훈련교사 : 전 은 석
email : euns_jun@naver.com

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 GUI를 포함한다.

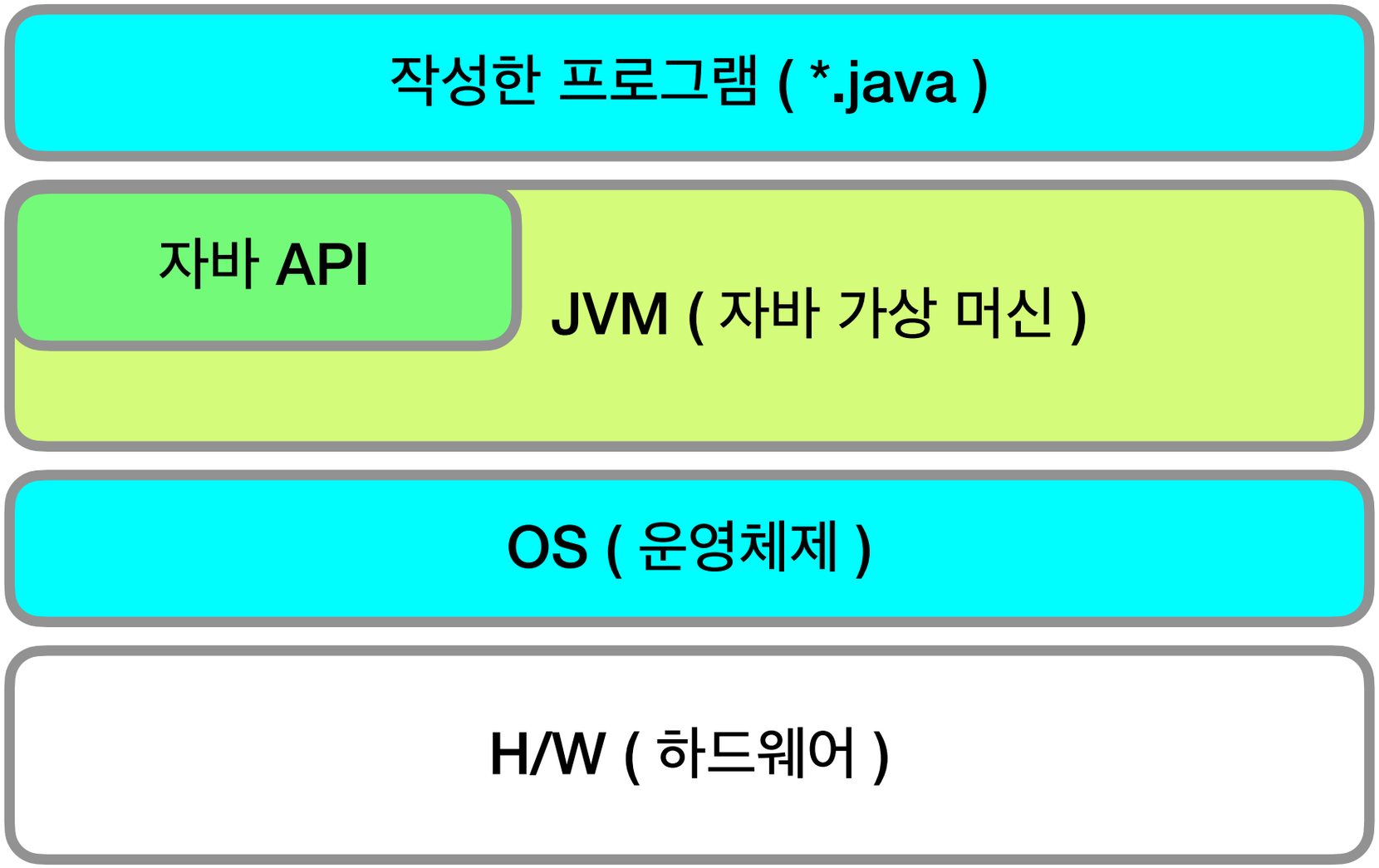
플랫폼은 소프트웨어 응용 프로그램들을 돌리는 데 쓰이는 하드웨어와 소프트웨어의 결합이다. 플랫폼은 하나의 운영 체제 또는 컴퓨터 아키텍처라고 단순히 말할 수 있으며 그 두 가지를 통칭해서 말할 수도 있다.

대중에게 가장 친근한 플랫폼은 x86 아키텍처에서 수행되는 마이크로소프트 윈도우다. 잘 알려진 다른 데스크톱 컴퓨터 플랫폼들은 리눅스와 OS X을 포함한다. 그러나 휴대 전화와 같은 많은 장치들은 효과적으로 컴퓨터 플랫폼이라고도 하지만 보통 그렇게 불리진 않는다.

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

플랫폼은 소프트웨어 개발 중에서도 핵심적이고도 기술적으로 어려운 부분이다. 플랫폼을 간략히 정의해 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 약속하는 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.

