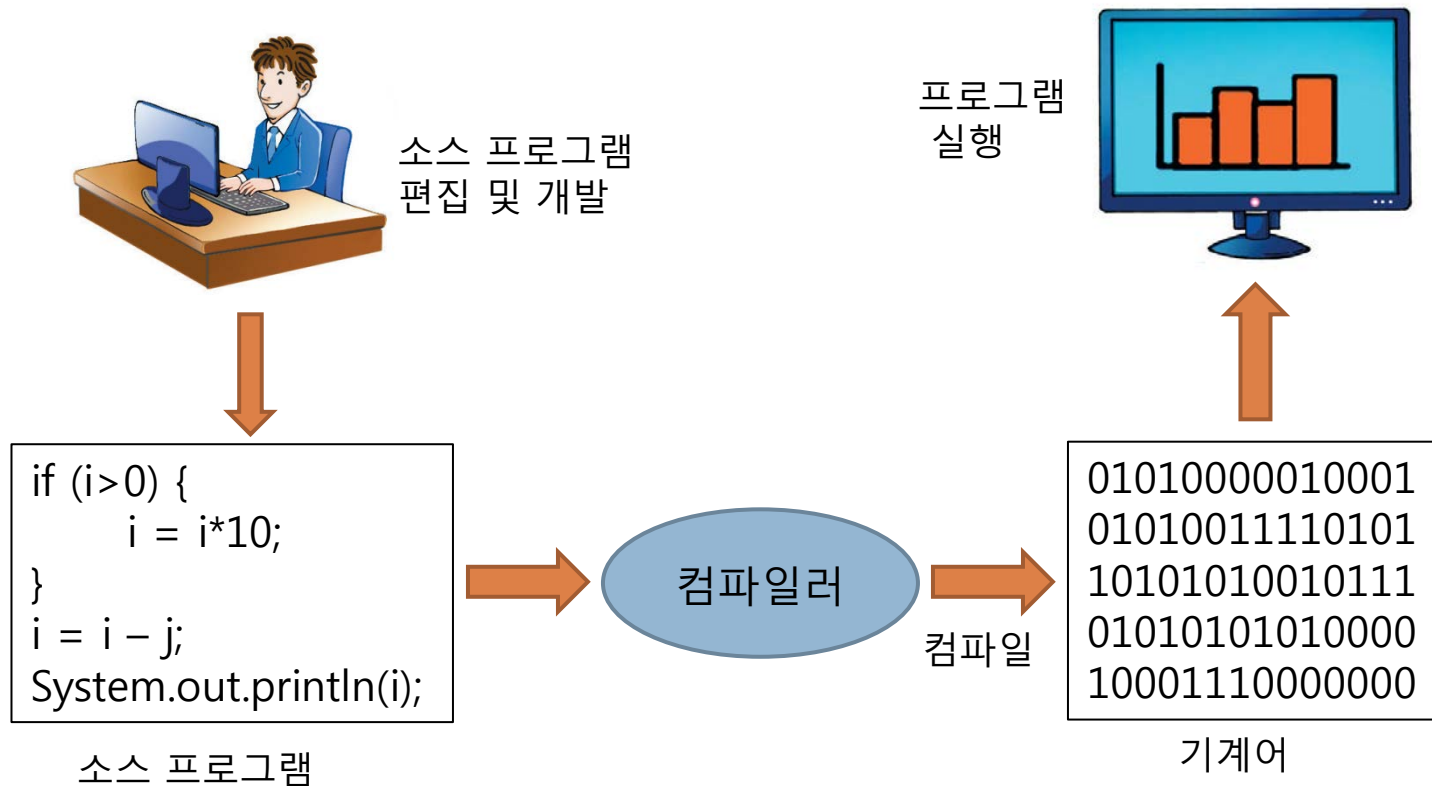


# 1. JAVA BASIC

# 컴파일

3

- 소스 : 프로그래밍 언어로 작성된 텍스트 파일
- 컴파일 : 소스 파일을 컴퓨터가 이해할 수 있는 기계어로 만드는 과정
  - ▣ 소스 파일 확장자와 컴파일 된 파일의 확장자
    - 자바 : **.java** -> **.class**
    - C : **.c** -> **.obj**-> **.exe**
    - C++ : **.cpp** -> **.obj** -> **.exe**



# 플랫폼 종속성(platform dependency)

4



인텔 CPU를 가진  
리눅스 환경에서  
개발

C/C++  
응용 프로그램

플랫폼 = 하드웨어 플랫폼 + 운영체제 플랫폼

프로그램의 플랫폼 호환성 없는 이유

- 기계어가 CPU마다 다름
- 운영체제마다 API 다름
- 운영체제마다 실행파일 형식 다름

실행



인텔 CPU + 리눅스

실행되지  
않음



Apple 사의 MAC PC

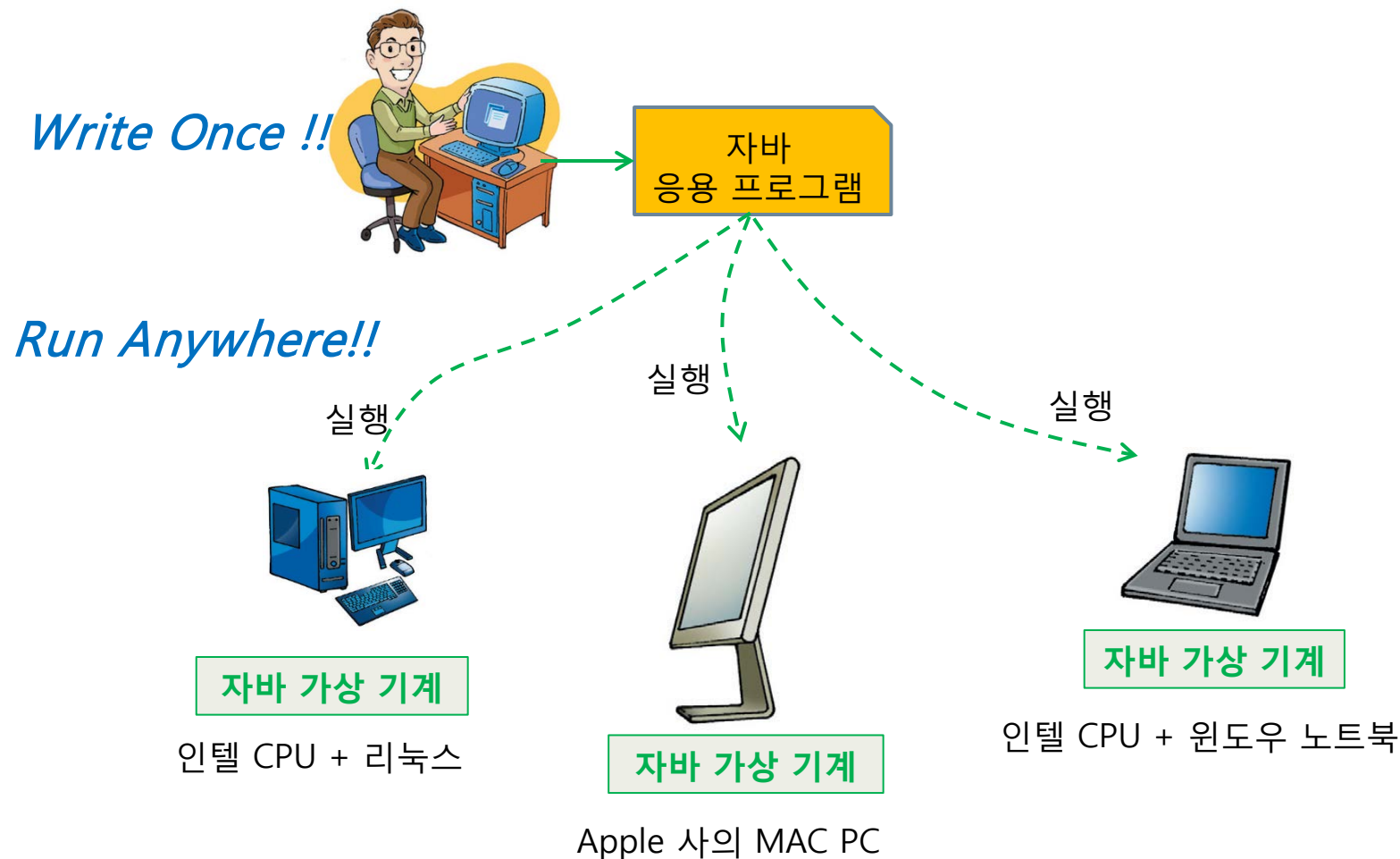
실행되지  
않음



인텔 CPU + 윈도우 노트북

# 자바의 플랫폼 독립성, WORA

5



# 자바의 실행 환경

6

## □ 바이트 코드

- 자바 가상 기계에서 실행 가능한 바이너리 코드
  - 바이트 코드는 컴퓨터 CPU에 의해 직접 실행되지 않음
  - 자바 가상 기계가 작동 중인 플랫폼에서 실행
  - 자바 가상 기계가 인터프리터 방식으로 바이트 코드 해석
- 클래스 파일(.class)에 저장

## □ 자바 가상 기계(JVM : Java Virtual Machine)

- 각기 다른 플랫폼에 설치
- 동일한 자바 실행 환경 제공
- 자바 가상 기계 자체는 플랫폼에 종속적
  - 자바 가상 기계는 플랫폼마다 각각 작성됨
  - 예) 리눅스에서 작동하는 자바 가상 기계는 윈도우에서 작동하지 않음
- 자바 가상 기계 개발 및 공급
  - 자바 개발사인 오라클 외 IBM, MS 등 다양한 회사에서 제작 공급

## □ 자바의 실행

- 자바 가상 기계가 클래스 파일(.class)의 바이트 코드 실행

# 자바와 오픈 소스

7

- 오픈 소스란?
  - ▣ 소프트웨어 제작자의 권리를 보존
  - ▣ 누구나 액세스할 수 있도록 소스 코드를 무상 공개한 소프트웨어
- 오픈 소스의 장점
  - ▣ 공개된 소스 코드를 참조함으로써 개발 시간 및 비용 단축
  - ▣ 공개된 소프트웨어를 다수의 인원이 참여 개량, 우수한 품질의 소프트웨어 개발
- 오픈 소스의 단점
  - ▣ 무단으로 상용 소프트웨어에 사용할 경우 저작권 침해 발생
  - ▣ 다양한 개량 버전의 소프트웨어로 인한 호환성 문제
- 오픈 소스 소프트웨어 사례
  - ▣ Linux, OpenOffice, Open Solaris, Mozilla, Apache, GNU, WebKit 등
  - ▣ 2006년 11월, 선마이크로시스템즈는 자바를 GPL 라이선스로 소스 오픈
  - ▣ <http://sourceforge.net> : 오픈 소스 사이트

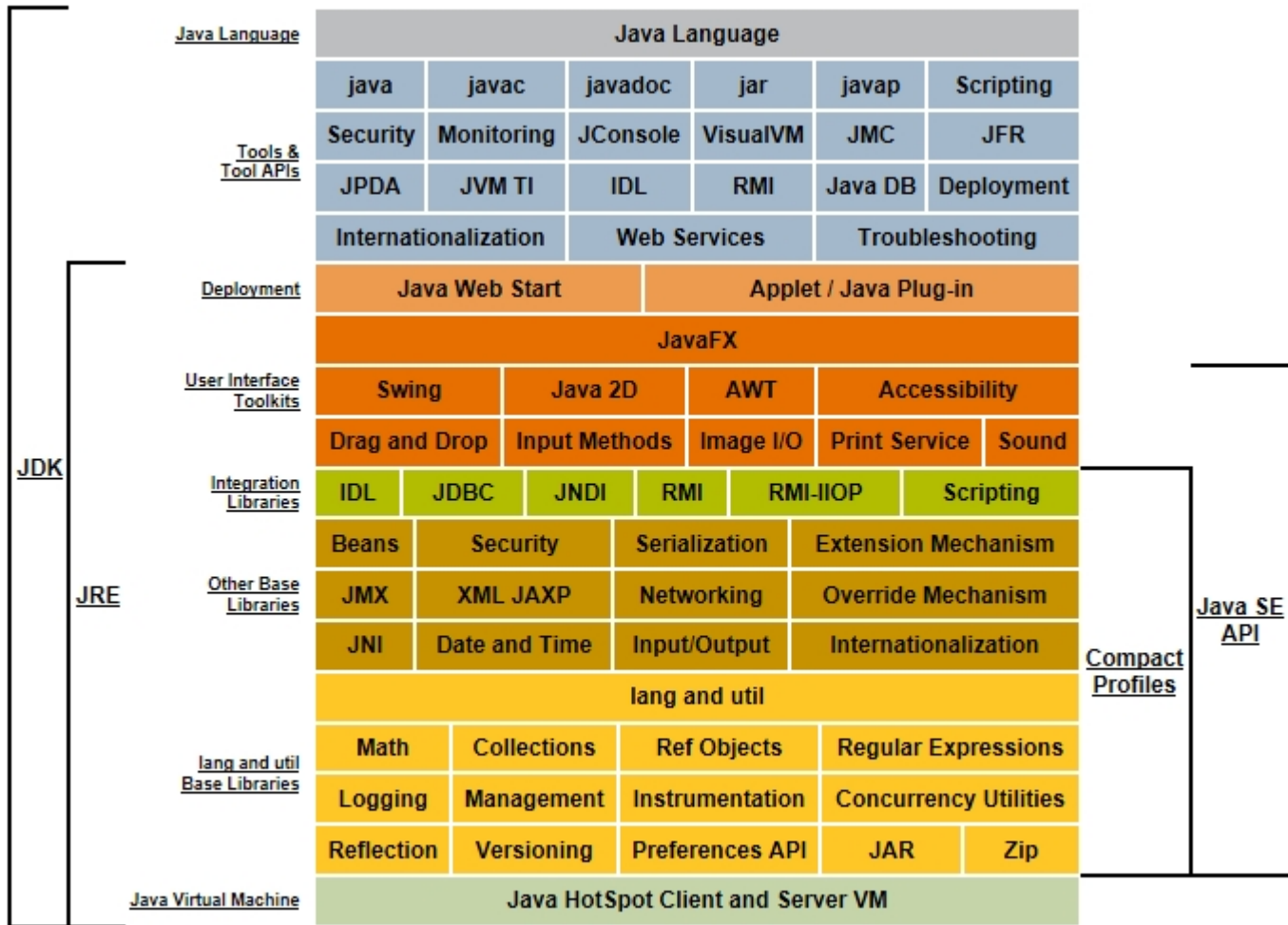
# 자바의 배포판 종류

8

- 오라클은 개발 환경에 따라 다양한 자바 배포판 제공
- Java SE
  - ▣ 자바 표준 배포판(Standard Edition)
  - ▣ 데스크탑과 서버 응용 개발 플랫폼
- Java ME
  - ▣ 자바 마이크로 배포판
    - 휴대 전화나 PDA, 셋톱박스 등 제한된 리소스를 갖는 하드웨어에서 응용 개발을 위한 플랫폼
    - 가장 작은 메모리 풋프린트
  - ▣ Java SE의 서브셋 + 임베디드 및 가전 제품을 위한 API 정의
- Java EE
  - ▣ 자바 기업용 배포판
    - 자바를 이용한 다중 사용자, 기업용 응용 개발을 위한 플랫폼
  - ▣ Java SE + 인터넷 기반의 서버사이드 컴퓨팅 관련 API 추가

