Java 1.8 PPT

나 종 민

Part 01.

JAVA
기초 프로그래밍

❖ 1. JAVA 개요

1. 자바란 ?

- 1991년부터 전자렌지, 냉장고, 전기밥솥 등 가정용 가전기기를 제어할 것을 목적으로 선 마이크로시스템(Sun Microsystem)사에서 개발하여 1996년 공식적으로 발표한 객체지향 프로그래밍 언어

2. 자바언어의 특징

- 플렛폼(OS) 독립성 : JVM(Java Virtual Machine)
- 객체지향 언어 : 상속, 캡슐화, 다형성
- 멀티 쓰레드 지원
- 자동 메모리 관리 : Garbage Collection
- 동적인 성능(로딩) 확장 제공

2. JAVA 개발환경구축

1. JDK(Java Development Kit) 설치
   - JDK 다운로드 설치
     https://www.oracle.com/java/technologies/downloads/#java8
   - JDK 설치 확인
     c:₩> java -version
   - 환경변수 설정
   - javac.exe : 자바 컴파일러, 자바소스코드를 바이트 코드로 컴파일
     javac Hello.java
   - java.exe : 자바 인터프리터, 컴파일러가 생성한 바이트코드를 해석하고 실행
     java Hello
   - javap.exe : 역 어셈블러, 컴파일된 클래스 파일을 원래의 소스로 변환
     javap Hello > Hello.java
   - javadoc.exe : 자동문성생성기, 소스파일에 있는 주석(/*** */)을 이용하여 Java API문서와 같은 형식의 문서를 자동 생성
     javadoc Hello.java
   - jar.exe : 압축프로그램, 클래스파일과 프로그램의 실행에 관련된 파일을 하나의 jar파일(.jar)로 압축하거나 압축해제
     압축시 : jar cvf Hello.jar Hello1.class Hello2.class
     해제시 : jar xvf Hello.jar

2. JAVA 개발환경구축

2. IDE(Integrated Development Environment) 설치

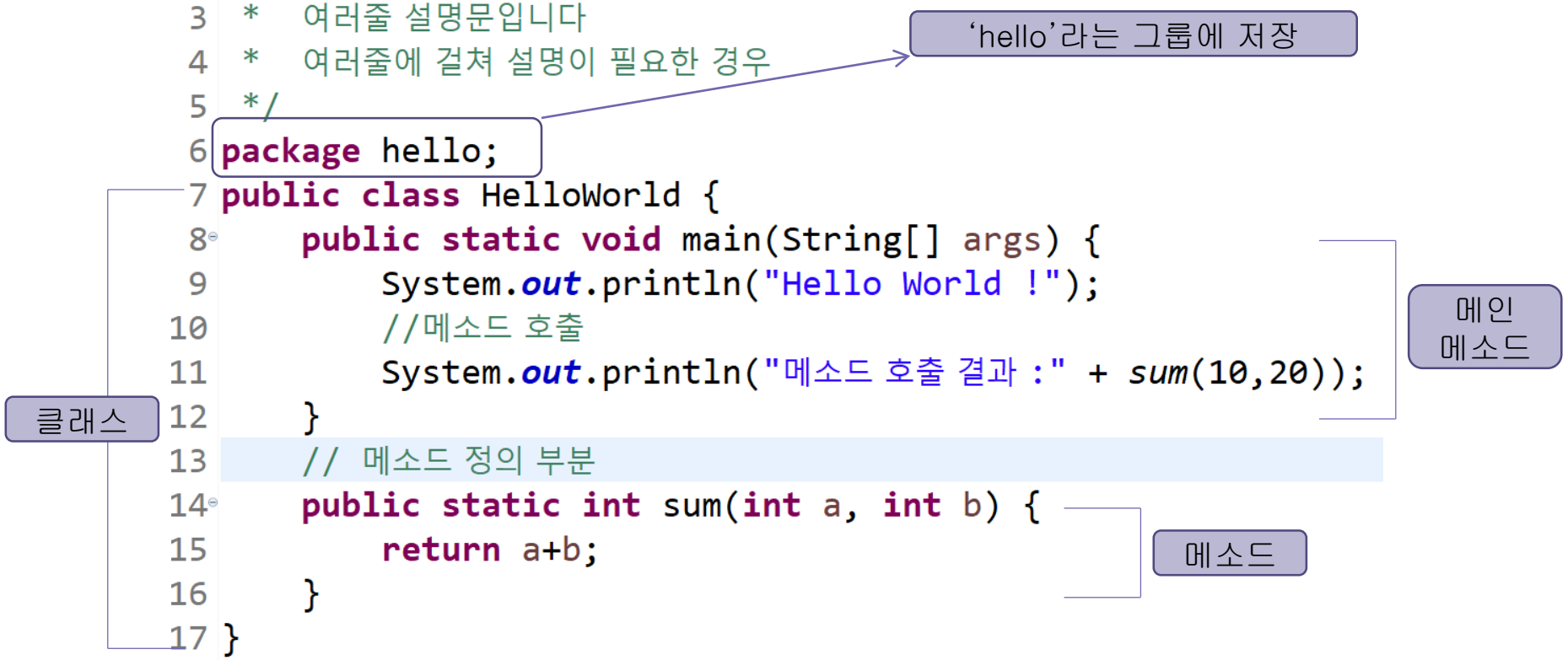
- eclipse 다운로드 설치
  https://www.eclipse.org/downloads/

번호	이클립스 버전명	요구사항
1	Eclipse 4.17 (2020-09)	A Java 11 or newer JRE/JDK is required, (Java 11 이상 JRE / JDK가 필요합니다.)
2	Eclipse 4.16 (2020-06)	A Java 8 or newer JRE/JDK is required, LTS release are preferred (Java 8 이상 JRE / JDK가 필요합니다. LTS 릴리스가 선호됨)
3	Eclipse 4.14 (2019-12)	A Java 8 or newer JRE/JDK is required. (Java 8 이상 JRE / JDK가 필요합니다.)

2. JAVA 개발환경구축

3. 자바프로그램의 기본적인 구조

```
1  // 한줄 설명문입니다
2  /*
3   *   여러줄 설명문입니다
4   *   여러줄에 걸쳐 설명이 필요한 경우
5   */
6  package hello;
7  public class HelloWorld {
8      public static void main(String[] args) {
9          System.out.println("Hello World !");
10         //메소드 호출
11         System.out.println("메소드 호출 결과 :" + sum(10,20));
12     }
13     // 메소드 정의 부분
14     public static int sum(int a, int b) {
15         return a+b;
16     }
17 }
```