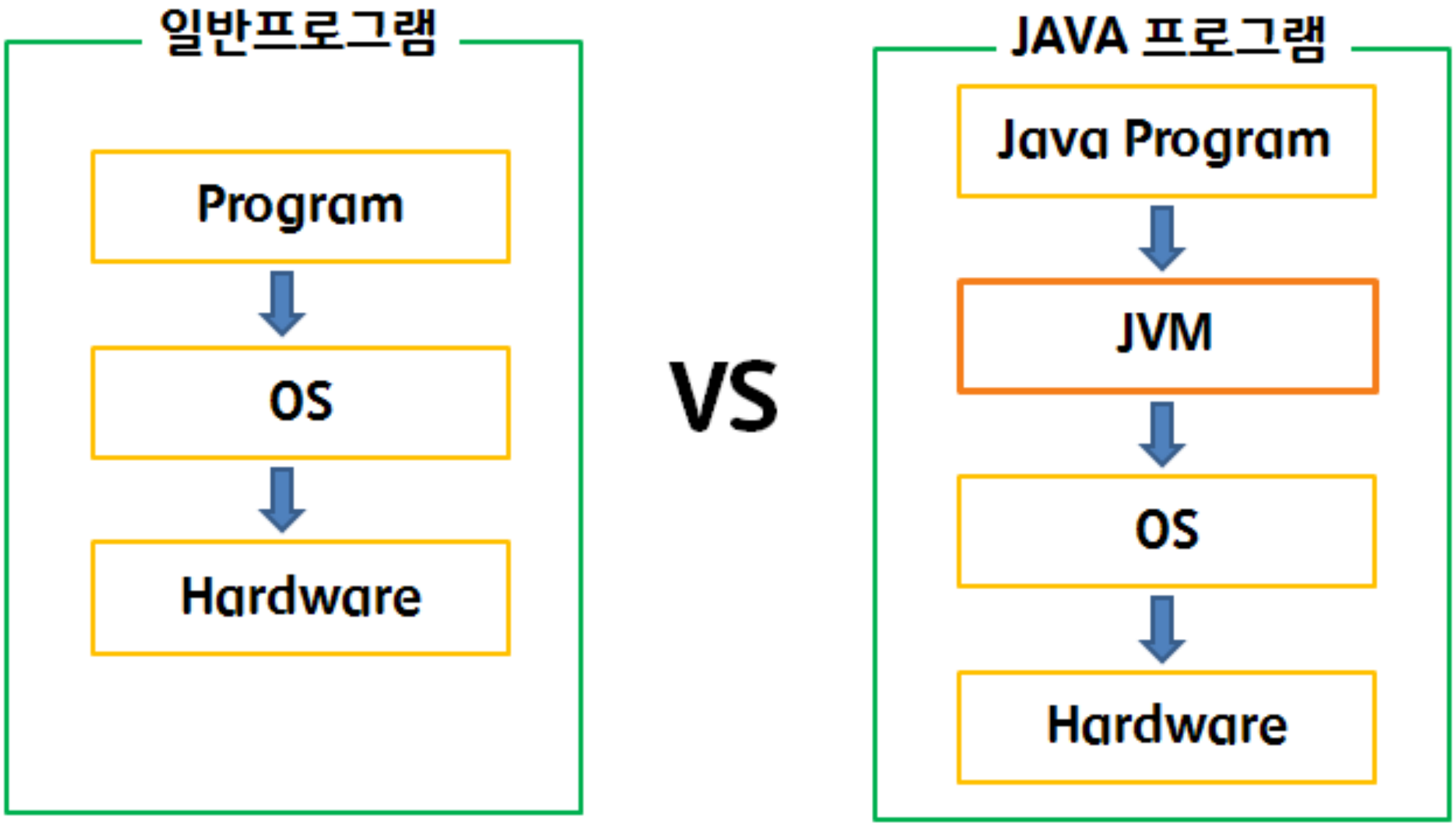
자바의 플랫폼



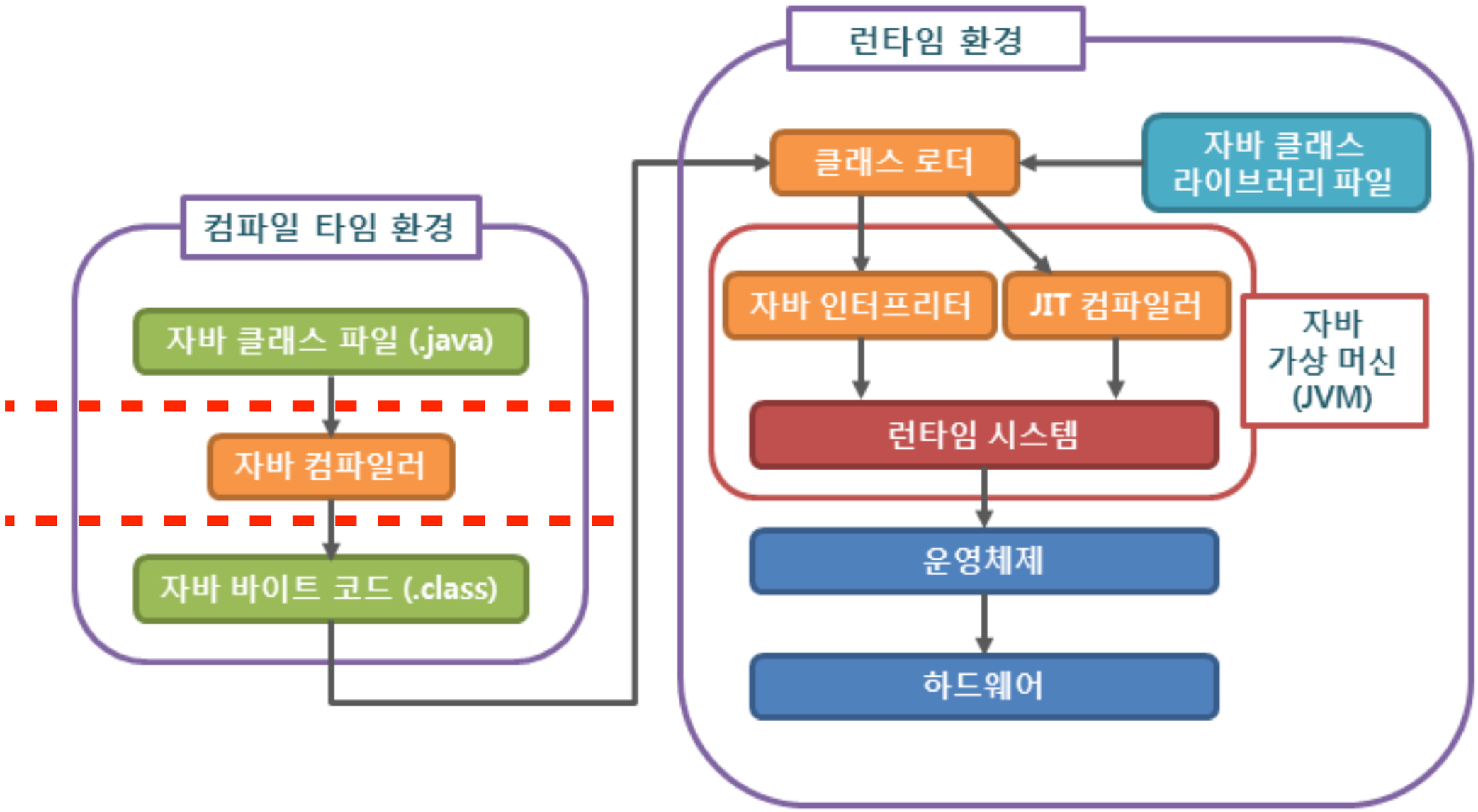
- **자바 API(Application Programming Interface)**
  GUI(Graphical User Interface)와 같은 작은 장치들과
  유용한 능력을 제공하는 많은 클래스와 인터페이스들의 묶음이며 패키지로 제공

- **JVM(자바가상머신, Java Virtual Machine)**

❖ 자바와 일반 프로그램 비교



자바의 실행 단계


컴파일 타임 환경
자바 클래스 파일 (.java)
자바 컴파일러
자바 바이트 코드 (.class)
런타임 환경
클래스 로더
자바 클래스
라이브러리 파일
자바 인터프리터
JIT 컴파일러
자바
가상 머신
(JVM)
런타임 시스템
운영체제
하드웨어

JVM(자바가상머신, Java Virtual Machine)

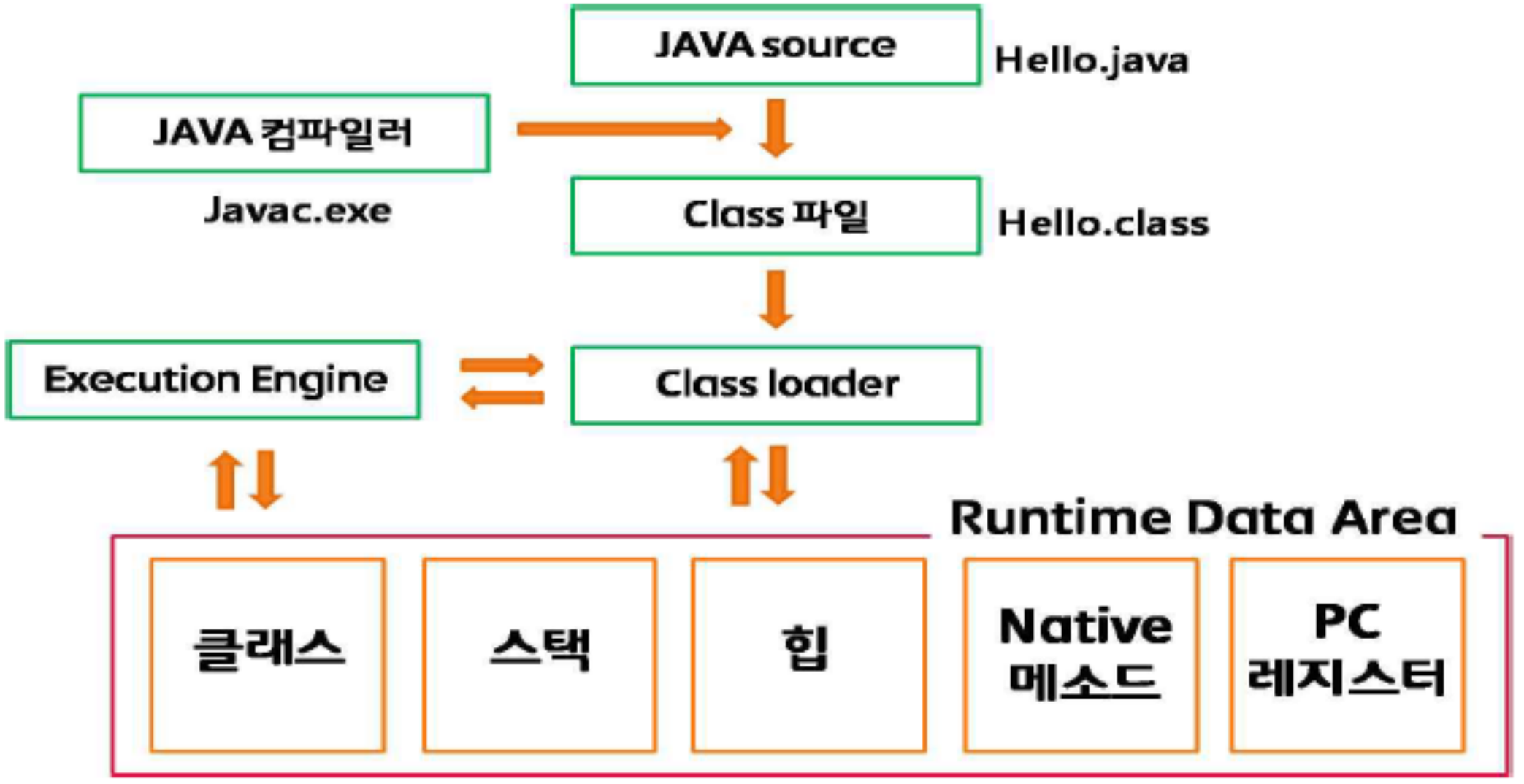
1. 자바 바이트코드를 수행할 수 있는 환경

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 스텍(Native Method Stack)
- PC Register