# Java SE 구성

9



출처: <http://download.oracle.com/javase/8/docs/>



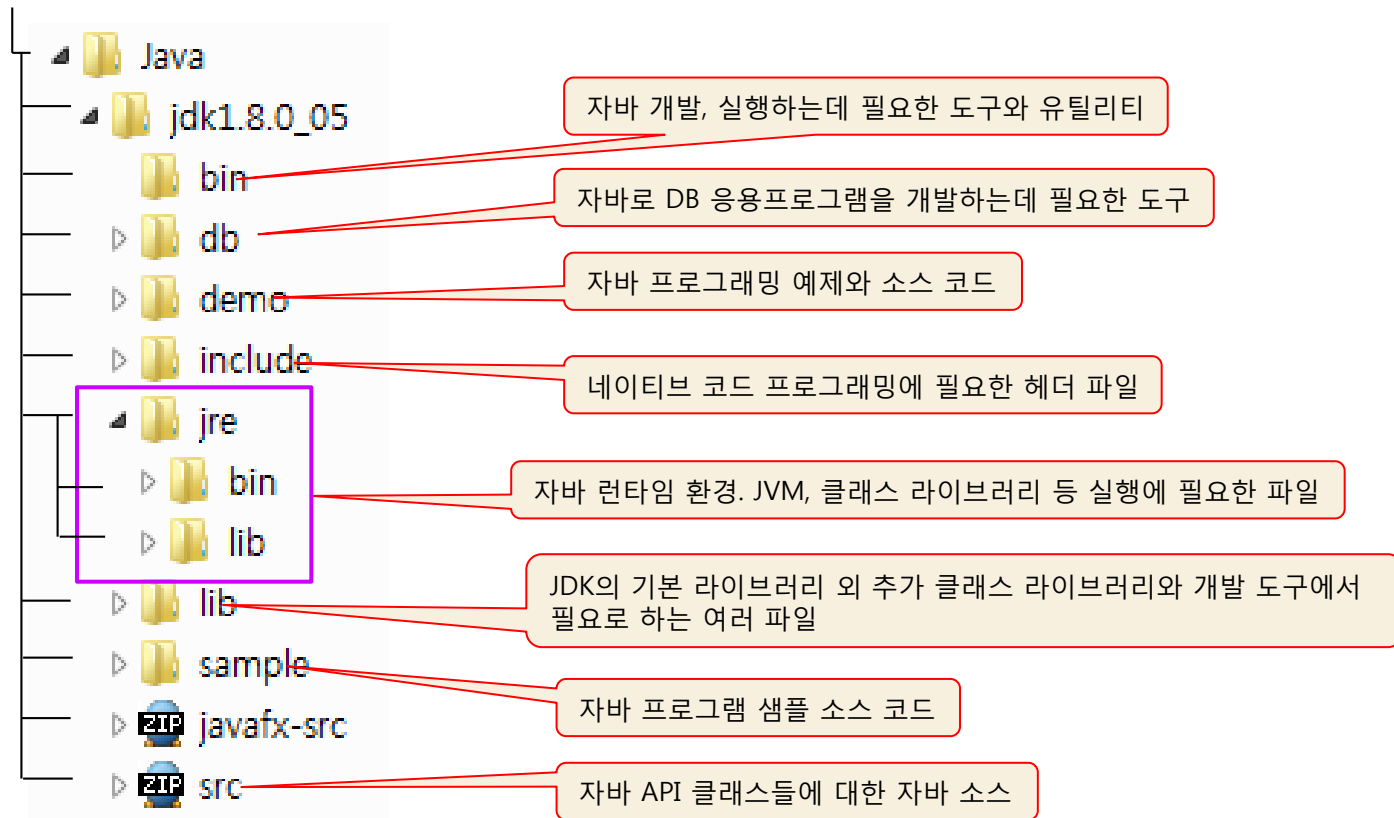
# JDK와 JRE

10

- JDK(Java Development Kit)
  - ▣ 자바 응용 개발 환경. 개발에 필요한 도구 포함
    - 컴파일러, JRE (Java Runtime Environment), 클래스 라이브러리, 샘플 등 포함
- JRE(Java Runtime Environment)
  - ▣ 자바 실행 환경. JVM 포함
  - ▣ 자바 실행 환경만 필요한 경우 JRE만 따로 다운 가능
- JDK와 JRE의 개발 및 배포
  - ▣ 오라클의 Technology Network의 자바 사이트에서 다운로드
    - <http://www.oracle.com/technetwork/java/index.html>
- JDK의 bin 디렉터리에 포함된 주요 개발 도구
  - ▣ javac - 자바 소스를 바이트 코드로 변환하는 컴파일러
  - ▣ java - jre의 bin 디렉터리에 있는 자바 응용프로그램 실행기
  - ▣ jar - 자바 아카이브 파일 (JAR)의 생성 및 관리하는 유틸리티
  - ▣ jdb - 자바 디버거
  - ▣ appletviewer - 웹 브라우저 없이 애플릿을 실행하는 유틸리티

# JDK 설치 후 디렉터리 구조

11



# 자바 API

12

- 자바 API(Application Programming Interface)란?
  - ▣ JDK에 포함된 클래스 라이브러리
    - 주요한 기능들을 미리 구현한 클래스 라이브러리의 집합
  - ▣ 개발자는 API를 이용하여 쉽고 빠르게 자바 프로그램 개발
    - API에서 정의한 규격에 따라 클래스 사용
- 자바 패키지(package)
  - ▣ 서로 관련된 클래스들을 분류하여 묶어 놓은 것
  - ▣ 계층구조로 되어 있음
    - 클래스의 이름에 패키지 이름도 포함
    - 다른 패키지에 동일한 이름의 클래스 존재 가능
  - ▣ 자바 API(클래스 라이브러리)는 JDK에 패키지 형태로 제공됨
    - 필요한 클래스가 속한 패키지만 import하여 사용
  - ▣ 개발자 자신의 패키지 생성 가능

# 자바 통합 개발 환경-이클립스(Eclipse)

14

- IDE(Integrated Development Environment )란?
  - ▣ 통합 개발 환경
  - ▣ 편집, 컴파일, 디버깅을 한번에 할 수 있는 통합된 개발 환경
- 이클립스(Eclipse)
  - ▣ 자바 응용 프로그램 개발을 위한 통합 개발 환경
  - ▣ IBM에 의해 개발된 오픈 소스 프로젝트
  - ▣ <http://www.eclipse.org/downloads/> 에서 다운로드

# 자바의 특성(1)

22

- 객체지향
  - ▣ 객체지향의 특징인 클래스 계층 구조, 상속성, 다형성, 캡슐화 등 지원
- 멀티스레드
  - ▣ 다수 스레드의 동시 수행 환경 지원
    - 자바는 운영체제의 도움 없이 자체적으로 멀티스레드 지원
    - C/C++ 등에서는 멀티스레드를 위해 운영체제 API를 호출
- 플랫폼 독립성
  - ▣ 자바 가상 기계가 바이트 코드 실행
    - 플랫폼에 종속성을 갖지 않음
- 소스(.java)와 클래스(.class) 파일
  - ▣ 하나의 소스 파일에 여러 클래스를 작성 가능
    - 하나의 public 클래스만 가능
  - ▣ 소스 파일의 이름과 public으로 선언된 클래스 이름은 같아야 함
  - ▣ 클래스 파일에는 단 하나 만의 클래스만 존재
    - 다수의 클래스를 가진 자바 소스를 컴파일하면 클래스마다 별도 클래스 파일 생성

# 자바의 특징(2)

23

- 실행 모듈
  - ▣ 한 개의 class 파일 또는 다수의 class 파일로 구성
  - ▣ 여러 폴더에 걸쳐 다수의 클래스 파일로 구성된 경우
    - jar 파일 형태로 배포 가능
- main() 메소드
  - ▣ 자바 응용프로그램의 실행은 main() 메소드에서 시작
  - ▣ 하나의 클래스 파일에 **하나 이상의 main() 메소드가 있을 수 없음**
    - 각 클래스 파일이 main() 메소드를 포함하는 것은 상관없음
- 클래스로 캡슐화
  - ▣ 자바의 모든 변수나 함수는 클래스 내에 선언
  - ▣ 클래스 안에서 새로운 클래스(내부 클래스) 작성 가능
- 패키지
  - ▣ 관련된 여러 클래스를 패키지로 묶어 관리
  - ▣ 패키지는 폴더 개념
    - 예) java.lang.System은 java\lang 디렉터리의 System.class 파일

# 자바 프로그램 구조 - 예제 1

24

```
/*  
 * 맞보기 예제.  
 * 소스 파일 : Hello2.java  
 */
```

```
public class Hello2 {
```

```
    public static int sum(int n, int m) {  
        return n + m;  
    }
```

메소드

```
    // main() 메소드에서 실행 시작
```

```
    public static void main(String[] args) {  
        int i = 20;  
        int s;  
        char a;
```

```
        s = sum(i, 10); // sum() 메소드 호출
```

```
        a = '?';
```

```
        System.out.println(a); // 문자 '?' 화면 출력
```

```
        System.out.println("Hello2"); // "Hello2" 문자열 화면 출력
```

```
        System.out.println(s); // 정수 s 값 화면 출력
```

```
    }
```

```
}
```

클래스

```
?  
Hello2  
30
```

메소드

# 맛보기 예제 설명

25

## □ 클래스 만들기

- Hello2 이름의 클래스 선언

```
public class Hello2 {  
}
```

- class 키워드로 클래스 정의(4장 참고)
- public으로 선언하면 다른 클래스에서도 접근 가능
- 클래스 코드는 {} 내에 모두 작성

## □ main() 메소드

- public static void으로 선언되어야 함

```
public static void main(String[] args) {  
}
```

- 자바 프로그램은 main() 메소드부터 실행 시작
- String[] args로 실행 인자를 전달 받음(3장 참고)

## □ 멤버 메소드

- 메소드 sum()

```
public static int sum(int n, int m) {  
    ...  
}
```

- 클래스에 속한 함수로서, 클래스 내에서만 선언

## □ 변수 선언

- 변수 타입과 변수 이름 선언

```
int i=20;  
int s;  
char a;
```

- 메소드 내에서 선언된 변수는 지역 변수
  - 지역 변수는 메소드 실행이 끝나면 저장 공간 반환

## □ 메소드 호출

- sum() 메소드 호출

```
s = sum(i, 10); // 메소드 sum() 호출
```

- sum() 메소드의 호출 시 변수 i의 값과 정수 10을 전달
- sum() 메소드의 n, m에 각각 20, 10 값 전달
- sum() 메소드는 n과 m 값을 더한 30 리턴
- 변수 s는 정수 30을 전달받아 저장



# sum() 메소드 호출과 리턴

26

```
public static int sum(int n, int m) {  
    return n + m; // 30 리턴  
}
```

n 20  
m 10

```
int i=20;
```

```
s = sum(i, 10);
```

s 30

sum() 메소드 호출

# 예제 설명(계속)

27

## 주석문

- ▣ //을 만나면 한 라인으로 주석으로 처리
- ▣ /\* 와 \*/ 사이의 여러 행을 주석으로 처리

## 화면 출력

- ▣ 표준 출력 스트림에 메시지 출력

```
System.out.println(a);      // 문자 ? 화면 출력
System.out.println("Hello2"); // "Hello2" 화면 출력
System.out.println(s);      // 정수 s 값 화면 출력
```

- ▣ 표준 출력 스트림 System.out의 println() 메소드 호출
- ▣ println()은 여러 타입의 데이터 출력 가능
- ▣ println()은 출력 후 다음 행으로 커서 이동

## 문장

- ▣ ;로 한 문장의 끝을 인식

```
int i=20;
b = '?';
s = sum(i, 20);
```

- ▣ 한 문장을 여러 줄에 작성해도 무방

```
b
= '?';
```

- ▣ 주석문 끝에는 ';'를 붙이지 않음

## 블록

- ▣ 블록은 { 로 시작하여 } 로 끝남

```
public class Hello2 {
    ....
} // Hello2 클래스 끝

public static void main(String[] args) {
    ...
} // main() 코드 끝
```

- ▣ 클래스와 메소드는 모두 블록으로 구성

# 식별자 (identifier)

28

- 식별자란?
  - ▣ 클래스, 변수, 상수, 메소드 등에 붙이는 이름
- 식별자의 원칙
  - ▣ '@', '#', '!'와 같은 특수 문자, 공백 또는 탭은 식별자로 사용할 수 없으나 '\_', '\$'는 사용 가능
  - ▣ 유니코드 문자 사용 가능. 한글 사용 가능
  - ▣ 자바 언어의 키워드는 식별자로 사용불가
  - ▣ 식별자의 첫 번째 문자로 숫자는 사용불가
  - ▣ '' 또는 '\$'를 식별자 첫 번째 문자로 사용할 수 있으나 일반적으로 잘 사용하지 않는다.
  - ▣ 불린 리터럴 (true, false)과 널 리터럴(null)은 식별자로 사용불가
  - ▣ 길이 제한 없음
- 대소문자 구별
  - ▣ Test와 test는 별개의 식별자

# 식별자 이름 사례

29

## □ 사용 가능한 예

```
int name;
char student_ID;           // '_' 사용 가능
void $func() { }           // '$' 사용 가능
class Monster3 { }         // 숫자 사용 가능
int whatsyournamemynameiskitae; // 길이 제한 없음
int barChart; int barchart; // 대소문자 구분. barChart와 barchart는 다름
int 가격;                  // 한글 이름 사용 가능
```

## □ 잘못된 예

```
int 3Chapter;              // 숫자 사용 불가
class if { }               // if는 자바의 예약어
char false;                // false는 사용 불가
void null() { }            // null 사용 불가
class %calc { }            // '%'는 특수문자
```

# 자바 키워드

30

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

# 식별자 이름 붙이는 관습

31

- 기본 : 헝그리안 이름 붙이기
- 클래스 이름

```
public class HelloWorld {}  
class Vehicle {}  
class AutoVendingMachine {}
```

- 첫 번째 문자는 대문자로 시작
- 여러 단어가 복합될 때 각 단어의 첫 번째 문자만 대문자
- 변수, 메소드 이름

```
int iAge;           // iAge의 i는 int의 i를 표시  
boolean bIsSingle; // bIsSingle의 처음 b는 boolean의 b를 표시  
String strName;     // strName의 str은 String의 str을 표시  
public int iGetAge() {} // iGetAge의 i는 int의 i를 표시
```

- 첫 단어 이후 각 단어의 첫 번째 문자는 대문자로 시작
- 상수 이름

```
final static double PI = 3.141592;
```

- 모든 문자를 대문자로 표시

# 자바의 데이터 타입

32

## □ 자바의 데이터 타입

### ▣ 기본 타입 : 8 개

- boolean
- char
- byte
- short
- int
- long
- float
- double

레퍼런스는 C/C++의 포인터와 유사한 개념  
그러나 메모리 주소값을 가지지 않음

### ▣ 레퍼런스 타입 : 1 개이며 용도는 다음 3 가지

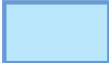
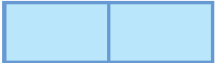

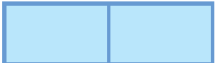
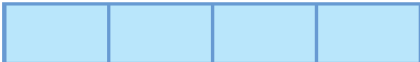