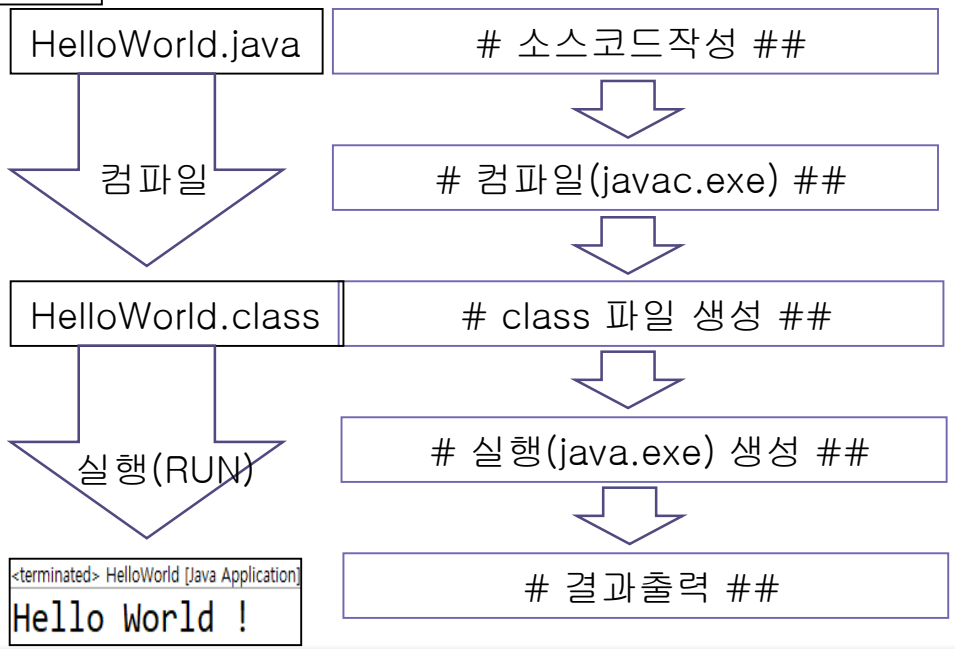


2. JAVA 개발환경구축

4. 자바 프로그램 작성 및 컴파일 단계

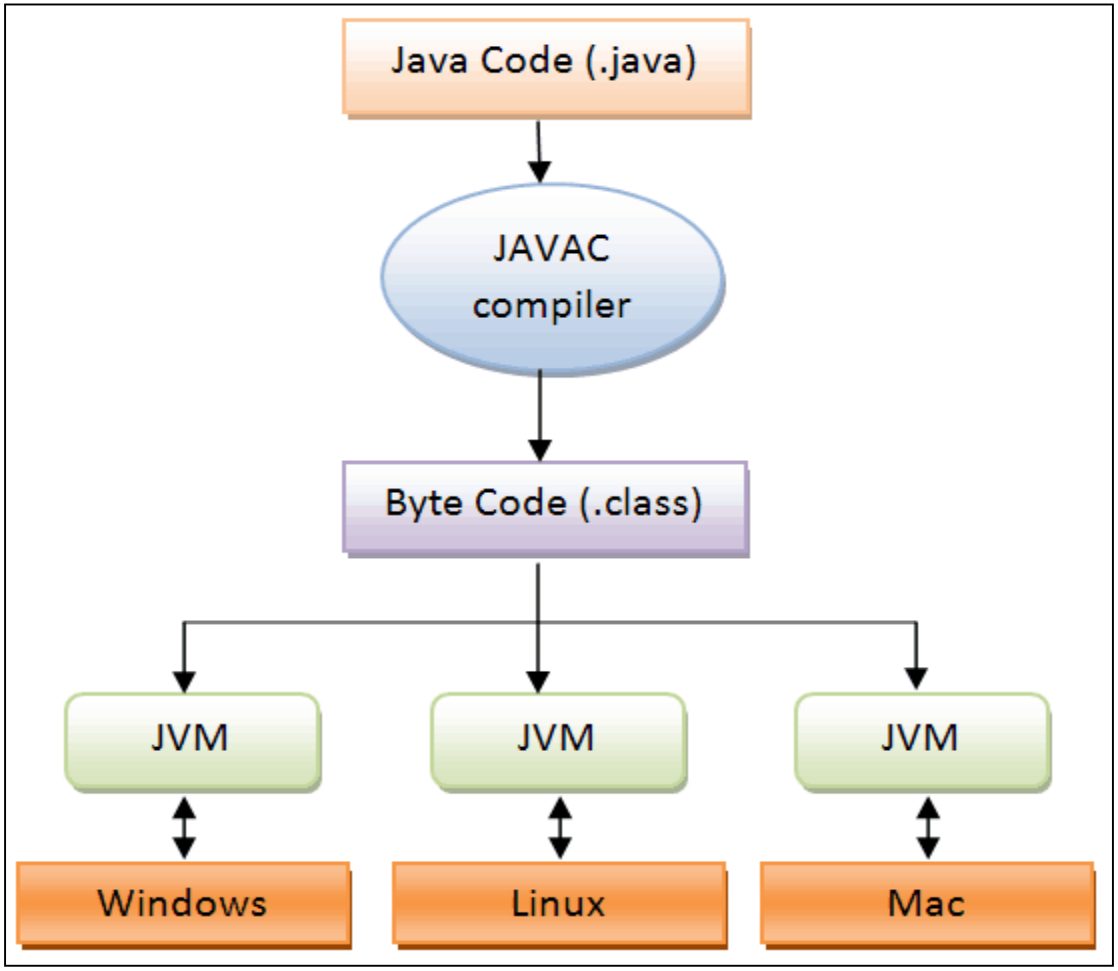
```java
//화면에 Hello World 출력하기

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World !");
    }
}

```

JavaSE-11(JDK11) 이후 버전
-> module-info.java(모듈 기술자 필요)
-> 명령프로픔트에서 프로젝트 실행시 필요
-> module-info.java {
requires java.se;
//java se가 제공하는 모든 모듈이 필요함
}



2. JAVA 개발환경구축

5. 자바 프로그램 실행 단계



3. 표준 입출력

1. 표준 출력(System.out)
   - System.out.print()
   - System.out.println()
   - System.out.printf()
2. 표준입력(System.in)
   - System.in.read()

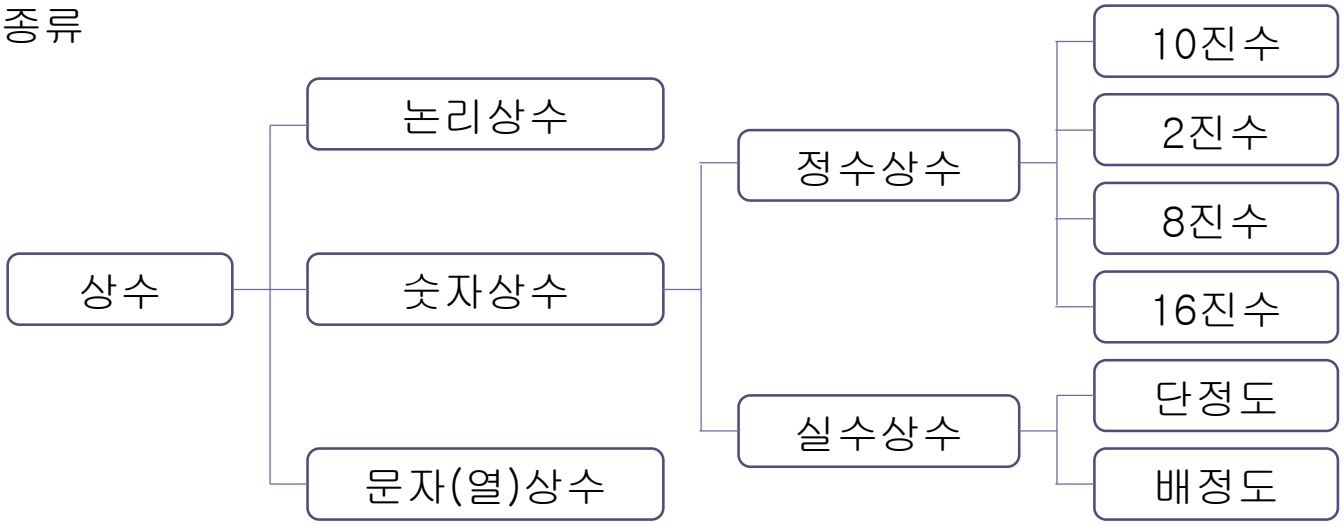
4. 상수, 변수

1. 상수 (constant & literal)
2. 변수 (variable)
3. 변수의 타입(자료형 :Data type)
4. 변수에 데이터(상수) 할당
5. Scanner 클래스

4. 상수, 변수

1. 상수 (constant & literal)

- 한번 정의되면 변하지 않는 값(data)
- 종류



4. 상수, 변수

2. 변수(variable)

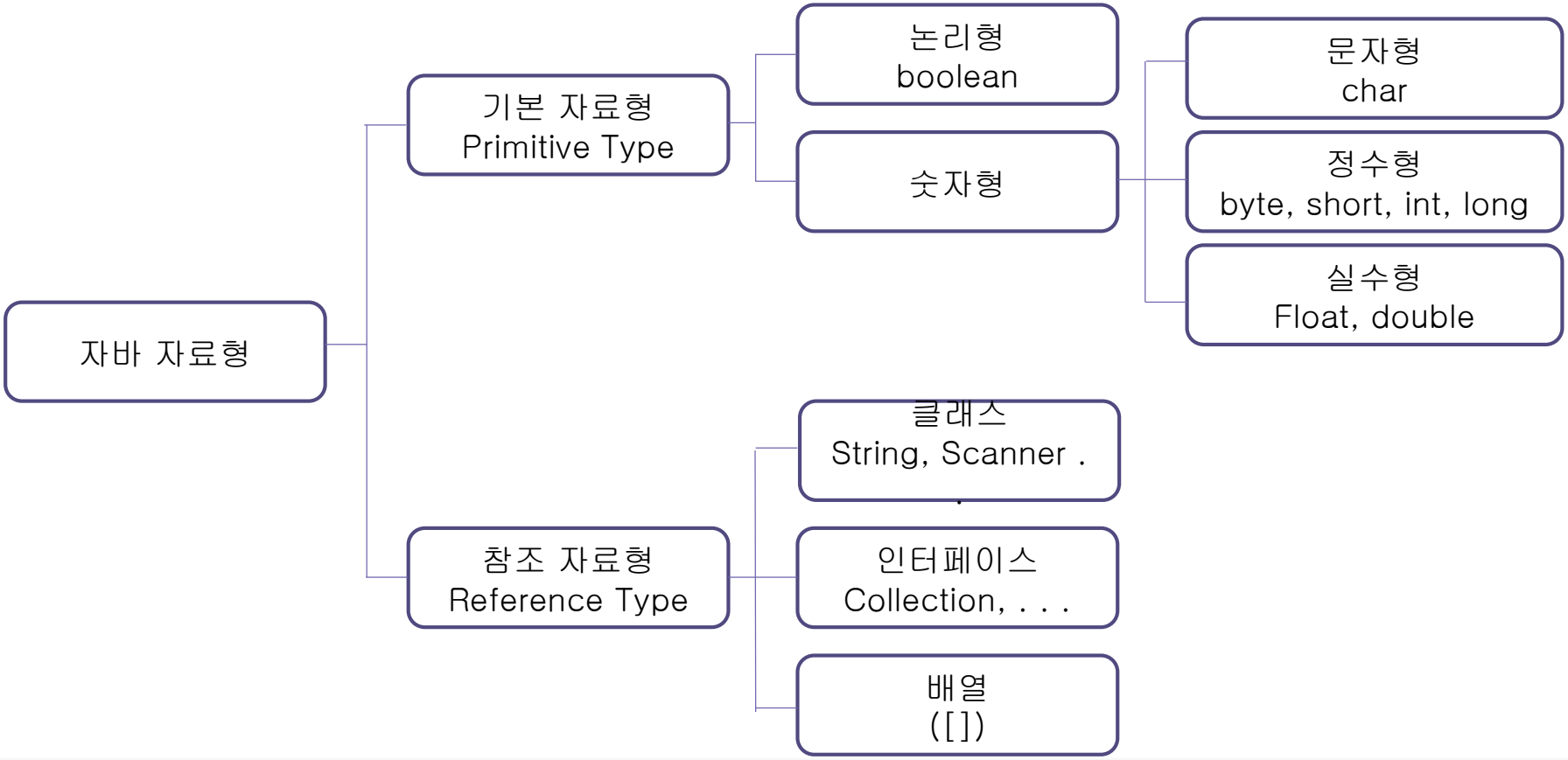
- 상수를 임시 저장하기 위한 공간
- 어떤 값을 주기억 장치에 기억하기 위해서 사용하는 공간을 의미
- 변수의 명명 규칙
  - 대소문자가 구분되며 길이에 제한이 없다.
  - 예약어 사용 불가
  - 숫자로 시작할 수 없다.
  - 특수문자는 '_', '$' 만 허용한다.

- 변수의 선언(생성)
  변수타입(데이터 형) 변수_이름;

4. 상수, 변수

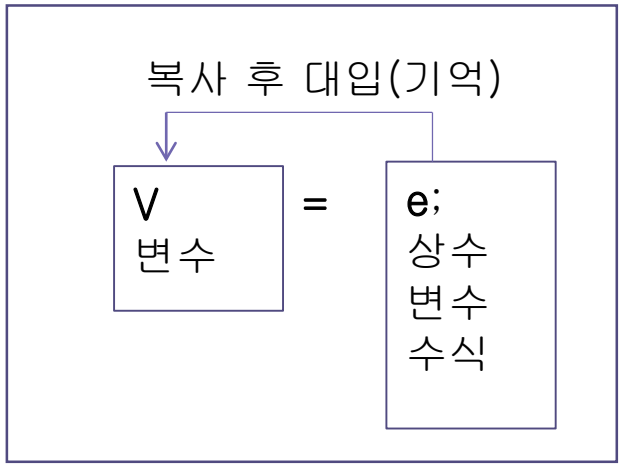
3. 변수의 타입(자료형 :Data type)

- 데이터 타입은 생성할 메모리의 크기(size)와 메모리의 형태(type)를 지정하기 위한 목적으로 사용



4. 상수, 변수

4. 변수에 데이터(상수) 할당



사용 예

```
 char ch = 'A';
 int kor = 70;
 int eng =90;
 int tot = kor+eng;
double ave = tot / 2.
String str="김학생";
```

- 대입형 변수
- 누적(카운트) 변수

4. 상수, 변수

5. Scanner 클래스

• 키보드로부터 데이터를 입력 받기 위한 클래스로
• System.in에게 키를 읽게 하고, 읽은 바이트를 문자, 정수, 실수, 불린, 문자열 등 다양한 타입으로 변환하여 리턴
• Scanner 클래스는 'Part 6 자바 기본 API'에서 자세히 배우므로 지금은 기본적인 사용법만 알아두자.