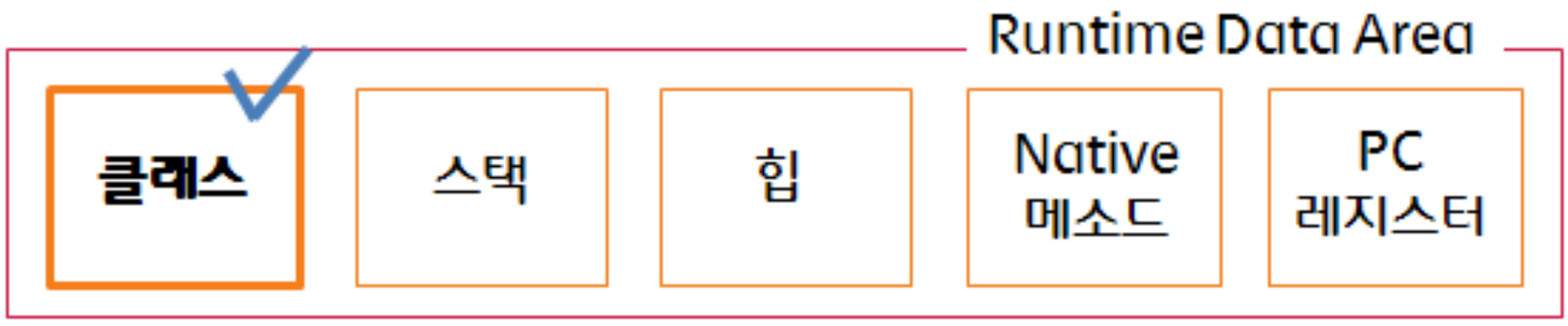
❖ JVM 메모리 구조

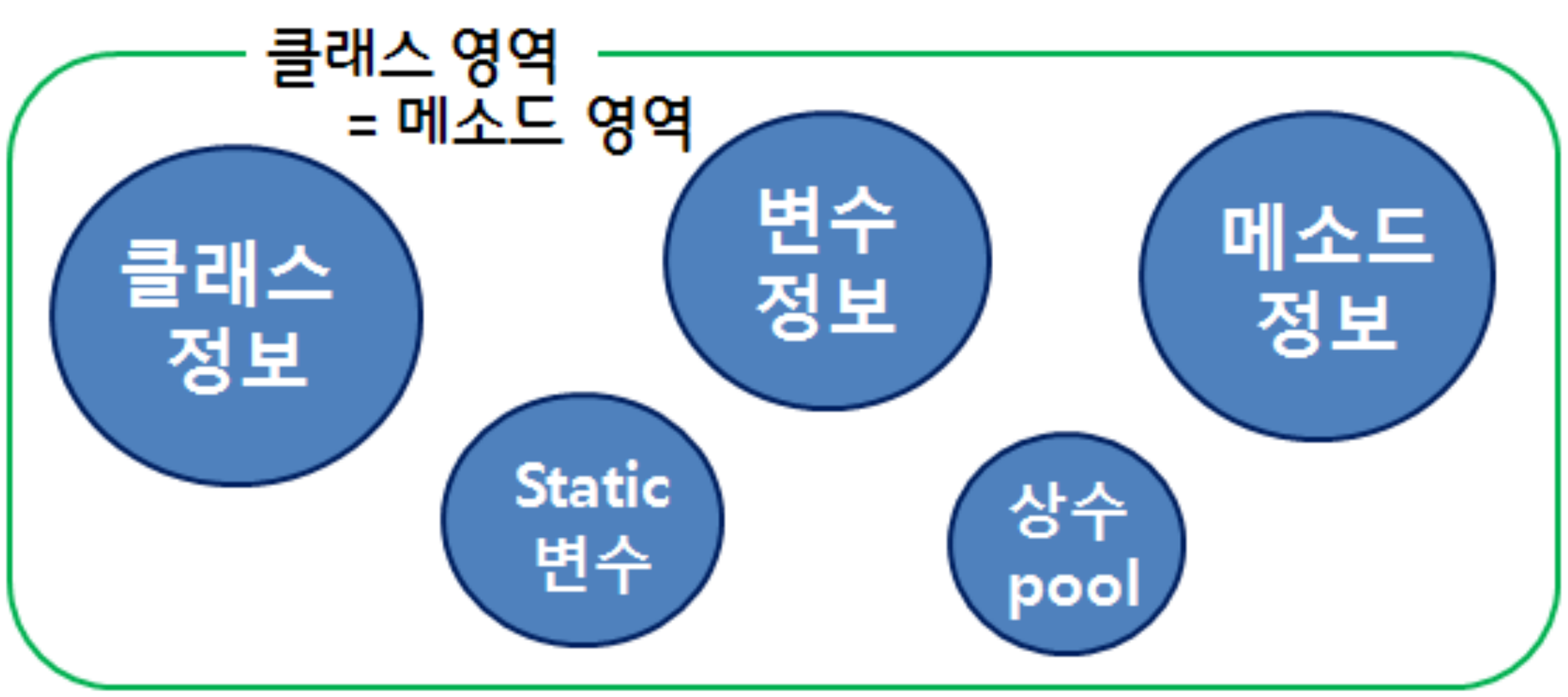


- JAVA Source : 사용자가 작성한 JAVA 코드
- JAVA Compiler : JAVA 코드를 Byte Code로 변환시켜주는 기능
- Class Loader : Class파일을 메모리(Runtime Data Area)에 적재하는 기능
- Execution Engine : Byte Code를 실행 가능하게 해석해주는 기능
- Runtime Data Area : 프로그램을 수행하기 위해 OS에서 할당 받은 메모리 공간

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

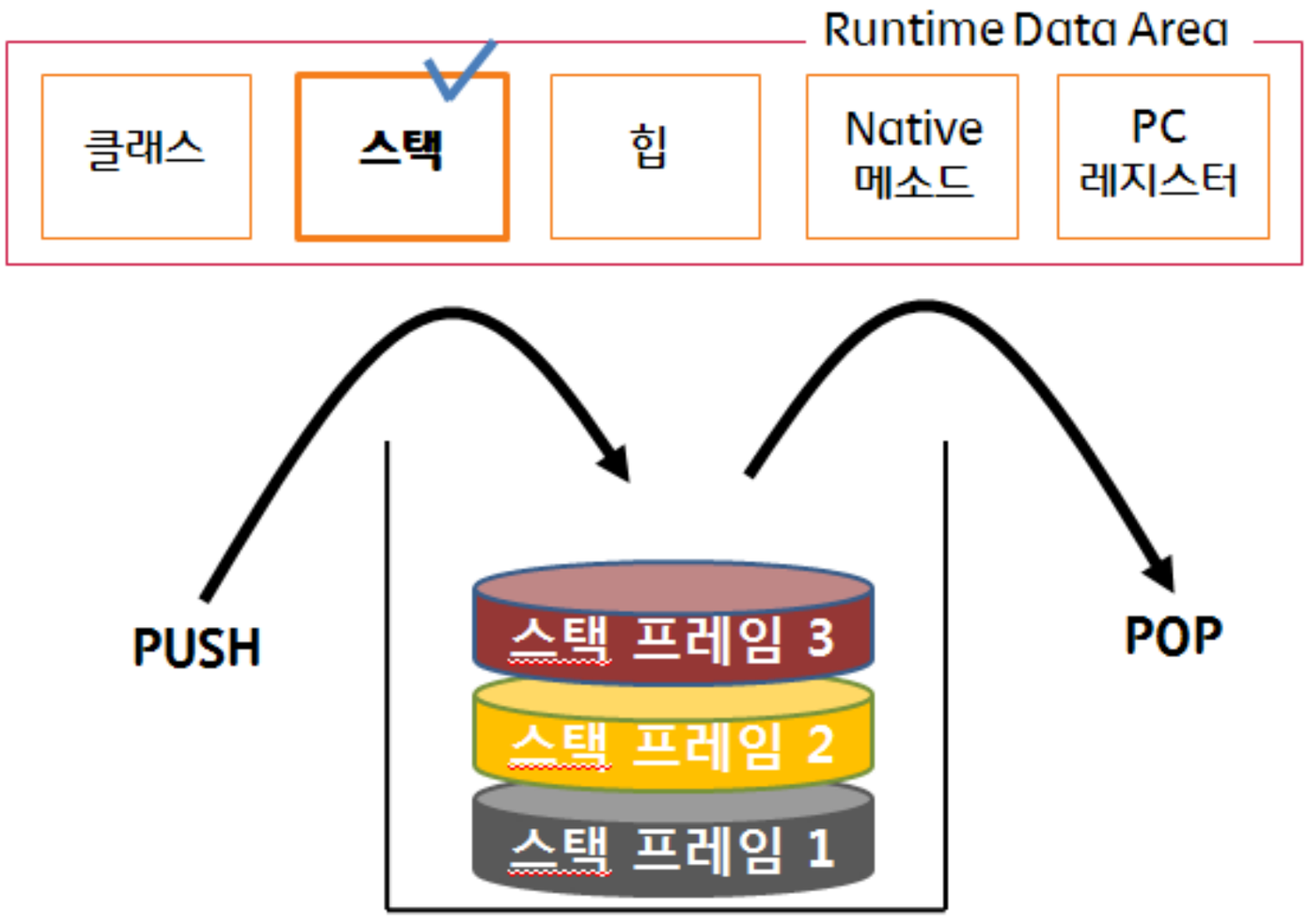
iv) 상수 풀(Constant Pool)
- Type에서 사용된 상수를 저장하는 곳(중복이 있을 시 기존의 상수 사용)
- 문자 상수, 타입, 필드, Method의 symbolic reference(객체 이름으로 참조하는 것)도 상수 풀에 저장