- 클래스(class)에 대한 레퍼런스
- 인터페이스(interface)에 대한 레퍼런스
- 배열(array)에 대한 레퍼런스

# 자바의 기본 데이터 타입

33

## □ 특징

- 기본 데이터 타입의 크기가 정해져 있음
- 기본 데이터 타입의 크기는 CPU나 운영체제에 따라 변하지 않음

논리 타입	boolean		(1바이트, true 또는 false)
문자 타입	char		(2바이트, Unicode)
정수 타입	byte		(1바이트, -128~127)
	short		(2바이트, -32,768~32,767)
	int		(4바이트, $-2^{31} \sim 2^{31}-1$ )
	long		(8바이트, $-2^{63} \sim 2^{63}-1$ )
실수 타입	float		(4바이트, $-3.4\text{E}38 \sim 3.4\text{E}38$ )
	double		(8바이트, $-1.7\text{E}308 \sim 1.7\text{E}308$ )



# 변수 선언 사례

35

## □ 변수 선언 사례

```
int radius;  
char c1, c2, c3; // 3 개의 변수를 한 번에 선언한다.  
double weight;
```

## □ 변수 선언과 초기화

### ▣ 선언과 동시에 초기값 지정

```
int radius = 10;  
char c1 = 'a', c2 = 'b', c3 = 'c';  
double weight = 75.56;
```

## □ 변수에 값 대입

### ▣ 대입 연산자인 = 다음에 식(expression)

```
radius = 10 * 5;  
c1 = 'r';  
weight = weight + 5.0;
```

# 변수와 선언

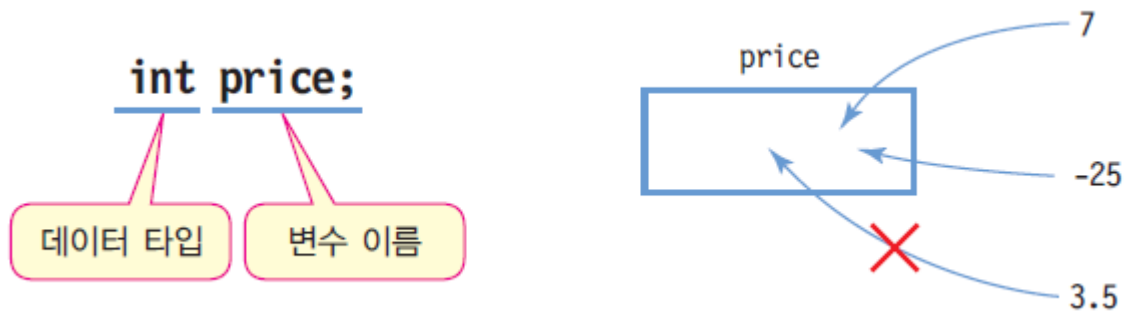
34

## □ 변수

- ▣ 프로그램 실행 중에 값을 임시 저장하기 위한 공간
  - 변수 값은 프로그램 수행 중 변경될 수 있음
- ▣ 데이터 타입에서 정한 크기의 메모리 할당
- ▣ 반드시 변수 선언과 값을 초기화 후 사용

## □ 변수 선언

- ▣ 변수의 타입 다음에 변수 이름을 적어 변수를 선언



# 정수 타입 리터럴

36

- 정수 타입 리터럴 : 정수 직접 표시
  - ▣ 8진수 : 0으로 시작하는 숫자는 모두 8진수
    - `int n = 015;` // 10진수로 13
  - ▣ 16진수 : 0x로 시작하는 숫자는 16진수
    - `int n = 0x15;` // 10진수로 21
  - ▣ 10진수 : 0으로 시작하지 않는 숫자는 10진수
    - 15, 3, 20, 55, 88
  - ▣ 2진수 : 0b로 시작하지 않는 2진 숫자는 2진수
    - `int n = 0b0101;` // 10진수로 5. 이진수 0101 -> 5
- ▣ 모든 정수 리터럴은 int 형으로 컴파일
- ▣ long 타입 리터럴은 숫자 뒤에 L 또는 l을 붙여 표시
  - ex) 24L, 3578l

# 실수 타입 리터럴

37

- 부동 소수점 실수 직접 표시
  - ▣ 소수점을 찍은 실수, 지수(exponent)식으로 표현한 실수
    - 12. 또는 12.0
    - .1234 또는 0.1234 또는 1234E-4
  - ▣ 숫자 뒤에 f(float)나 d(double)을 명시적으로 붙이기도 함
    - 0.1234 또는 0.1234D 또는 0.1234d → double 타입
    - 0.1234f 또는 0.1234F → float 타입
    - 1234D 또는 1234d → 1234.0과 같으며 double 타입
    - 1234F 또는 1234f → 1234.0과 같으며 float 타입
  - ▣ 실수 타입 리터럴은 double 타입으로 컴파일

# 문자 타입 리터럴

38

- 단일 인용부호(' ')로 문자 표현
  - 'a', 'W', '가', '\*', '3', '7'
- `₩u`다음에 4자리 16진수로, 2 바이트의 유니코드(Unicode)
  - `₩u0041` -> 문자 'A'의 유니코드(0041)
  - `₩uae00` -> 한글문자 '글'의 유니코드(ae00)
- 특수 기호는 `₩`로 시작

특수문자 리터럴 (이스케이프 시퀀스)	의미
<code>'\b'</code>	백스페이스(backspace)
<code>'\t'</code>	탭(tab)
<code>'\n'</code>	라인피드(line feed)
<code>'\f'</code>	폼피드(form feed)
<code>'\r'</code>	캐리지 리턴(carriage return)
<code>'\"'</code>	이중 인용부호(double quote)
<code>'\''</code>	단일 인용부호(single quote)
<code>'\\'</code>	백슬래시(backslash)

# 논리 타입 리터럴

39

## □ 논리 값 표시

- ▣ true 또는 false 뿐

```
boolean b = true;  
boolean c = 10 > 0; // 10>0이 참이므로 c 값은 true
```

## □ 논리 타입과 정수타입 사이의 타입 변환 허용 안 됨

```
int i;  
if ((boolean)i){ } // 정수 i를 논리 타입으로 변환할 수 없음. 컴파일 오류
```

- ▣ (i==1) 또는 (i!=0)과 같은 명확한 논리 연산 사용해야 함

# Tip: 기본 타입 이외 리터럴

40

## □ null 리터럴

- 어떠한 레퍼런스 타입의 값으로도 사용 가능
  - ~~int n = null;~~ // 기본 데이터 타입에는 사용 불가
  - String str = null;

## □ 문자열 리터럴

- 이중 인용부호로 묶어서 표현
  - "Good", "Morning", "자바", "3.19", "26", "a"
- 자바에서 문자열은 객체이므로 기본 타입 아님
- 문자열 리터럴은 String 객체로 자동 처리

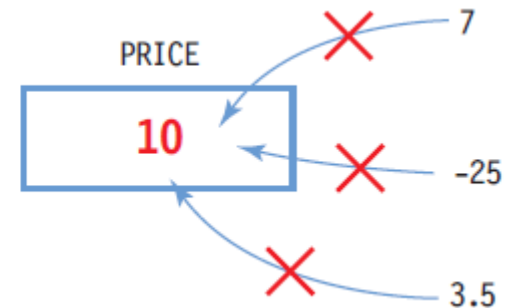
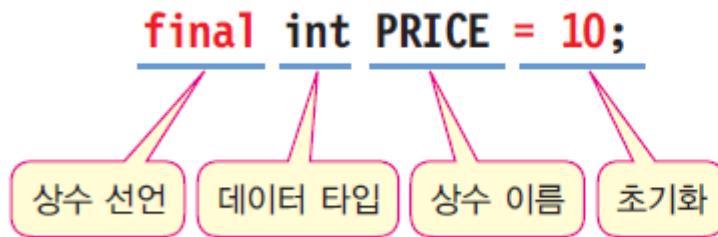
```
String str1 = "Welcome";  
String str2 = null;  
System.out.println(str1);
```

# 상수

41

## □ 상수 선언

- ▣ final 키워드 사용
- ▣ 선언 시 초기값 지정
- ▣ 실행 중 값 변경 불가



## □ 상수 선언 사례

```
final double PI = 3.141592;  
final int LENGTH = 20;
```



# 예제 2-1 : 변수, 리터럴, 상수 사용하기

42

원의 면적을 구하는 프로그램을 작성해보자.

```
public class CircleArea {  
    public static void main(String[] args) {  
        final double PI = 3.14; // 원주율을 상수로 선언  
        double radius = 10; // 원의 반지름  
        double circleArea = 0; // 원의 면적  
  
        circleArea = radius*radius*PI; // 원의 면적 계산  
  
        // 원의 면적을 화면에 출력한다.  
        System.out.print("원의 면적 = ");  
        System.out.println(circleArea);  
    }  
}
```

원의 면적 = 314.0

# 타입 변환 - 자동 타입 변환

43

## □ 자동 타입 변환이 발생하는 경우

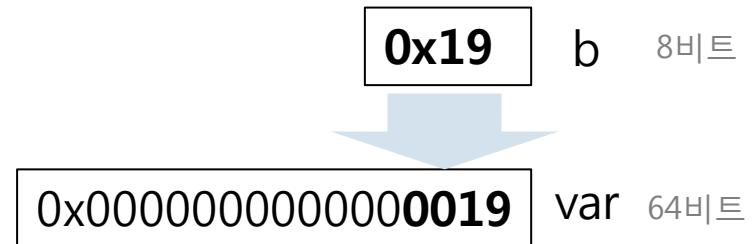
### ▣ 원래의 타입보다 큰 타입으로 바뀔 때

byte >> short/char >> int >> long >> float >> double

### ▣ 원본 값 보존

```
long var;  
byte b = 25; // 0x19  
var = b;
```

롱타입 변수      자동타입변환      바이트타입 변수

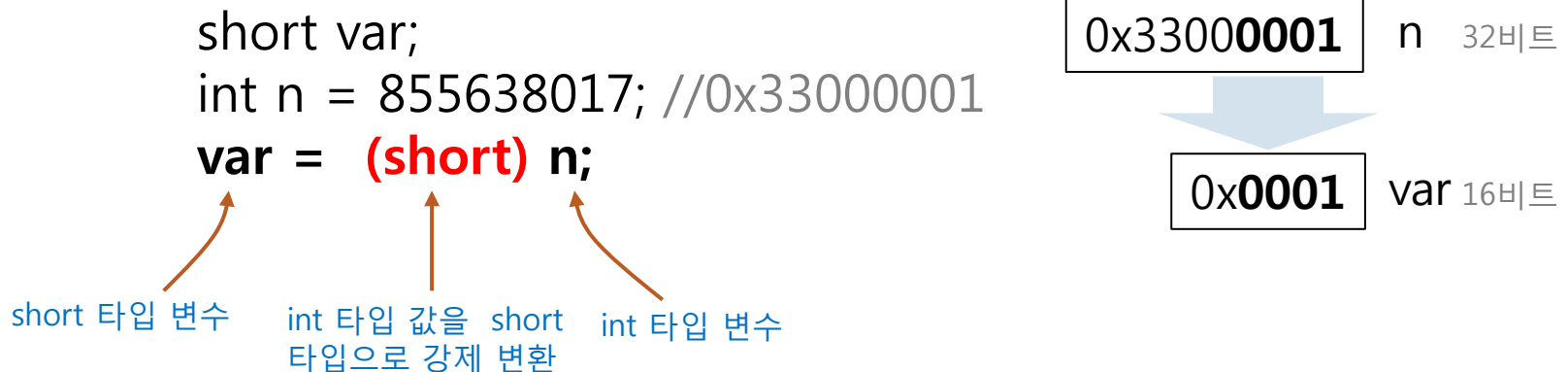


```
long var;  
int n = 32555;  
byte b = 25; // 0x19  
var = n; // int 타입에서 long 타입으로 자동 변환. var 값은 32555  
var = b; // byte 타입에서 long 타입으로 자동 변환. var 값은 25
```

# 강제 타입 변환

44

- 강제 타입 변환 : 개발자의 의도적으로 타입 변환
  - ▣ 개발자가 코드에 명시적으로 타입 변환 지정
    - 실수 타입이 정수 타입으로 강제 변환되면 소수점 아래가 버려짐
      - 데이터 손실



```
short var;  
int n = 855638017; // n의 16진수 값은 0x33000001  
var = (short) n; // int 타입에서 short 타입으로 강제 변환. var 값은 1  
  
double d = 1.9;  
int n = (int)d; // n은 1이 된다.
```

# 예제 2-2 : 자동 타입 변환, 강제 타입 변환

45

자동 타입 변환과 강제 타입 변환의 이해를 위한 예제이다.  
다음 소스의 실행 결과는 무엇인가?

```
public class TypeConversion {  
    public static void main(String[] args) {  
        byte b = 127;  
        int i = 100;  
        System.out.println(b+i);  
        System.out.println(10/4);  
        System.out.println(10.0/4);  
        System.out.println((char)0x12340041);  
        System.out.println((byte)(b+i));  
        System.out.println((int)2.9 + 1.8);  
        System.out.println((int)(2.9 + 1.8));  
        System.out.println((int)2.9 + (int)1.8);  
    }  
}
```

227  
2  
2.5  
A  
-29  
3.8  
4  
3

# Scanner를 이용한 키 입력 - 강추

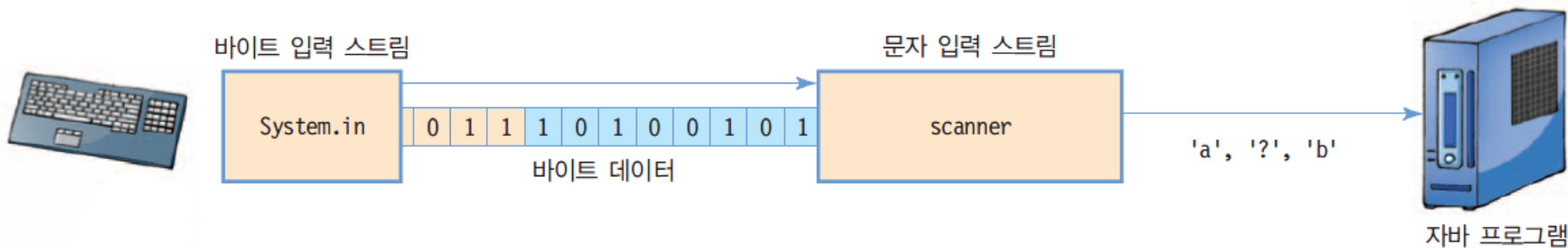
46

## □ Scanner 클래스

### ▣ java.util.Scanner 클래스

- System.in에게 키를 읽게 하고, 읽은 바이트를 문자, 정수, 실수, 불린, 문자열 등 다양한 타입으로 변환하여 리턴

```
Scanner a = new Scanner(System.in);
```



### ▣ import문 필요

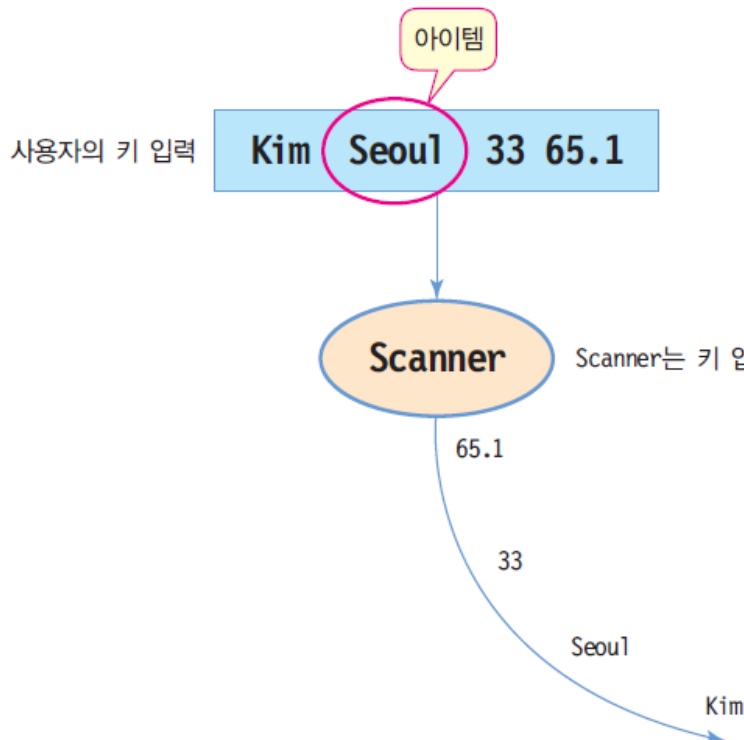
- 소스 맨 앞줄에 사용

```
import java.util.Scanner;
```

# Scanner를 이용한 키 입력

47

- Scanner에서 키 입력 받기
  - Scanner는 입력되는 키 값을 공백으로 구분되는 아이템 단위로 읽음
  - 공백 문자 : 'wt', 'wf', 'wr', ' ', 'wn'
- 개발자가 원하는 다양한 타입 값을 쉽게 읽을 수 있음



```
Scanner scanner = new Scanner(System.in);  
String name = scanner.next();    // "Kim"  
String addr = scanner.next();    // "Seoul"  
int age = scanner.nextInt();     // 33  
double weight = scanner.nextDouble(); // 65.1
```

Scanner는 키 입력을 공백 단위로 구분하여 읽는다.