기본 사용법

- Scanner 클래스를 사용하기 위해서는 import java.util.Scanner 를 해주어야 한다.
  import java.util.*; 또는 import java.util.Scanner;
  * eclipse에서 import 하기[단축키 : Ctrl + Shift + o(영문자)]

- Scanner 객체 생성
  Scanner scn = new Scanner(System.in);

- 정수 입력 nextInt(), 실수 입력 nextDouble(), 문자열 입력 next(),
- 문자열(한줄전체) nextLine()

5. 연산자

1. 단항 연산자
2. 산술 연산자
3. 관계(비교) 연산자
4. 논리 연산자
5. 기타 연산자

5. 연산자

1. 단항 연산자

연산자	연산대상	연산내용
[]	모든 데이터형	배열요소 지정
.	레퍼런스	객체멤버 지정
++	정수형, 실수형	값 증가
--	정수형, 실수형	값 감소
+, -	정수형, 실수형	부호에 사용
~	정수형	비트반전
!	논리형	논리반전
new	레퍼런스형	객체생성
(type)	모든 데이터형	캐스팅연산자

5. 연산자

2. 산술 연산자

연산자	연산대상	연산내용
*, / , %	정수형, 실수형	곰셈, 나눗셈, 나머지
+, -	정수형, 실수형	덧셈, 뺄셈
+	String 객체	문자열 결합

연산자	연산대상	연산내용
<<	정수형	비트 좌이동
>>	정수형	비트 우이동
>>>	정수형	비트 부호화 우이동

5. 연산자

3. 관계(비교) 연산자와 상등연산자

연산자	연산대상	연산내용
<, <=, >, >=	정수형, 실수형	값 비교
instanceof	레퍼런스형	객체타입 비교

연산자	연산대상	연산내용
==	기본 데이터형	같다
!=	기본 데이터형	같지않다

5. 연산자

4. 논리 연산자

연산자	연산대상	연산내용
&	정수형, 논리형	비트 AND, 논리 AND
^	정수형, 논리형	비트 XOR, 논리 XOR
|	정수형, 논리형	비트 OR, 논리 OR

연산자	연산대상	연산내용
&&	논리형	논리 AND
||	논리형	논리 OR

5. 연산자

5. 조건연산자, 대입연산자

연산자	연산대상	연산내용
? :	논리형, 모든 데이터형	조건 사항

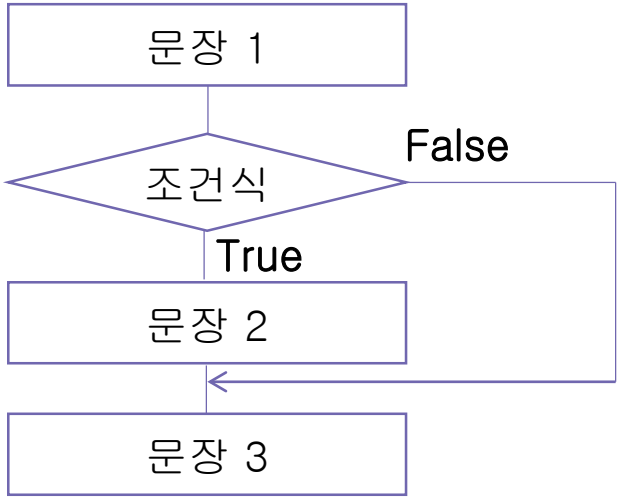
연산자	연산대상	연산내용
= , *=, /=, %=, +=, -=, <<=, >>= , >>>=, &=, ^=, |=	변수, 모든 데이터형	대입연산, 연산 후 대입

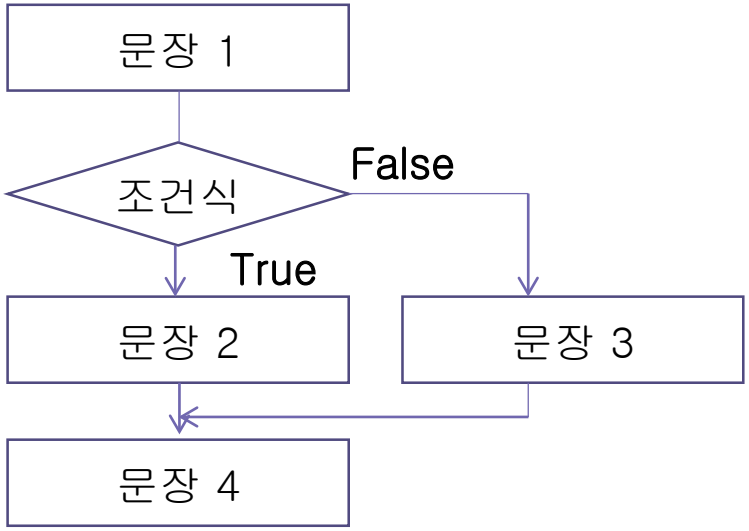
6. 조건(비교판단) 문

1. if ~ else
2. 중첩 if 문(if ~ (if~ else) else
3. if ~ else if ~ else
4. switch ~ case

6. 조건(비교판단) 문

1. if ~ else 문



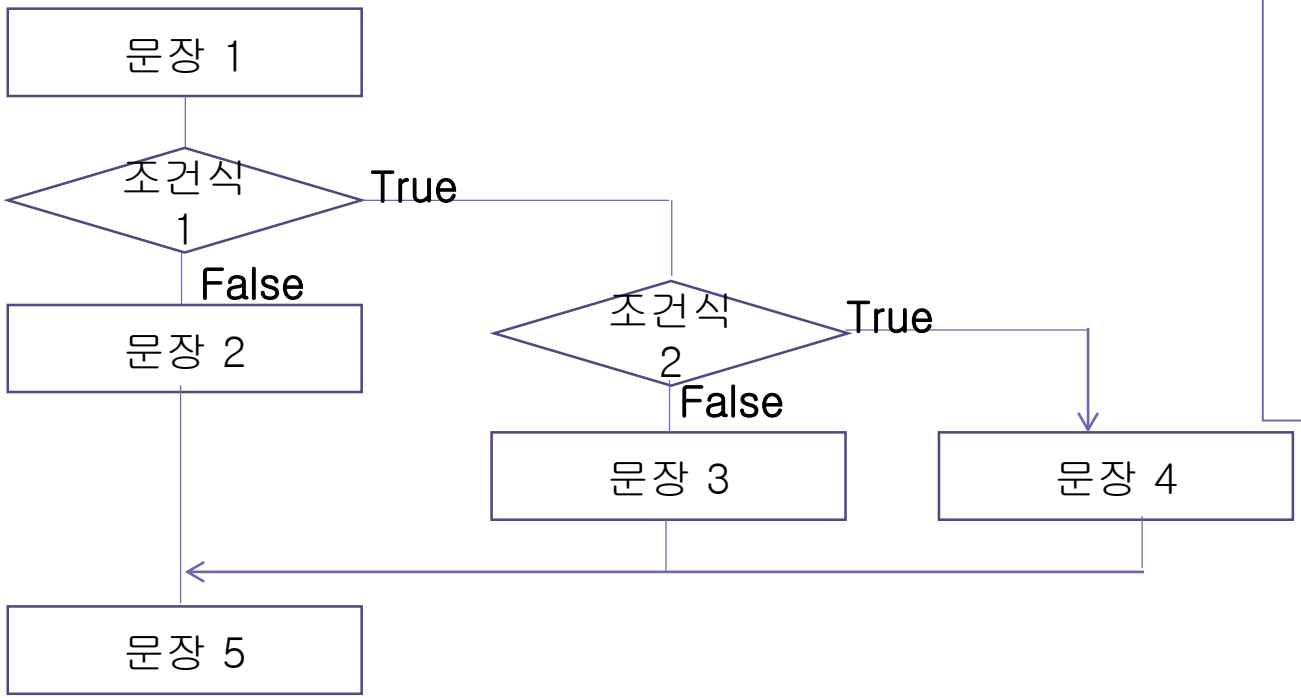



```
문법 형식
    문장 1;
    if(조건식) {
        문장2;
    }
    문장 3;
```

```
문법 형식
    문장 1;
    if(조건식) {
        문장2;
    }else{
        문장 3;
    }
    문장 4;
```

6. 조건(비교판단) 문

2. 중첩 if 문(if ~ (if~ else) else)