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

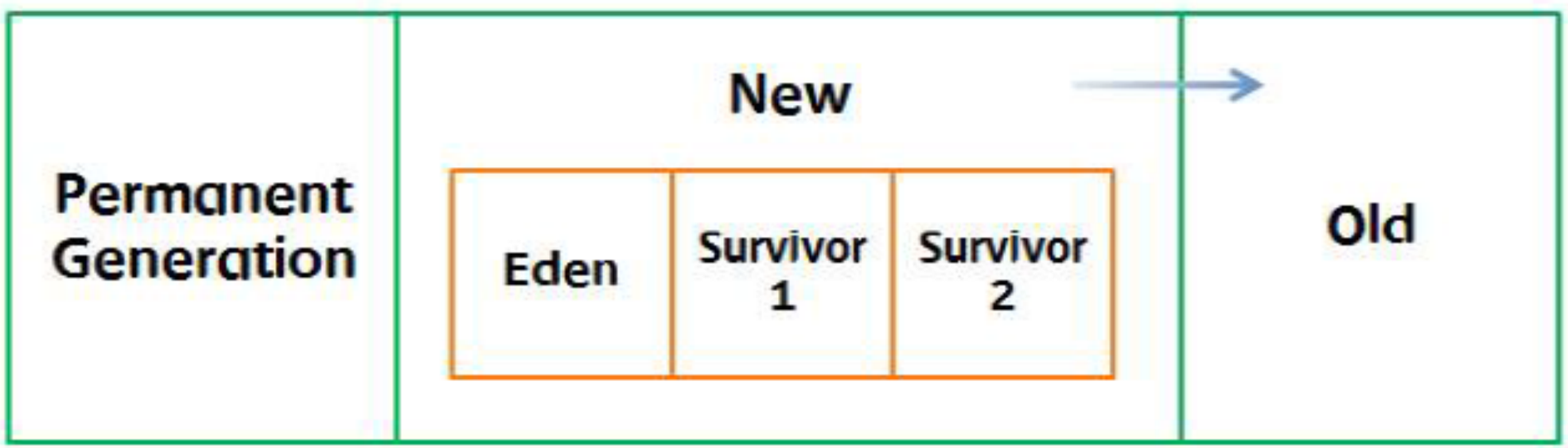
Stack Area



- 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

콘솔창에 출력하기

- System.out.println(내용);
  - 내용을 출력하고 줄바꿈이 된다.

- System.out.print(내용);
  - 내용을 출력하고 줄바꿈이 안된다.

- System.out.printf("형식형식형식 ... ", 내용, 내용, 내용, ...);
  - 여러 내용을 지정한 형식으로 출력한다.

형식화문자	내 용
%d	정수값을 10진수로 출력
%o	정수값을 8진수로 출력
%x	정수값을 16진수로 출력
%f	실수값을 소수 방식으로 출력
%e	실수값을 지수 방식으로 출력
%c	문자를 출력
%s	문자열을 출력
%b	논리값을 출력
%n	줄바꿈

예제

%10d

: 10진수를 10자리에 맞춰 출력

%10.4f

: 실수를 10자리에 맞게 출력하는데 소수이하는 4자리를 출력

```
System.out.printf(
        "%5d,%n%5d,%n%5d",
        168 & 245,
        168 | 245,
        168 ^ 245);
```

❖ 자바 프로그램에서 사용하는 단어의 종류

예약어(키워드), 상수, 식별자

◆ 식별자

식별(識別)자란?

- 보고 느낄 수 있는 모든 사물(객체)들을 각각 구별할 수 있는 것을 의미 한다.
- 클래스, 메소드, 변수, 상수에 프로그래머가 정하는 이름
  (프로그래밍에서 사용하는 모든 이름)

* 대소문자가 구분되며 길이에 제한이 없다. (Abcd와 abcd는 다른것으로 간주)
* 숫자로 시작해서는 안된다.
* 식별자에는 예약어(키워드), 상수를 쓸수 없다.
* 식별자가 겹쳐져서 구분해줘야 할 경우 마침표(.)로 연결해서 소속을 표시

◆ 상수 : true, false, null

◆ 예약어 (키워드) : 이미 약속된 의미를 갖는 단어

abstract	assert	boolean	break	byte	case	catch	char	class	const
continue	default	do	double	else	enum	extends	final	finally	float
for	goto	if	implements	import	instanceof	int	interface	long	native
new	package	private	protected	public	return	short	static	strictfp	super
switch	synchronized	this	throw	throws	transient	try	void	volatile	while

클래스 정의 규칙

- 첫 글자는 대문자로 한다.
- 두 단어 이상으로 이루어지는 경우 각 단어의 첫글자는 대문자로 한다.(Camel 표기법)
- 한글도 쓸수 있다.(가능하면 피할 것)
- 특수 문자는 _, $ 는 쓸 수 있다(첫 글자에도 가능). 나머지는 쓸 수 없다.
- 첫 글자로 숫자가 올 수 없다.

변수, 메소드 정의 규칙

- 첫 글자는 소문자로 한다.
- 두단어 이상으로 이루어지는 경우 두번째 단어 이후 첫글자는 대문자로 한다. (Camel 표기법)

상수 정의 규칙

- 모든 문자는 대문자를 사용한다.
- 두 단어 이상이 될 경우 각 단어를 언더 스코어(_) 로 연결한다.

자바에서의 기호들

- 소괄호, 중괄호, 대괄호 (( ), { }, [ ]) : 영역을 구분
- 작은 따옴표 : 문자를 표시
- 큰 따옴표 : 문자열을 표시
- 세미콜론(;) : 명령문의 끝을 표시하는 기호

❖ 자바 프로그램의 구조

자바프로그램은 반드시 모든 내용이 하나의 class 안에 포함되어야 한다.

◆ class를 만드는 방법

```
[접근지정자] [속성] class 클래스이름 {

}
```

- 예제

```
public class Test {

}
```

* 형식에 있어서 [ ] 안의 내용은 생략해도 무방하다.

훈련교사 : 전 은 석
email : euns_jun@naver.com

◈ 파일이름 지정하는 방법

1. 하나의 파일에는 하나의 class 만 만드는 것을 원칙으로 한다.
그러므로 파일의 이름은 class 의 이름과 동일하게 만들어야 한다.

2. 어쩔수 없이 하나의 파일에 두개 이상의 class를 만들게 되는 경우
   1) 그중에 public 클래스의 이름으로 정한다.
   (하나의 파일에는 public 클래스는 0개 ~ 1개까지만 허락한다.)
   2) 만약 public class가 없으면
      a. main 함수가 있는 클래스이름으로 한다.
      b. main 함수가 없거나 main() 가 여러개 있으면
      아무 클래스이름으로 하면된다.

훈련교사 : 전 은 석
email : euns_jun@naver.com

진입점 함수

클래스가 실행되기 위해서는 JVM에 의해서 실행되는 함수가 있어야 한다.
JVM은 오직 "public static void main(String[] args)" 함수만 사용한다.

따라서 만약 어떤 클래스를 만들고 그 클래스가 실행되기 위해서는 반드시
public static void main(String[] args)가 존재해야만 한다.

이처럼 어떤 프로그램을 시작하는 함수를 **진입점 함수**라고 부른다.

* class 안에는 필요한 함수나 필요한 변수를 포함할 수 있다.

❖ 자료형

◆ 기본 자료형 (Primitive Data Type)

자바에서 사용하는 기본 데이터 타입

* : 숫자형 기본 입력 타입

구 분	자료형	byte	범 위
논리형	boolean	1	true, false
문자형	char	2	0 ~ 2^{16} - 1(0 ~ 65535)
정수형	byte	1	-128 ~ 127 (-2^7 ~ 2^7 - 1)
	short	2	-32,768 ~ 32,767 (-2^{15} ~ 2^{15} - 1)
	int *	4	-2147483648 ~ 2147483647 (-2^{31} ~ 2^{31} - 1)
	long	8	-9223372036854775808 ~ 9223372036854775807 (-2^{63} ~ 2^{63} - 1)
실수형	float	4	1.4E-45 ~ 3.4028235E38
	double *	8	4.9E-324 ~ 1.79769313486231570E308

* 실수형은 양의 값만 표시 됨

변수

- 아직 알려지지 않거나 어느 정도까지만 알려져 있는 양이나 정보에 대한 상징적인 이름
- 다른 언어에서는 스칼라(scalar)라 부르기도 한다.
- 데이터를 담는 일종의 그릇

종류

- 기본형 (Primitive Type)
  - 기본 자료형 8개를 저장
  - 실제 값이 저장
- 참조형(Reference Type)
  - 객체 주소를 저장
  - 기본형을 제외한 나머지 타입
  - 변수의 타입으로 클래스의 이름을 사용

형식

- 기본자료형 변수이름; → 변수 선언
- 기본자료형 변수이름 = 데이터; → 변수 선언 & 초기화
- 클래스이름 변수이름;
- 클래스이름 변수이름 = new 클래스이름();

상수

- 알려져 있는 양이나 정보에 대한 상징적인 이름
- 한번 정하면 수정할 수 없다.
- 상수명은 대문자로 한다.
- 상수는 선언과 동시에 초기화 해야한다.(값을 변경할 수 없으므로...)