# Scanner 주요 메소드

48

메소드	설명
<code>String next()</code>	다음 아이템을 문자열 타입으로 리턴한다.
<code>byte nextByte()</code>	다음 아이템을 <code>byte</code> 타입으로 리턴한다.
<code>short nextShort()</code>	다음 아이템을 <code>short</code> 타입으로 리턴한다.
<code>int nextInt()</code>	다음 아이템을 <code>int</code> 타입으로 리턴한다.
<code>long nextLong()</code>	다음 아이템을 <code>long</code> 타입으로 리턴한다.
<code>float nextFloat()</code>	다음 아이템을 <code>float</code> 타입으로 리턴한다.
<code>double nextDouble()</code>	다음 아이템을 <code>double</code> 타입으로 리턴한다.
<code>String nextLine()</code>	<del>한 라인 전체('\n' 포함)를 문자열 타입으로 리턴한다.</del>

한 라인 전체('Wn' 포함하지 않음)를 읽고  
'Wn'을 제외한 문자열 반환

## 예제 2-4 : Scanner를 이용한 키 입력 연습

49

Scanner를 이용하여 나이, 체중, 신장 데이터를 키보드에서 입력 받아 다시 출력하는 프로그램을 작성해보자.

```
import java.util.Scanner;

public class ScannerExam {
    public static void main (String args[]) {
        Scanner a = new Scanner(System.in);
        System.out.println("나이, 체중, 신장을 빈칸으로 분리하여 순서대로 입력하세요");
        System.out.println("당신의 나이는 " + a.nextInt() + "살입니다.");
        System.out.println("당신의 체중은 " + a.nextDouble() + "kg입니다.");
        System.out.println("당신의 신장은 " + a.nextDouble()+ "cm입니다.");
    }
}
```

나이, 체중, 신장을 빈칸으로 분리하여 순서대로 입력하세요

35 75 175

당신의 나이는 35살입니다.

당신의 체중은 75.0kg입니다.

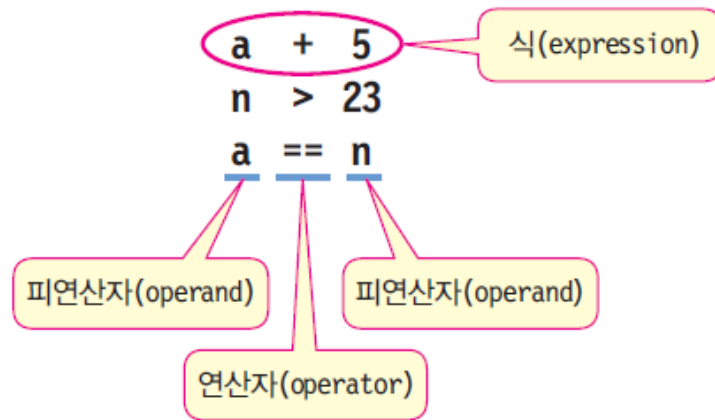
당신의 신장은 175.0cm입니다.



# 식과 연산자

50

- 연산 : 주어진 식을 계산하여 결과를 얻어내는 과정



연산의 종류	연산자
증감	<code>++ --</code>
산술	<code>+ - * / %</code>
시프트	<code>&gt;&gt; &lt;&lt; &gt;&gt;&gt;</code>
비교	<code>&gt; &lt; &gt;= &lt;= == !=</code>
비트	<code>&amp;   ^ ~</code>
논리	<code>&amp;&amp;    ! ^</code>
조건	<code>? :</code>
대입	<code>= *= /= += -= &amp;= ^=  = &lt;&lt;= &gt;&gt;= &gt;&gt;&gt;=</code>

# 연산자 우선 순위

51

<div> <div>높음</div> <div>↓</div> <div>낮음</div> </div>	++(postfix) --(postfix)
	+(양수 부호) -(음수 부호) ++(prefix) --(prefix) ~ !
	형 변환(type casting)
	* / %
	+(덧셈) -(뺄셈)
	<< >> >>>
	<> <= >= instanceof
	== !=
	& (비트 AND)
	^ (비트 XOR)
	(비트 OR)
	&& (논리 AND)
	(논리 OR)
	? : (조건)
	= += -= *= /= %= &= ^=  = <<= >>= >>>=

- 같은 우선순위의 연산자
  - ▣ 왼쪽에서 오른쪽으로 처리
  - ▣ 예외)오른쪽에서 왼쪽으로
    - 대입 연산자, --, ++, +,-(양수 음수 부호), !, 형 변환은 오른쪽에서 왼쪽으로 처리
- 괄호는 최우선순위
  - ▣ 괄호가 다시 괄호를 포함한 경우는 가장 안쪽의 괄호부터 먼저 처리

# 산술 연산자

52

## □ /와 % 연산자

- ▣ 정수 연산, /은 정수 몫. %는 정수 나머지
- ▣ %의 이용 사례 : 홀수 짝수 판별
  - `int r = x % 2; // r이 1이면 x는 홀수`

산술 연산자	의미	예	결과
+	더하기	25.5 + 3.6	29.1
-	빼기	3 - 5	-2
*	곱하기	2.5 * 4.0	10.0
/	나누기	5/2	2
%	나머지	5%2	1

## 예제 2-5 : 산술 연산 예제

53

정수를 입력 받고 입력 받은 정수의 초를 몇 시간, 몇 분, 몇 초인가를 구하는 프로그램을 작성하시오.

```
import java.util.Scanner;

public class ArithmeticOperator {
    public static void main (String[] args) {
        int time;
        int second;
        int minute;
        int hour;
        Scanner sc = new Scanner(System.in);
        System.out.print("정수를 입력하세요:"); // 시,분,초로 변환될 정수 입력

        time = sc.nextInt();
        second = time % 60; // 60으로 나눈 나머지는 초를 의미
        minute = (time / 60) % 60; // 60으로 나눈 몫을 다시 60으로 나눈 나머지는 분을 의미
        hour = (time / 60) / 60; // 60으로 나눈 몫을 다시 60으로 나눈 몫은 시간을 의미

        System.out.print(time + "초는 ");
        System.out.print(hour + "시간, ");
        System.out.print(minute + "분, ");
        System.out.println(second + "초입니다.");
    }
}
```

정수를 입력하세요:500  
500초는 0시간, 8분, 20초입니다.

# 비트 연산자

54

## □ 피 연산자의 각 비트들을 대상으로 하는 연산

비트 연산자	내용
$a \& b$	a와 b의 각 비트들의 AND 연산. 두 비트 모두 1일 때만 1이 되며 나머지는 0이 된다.
$a   b$	a와 b의 각 비트들의 OR 연산. 두 비트 모두 0일 때만 0이 되며 나머지는 1이 된다.
$a \wedge b$	a와 b의 각 비트들의 XOR 연산. 두 비트가 서로 다르면 1, 같으면 0이다.
$\sim a$	단항 연산자로서 a의 각 비트들에 NOT 연산. 1을 0으로, 0을 1로 변환한다.

# 비트 연산자의 사례

55

$$\begin{array}{r} 01101010 \\ \& 11001101 \\ \hline 01001000 \end{array}$$

모두 1이므로  
결과는 1

둘 중 하나라도 0이 되면  
결과는 0

$$\begin{array}{r} 01101010 \\ | 11001101 \\ \hline 11101111 \end{array}$$

모두 0이므로  
결과는 0

둘 중 하나라도 1이 되면  
결과는 1

$$\begin{array}{r} 01101010 \\ \wedge 11001101 \\ \hline 10100111 \end{array}$$

두 비트가  
모두 같으므로  
결과는 0

두 비트가  
서로 다르므로  
결과는 1

$$\begin{array}{r} \sim 01101010 \\ \hline 10010101 \end{array}$$

1은 0으로 변환

0은 1로 변환

# 시프트 연산자

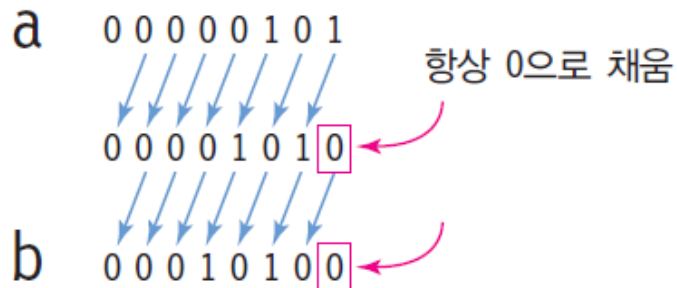
56

시프트 연산자	내용
$a \gg b$	a의 각 비트를 오른쪽으로 b번 시프트한다. 최상위 비트의 빈자리는 시프트 전의 최상위 비트로 다시 채운다. 산술적 오른쪽 시프트라고 한다.
$a \ggg b$	a의 각 비트를 오른쪽으로 b번 시프트한다. 그리고 최상위 비트의 빈자리는 0으로 채운다. 논리적 오른쪽 시프트라고 한다.
$a \ll b$	a의 각 비트를 왼쪽으로 b번 시프트한다. 그리고 최하위 비트의 빈자리는 0으로 채운다. 산술적 왼쪽 시프트라고 한다.

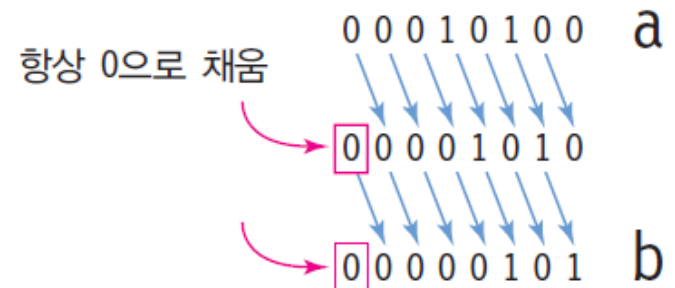
# 시프트 연산자의 사례

57

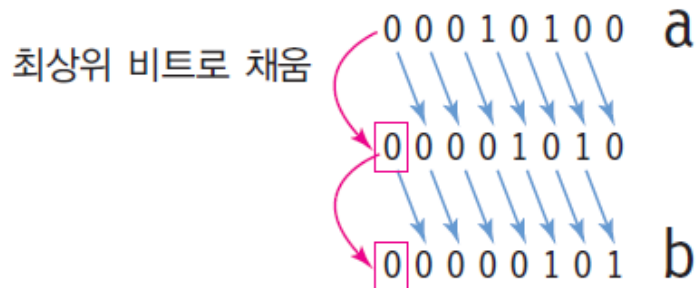
```
byte a = 5; // 5  
byte b = (byte)(a << 2); // 20
```



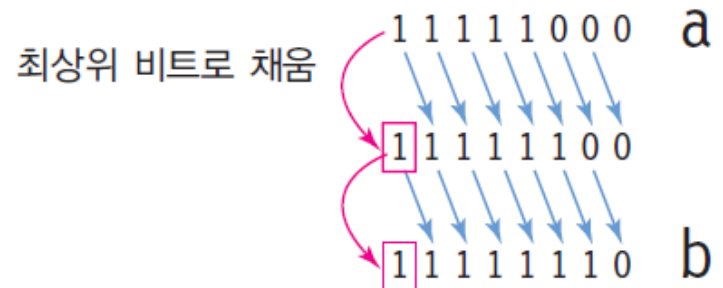
```
byte a = 20; // 20  
byte b = (byte)(a >> 2); // 5
```



```
byte a = 20; // 20  
byte b = (byte)(a >> 2); // 5
```



```
byte a = (byte)0xf8; // -8  
byte b = (byte)(a >> 2); // -2
```





# Tip: 산술적 시프트와 논리적 시프트

58

- 산술적 오른쪽 시프트
  - ▣ >>는 1비트 오른쪽으로 시프트 : 나누기 2의 결과
- 산술적 왼쪽 시프트
  - ▣ <<는 1비트 왼쪽 시프트 : 2로 곱하는 결과
  - ▣ 시프트 결과 음수(최상위 비트가 1)가 최상위 비트가 0인 양수가 되는 오버플로우 발생 가능 주의
- 논리적 오른쪽 시프트
  - ▣ >>>는 시프트 시 최상위 비트에 항상 0이 삽입
  - ▣ 나누기의 산술적 효과 없음
- byte, short, char 타입의 시프트 연산 시 주의 사항
  - ▣ int 타입으로 변환되어 연산, 원하지 않는 결과 발생 가능

# 예제 2-6 : 비트 연산자와 시프트 연산자 사용 예

59

다음 소스의 실행 결과는 무엇인가?

```
public class BitShiftOperator {
    public static void main (String[] args) {
        short a = (short)0x55ff;
        short b = 0x00ff;

        // 비트 연산
        System.out.printf("%xWn", a & b);
        System.out.printf("%xWn", a | b);
        System.out.printf("%xWn", a ^ b);
        System.out.printf("%xWn", ~a);

        byte c = 20; // 0x14
        byte d = -8; // 0xf8

        // 시프트 연산
        System.out.println(c << 2); // c를 2비트 왼쪽 시프트
        System.out.println(c >> 2); // c를 2비트 오른쪽 시프트. 0 삽입
        System.out.println(d >> 2); // d를 2비트 오른쪽 시프트. 1 삽입
        System.out.printf("%xWn", d >>> 2); // d를 2비트 오른쪽 시프트. 0 삽입
    }
}
```

printf("%xWn", ...)는 결과 값을 16진수 형식으로 출력

ff  
55ff  
5500  
ffffaa00  
80  
5  
-2  
3ffffffe

# 비교연산자

60

비교 연산자	내용	예제	결과
<code>a &lt; b</code>	a가 b보다 작으면 true 아니면 false	<code>3 &lt; 5</code>	true
<code>a &gt; b</code>	a가 b보다 크면 true 아니면 false	<code>3 &gt; 5</code>	false
<code>a &lt;= b</code>	a가 b보다 작거나 같으면 true 아니면 false	<code>1 &lt;= 0</code>	false
<code>a &gt;= b</code>	a가 b보다 크거나 같으면 true 아니면 false	<code>10 &gt;= 10</code>	true
<code>a == b</code>	a가 b와 같으면 true 아니면 false	<code>1 == 3</code>	false
<code>a != b</code>	a가 b와 같지 않으면 true 아니면 false	<code>1 != 3</code>	true