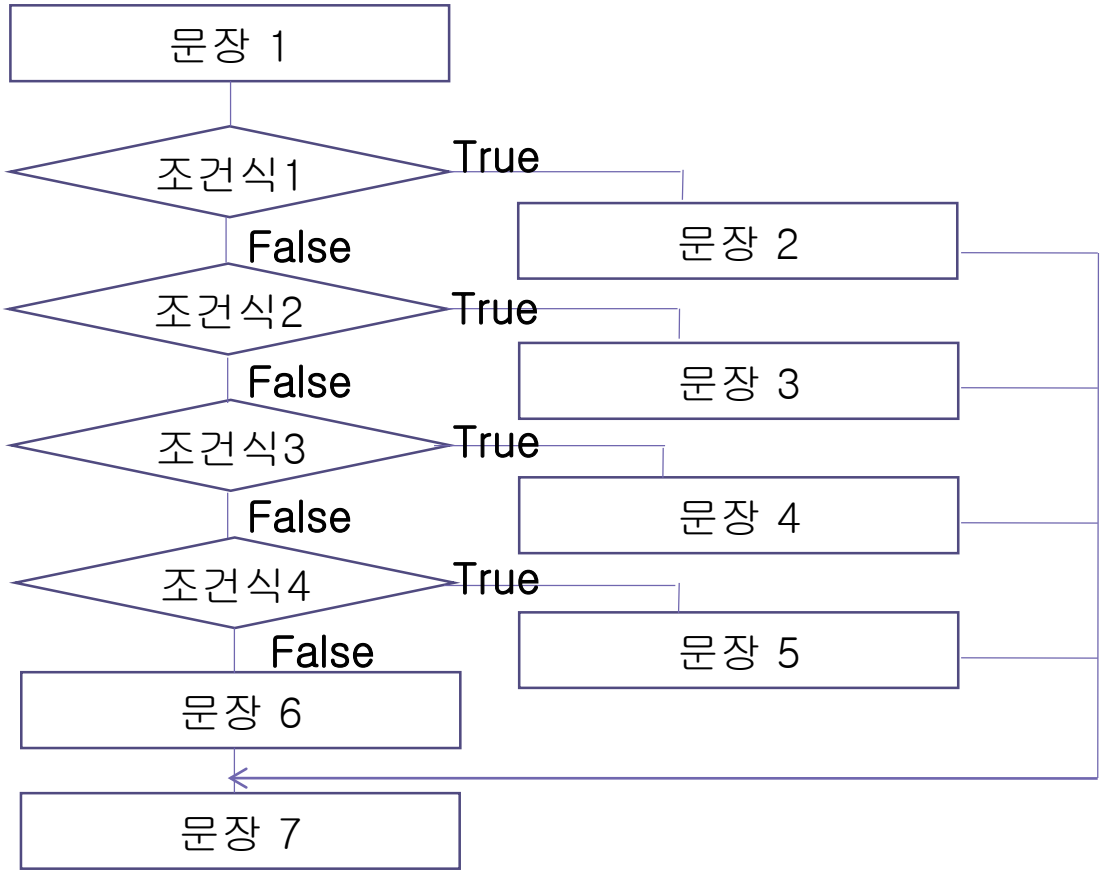


(1) 형식 -2

```
문장 1
if(조건식 1)
    if(조건식 2)
        문장 4;
    else
        문장 3;
else
    문장 2
문장 5;
```

6. 조건(비교판단) 문

3. if ~ else if ~ else 문

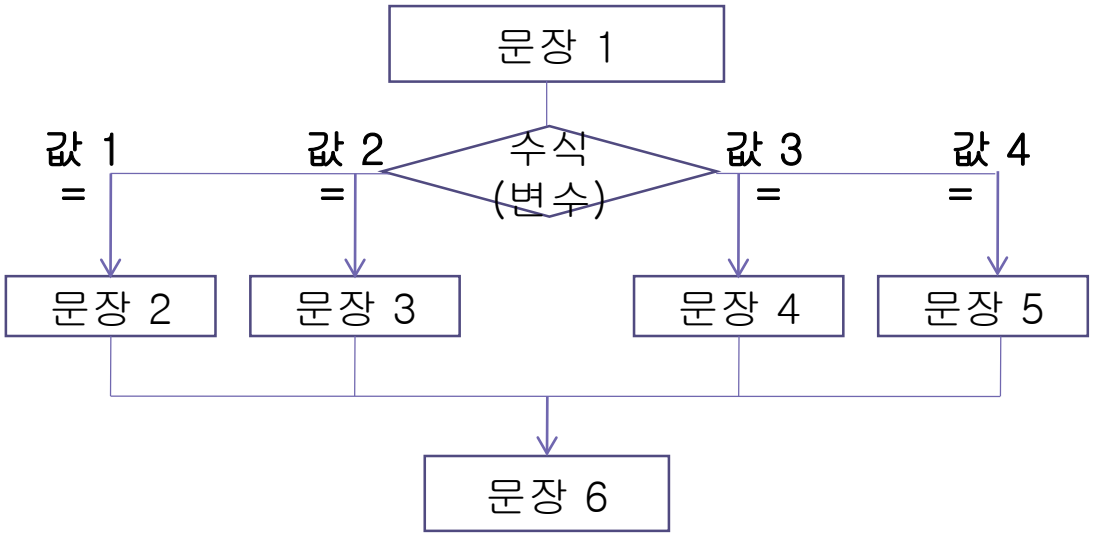


(1) 형식 -2

```
문장 1
if(조건식 1)
    문장2;
else if(조건식 2)
    문장 3;
else if(조건식 3)
    문장 4;
else if(조건식 4)
    문장 5;
else
    문장 6;
문장 7;
```

6. 조건(비교판단) 문

4. 조건문(switch case)



(1) 형식

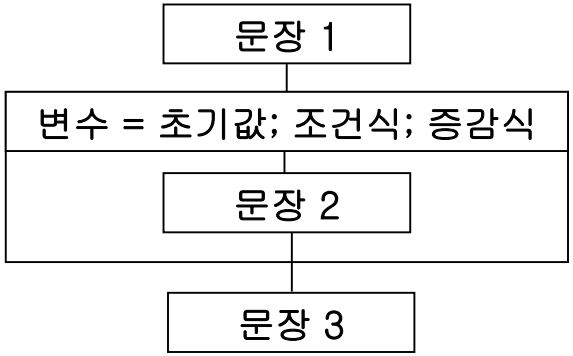
```
문장 1
switch (변수 또는 수식){
    case 값1:
                문장2;
    case 값2:
                문장3;
                break;
    case 값3:
                문장4;
                break;
    case 값4:
                문장5;
                break;
      .   .   .   .
     default:
                문장6;
}
문장 7;
```

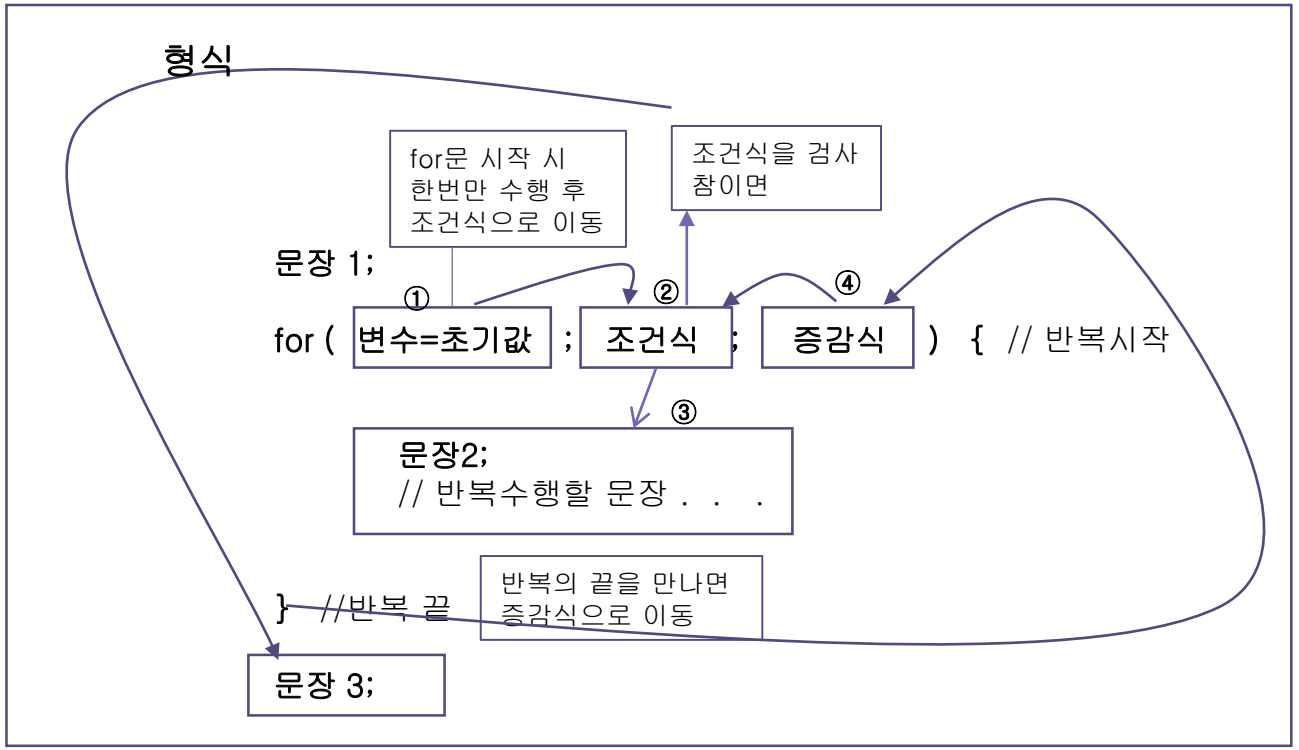
7. 반복문

1. for 문
2. 중첩 for 문
3. while 문
4. do ~ while 문

7. 반복문

1. for 문

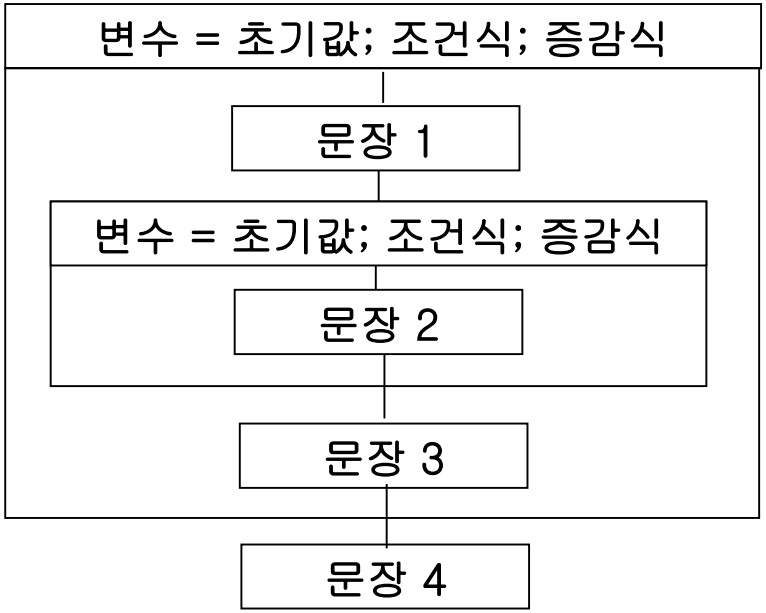




break; 는 반복문을 빠져나가고
continue; while문은 조건식으로 이동
for문은 증감식으로 이동

7. 반복문

2. 중첩 for 문



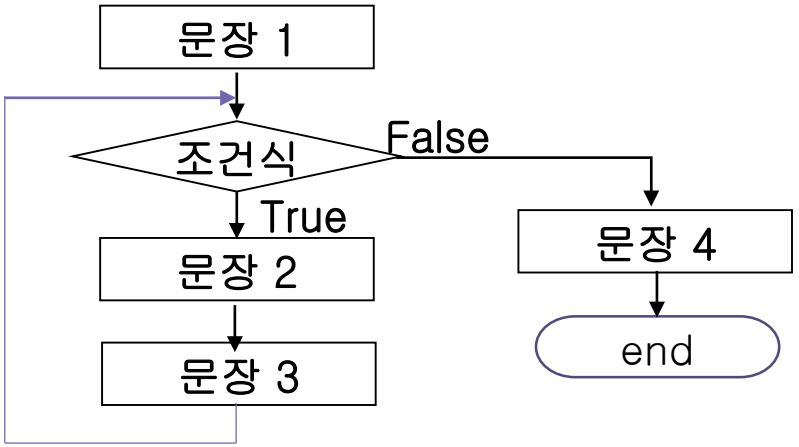
(1) 형식

```
문장 1;
for(변수=초기값; 조건식; 증감식){
    문장2;
    for(변수=초기값; 조건식; 증감식){
        문장3;
        .   .
    }
}
문장 4;
```