◆ 형식

- final 데이터타입 상수명(대문자) = 데이터; → 상수 선언 & 초기화

변수 선언의 의미

변수 선언은 단순히 데이터를 담는 것만이 아니다.
데이터의 형태와 데이터의 크기, 필요한 메모리의 크기도 지정한다는 의미

int num ; => num이라는 변수에
int 형의 데이터 &
4바이트의 데이터 &
메모리 4바이트 할당받는다.

❖ 형변환

값의 자료형을 원하는 자료형으로 변환하는 작업

❖ 리터럴 형변환

데이터를 입력하면 리터럴 영역에 저장이 먼저 된다.
이때 테이터의 타입이 자동으로 결정되고 크기도 정해진다. 이것을 리터럴 타입이라 한다.
정수는 int로 자동 결정되고 실수는 double 타입으로 자동으로 결정이 된다.

이때 메모리의 크기를 정해줄 수도 있는데 이때 쓰는 방법이 리터럴 형변환이다.

＊ 방법

입력할 데이터뒤에 알파벳을 붙여준다.

- 데이터L; → long타입
- 데이터F; → float타입

📌 정수형 데이터는 리터럴 형변환 없이 사용할 수 있다.

❖ 형변환

❖ 자동 형변환

지정하지 않아도 자동적으로 형태를 바꿔서 사용되는 경우
: 작은 형태의 데이터가 큰 형태의 데이터로 필요한 경우 자동 형변환이 일어난다.

예제

```
float num1 = 10;     // 성공
float num2 = 10.;    // 실패
```

❖ 강제 형변환

자동 형변환이 불가능한 경우 개발자가 강제로 형 변환을 할 필요가 있다.

◆ 형식

```
(변환할 데이터형) 데이터;
```

예제

```
(float) 10.0;
```

❖ 연산자

연산자란
자료의 가공을 위해 정해진 방식에 따라 계산하고 결과를 얻기 위한 행위를 의미하는 기호들의 총칭이다.
각 연산자들은 연산을 하기 위해 인식하는 자료형들이 정해져 있다.

❖ 연산자의 종류와 우선순위

종류	연산자	우선순위
증감 연산자	++, --	1순위
산술 연산자	+, -, *, /, %	2순위
시프트 연산자	〉〉, 〈〈, 〉〉〉	3순위
비교 연산자	〉, 〈, 〉=, 〈=, = =, !=	4순위
비트 연산자	&, |, ^, ~	~만 1순위, 나머지는 5순위
논리 연산자	&&, ||, !	!만 1순위, 나머지는 6순위
조건(삼항) 연산자	?, :	7순위
대입 연산자	=, *=, /=, %=, +=, -=	8순위

❖ 대입 연산자

특정한 상수 값이나 변수 값 또는 객체를 변수에 전달하여 기억시킬 때 사용하는 연산자이다

❖ 대입 연산자의 종류

구분	연산자	의미
대입 연산자	=	연산자를 중심으로 오른쪽 변수값을 왼쪽 변수에 대입한다.
	+=	왼쪽 변수에 더하면서 대입한다.
	-=	왼쪽 변수값에서 빼면서 대입한다.
	*=	왼쪽 변수에 곱하면서 대입한다.
	/=	왼쪽 변수에 나누면서 대입한다.
	%=	왼쪽 변수에 나머지 값을 구하면서 대입한다.

예제

```
int no1 = 10;
no1 += 10;
System.out.println( no1 );
```

❖ 산술연산자

4칙 연산(+, -, *, /)과 나머지 값을 구하는 연산자(%)

❖ 산술연산자의 종류

구분	연산자	의미
산술 연산자	+	더하기
	-	빼기
	*	곱하기
	/	나누기
	%	나머지 값 구하기

예제

```
int no1 = 10 / 3 ;
System.out.println("no1 = " + no1);
```

```
int no1 = 10%3;
System.out.println("no1 = " + no1);
```

❖ 증감연산자

1씩 증가 또는 감소시키는 연산자이다.
무엇보다 중요한 것은 ++ 또는 --와 같은
연산자가 변수 앞에 위치하느냐? 아니면 변수 뒤에 위치하느냐?가 더 중요한 연산자이다.

❖ 증감연산자의 종류

구분	연산자	의미
증감 연산자	++	1씩 증가시킨다.
	--	1씩 감소시킨다.

예제

```
int no1 = 10;
System.out.println("no1 = " + (++no1));
```

```
int no1 = 10;
System.out.println("no1 = " + (no1++));
System.out.println("no1 = " + no1);
```

예제

```
int     no1 = 10;
int     no2 = no1++ + no1++;
System.out.println( no2 );
```

예제

```
int no1 = 10;
int no2 = ++no1 + no1++;

System.out.println( no2 );
```

❖ 비교 연산자

변수나 상수의 값을 비교할 때 쓰이는 연산자로서
결과가 항상 true 또는 false인 논리값(boolean)이어야 한다.

❖ 비교 연산자의 종류

구분	연산자	의미
비교 연산자	〉	크다.
	〈	작다.
	〉=	크거나 같다.
	〈=	작거나 같다.
	= =	피연산자들의 값이 같다.
	!=	피연산자들의 값이 같지 않다.

예제

```
System.out.println(1 > 2 );
```

❖ 논리 연산자

true나 false인 논리 값을 가지고 다시 한번 조건 연산하는 연산자이다.
하나 이상의 처리 조건이 있어야 하며 먼저 처리되는 조건에 따라
다음의 처리 조건을 처리할지 안 할지를 결정하는 말 그대로 논리적인 연산자이다.

❖ 논리 연산자의 종류

연산자	의미	설명
&	and(논리곱)	주어진 조건들이 모두 true일 때만 true를 나타낸다.
&&		
|	or(논리합)	주어진 조건들 중 하나라도 true이면 true를 나타낸다.
||		
!	not(부정)	true는 false로 false는 true로 나타낸다.

✻ **&& 와 || 는 절삭 연산을 한다.**

연산자	설명
&&	선조건이 true일 때만 후조건을 실행하며 선조건이 false이면 후조건을 실행하지 않는다.
||	선조건이 true이면 후조건을 실행하지 않으며 선조건이 false일 때만 후조건을 실행한다.

예제

```
System.out.println(true & false );
System.out.println(true || false );
```

예제

```
int no1 = 10;
int no2 = 11;
int no3 = 11;

System.out.println(no1++ > no2 && no1 < no3 );
```

❖ 시프트 연산자

시프트 연산은 대상 필드의 값을 비트로 바꾼 후 비트 수만큼 이동시켜서 값을 얻는 연산이다.
이연산은 boolean, float, double 형의 경우는 사용할 수 없다.

많은 CPU에서는 상수 곱셈 등의 연산을 시프트 연산자로 처리하는 것이
산술 논리 장치를 거치는 것보다 빠르기 때문에,
컴파일러에서는 곱셈/나눗셈 연산을 자동적으로 산술 시프트 명령어로 변환한다.

❖ 시프트 연산자의 종류

구분	연산자	의미
시프트 연산자	>>	bit값을 오른쪽으로 이동(빈 자리는 부호값으로 대입)한다.
	<<	bit값을 왼쪽으로 이동(빈 자리는 0으로 대입)한다.
	>>>	bit값을 오른쪽으로 이동(빈 자리는 0으로 대입)한다.

예제

```
System.out.println(11 << 2 );
```

← 11×2^2

❖ 비트 연산자

피연산자 즉 연산의 대상이 되는 값들을
내부적으로 bit단위로 변경한 후 연산을 수행하는 연산자이다.

❖ 비트 연산자의 종류

구분	연산자	의미
비트 연산자	&	비트 단위의 AND
	|	비트 단위의 OR
	^	XOR(배타적 OR)

예제

```
int no1 = 168 ;
int no2 = 245 ;
System.out.println( no1 & no2);
System.out.println( no1 | no2);
System.out.println( no1 ^ no2);
```

	128	64	32	16	8	4	2	1	
no1	1	0	1	0	1	0	0	0	168
no2	1	1	1	1	0	1	0	1	245
&	1	0	1	0	0	0	0	0	160
|	1	1	1	1	1	1	0	1	253
^	0	1	0	1	1	1	0	1	93

조건 연산자 (삼항 연산자)

하나의 조건을 정의하여 만족 시에는 '참값'을 반환하고 만족하지 못할 시에는 '거짓값'을 반환하여 단순 비교에 의해 변화를 유도하는 연산자이다.

