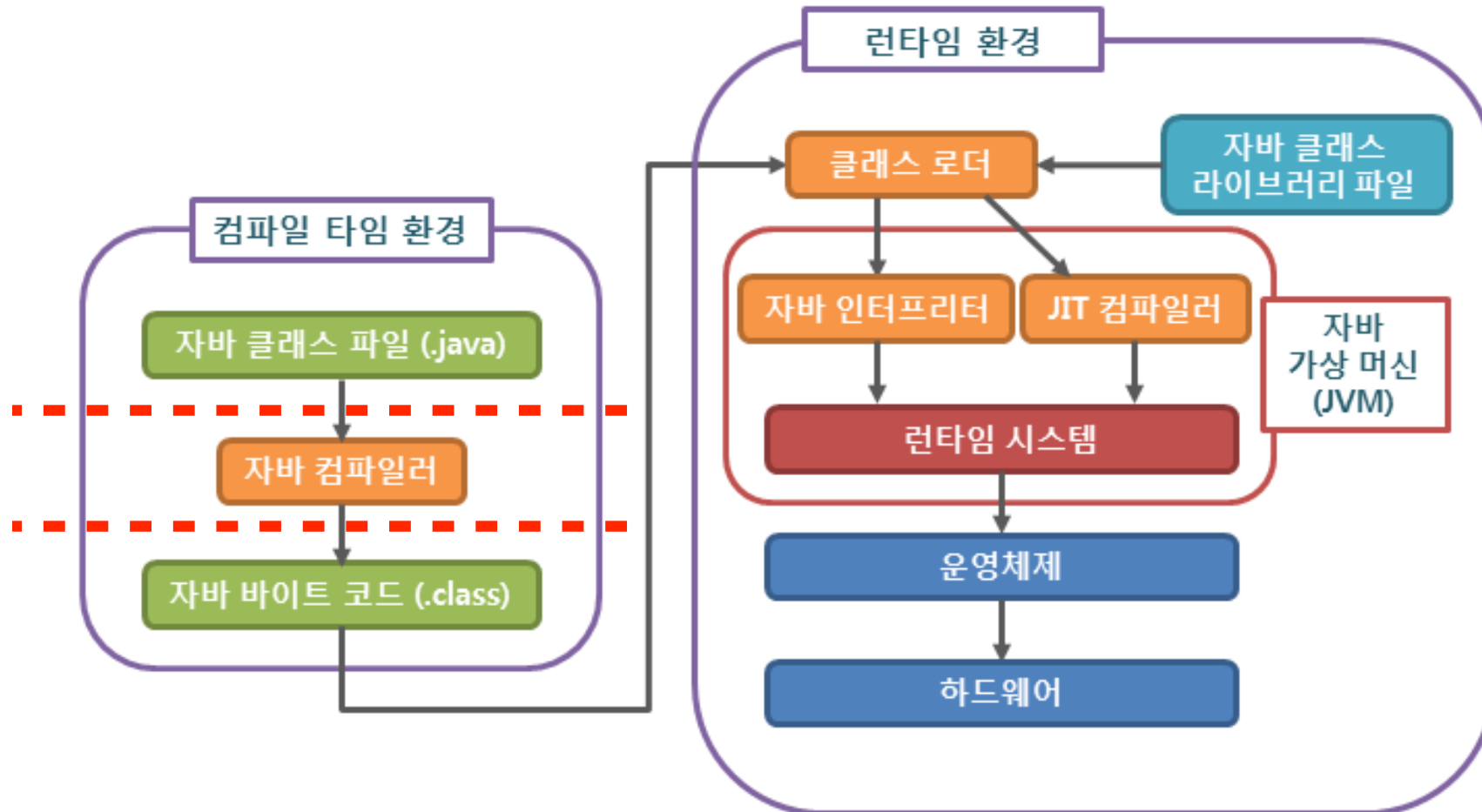


- ◆ 자바 API(Application Programming Interface)
GUI(Graphical User Interface)와 같은 작은 장치들과
유용한 능력을 제공하는 많은 클래스와 인터페이스들의 묶음이며 패키지로 제공
- ◆ JVM(자바가상머신, Java Virtual Machine)