# 논리 연산자

a	!a	예제
true	false	!(3 < 5)는 false
false	true	!(3 > 5)는 true

a	b	a ^ b	예제
true	true	false	(3 < 5) ^ (1 == 1)은 false
true	false	true	(3 < 5) ^ (1 == 2)은 true
false	true	true	(3 > 5) ^ (1 == 1)은 true
false	false	false	(3 > 5) ^ (1 == 2)은 false

a	b	a    b	예제
true	true	true	(3 < 5)    (1 == 1)은 true
true	false	true	(3 < 5)    (1 == 2)은 true
false	true	true	(3 > 5)    (1 == 1)은 true
false	false	false	(3 > 5)    (1 == 2)은 false

a	b	a && b	예제
true	true	true	(3 < 5) && (1 == 1)은 true
true	false	false	(3 < 5) && (1 == 2)은 false
false	true	false	(3 > 5) && (1 == 1)은 false
false	false	false	(3 > 5) && (1 == 2)은 false

## 예제 2-7 : 비교 연산자와 논리 연산자 사용하기

62

다음 소스의 실행 결과는 무엇인가?

```
public class LogicalOperator {  
    public static void main (String[] args) {  
        System.out.println('a' > 'b');  
        System.out.println(3 >= 2);  
        System.out.println(-1 < 0);  
        System.out.println(3.45 <= 2);  
        System.out.println(3 == 2);  
        System.out.println(3 != 2);  
        System.out.println(!(3 != 2));  
        System.out.println((3 > 2) && (3 > 4));  
        System.out.println((3 != 2) || (-1 > 0));  
        System.out.println((3 != 2) ^ (-1 > 0));  
    }  
}
```

false  
true  
true  
false  
false  
false  
true  
false  
false  
true  
true

# 대입 연산자, 증감 연산자

63

대입 연산자	내용
<code>a = b</code>	b의 값을 a에 대입
<code>a += b</code>	<code>a = a + b</code> 와 동일
<code>a -= b</code>	<code>a = a - b</code> 와 동일
<code>a *= b</code>	<code>a = a * b</code> 와 동일
<code>a /= b</code>	<code>a = a / b</code> 와 동일
<code>a %= b</code>	<code>a = a % b</code> 와 동일

대입 연산자	내용
<code>a &amp;= b</code>	<code>a = a &amp; b</code> 와 동일
<code>a ^= b</code>	<code>a = a ^ b</code> 와 동일
<code>a  = b</code>	<code>a = a   b</code> 와 동일
<code>a &lt;&lt;= b</code>	<code>a = a &lt;&lt; b</code> 와 동일
<code>a &gt;&gt;= b</code>	<code>a = a &gt;&gt; b</code> 와 동일
<code>a &gt;&gt;&gt;= b</code>	<code>a = a &gt;&gt;&gt; b</code> 와 동일

증감 연산자	내용
<code>a++</code>	a를 먼저 사용한 후에 1 증가
<code>a--</code>	a를 먼저 사용한 후에 1 감소
<code>++a</code>	a를 먼저 1 증가한 후에 사용
<code>--a</code>	a를 먼저 1 감소한 후에 사용

# 증감 연산자

64

## □ 증감 연산의 순서

- ▣ 연산자가 피연산자 뒤에 붙는 경우

```
int a, b = 4;  
a = b++;  
// 결과 a=4, b=5
```

- ▣ 연산자가 피연산자 앞에 붙는 경우

```
int a, b = 4;  
a = ++b;  
// 결과 a=5, b=5
```

## 예제 2-8 : 대입 연산자와 증감 연산자 사용하기

65

다음 소스의 실행 결과는 무엇인가?

```
public class UnaryOperator {  
    public static void main(String[] args) {  
        int opr = 0;  
        opr += 3;           // opr = opr + 3  
        System.out.println(opr++); // opr 출력 후 증가  
        System.out.println(opr);  
        System.out.println(++opr); // opr 증가 후 출력  
        System.out.println(opr);  
        System.out.println(opr--); // opr 출력 후 감소  
        System.out.println(opr);  
        System.out.println(--opr); // opr 감소 후 출력  
        System.out.println(opr);  
    }  
}
```

3  
4  
5  
5  
5  
4  
3  
3



# 조건 연산자 ?:


66

## □ opr1?opr2:opr3

- 세 개의 피연산자로 구성된 삼항(ternary) 연산자
  - opr1이 true이면, 연산식의 결과는 opr2, false이면 opr3
- if-else를 간결하게 표현할 수 있음

```
int x = 5;  
int y = 3;
```

```
int s;  
if(x>y)  
    s = 1;  
else  
    s = -1;
```



```
int s = (x>y)?1:-1;
```

## 예제 2-9 : 조건 연산자 사용하기

67

다음 소스의 실행 결과는 무엇인가?

```
public class TernaryOperator {  
    public static void main (String[] args) {  
        int a = 3, b = 5;  
  
        System.out.println("두 수의 차는 " + ((a>b)?(a-b):(b-a)));  
    }  
}
```

두 수의 차는 2

# 조건문 - if문

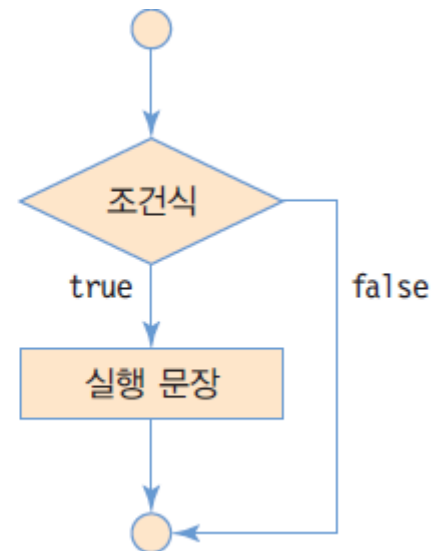
68

## □ 단순 if 문

- ▣ if 다음의 괄호 안에는 조건식(논리형 변수나 논리 연산)
- ▣ 조건식의 값
  - true인 경우, if문을 벗어나 다음 문장이 실행된다.
  - false의 경우에는 if 다음의 문장이 실행되지 않고 if 문을 빠져 나온다.
- ▣ 실행문장이 단일 문장인 경우 둘러싸는 { } 생략 가능

```
if(조건식) {  
    ...실행 문장...  
}
```

if 키워드



## 예제 2-10 : if문 사용하기

69

시험 점수가 80점이 이상이면 합격 판별을 하는 프로그램을 작성하시오.

```
import java.util.Scanner;

public class SuccessOrFail {
    public static void main (String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.print("점수를 입력하시오: ");
        int score = in.nextInt();
        if (score >= 80)
            System.out.println("축하합니다! 합격입니다.");
    }
}
```

점수를 입력하시오: 95  
축하합니다! 합격입니다.

# 조건문 – if-else

70

## □ if-else 문

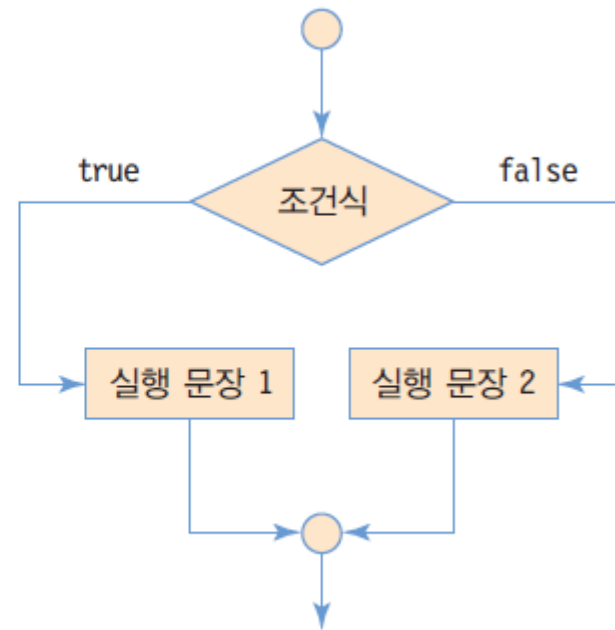
- 조건식이 true면 실행문장1 실행 후 if-else문을 벗어남
- false인 경우에 실행문장2 실행후, if-else문을 벗어남

2/3

```
if(조건식) {  
    ...실행 문장 1...  
}  
else {  
    ...실행 문장 2...  
}
```

if 키워드

else 키워드



# 예제 2-11 : if-else 사용하기

71

입력된 수가 3의 배수인지 판별하는 프로그램을 작성하시오.

```
import java.util.Scanner;

public class MultipleOfThree {
    public static void main (String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.print("수를 입력하시오: ");
        int number = in.nextInt();

        if (number % 3 == 0)
            System.out.println("3의 배수입니다.");
        else
            System.out.println("3의 배수가 아닙니다.");
    }
}
```

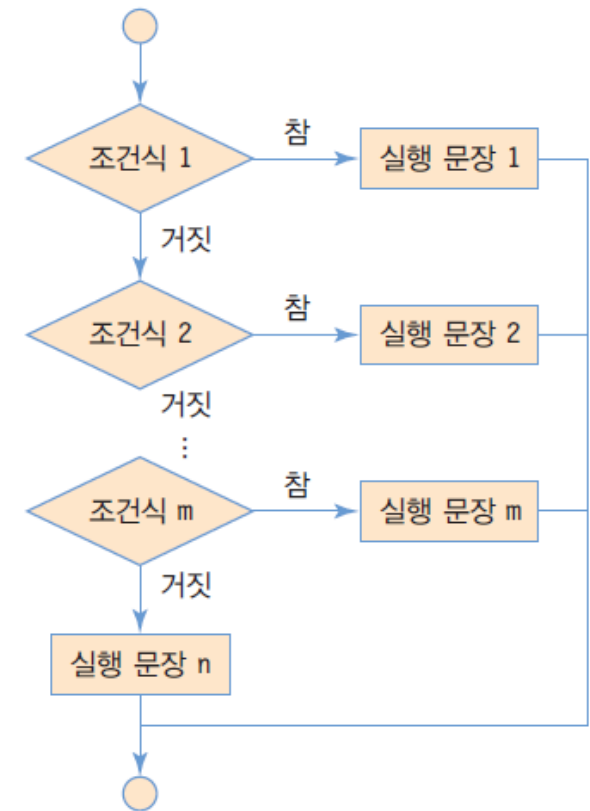
수를 입력하시오: 129  
3의 배수입니다.

# 조건문 - 다중 if

72

## □ 다중 if문

- ▣ 조건문이 너무 많은 경우, switch 문 사용 권장



## 예제 2-12 : 학점 매기기

73

if-else문을 이용하여 키보드 입력된 성적에 대해 학점을 부여하는 프로그램을 작성해보자.

```
import java.util.Scanner;

public class Grading {
    public static void main (String[] args) {
        char grade;
        Scanner a = new Scanner(System.in);
        while (a.hasNext()) {
            int score = a.nextInt();
            if(score >= 90) // score가 90 이상인 경우
                grade = 'A';
            else if(score >= 80) // score가 80 이상이면서 90.0 미만인 경우
                grade = 'B';
            else if(score >= 70) // score가 70 이상이면서 80 미만인 경우
                grade = 'C';
            else if(score >= 60) // score가 60 이상이면서 70 미만인 경우
                grade = 'D';
            else // score가 60 미만인 경우
                grade = 'F';
            System.out.println("학점은 "+grade+"입니다");
        }
    }
}
```

키가 입력될 때까지 기다리며, 입력된 키가 있는 경우 true 리턴. 라인의 첫 문자로 ctrl-z 키가 입력되면 false 리턴

80  
학점은 B입니다  
90  
학점은 A입니다  
76  
학점은 C입니다

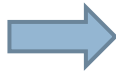


## Tip: if문과 조건 연산자 ?:

74

- 조건 연산자 ?:는 if-else로 바꿀 수 있음

```
i = a > b ? a - b : b - a;
```

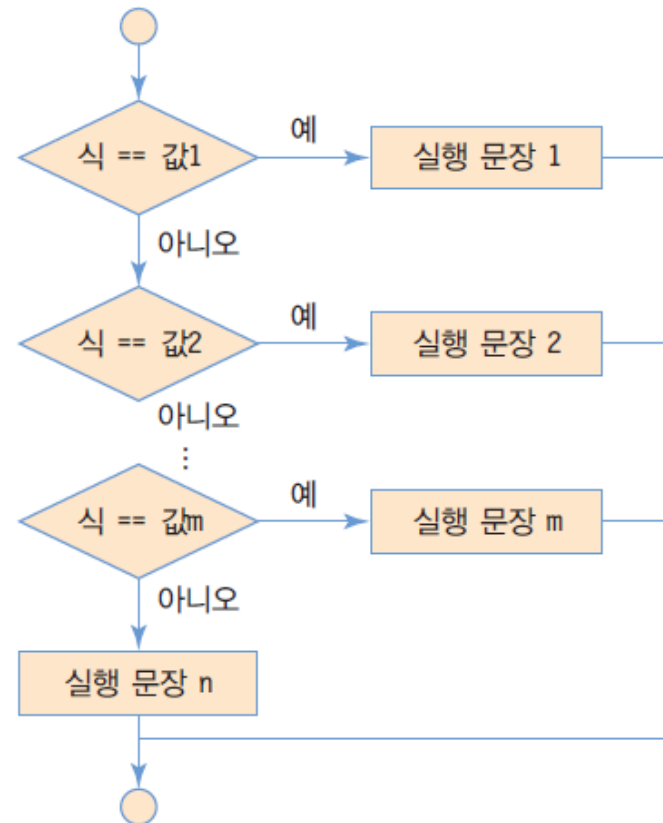
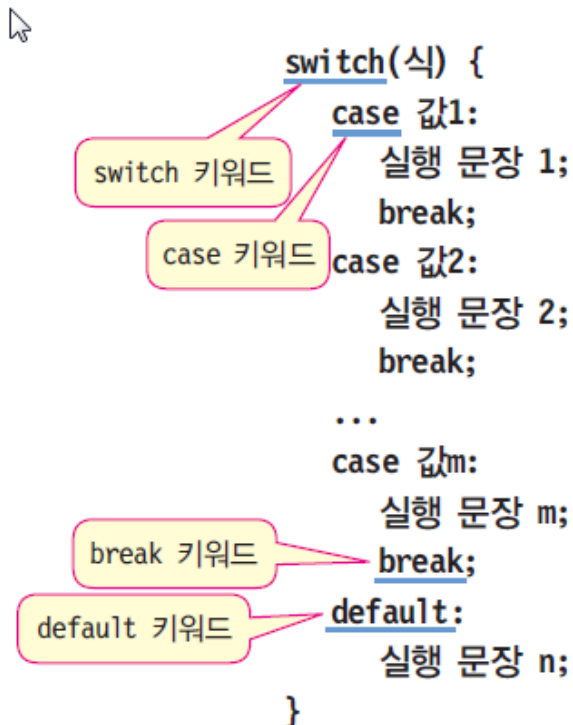


```
if (a > b)
    i = a - b;
else
    i = b - a;
```

# switch문

75

- switch문은 식과 case 문의 값과 비교
  - ▣ case의 비교 값과 일치하면 해당 case의 실행문장 수행
    - break를 만나면 switch문을 벗어남
  - ▣ case의 비교 값과 일치하는 것이 없으면 default 문 실행
- default문은 생략 가능



# switch문에서 벗어나기

76

- switch문 내의 break문
  - ▣ break문 만나면 switch문 벗어남
  - ▣ case 문에 break문이 없다면, 다음 case문으로 실행 계속
    - 언젠가 break를 만날 때까지 계속 내려 가면서 실행

```
char grade='A';
switch (grade) {
    case 'A':
        System.out.println("90 ~ 100점입니다.");
        break;
    case 'B':
        System.out.println("80 ~ 89점입니다.");
        break;
    case 'C':
        System.out.println("70 ~ 79점입니다.");
        break;
}
```

90 ~ 100점입니다.  
80 ~ 89점입니다.