형식

```
조건식 ? 참일때 반환값 : 거짓일때 반환값 ;
```

예제

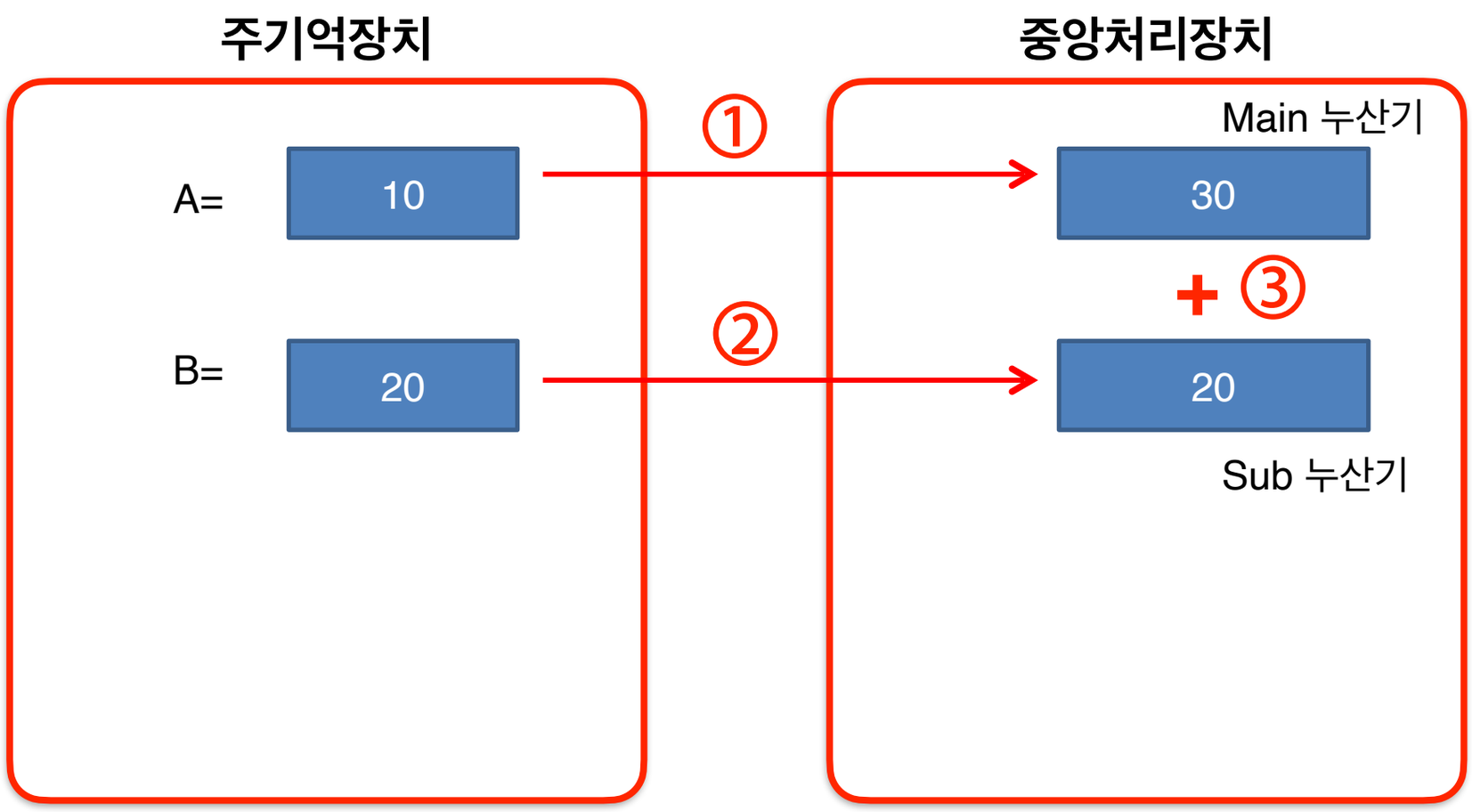
```
int no1 = 10;
int no2 = 20;
String str1 = ( no1 > no2 ) ? ("no1이 no2보다 크다.") : ("no1이 no2 보다 작다.");
```