❖ 자바와 일반 프로그램 비교



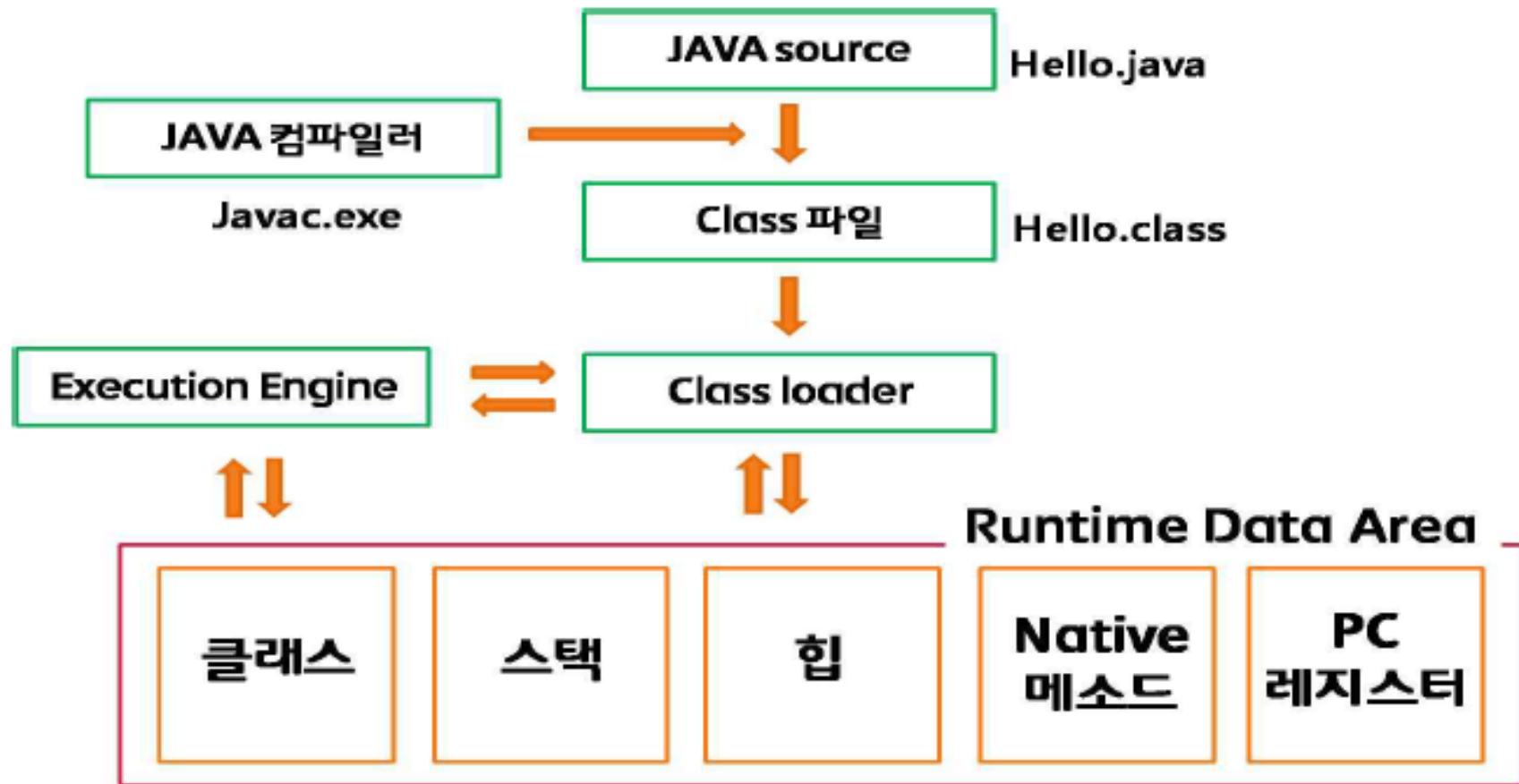
❖ 자바의 실행 단계



❖ JVM(자바가상머신, Java Virtual Machine)