break; 반복문을 하나만 빠져나가고
continue; while문은 조건식으로 이동
for문은 증감식으로 이동

7. 반복문

3. while 문



형식

```
문장 1;
while( 조건식 ) { //while 시작
    문장2;
    문장3;
}//while 끝
문장 n;
```

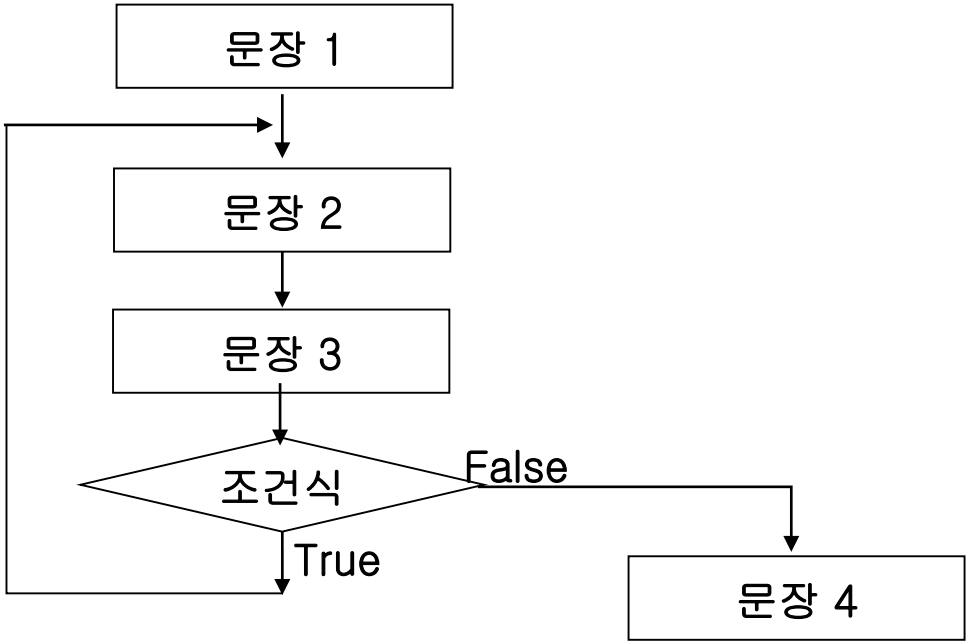
조건식이 거짓이면 문장n 으로 이동

조건식이 참이면 문장2, 3을 수행

break; 는 반복문을 빠져나가고
continue; while문은 조건식으로 이동
for문은 증감식으로 이동

7. 반복문

4. do ~ while 문



(1) 형식

```
문장 1;
do {
        문장2;
          . . .
}while(조건식);
문장 4;
```

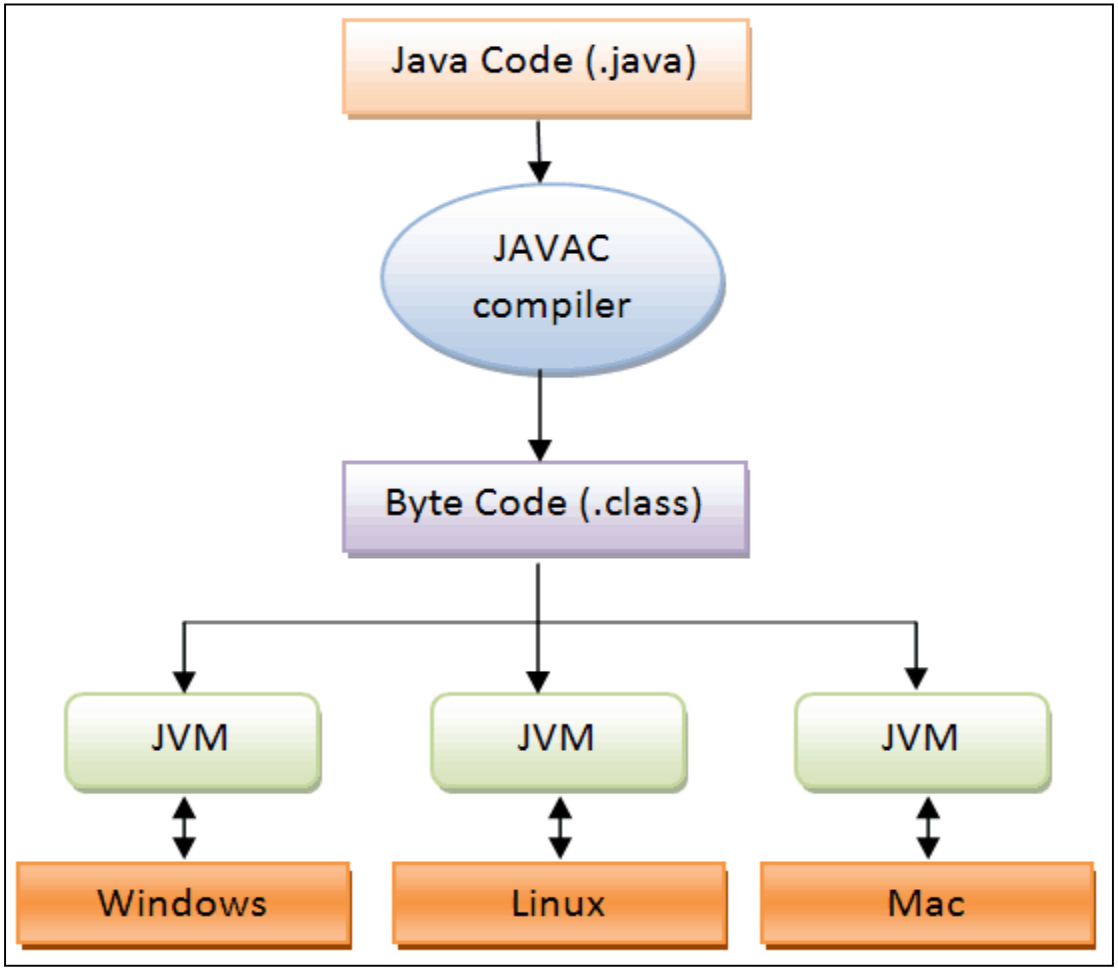
8. JVM (Java Virtual Machine)

1. Java 프로그램 실행 단계

2. JVM 구조 및 기능

8. JVM (Java Virtual Machine)

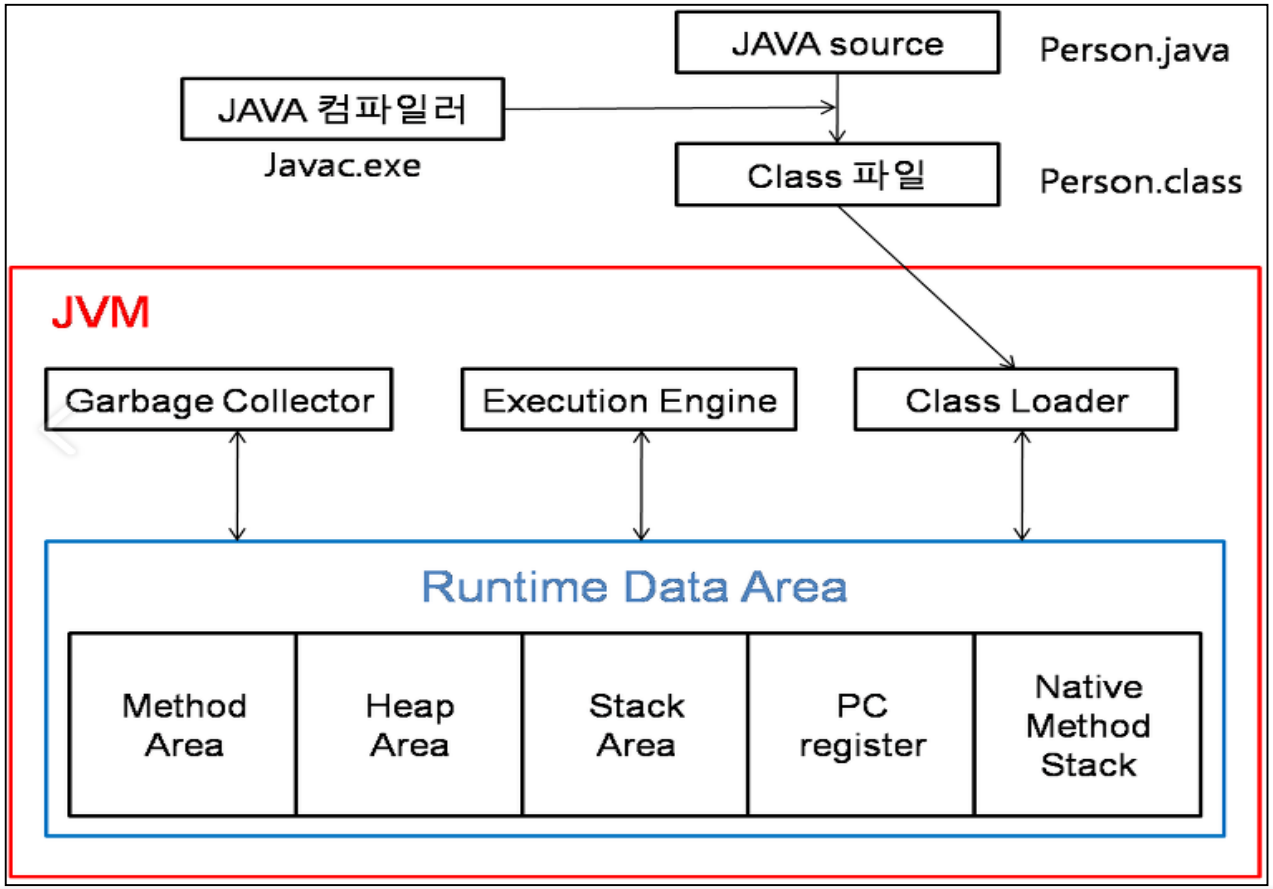
1. 자바 프로그램의 실행단계



8. JVM (Java Virtual Machine)

2. JVM 구조 및 기능

JVM의 구조는 크게 보면, Garbage Collector, Execution Engine, Class Loader, Runtime Data Area로, 4가지로 나눌 수 있다.