랜덤한 숫자 얻는 방법

```
int num = (int) Math.random() * (큰수 - 작은수 +1) + 작은수;
```

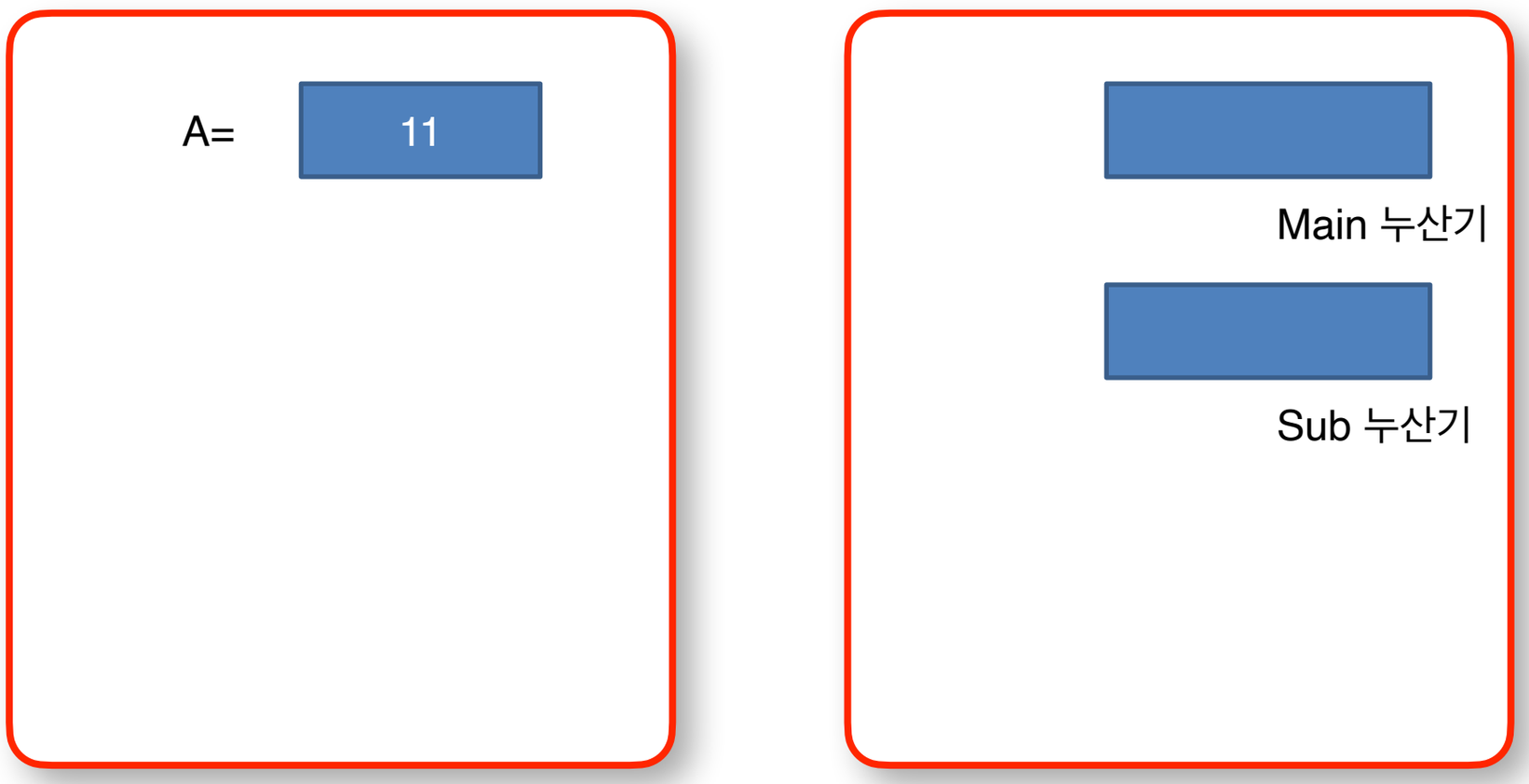
연산 처리 과정

A + B



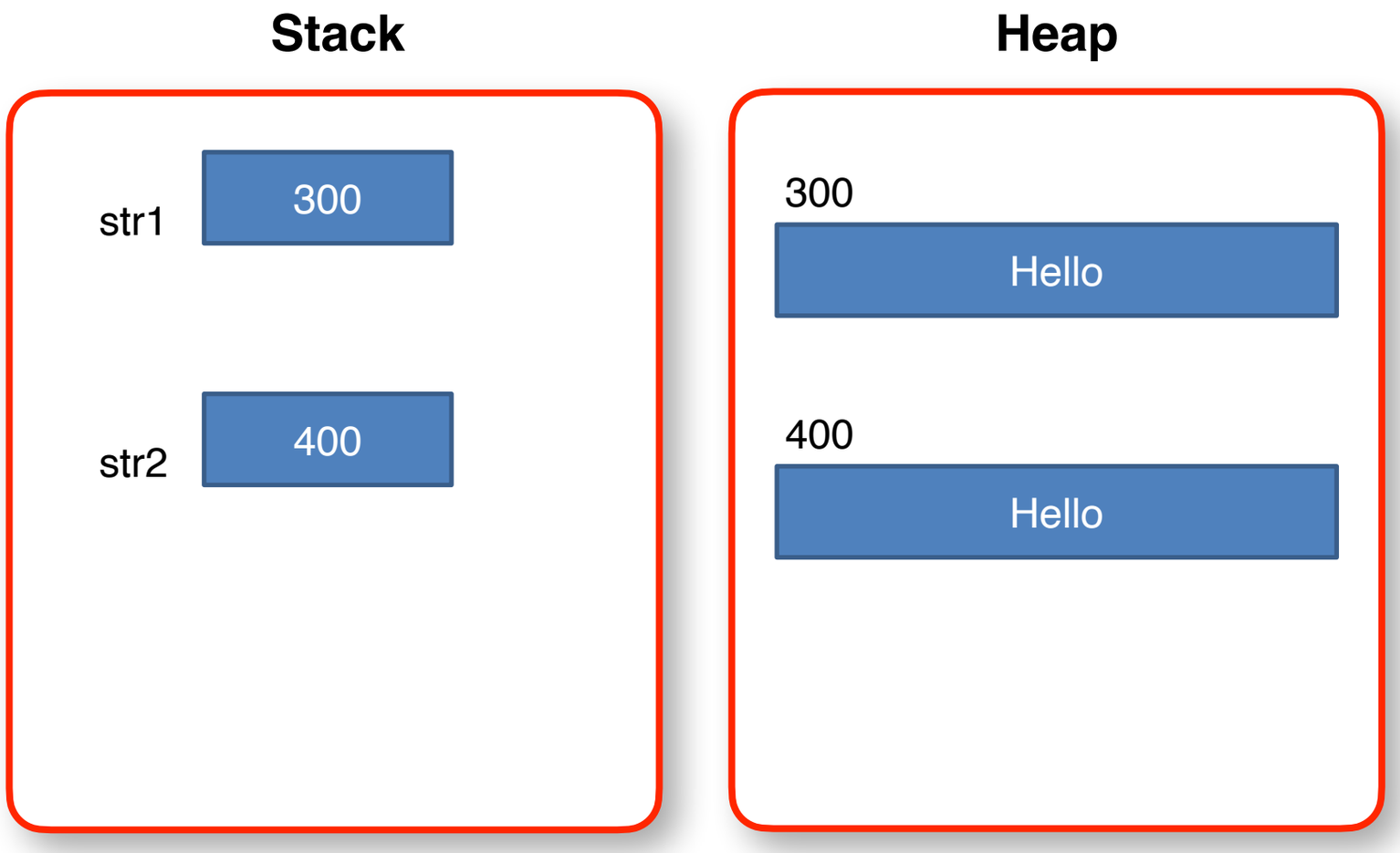
연산 처리 과정

++A



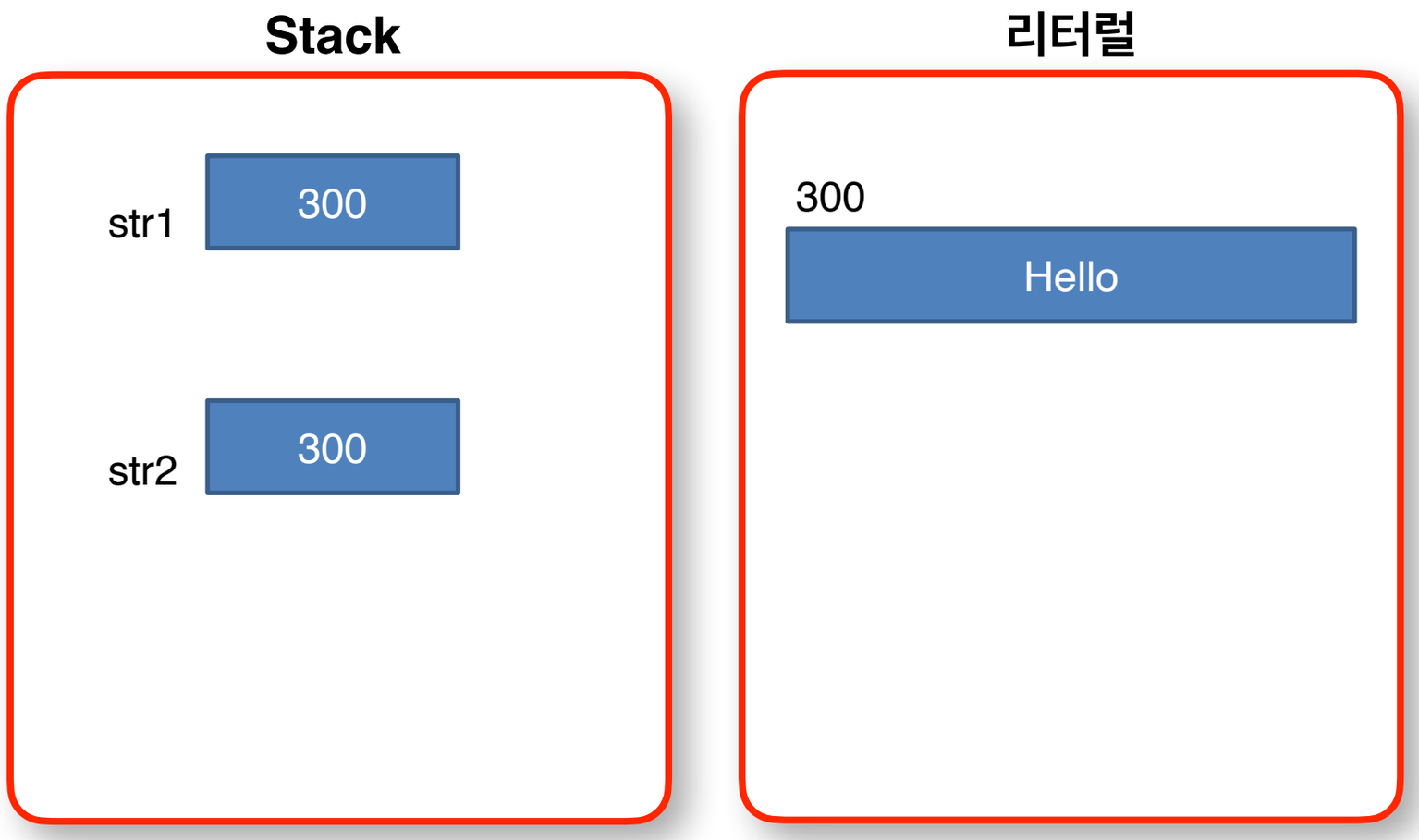
❖ 문자열 처리 원리

String str1 = new String(“Hello”);
String str2 = new String(“Hello”);



❖ 문자열 처리 원리

String str1 = “Hello”;
String str2 = “Hello”;



주석문

주석 종류	의미	설명
// 주석문	단행 주석처리	현재 행에서 //의 뒷문장부터 주석으로 처리된다.
/* 주석문 */	다행 주석처리	/*에서 */ 사이의 문장이 주석으로 처리된다.
/** 주석문 */	HTML 문서화 주석처리	/**에서 */ 사이의 문장이 주석으로 처리된다. 장점은 HTML 문서화로 주석이 처리되므로 API와 같은 도움말 페이지를 만들 수 있다.

키보드를 이용해서 데이터 입력받기

1. class가 만들어지기 이전에
   import java.util.*;
   이라는 명령을 이용해서 사용할 라이브러리를 등록한다.
2. 프로그램에 들어가서(우리는 주로 main 함수 안에서)
   Scanner sc = new Scanner(System.in);
   이라는 명령을 이용해서 키보드를 통해서 입력 받을 도구를 준비한다.
3. 키보드를 통해서 입력받을 필요가 생기면....
   변수 = sc.nextXXX();
   명령을 이용해서 데이터를 입력 받으면 된다.
   이때 XXX는 입력받을 데이터의 종류에 따라 달라진다.

1. sc.nextLine(); 문자열
2. sc.nextInt(); 정수
3. sc.nextFloat(); 실수

❖ 제어문

프로그램의 흐름에 영향을 주고 때에 따라 제어가 가능하도록 하는 것 (조건문, 반복문)

❖ 조건문

조건을 주고 조건의 일치 여부에 따라 실행을 결정하는 명령문

◆ if 문

조건이 맞을때에만 실행문을 실행할 조건문

✓ 형식

```
if ( 조건식 ) {  실행문 }
```

◆ if-else문

조건이 맞았을때와 맞지 않았을때의 실행문을 지정해 놓은 조건문

✓ 형식

```
if ( 조건식 ) {
    조건 일치시 실행문
} else {
    조건 불일치시 실행문
}
```

훈련교사 : 전 은 석
email : euns_jun@naver.com

◆ 다중 if(if-else if) 문

조건이 두개 이상일 경우 각 경우에 따라 실행 명령을 지정해 놓은 조건문

✤ 형식

```
if ( 조건식1 ) {
    조건식1 일치시 실행문
} else if(조건식2){
    조건식2 일치시 실행문
} else if(조건식3){
    조건식3 일치시 실행문
}
. . .
else {
    모든 조건에 맞지 않을때 실행문
}
```

훈련교사 : 전 은 석
email : euns_jun@naver.com

◆ switch문

인자값의 경우 따라 실행문장을 만들어두는 조건문

*** 수행문장 다음에 반드시 "break;" 문을 기술한다.

형식

```
switch ( 인자값 ) {
    case 조건값1:
        실행문;
        break;
    case 조건값2:
        실행문;
        break;
    case 조건값3:
        실행문;
        break;
    break;
    . . .
    default:
        실행문;
}
```

❖ 반복문

실행문을 반복 수행하게 할때 사용하는 제어문

- 카운터 변수 : 반복횟수를 기록할 변수
- 조건식 : 반복을 종료할 조건식
- 증감식 : 반복될때마다 카운터 변수를 증감 해주는 명령

◆ for 반복문

카운터변수와 반복이 종료될 조건식이 있는 반복 명령

형식

```
                1                2                3
for( 카운터변수 초기화; 조건식 ; 카운터변수 증감식){
        실행문      4
}
5
```

순서 : 1 -> 2 -> 4 -> 3 -> 2 -> 4 -> 3 -> . . . 2 -> 5

예제

```
for(int i = 0 ; i < 10 ; i++){
        System.out.println(i);
}
```

향상된 for 명령

시퀀스 자료형(인덱스가 있는 자료형)에서 유용하게 쓸 수 있는 반복 명령
인덱스 순서로 하나씩 꺼내서 변수에 대입한다.