## 예제 2-13 : switch문의 break 사용하기

77

학점이 A, B 인 학생에게는 "참 잘하였습니다.", 학점이 C, D인 학생에게는 "좀 더 노력하세요.", 학점이 F인 학생에게는 "다음 학기에 다시 수강하세요."를 출력하는 프로그램을 switch문의 break를 잘 활용하여 작성하여라.

```
public class GradeSwitch {  
    public static void main(String[] args) {  
        char grade='C';  
        switch (grade) {  
            case 'A':  
            case 'B':  
                System.out.println("참 잘하였습니다.");  
                break;  
            case 'C':  
            case 'D':  
                System.out.println("좀 더 노력하세요.");  
                break;  
            case 'F':  
                System.out.println("다음 학기에 다시 수강하세요.");  
                break;  
            default:  
                System.out.println("잘못된 학점입니다.");  
        }  
    }  
}
```

좀 더 노력하세요.

# case 문의 값

78

## □ case 문의 값의 특징

- switch 문은 식의 결과 값을 case 문과 비교
- 사용 가능한 case문의 값
  - 문자, 정수, 문자열 리터럴(JDK 1.7부터)만 허용
  - 실수 리터럴은 허용되지 않음

```
int c = 25;
switch(c%2) {
    case 1 :           // 정수 리터럴
        ...;
        break;
    case 2 :           // 정수 리터럴
        ...;
        break;
}

String s = "예";
switch(s) {
    case "예" :        // 문자열 리터럴. JDK1.7부터 적용
        ...;
        break;
    case "아니요" :    // 문자열 리터럴. JDK1.7부터 적용
        ...;
        break;
}
```

정상적인  
case 문

```
char grade='C';
switch (grade) {
    case 'A' :         // 문자 리터럴
        ...;
        break;
    case 'B' :         // 문자 리터럴
        ...;
        break;
}
```

정상적인  
case 문

```
switch(a) {
    case a :           // 오류. 변수 사용 안됨
    case a > 3 :        // 오류. 수식 안됨
    case a == 1 :       // 오류. 수식 안됨
}
```

잘못된  
case 문

## 예제 2-14 : 성적 분류

79

앞의 다중 if문을 이용한  
성적 분류 프로그램을  
switch문으로 바꾸시오.

100  
학점은 A입니다  
55  
학점은 F입니다  
76  
학점은 C입니다

```
import java.util.Scanner;

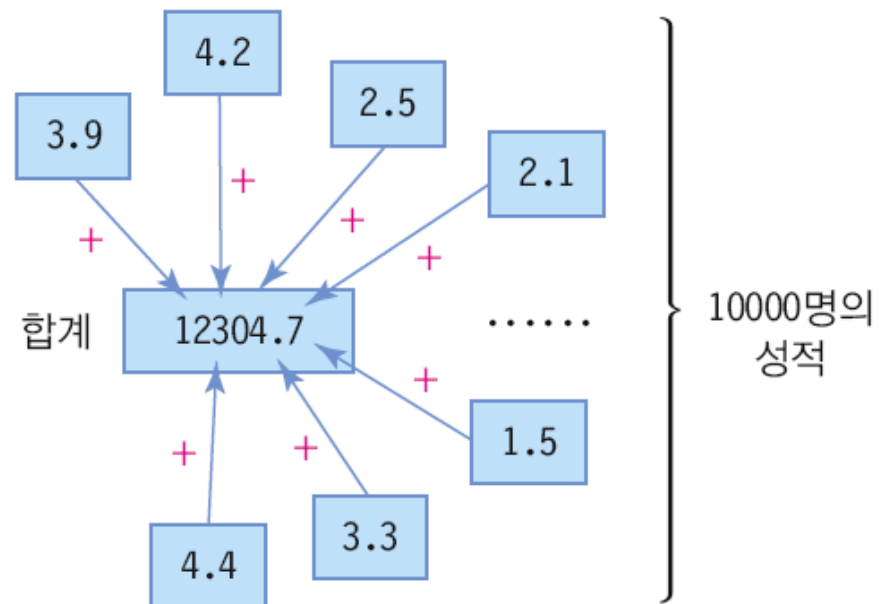
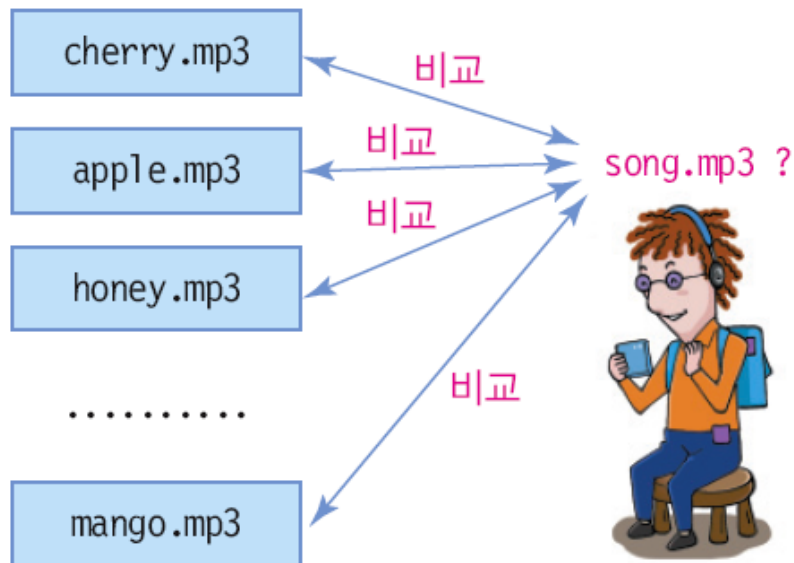
public class Grading2 {
    public static void main (String[] args) {
        char grade;
        Scanner a = new Scanner(System.in);
        while (a.hasNext()) {
            int score = a.nextInt();
            switch (score/10) {
                case 10:
                case 9:
                    grade = 'A';
                    break;
                case 8:
                    grade = 'B';
                    break;
                case 7:
                    grade = 'C';
                    break;
                case 6:
                    grade = 'D';
                    break;
                default:
                    grade = 'F';
            }
            System.out.println("학점은 "+grade+"입니다");
        }
    }
}
```

# 반복문의 특징

80

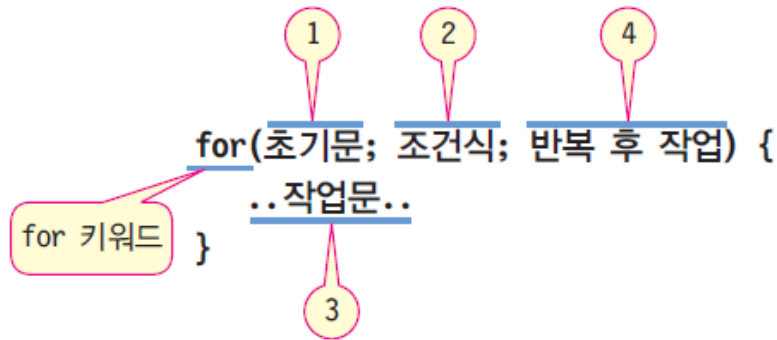
## □ 자바 반복문의 종류

- ▣ for 문
- ▣ while 문
- ▣ do while 문



# for 문의 구성

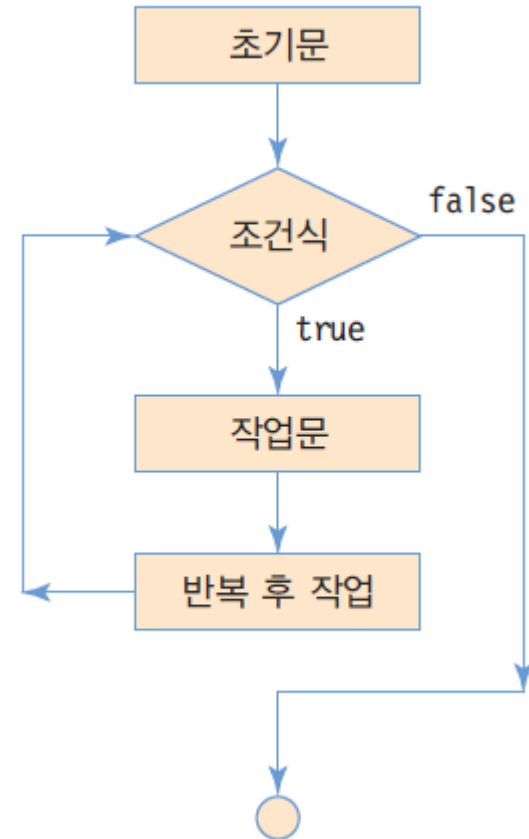
81



- ①
- for 문이 실행한 후 오직 한번만 실행되는 초기화 작업
  - 콤마(',')로 구분하여 여러 문장 나열 가능
  - 초기화할 일 없으면 비어둘 수 있음

- ②
- 논리형 변수나 논리 연산만 가능
  - 반복 조건이 true이면 반복 계속, false이면 반복 종료
  - 반복 조건이 true 상수인 경우, 무한 반복
  - 반복 조건이 비어 있으면 true로 간주

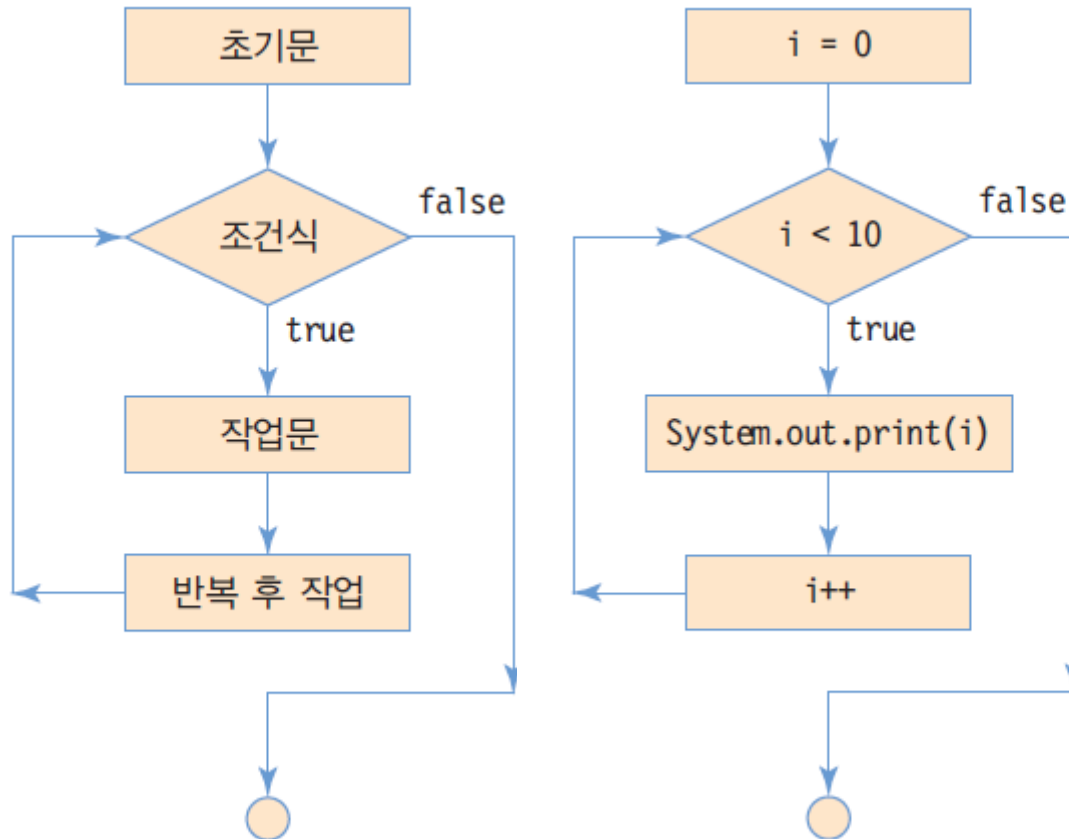
- ④
- 반복 작업 문장들의 실행 후 처리 작업
  - 콤마(',')로 구분하여 여러 문장 나열 가능





# for문의 실행 과정을 나타내는 순서도

82



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

# for문의 예시

83

- 0에서 9까지 정수 출력

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

```
int i;  
for(i = 0; i < 10; i++)  
    System.out.print(i);
```

- 반복문에 변수 선언 가능

```
for(int i = 0; i < 10; i++) // 변수 i는 for문을 벗어나서 사용할 수 없음  
    System.out.print(i);
```

- 0에서 100까지의 합 구하기

```
int sum = 0;  
for(int i = 0; i <= 100; i++)  
    sum += i;
```

```
int sum;  
for(int i = 0, sum=0; i <= 100; i++)  
    sum += i;
```

```
int sum = 0;  
for(int i = 100; i >= 0; i--)  
    sum += i;
```

# for문의 특이한 형태

84

```
for(초기작업; true; 반복후작업) { // 반복 조건이 true이면 무한 반복  
.....  
}
```

```
for(초기작업; ; 반복후작업) { // 반복조건이 비어 있으면 true로 간주, 무한 반복  
.....  
}
```

```
// 초기 작업과 반복후작업은 ';'로 분리하여 여러 문장 나열 가능  
for(i=0; i<10; i++, System.out.println(i)) {  
.....  
}
```

```
// for문 내에 변수 선언  
for(int i=0; i<10; i++) { // 변수 i는 for문 내에서만 사용 가능  
.....  
}
```

# 예제 3-1 : 1부터 10까지 숫자의 합을 출력

85

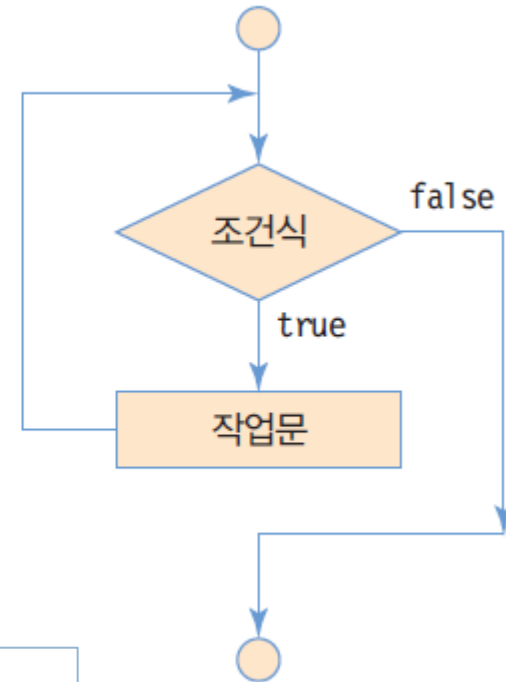
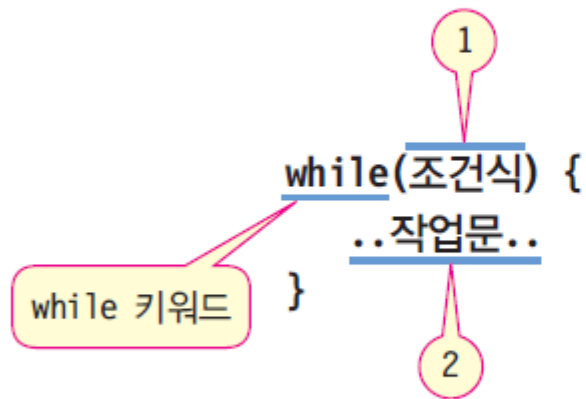
for문을 이용하여 1부터 10까지 덧셈을 표시하고 합을 구하시오.

```
public class ForSample {  
    public static void main (String[] args) {  
        int i, j;  
  
        for (j=0,i=1; i <= 10; i++) {  
            j += i;  
            System.out.print(i);  
            if(i==10) {  
                System.out.print("=");  
                System.out.print(j);  
            }  
            else  
                System.out.print("+");  
        }  
    }  
}
```