1. 자바 바이트코드를 수행할 수 있는 환경
2. 대부분의 운영체제나 웹 브라우저 등 여러가지 플랫폼에 설치되어 사용될 수 있고, 휴대전화나 가전기기에도 설치가능
3. JVM의 구성
 - ◆ 클래스 영역 : 클래스코드를 저장하는 영역
 - ◆ 자바스택(Java Stack) : 메서드를 호출할 때 관련정보를 저장하는 영역
 - ◆ 힙(Heap) : new라는 키워드를 통해 객체가 생성될 때 할당받는 영역
 - ◆ 네이티브 메서드 스택(Native Method Stack)
 - ◆ PC Register

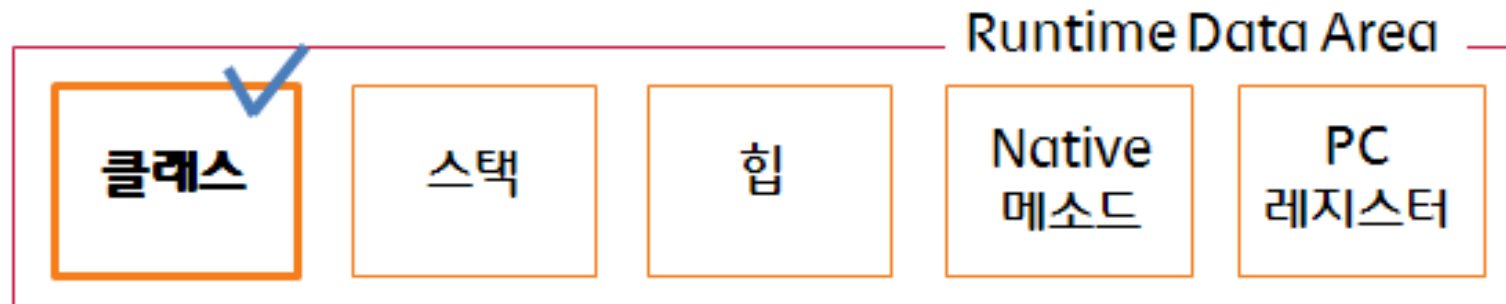
❖ JVM 메모리 구조



- 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

❖ JVM 구성

◆ Class Area



❖ JVM 구성

◆ Class Area

- Method Area, Code Area, Static Area 로 불리어짐

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

ii) Method Information : 메서드의 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