- 형식 : 매개변수에 담겨있는 데이터 타입과 동일한 타입의 변수를 만들어준다.

```
for( 데이터타입    변수이름 : 시퀀스자료 변수이름 ){
    실행문;
}
```

- 예제] 정수 자료 6개(1, 2, 3, 4, 5, 6)를 넣은 배열의 데이터를 순서대로 출력

```
int[] noArray = {1, 2, 3, 4, 5, 6};
int cnt = 0;
for(int num : noArray){
    System.out.println("index : " + cnt++ + " / 데이터 : " +  num);
}
```

while 명령

for 반복문과 비교해서 카운터변수를 따로 만들어줘야한다는 차이점이 있다.
조건이 참이면 반복하는 명령
카운터변수의 생성과 처리는 따로 해줘야 한다.

형식

```
카운터변수 선언;
while( 반복조건 ){
    실행문;
    카운터변수 증감식;
}
```

- 예제] 정수 자료 6개(1, 2, 3, 4, 5, 6)를 넣은 배열의 데이터를 순서대로 출력

```
int[] noArray = {1, 2, 3, 4, 5, 6};
int cnt = 0;
while( cnt < noArray.length ){
    System.out.println("index : " + cnt++ + " / 데이터 : " +  num);
}
```

훈련교사 : 전 은 석
email : euns_jun@naver.com

do - while 명령

다른 반복문과 비교해서 최소 한번은 반드시 실행한다는 차이점이 있다.
while 문과 비교해 조건이 맨 마지막에 온다는 점이 다르다.

형식

```
카운터변수 선언;
do {
    실행문;
    카운터변수 증감식;
} while ( 반복조건 );
```

- 예제] 정수 자료 6개(1, 2, 3, 4, 5, 6)를 넣은 배열의 데이터를 순서대로 출력

```
int[] noArray = {1, 2, 3, 4, 5, 6};
int cnt = 0;
do{
    System.out.println("index : " + cnt + " / 데이터 : " +  num);
} while( cnt++ < noArray.length )
```

break 명령

반복문과 switch문 또는 레이블이 붙은 반복문을 종료
가장 가까운 반복문을 종료

continue 명령

반복문을 다음 회차로 다시 반복
가장 가까운 반복문을 다시 실행
continue 다음에 레이블이 붙을 경우 해당 레이블이 붙은 반복문의 다음회차로 진행

형식

```
반복문( 반복조건 ){
      if( 조건식 ) {
          break( 또는 continue ) [레이블];
     }
     실행문;
}
```

- 예제] 정수를 입력받아 그 숫자가 짝수인지 홀수인지 출력하세요.
  7번 반복하고 중간에 3이 나오면 종료하세요.

```
import java.util.*;
... 생략 ...
Scanner sc = new Scanner(System.in);
int no = 0;
int cnt = 0;

while(cnt++ < 7){
    no = sc.nextInt();
    if ( no == 3 ){
        break;
    } else {
        System.out.println(
          cnt + " 번째 입력값 : " + no + " / " + (no % 2 == 0 ? "짝수" : "홀수");
        continue;
    }
}
```

훈련교사 : 전 은 석
email : euns_jun@naver.com

배열

같은 자료형들끼리 모아두는 하나의 묶음

담을수 있는 자료의 타입과 길이가 먼저 정해진다.

선언 형식1

```
데이터타입[] 변수이름;
```

선언 형식2

```
데이터타입 변수이름[];
```

❖ 배열

배열 생성 형식1 : 배열 객체만 생성하는 방법

```
데이터타입 변수이름[] = new 데이터타입[길이];
```

예제] 정수 자료 6개를 넣을 배열을 만드세요.

```
int[] noArray = new int[6];
```

배열 생성 형식2 : 데이터를 생성과 동시에 입력하는 방법

```
데이터타입 변수이름[] = {데이터1, 데이터2, 데이터3, . . . };
```

예제] 정수 자료 6개(1, 2, 3, 4, 5, 6)를 넣을 배열을 만드세요.

```
int[] noArray = {1, 2, 3, 4, 5, 6};
```

훈련교사 : 전 은 석
email : euns_jun@naver.com

배열

같은 자료형들끼리 모아두는 하나의 묶음

담을수 있는 자료의 타입과 길이가 먼저 정해진다.

선언 형식1

```
데이터타입[] 변수이름;
```

선언 형식2

```
데이터타입 변수이름[];
```

❖ 배열

배열 생성 형식1 : 배열 객체만 생성하는 방법

```
데이터타입 변수이름[] = new 데이터타입[길이];
```

예제] 정수 자료 6개를 넣을 배열을 만드세요.

```
int[] noArray = new int[6];
```

배열 생성 형식2 : 데이터를 생성과 동시에 입력하는 방법

```
데이터타입 변수이름[] = {데이터1, 데이터2, 데이터3, . . . };
```

예제] 정수 자료 6개(1, 2, 3, 4, 5, 6)를 넣을 배열을 만드세요.

```
int[] noArray = {1, 2, 3, 4, 5, 6};
```

향상된 for 명령

시퀀스 자료형(인덱스가 있는 자료형)에서 유용하게 쓸 수 있는 반복 명령
인덱스 순서로 하나씩 꺼내서 변수에 대입한다.

- 형식 : 매개변수에 담겨있는 데이터 타입과 동일한 타입의 변수를 만들어준다.

```
for( 데이터타입    변수이름 : 시퀀스자료 변수이름 ){
    실행문;
}
```

- 예제] 정수 자료 6개(1, 2, 3, 4, 5, 6)를 넣은 배열의 데이터를 순서대로 출력

```
int[] noArray = {1, 2, 3, 4, 5, 6};
int cnt = 0;
for(int num : noArray){
    System.out.println("index : " + cnt++ + " / 데이터 : " +  num);
}
```

while 명령

for 반복문과 비교해서 카운터변수를 따로 만들어줘야한다는 차이점이 있다.
조건이 참이면 반복하는 명령
카운터변수의 생성과 처리는 따로 해줘야 한다.

형식

```
카운터변수 선언;
while( 반복조건 ){
    실행문;
    카운터변수 증감식;
}
```

- 예제] 정수 자료 6개(1, 2, 3, 4, 5, 6)를 넣은 배열의 데이터를 순서대로 출력

```
int[] noArray = {1, 2, 3, 4, 5, 6};
int cnt = 0;
while( cnt < noArray.length ){
    System.out.println("index : " + cnt++ + " / 데이터 : " +  num);
}
```

훈련교사 : 전 은 석
email : euns_jun@naver.com

do - while 명령

다른 반복문과 비교해서 최소 한번은 반드시 실행한다는 차이점이 있다.
while 문과 비교해 조건이 맨 마지막에 온다는 점이 다르다.

형식

```
카운터변수 선언;
do {
    실행문;
    카운터변수 증감식;
} while ( 반복조건 );
```

- 예제] 정수 자료 6개(1, 2, 3, 4, 5, 6)를 넣은 배열의 데이터를 순서대로 출력

```
int[] noArray = {1, 2, 3, 4, 5, 6};
int cnt = 0;
do{
    System.out.println("index : " + cnt + " / 데이터 : " +  num);
} while( cnt++ < noArray.length )
```

break 명령

반복문과 switch문 또는 레이블이 붙은 반복문을 종료
가장 가까운 반복문을 종료

continue 명령

반복문을 다음 회차로 다시 반복
가장 가까운 반복문을 다시 실행
continue 다음에 레이블이 붙을 경우 해당 레이블이 붙은 반복문의 다음회차로 진행

형식

```
반복문( 반복조건 ){
      if( 조건식 ) {
          break( 또는 continue ) [레이블];
     }
     실행문;
}
```

- 예제] 정수를 입력받아 그 숫자가 짝수인지 홀수인지 출력하세요.
  7번 반복하고 중간에 3이 나오면 종료하세요.

```
import java.util.*;
... 생략 ...
Scanner sc = new Scanner(System.in);
int no = 0;
int cnt = 0;

while(cnt++ < 7){
    no = sc.nextInt();
    if ( no == 3 ){
        break;
    } else {
        System.out.println(
          cnt + " 번째 입력값 : " + no + " / " + (no % 2 == 0 ? "짝수" : "홀수");
        continue;
    }
}
```