1+2+3+4+5+6+7+8+9+10=55

# while 문의 구성

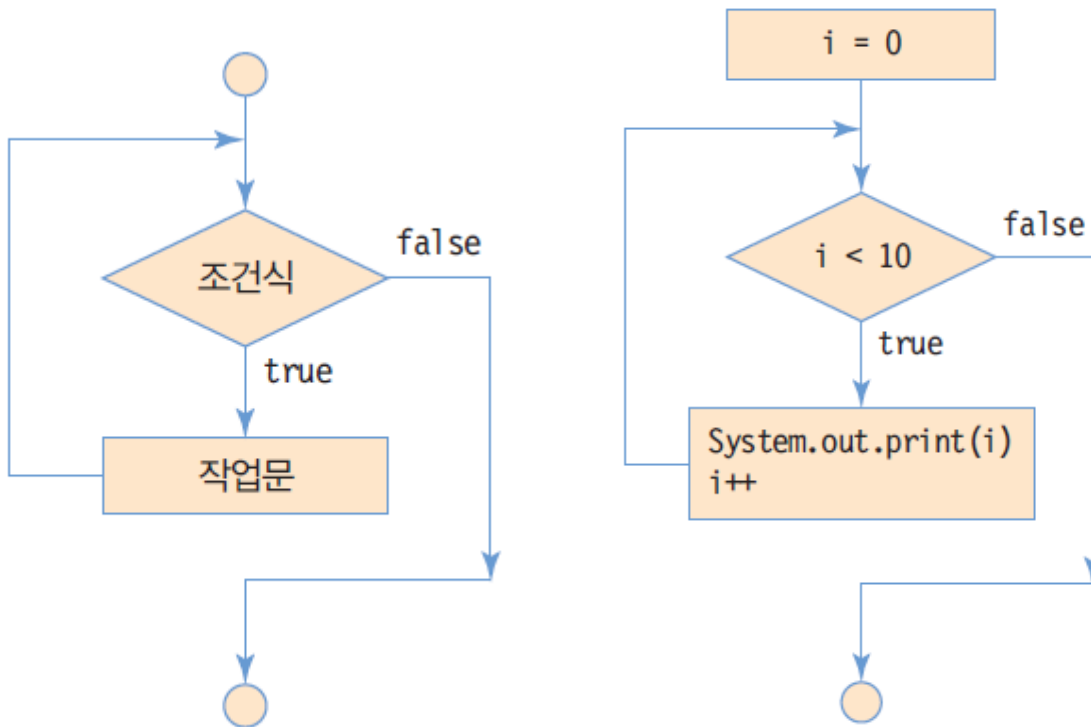
86



- 반복 조건이 true이면 반복, false이면 반복 종료
- 반복 조건이 없으면 컴파일 오류
- 처음부터 반복조건을 통과한 후 작업문 수행

# while문의 실행 과정을 나타내는 순서도

87



```
i = 0;  
while(i<10) {  
    System.out.print(i);  
    i++;  
}
```

## 예제 3-2 : 입력된 수의 평균 구하기

88

while문을 이용하여 키보드에서 숫자를 입력 받아 입력 받은 모든 수의 평균을 출력하는 프로그램을 작성해보자.

0이 입력되면 입력이 종료되고 평균을 구하여 출력한다.

```
import java.util.Scanner;
public class WhileSample {
    public static void main (String[] args) {
        Scanner rd = new Scanner(System.in);
        int n = 0;
        double sum = 0;
        int i=0;
        while ((i = rd.nextInt()) != 0) {
            sum += i;
            n++;
        }
        System.out.println("입력된 수의 개수는 " + n + "개이며 평균은 " + sum / n + "입니다.");
    }
}
```

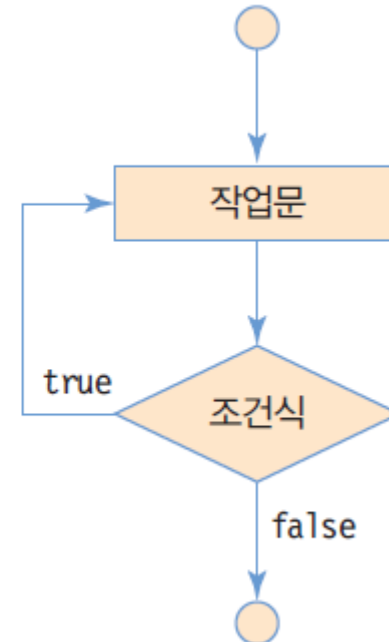
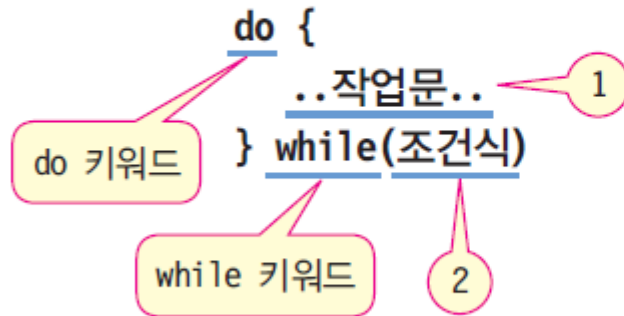
10  
20  
30  
40  
0

마지막 입력을 뜻함

입력된 수의 개수는 4개이며 평균은 25.0입니다.

# do-while 문의 구성

89

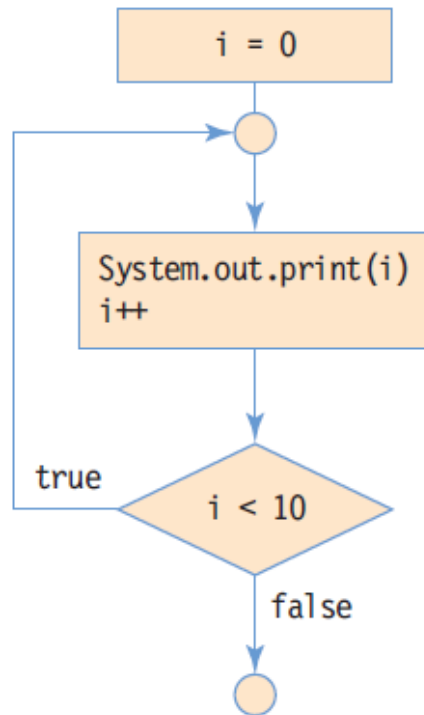
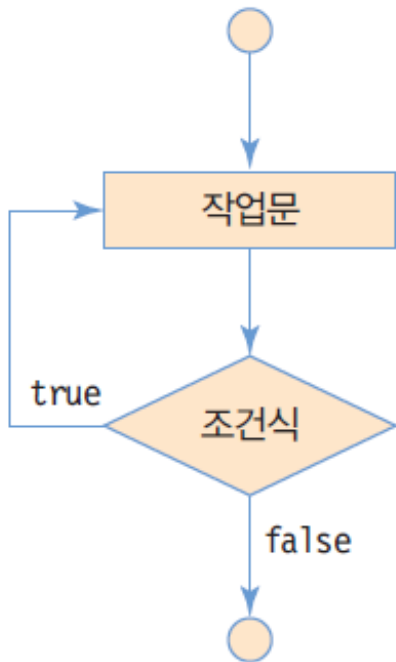


- ❶
  - 무조건 최소 한번은 실행
- ❷
  - 반복 조건이 true이면 반복, false이면 반복 종료
  - 반복 조건이 없으며 컴파일 오류



# do-while문의 실행 과정을 나타내는 순서도

90



```
i = 0;  
do {  
    System.out.print(i);  
    i++;  
} while(i<10);
```

## 예제 3-3 : a-z까지 출력

91

do-while문을 이용하여 'a'부터 'z'까지 출력하는 프로그램을 작성하시오.

```
public class DoWhileSample {  
    public static void main (String[] args) {  
        char a = 'a';  
  
        do {  
            System.out.print(a);  
            a = (char) (a + 1);  
        } while (a <= 'z');  
    }  
}
```

abcdefghijklmnopqrstuvwxyz

# 중첩 반복

92

- 중첩 반복
  - ▣ 반복문이 다른 반복문을 내포하는 구조
  - ▣ 이론적으로는 몇 번이고 중첩 반복 가능
  - ▣ 너무 많은 중첩 반복은 프로그램 구조를 복잡하게 하므로 2중 또는 3중 반복이 적당

```
for(i=0; i<100; i++) { // 100 개의 학교 성적을 모두 더한다.
```

```
.....
```

```
    for(j=0; j<10000; j++) { // 10000 명의 학생 성적을 모두 더한다.
```

```
        .....
```

```
        .....
```

```
    }
```

```
.....
```

```
}
```

10000명의 학생이 있는 100개 대학의 모든 학생 성적의 합을 구할 때,  
for 문을 이용한 이중 중첩 구조

## 예제 3-4 : 구구단

93

2중 중첩된 for문을 사용하여 구구단을 출력하는 프로그램을 작성하시오.  
한 줄에 한 단씩 출력한다.

```
public class NestedLoop {  
    public static void main (String[] args) {  
        int i, j;  
  
        for (i = 1; i < 10; i++, System.out.println()) {  
            for (j = 1; j < 10; j++, System.out.print("\t")) {  
                System.out.print(i + "*" + j + "=" + i*j);  
            }  
        }  
    }  
}
```

1*1=1	1*2=2	1*3=3	1*4=4	1*5=5	1*6=6	1*7=7	1*8=8	1*9=9
2*1=2	2*2=4	2*3=6	2*4=8	2*5=10	2*6=12	2*7=14	2*8=16	2*9=18
3*1=3	3*2=6	3*3=9	3*4=12	3*5=15	3*6=18	3*7=21	3*8=24	3*9=27
4*1=4	4*2=8	4*3=12	4*4=16	4*5=20	4*6=24	4*7=28	4*8=32	4*9=36
5*1=5	5*2=10	5*3=15	5*4=20	5*5=25	5*6=30	5*7=35	5*8=40	5*9=45
6*1=6	6*2=12	6*3=18	6*4=24	6*5=30	6*6=36	6*7=42	6*8=48	6*9=54
7*1=7	7*2=14	7*3=21	7*4=28	7*5=35	7*6=42	7*7=49	7*8=56	7*9=63
8*1=8	8*2=16	8*3=24	8*4=32	8*5=40	8*6=48	8*7=56	8*8=64	8*9=72
9*1=9	9*2=18	9*3=27	9*4=36	9*5=45	9*6=54	9*7=63	9*8=72	9*9=81

# continue문

94

## □ continue 문

- 반복문을 빠져 나가지 않으면서
- 반복문 실행 도중 다음 반복을 진행

```
for (초기문; 조건식; 반복후작업) {  
    .....  
    continue;  
    .....  
}
```

분기

```
while (조건식) {  
    .....  
    continue;  
    .....  
}
```

조건식으로  
분기


```
do {  
    .....  
    continue;  
    .....  
} while (조건식);
```

조건식으로  
분기

## 예제 3-5 : 1부터 100까지 짝수의 합

95

for와 continue문을 사용하여 1부터 100까지 짝수의 합을 구해보자.

```
public class ContinueExample {  
    public static void main (String[] args) {  
        int sum = 0;  
        for (int i = 1; i <= 100; i++) {  
            if (i%2 == 1)  
                continue;   
            else  
                sum += i;  
        }  
        System.out.println("1부터 100까지 짝수의 합은 " + sum);  
    }  
}
```

1부터 100까지 짝수의 합은 2550

# break문

96

## □ break 문

- ▣ 반복문 하나를 완전히 빠져 나갈 때 사용
- ▣ break문은 하나의 반복문만 벗어남
  - 중첩 반복의 경우 안쪽 반복문의 break 문이 실행되면 안쪽 반복문만 벗어남

## 예제 3-6 : 입력된 숫자 개수 세기

97

while문과 break문을 사용하여 -1이 입력될 때까지 입력된 숫자의 개수를 출력하시오.

```
import java.util.Scanner;
public class BreakExample {
    public static void main (String[] args) {
        Scanner in = new Scanner(System.in);
        int num = 0;

        while (true) {
            if (in.nextInt() == -1)
                break;
            num++;
        }
        System.out.println("입력된 숫자 개수는 " + num);
    }
}
```

10  
8  
9  
5  
-1  
입력된 숫자 개수는 4

마지막 입력을 뜻함



# Tip: 라벨로 분기

98

## □ 라벨로 분기하는 경우

### ▣ **continue** 라벨;

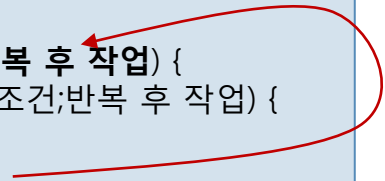
- 특정 라벨의 다음 반복으로 분기
- 중첩 반복(nested loop)에서 바깥의 반복문으로 빠져 나갈 때 주로 사용

### ▣ **break** 라벨;

- 라벨이 붙은 반복문을 벗어남.
- 중첩 반복문을 한 번에 벗어날 때 주로 사용

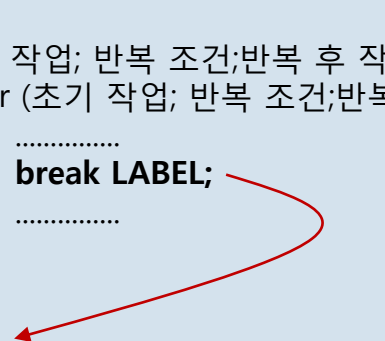
**LABEL:**

```
for (초기 작업; 반복 조건;반복 후 작업) {  
    for (초기 작업; 반복 조건;반복 후 작업) {  
        .....  
        continue LABEL; .....  
    }  
}
```



**LABEL:**

```
for (초기 작업; 반복 조건;반복 후 작업) {  
    for (초기 작업; 반복 조건;반복 후 작업) {  
        .....  
        break LABEL; .....  
    }  
} .....  
.....
```