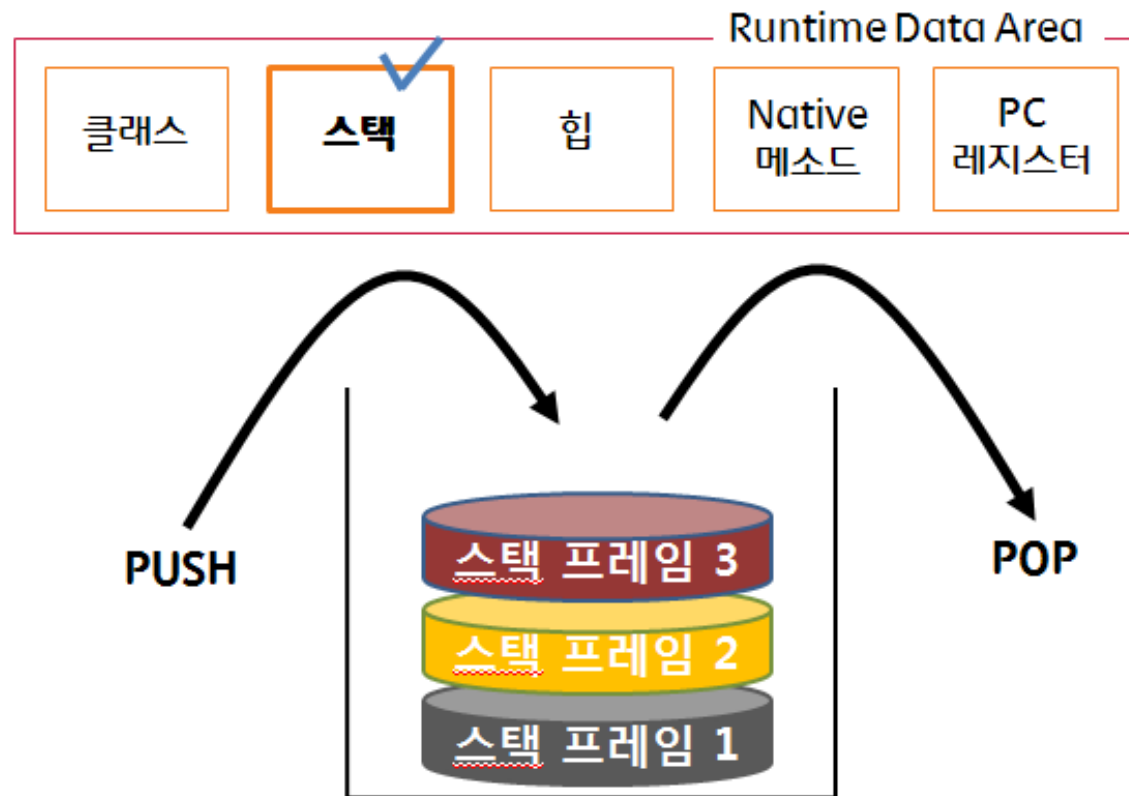
- 모든 객체가 공유 할 수 있고, 객체 생성 없이 접근 가능

vi) Class 사용 이전에 메모리 할당

- final class 변수의 경우(상수로 치환되어) 상수 풀에 값 복사

❖ JVM 구성

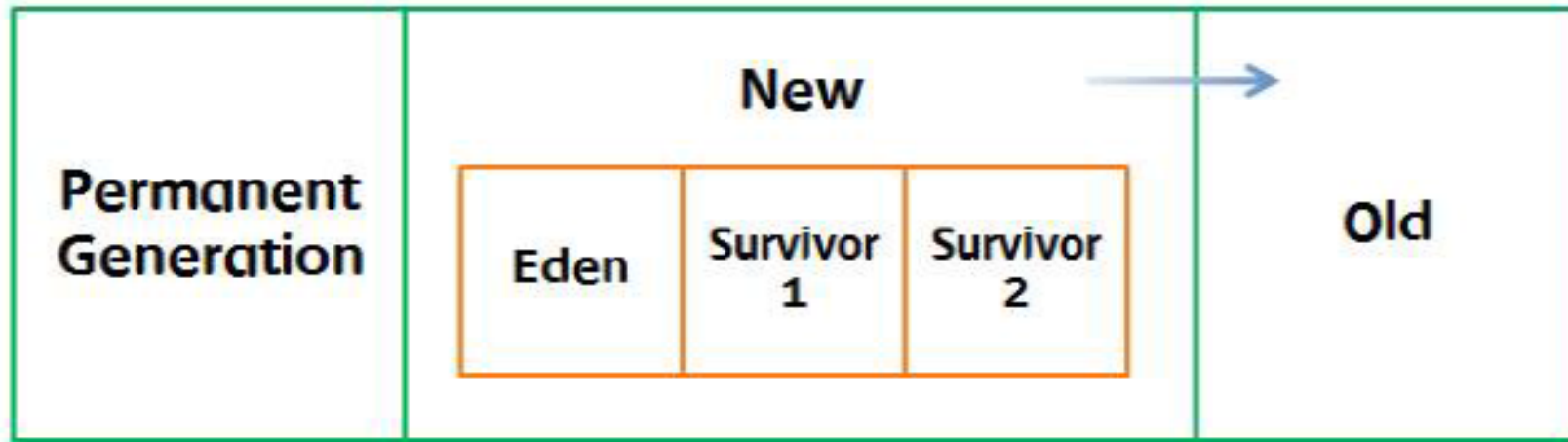
◆ Stack Area



- Last In First Out (LIFO)
- 메서드 호출 시마다 각각의 스택프레임(그 메서드만을 위한 공간)이 생성
- 메서드 안에서 사용되어지는 값들 저장, 호출된 메서드의 매개변수, 지역변수, 리턴 값 및 연산 시 일어나는 값들을 임시로 저장
- 메서드 수행이 끝나면 프레임별로 삭제

❖ JVM 구성

◆ 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

❖ JVM 구성

◆ Native method stack area

- 자바 외의 다른 언어에서 제공되는 메서드들이 저장되는 공간

◆ PC Register

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