8. JVM (Java Virtual Machine)

(1) Class Loader

JVM 내로 클래스 파일을 로드하고, 링크를 통해 배치하는 작업을 수행하는 모듈입니다. 런타임 시에 동적으로 클래스를 로드합니다.

(2) Execution Engine

클래스 로더를 통해 JVM 내의 Runtime Data Area에 배치된 바이트 코드들을 명령어 단위로 읽어서 실행합니다. 최초 JVM이 나왔을 당시에는 인터프리터 방식이었기때문에 속도가 느리다는 단점이 있었지만 JIT 컴파일러 방식을 통해 이 점을 보완하였습니다. JIT는 바이트 코드를 어셈블러 같은 네이티브 코드로 바꿈으로써 실행이 빠르지만 역시 변환하는데 비용이 발생하였습니다. 이 같은 이유로 JVM은 모든 코드를 JIT 컴파일러 방식으로 실행하지 않고, 인터프리터 방식을 사용하다가 일정한 기준이 넘어가면 JIT 컴파일러 방식으로 실행합니다.

(3) Garbage Collector

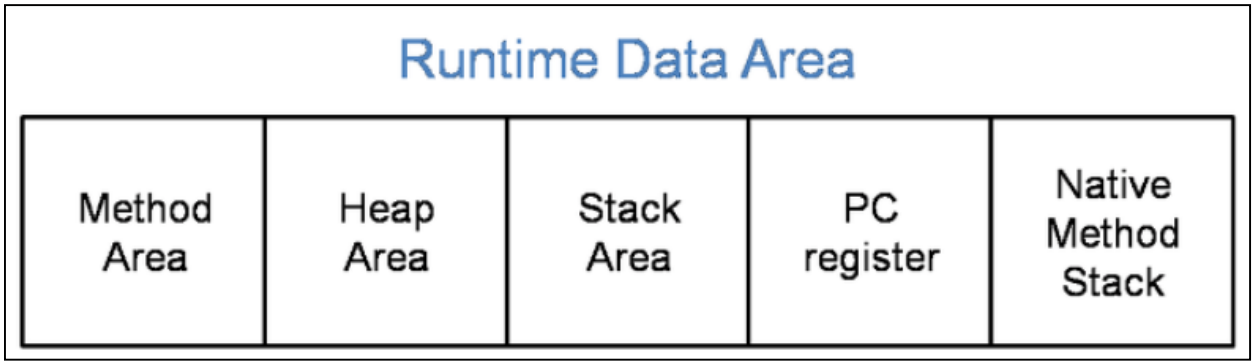
Garbage Collector(GC)는 힙 메모리 영역에 생성된 객체들 중에서 참조되지 않은 객체들을 탐색 후 제거하는 역할을 합니다. 이때, GC가 역할을 하는 시간은 언제인지 정확히 알 수 없습니다.

(4) Runtime Data Area

JVM의 메모리 영역으로 자바 애플리케이션을 실행할 때 사용되는 데이터들을 적재하는 영역입니다. 이 영역은 크게 Method Area, Heap Area, Stack Area, PC Register, Native Method Stack로 나눌 수 있습니다.

8. JVM (Java Virtual Machine)

3. Runtime Data Area



(1) Method area

모든 쓰레드가 공유하는 메모리 영역입니다. 메소드 영역은 클래스, 인터페이스, 메소드, 필드, Static 변수 등의 바이트 코드를 보관합니다.

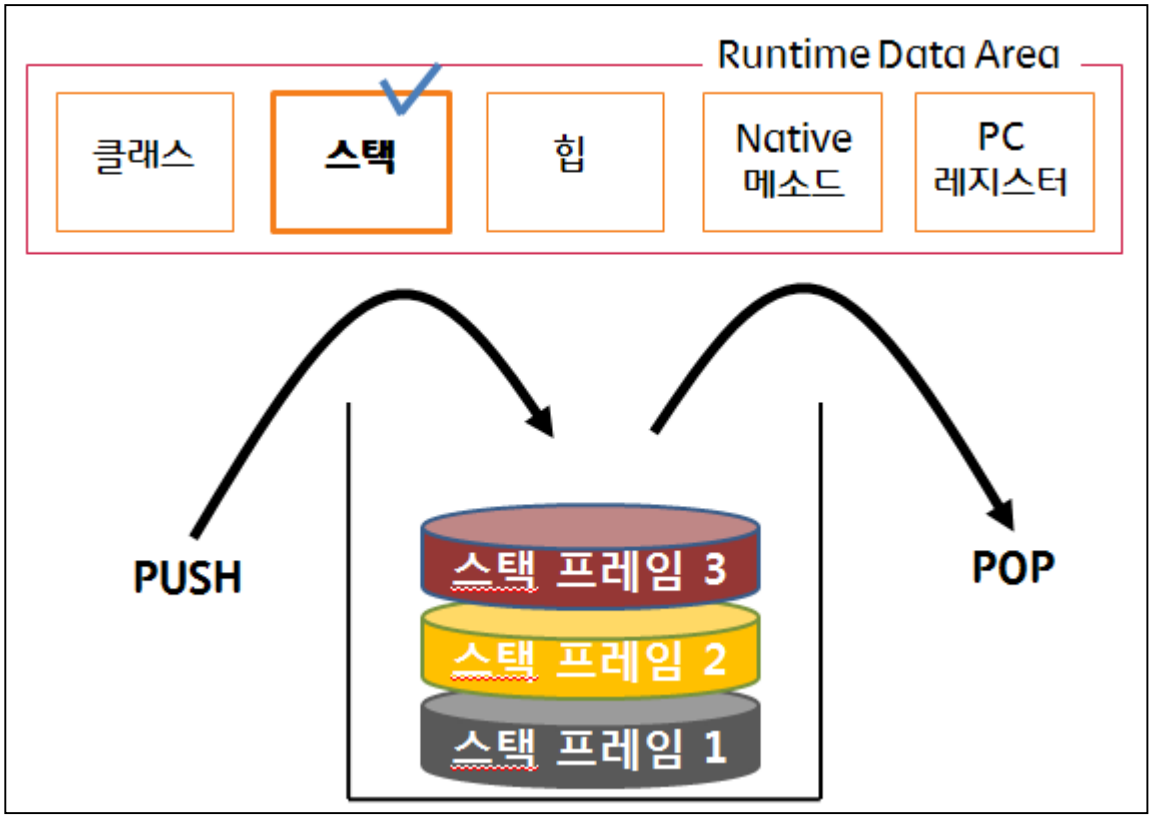
(2) Heap area

모든 쓰레드가 공유하며, new 키워드로 생성된 객체와 배열이 생성되는 영역입니다. 또한, 메소드 영역에 로드된 클래스만 생성이 가능하고 Garbage Collector가 참조되지 않는 메모리를 확인하고 제거하는 영역입니다

8. JVM (Java Virtual Machine)

(3) Stack area

메서드 호출 시마다 각각의 스택 프레임(그 메서드만을 위한 공간)이 생성합니다. 그리고 메서드 안에서 사용되는 값들을 저장하고, 호출된 메서드의 매개변수, 지역변수, 리턴 값 및 연산 시 일어나는 값들을 임시로 저장합니다. 마지막으로, 메서드 수행이 끝나면 프레임별로 삭제합니다.



8. JVM (Java Virtual Machine)

(4) PC Register

쓰레드가 시작될 때 생성되며, 생성될 때마다 생성되는 공간으로 쓰레드마다 하나씩 존재합니다. 쓰레드가 어떤 부분을 무슨 명령으로 실행해야할 지에 대한 기록을 하는 부분으로 현재 수행중인 JVM 명령의 주소를 갖습니다.

(5) Native method stack

자바 외 언어로 작성된 네이티브 코드를 위한 메모리 영역입니다.

9. 배열(Array)

1. 배열이란 ?

3. 1차원 배열

4. 2차원 배열

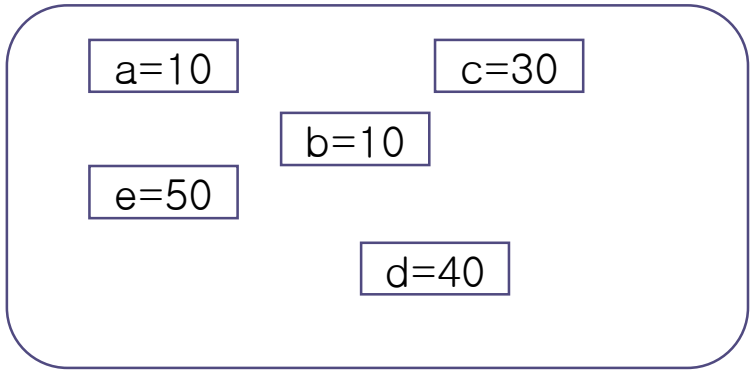
9. 배열(Array)

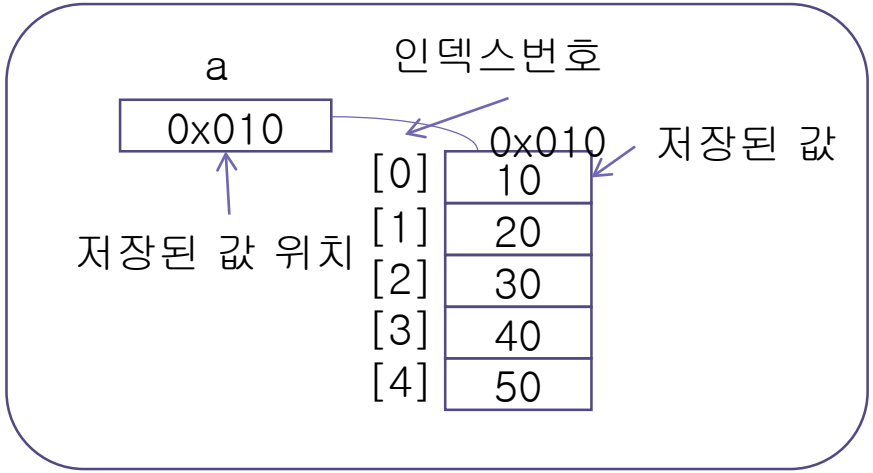
1. 배열이란?

- 같은 형(type)의 데이터를 하나의 자료구조(동일한 이름을 가진 기억장소의 집합:아파트)
- 배열을 이용하면 한 번에 많은 메모리 공간 할당 가능
- 인덱스를 이용하여 원소 데이터 접근
- 배열 인덱스는 0부터 시작
- 배열이 초기화 되지 않을 경우에는 묵시적인 값으로 자동 설정된다

5개의 정수 변수를 사용한 경우
int a=10. b=20, c=30, d=40, e=50

5개의 정수로 구성된 a 배열을 사용하는 경우
int[] a = {10,20,30,40,50}





배열을 문자열로 변환
Arrays.*toString(배열명)*

9. 배열(Array)

2. 배열의 선언과 할당

2_1) 배열의 선언(괄호 [])

```
자료형[] 배열변수이름;
int      intArray[];
char[]   charArray;
String   strArray[];
```

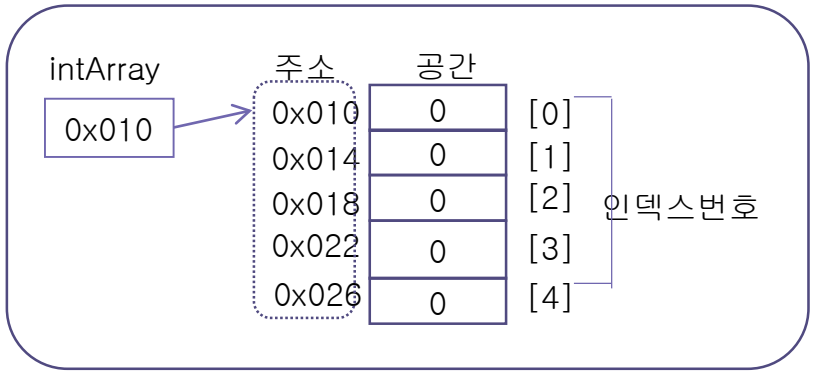
- 배열을 선언하면 메모리에 배열 변수가 다음과 같이 만들어진다
- 배열변수는 첫 번째 요소의 위치(주소)을 저장하는 변수

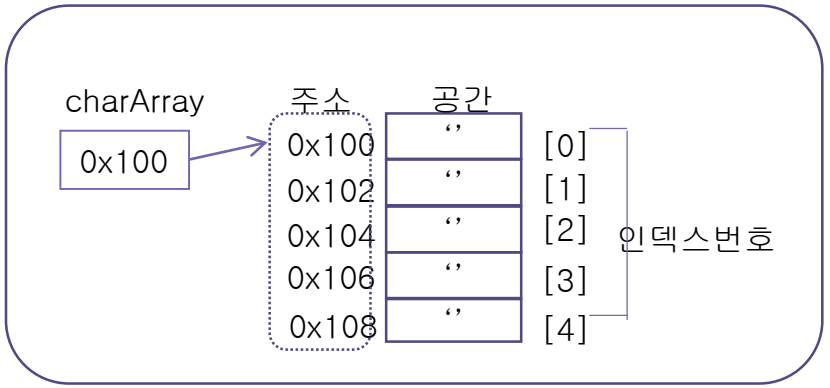
9. 배열(Array)

2_2) 배열의 할당(생성)

```
intArray = new  int[5];
charArray = new  char[5];
strArray = new String[5];
```

배열이 할당되면 묵시적인 값으로 자동 설정된다
(정수형은 0, 실수형은 0.0, 논리형은 false, 문자형은 '')

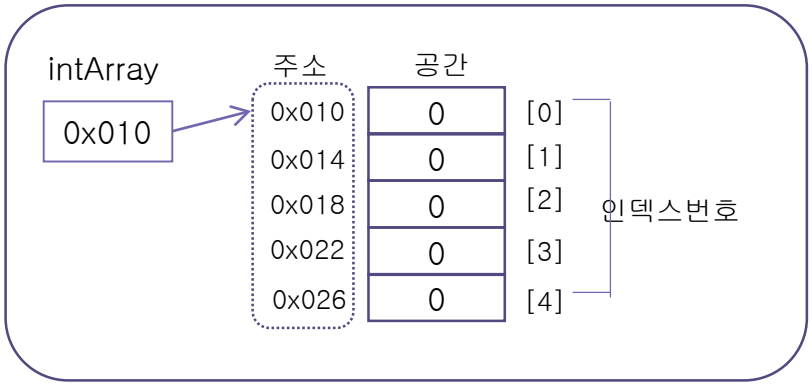




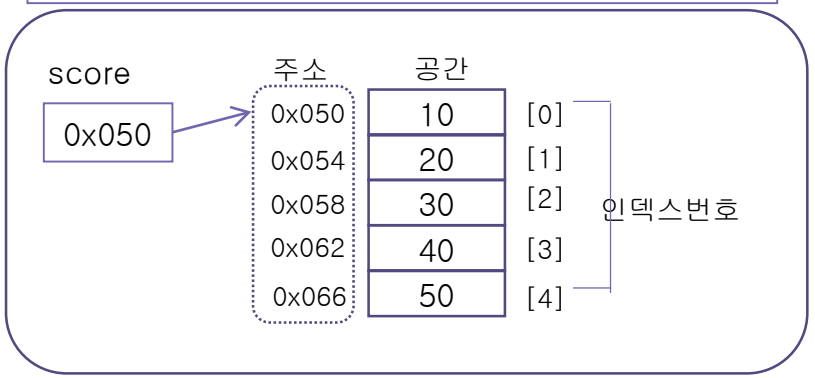
9. 배열(Array)

2_3) 배열의 선언과 동시 할당 및 초기화

```
//선언과 동시 묵시적 초기화
int intArray[] = new int[5];
```




```
//선언과 동시에 명시적 초기화
int score[] = {10,20,30,40,50};
```



9. 배열(Array)

3. 배열 인덱스와 원소 접근

- 배열의 인덱스는 0부터 시작
- 배열의 마지막 항목의 인덱스는 (배열 크기 – 1)
- 배열의 길이 (배열명.length)

```
int intArray = new int[5]; // 원소가 5개인 배열 생성. 인덱스는 0~4까지 가능
intArray[0] = 5; // 원소 0에 5 저장
intArray[3] = 6; // 원소 3에 6 저장
int n = intArray[3]; // 원소 3의 값을 읽어 n에 저장. n은 6이 됨

Int count = intArry.length;  // intArry의 길이 5를 반환
```

– 전체 배열 요소 출력

```
for(int i=0; i<intArry.length; i++) {
   System.out.print(intArray[i]);
}
```

– 전체 배열 요소 출력

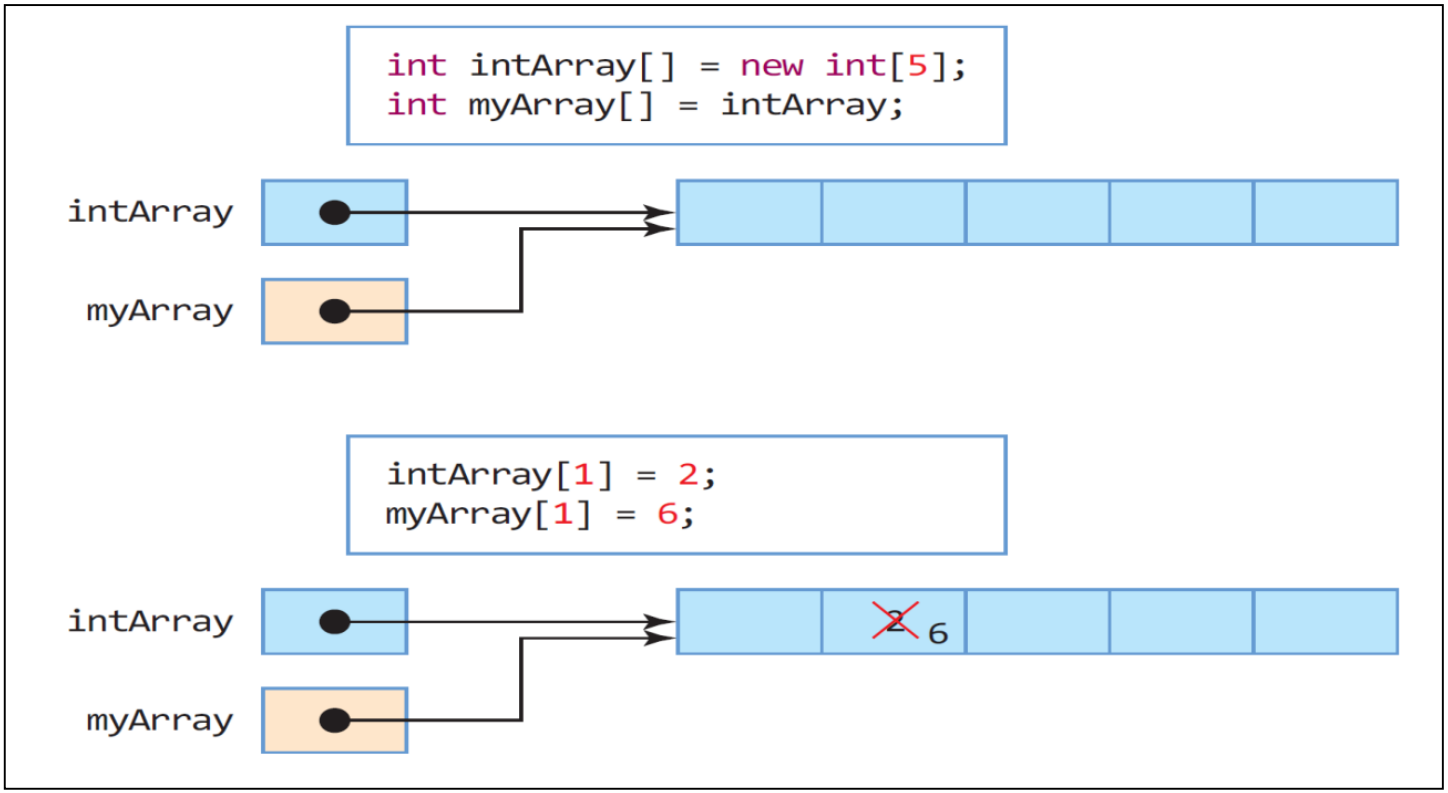
```
for(int a  :  intArry) {
   System.out.print(a);
}
```

9. 배열(Array)

4. 레퍼런스 치환과 배열 공유, 배열 복제

4-1) 배열공유

- 하나의 배열을 다수의 레퍼런스가 참조 가능



9. 배열(Array)

4_2) 배열 복제 :

- 자신을 복제하여 새로운 인스턴스 생성
- 어떤 인스턴스에 대해 작업을 할 때 원래의 인스턴스는 보존하고, clone() 메소드를 이용해서 새로운 인스턴을 생성하여 작업을 하면 작업이전의 값이 보존되므로 작업에 실패해서 원래의 상태로 되돌리거나 변경되기전의 값을 참고하는데 도움

- for()을 이용한 복제
  - 새로운 배열 생성 후 복제

- Object 클래스 : clone() 메서드

```
int intArray2[] = intArray.clone();
```

- System.arraycopy() 메소드
  - 새로운 배열 생성후 복제

```
System.arraycopy(arr, 0, newArr, 0, arr.length)
```

9. 배열(Array)

5. 다차원 배열의 선언과 할당

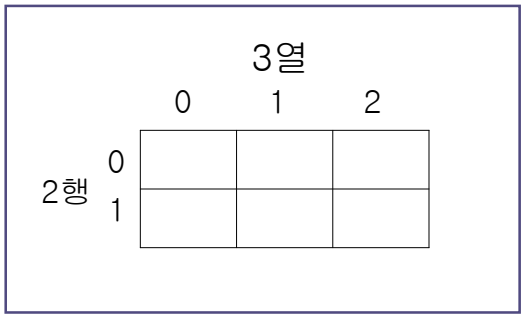
5_1) 2차원 배열의 선언 및 할당

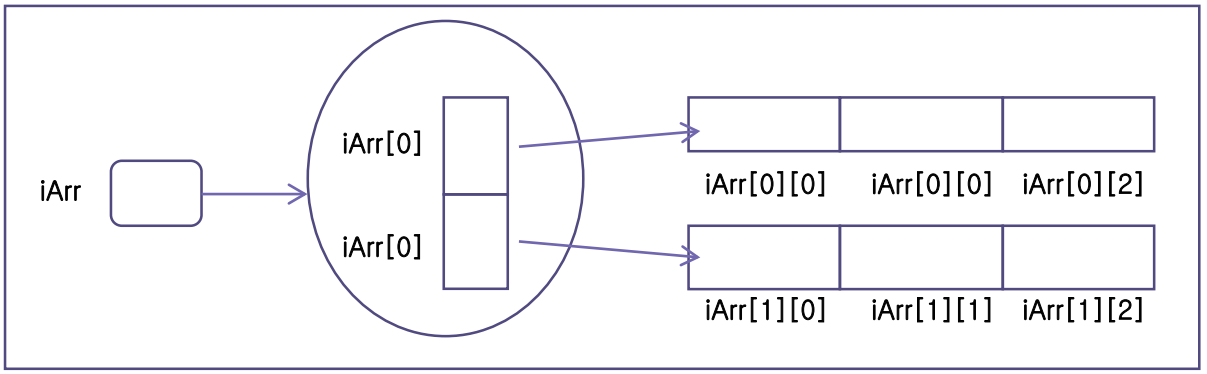
```
//선언
int      intArray[][]; // 정수 배열
char[][]  charArray; //문자 배열

//할당
intArray = new int[2][3];
charArray = new char[3][5];
```

9. 배열(Array)

5_2) 2차원 배열의 모양

예) int iArr = new int[2][3]; 선언한 경우
2행 3열의 2차원 테이블 형태의 배열로 2*3 = 6 개의 int형 공간 할당된다





9. 배열(Array)

5_3) 2차원 배열의 length
iArr.length -> 2차원 배열의 행의 개수로서 2
iArr[n].length는 n번째 행의 열의 개수
iArr[0].length -> 0번째 행의 열의 개수로서 3
iArr[1].length -> 1번째 행의 열의 개수로서 3

5_4) 2차원 가변 배열 선언

```
int score[][] = new int[5][3];
        또는
int score[][] = new int[5][];
score[0] = new int[3];
score[0] = new int[3];
score[0] = new int[3];
```

5_5) 2차원 가변 배열

```
int score[][] = new int[5][];
 score[0] = new int[3];
 score[0] = new int[4];
 score[0] = new int[5];
```

2차원 가변 배열의 선언과 명시적 초기화

```
int score[][] = {{1,2,3},
                 {4,5,6},
                 {7,8,9} }
```

2차원 가변 배열 선언과 명시적 초기화

```
int score[][] = {{1,2,3},
                 {1,2,3,4},
                 {1,2,3,4,5} }
```

9. 배열(Array)

2차원 배열의 전체 요소 출력

```
int arr[][] = {{1,2,3}, {1,2,3,4}, {1,2,3,4,5} }

System.out.println("for() 배열 요소 출력");
for(int i=0; i<arr1.length; i++) {
   for(int j=0; j<arr1[i].length; j++) {
      System.out.print(arr1[i][j]+"₩t");
   }
   System.out.println();
}

System.out.println("확장 for() 배열 요소 출력");
for(int a[] : arr1) {
    for(int b : a) {
       System.out.print(b + "₩t");
   }
   System.out.println();
}
```

9. 배열(Array)

2차원 배열 복제

```
public static void main(String[] args) {
      int a[][] = {{1,2,3},{4,5,6,},{7,8,9}};
      int[][] b = new int[a.length][a[0].length];
      for(int i=0; i<a.length; i++) {
         for(int j=0; j<a[i].length; j++) {
            b[i][j] = a[i][j];
         }
      }
 }
```

```
public static void main(String[] args) {
      int a[][] = {{1,2,3},{4,5,6,},{7,8,9}};
      int b[][] = new int[a.length][a[0].length];
      for(int i=0; i<b.length; i++){
         System.arraycopy(a[i], 0, b[i], 0, a[0].length);
      }
}
```

10. 메소드

1. 메소드의 정의 및 호출

- 메소드(method)란 어떠한 특정 작업을 수행하기 위한 명령문의 집합(다른 언어에서는 함수)

- 메소드 선언 및 호출

```
// 메소드 선언
static  [ void 또는 리턴 또는 반환타입 ]   메소드이름 (인자 또는 매개변수 리스트[생략가능]) {

      내용;

     return 값; // void일 경우 생략
}


// 메소드 호출(main()메소드에서 호출)
 public static void main(String[] args) {
    메소드이름();

    변수 = 메소드이름(인자리스트); //반환타입이 있는 경우
 }
```

10. 메소드

2. 메소드 유형

- 리턴 또는 반환타입 X, 인자 또는 매개변수 X
- 리턴 또는 반환타입 O, 인자 또는 매개변수 X
- 리턴 또는 반환타입 X, 인자 또는 매개변수 O
- 리턴 또는 반환타입 O, 인자 또는 매개변수 O

3. 가변인자

- 하나의 함수에서 매개변수를 동적으로 받을 수 있는 방법으로 가변인자를 사용 할 수 있다.
- 가변인자는 가변인자를 나타내는 기호(...)를 사용

```
void  test(String ... str)
```

10. 메소드

4. 메소드 호출 유형(Call By Value, Call By Reference)

```
## 메소드 호출 (값에의한 호출/ 참조에위한 호출) 예제 ##
public class Exam_08 {
   static void swap(int x, int y) {
      int temp =x;
      x = y;
      y = temp;
   }
   static void swapArr(int[] tmp) {
      int temp = tmp[0];
      tmp[0] = tmp[1];
      tmp[1] = temp;
   }
   public static void main(String[] args) {
      int a=100;
      int b=200;
      System.out.println("a="+a+", b="+b);
      swap(a,b); // 값에의한 호출
      System.out.println("a="+a+", b="+b);

      int arr[] = new int[2];
      arr[0]=100;
      arr[1]=200;
      System.out.println("arr[0]="+arr[0]+", arr[1]="+arr[1]);
      swapArr(arr); //참조에 의한 호출
      System.out.println("arr[0]="+arr[0]+", arr[1]"=+arr[1]);
   }
}
```

```
##  실행 결과 ##
 1. call by value(값에의한 호출)인 경우
    두변수의 값이 변경 안됨

 데이터 교환 메소드 호출 전
 a=100, b=200

 데이터 교환 메소드 호출 후
 a=100, b=200

 2. call by reference(참조에 의한 호출) 값이 변경

배열 데이터 교환 메소드 호출 전
arr[0]=100, arr[1]=200

배열 데이터 교환 메소드 호출 전
arr[0]=200, arr[1]=100
```

10. 메소드

5. 재귀 (Recursion) 함수란

- 특정 함수 내에서 자기 자신을 다시 호출하여 문제를 해결해나가는 함수입니다.
- 문제를 해결하기 위해 원래 범위의 문제에서 더 작은 범위의 하위 문제를 먼저 해결함으로써 원래 문제를 해결해 나가는 방식
- 재귀 함수는 함수 내에서 자기 자신을 계속 호출하는 방식이기 때문에 함수 안에 반드시 종료 구간이 되는 Base Case를 생각하며 코드를 구현

```
public class Recursion_Ex01 {
    public static void main(String[] args) {
        function();
    }

    public static void function() {
        System.out.println("Hello World");
        function();
    }
}

//무한 루프에 빠짐
```

10. 메소드

1부터 n까지 합을 구하는 함수를 구현

```
public class Recursion_Ex02 {
    public static void main(String[] args) {
        System.out.println(function(5));
    }

    public static int function(int num) {
        if(num==1)
            return 1;
        return num + function(num-1);
    }
}
```