# 배열이란?

99

## □ 배열(array)

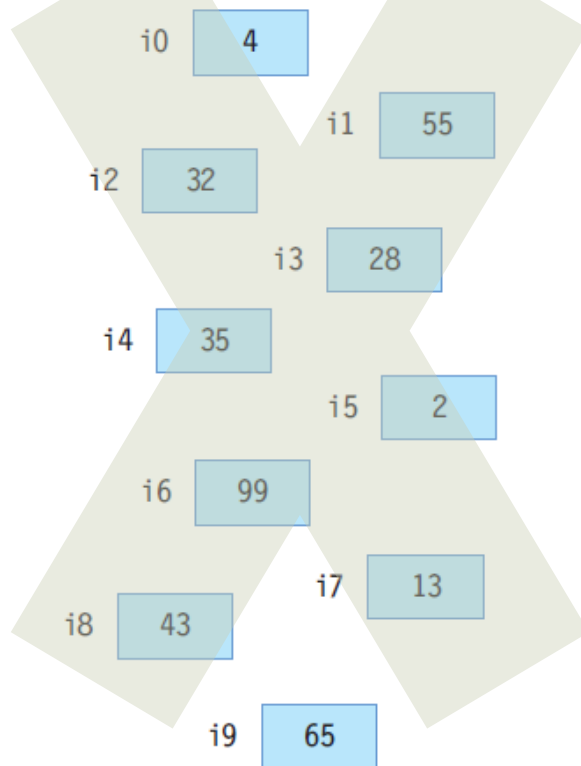
- ▣ 인덱스와 인덱스에 대응하는 데이터들로 이루어진 자료 구조
  - 배열을 이용하면 한 번에 많은 메모리 공간 선언 가능
- ▣ 배열은 같은 타입의 데이터들이 순차적으로 저장되는 공간
  - 원소 데이터들이 순차적으로 저장됨
  - 인덱스를 이용하여 원소 데이터 접근
  - 반복문을 이용하여 처리하기에 적합한 자료 구조
- ▣ 배열 인덱스
  - 0부터 시작
  - 인덱스는 배열의 시작 위치에서부터 데이터가 있는 상대 위치

# 자바 배열의 필요성과 모양

100

(1) 10개의 정수형 변수를 선언하는 경우

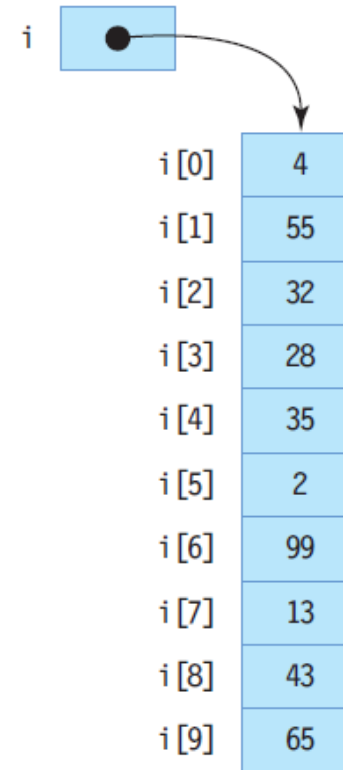
```
int i0, i1, i2, i3, i4, i5, i6, i7, i8, i9;
```



```
sum = i0+i1+i2+i3+i4+i5+i6+i7+i8+i9;
```

(2) 10개의 정수로 구성된 배열을 선언하는 경우

```
int i[] = new int[10];
```



```
for(sum=0, n=0; n<10; n++)  
    sum += i[n];
```

# 일차원 배열의 선언

101

- 배열 선언과 배열 생성의 두 단계 필요
- 배열 선언

```
int    intArray[];  
char   charArray[];
```

또는

```
int[]  intArray;  
char[] charArray;
```

- 배열 생성

```
intArray = new int[10];  
charArray = new char[20];
```

또는

```
int intArray[] = new int[10];  
char charArray[] = new char[20];
```

- 선언과 초기화
  - 배열 생성과 값 초기화

```
int intArray[] = {0,1,2,3,4,5,6,7,8,9}; // 총 10개의 정수 배열 생성 및 값 초기화
```

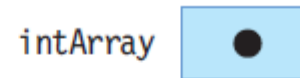
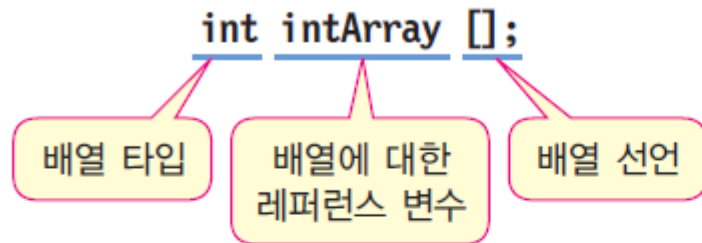
- 잘못된 배열 선언

```
int intArray[10]; // 컴파일 오류. 배열의 크기를 지정할 수 없음
```

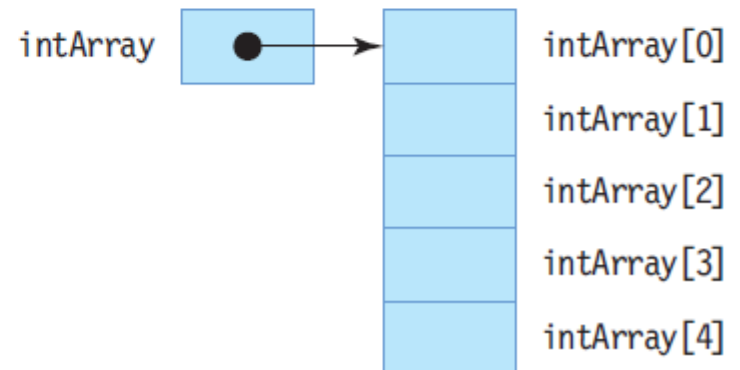
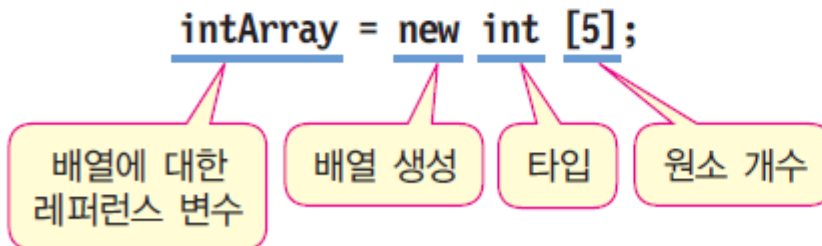
# 배열 선언과 생성

102

(1) 배열에 대한 레퍼런스 변수 `intArray` 선언



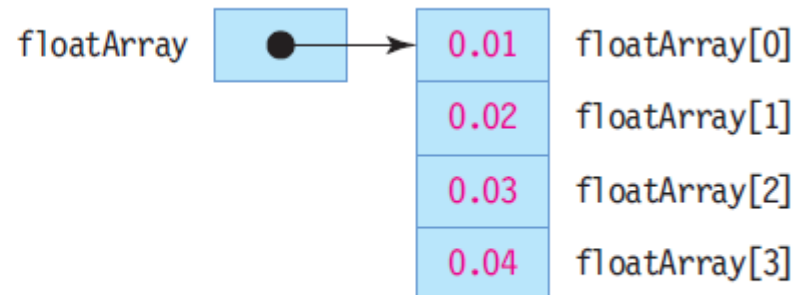
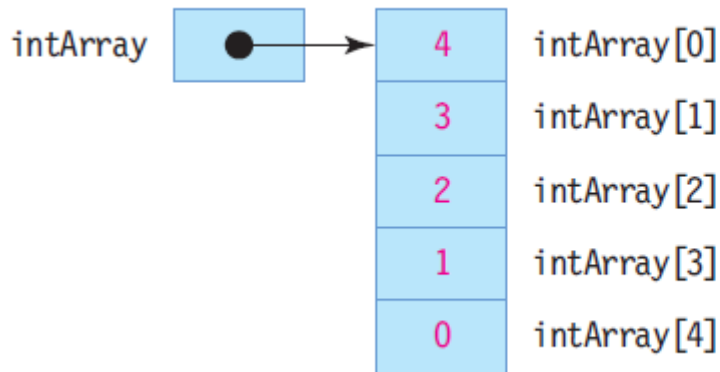
(2) 배열 생성



# 배열을 초기화하면서 생성한 결과

103

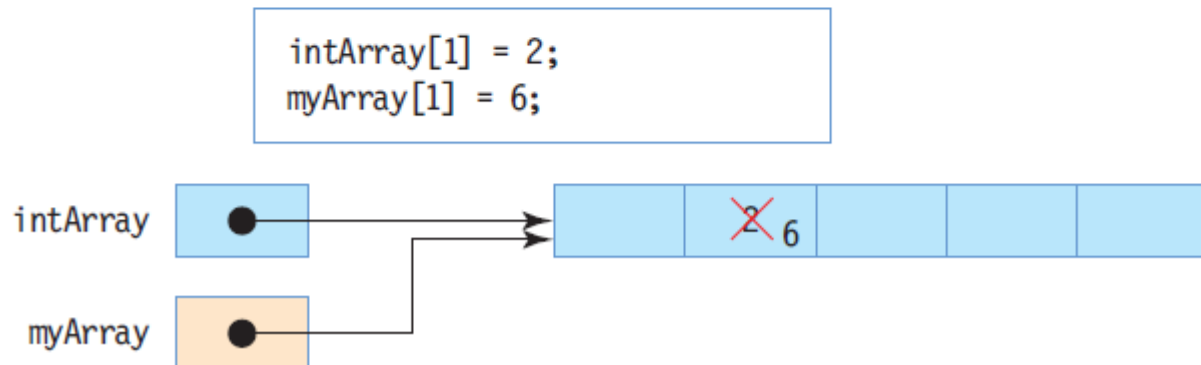
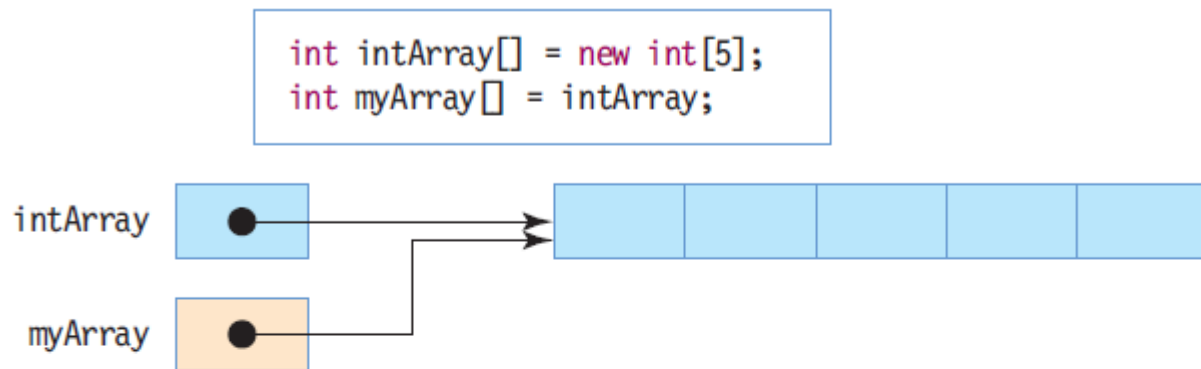
```
int intArray[] = {4, 3, 2, 1, 0};  
float floatArray[] = {0.01, 0.02, 0.03, 0.04};
```



# 배열 참조

104

- ▣ 생성된 하나의 배열을 다수의 레퍼런스가 참조 가능



# 배열 접근 방법

105

## □ 배열 원소 접근

### ▣ 반드시 배열 생성 후 접근

```
int intArray [];  
intArray[4] = 8; // 오류, intArray 배열의 메모리가 할당되지 않았음
```

### ▣ 배열 변수명과 [] 사이에 원소의 인덱스를 적어 접근

- 배열의 인덱스는 0부터 시작
- 배열의 마지막 항목의 인덱스는 (배열 크기 - 1)

```
int[] intArray;  
intArray = new int[10];  
  
intArray[3]=6; // 배열에 값을 저장  
int n = intArray[3]; // 배열로부터 값을 읽음
```



# 예제 3-7 : 배열에 입력 받은 수 중 제일 큰 수 찾기

106

양수 5개를 입력 받아 배열에 저장하고, 제일 큰 수를 화면에 출력하는 프로그램을 작성하시오.

```
import java.util.Scanner;
public class ArrayAccess {
    public static void main (String[] args) {
        Scanner in = new Scanner(System.in);
        int intArray[] = new int[5];
        int max = 0;

        for (int i = 0; i < 5; i++) {
            intArray[i] = in.nextInt();
            if (intArray[i] > max)
                max = intArray[i];
        }
        System.out.print("입력된 수에서 가장 큰 수는 " + max + "입니다.");
    }
}
```

```
1
39
78
100
99
```

입력된 수에서 가장 큰 수는 100입니다.

# 배열의 크기와 인덱스

107

## □ 배열 인덱스

- ▣ 인덱스는 0부터 시작하며 마지막 인덱스는 (배열 크기 -1)
- ▣ 인덱스는 정수 타입만 가능

```
int intArray [] = new int[5]; // 인덱스는 0~4까지 가능  
int n = intArray[-2]; // 실행 오류. -2는 인덱스로 적합하지 않음  
int m = intArray[5]; // 실행 오류. 5는 인덱스의 범위(0~4)를 넘었음
```

## □ 배열의 크기

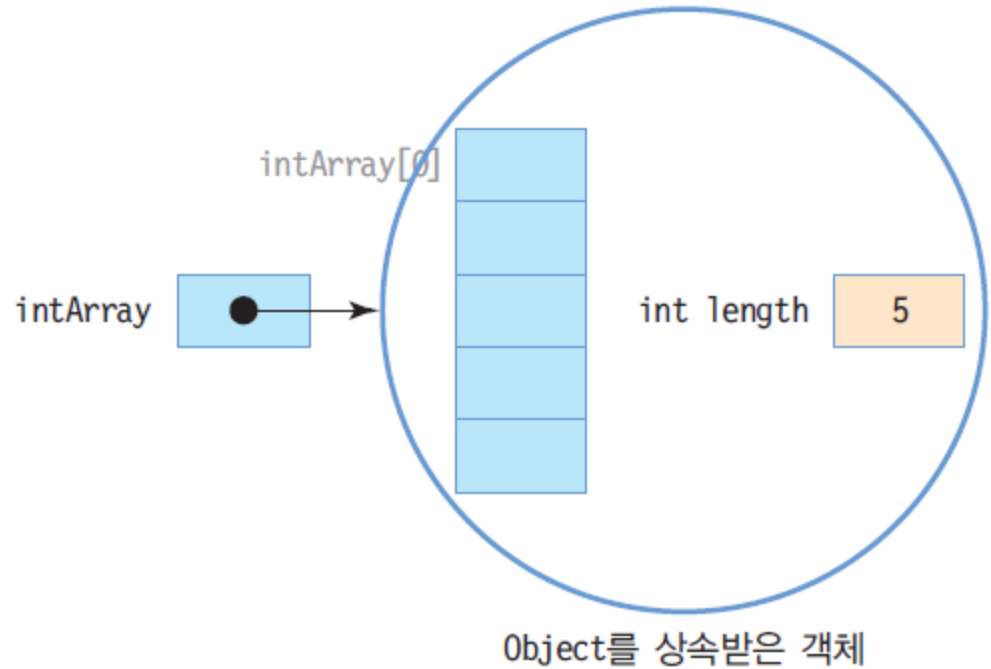
- ▣ 배열의 크기는 배열 레퍼런스 변수를 선언할 때 결정되지 않음
  - 배열의 크기는 배열 생성 시에 결정되며, 나중에 바꿀 수 없음
- ▣ 배열의 크기는 배열의 length 필드에 저장

```
int size = intArray.length;
```

# 배열은 객체로 관리

108

```
int intArray[];  
intArray = new int[5];  
  
int size = intArray.length;  
// size는 5
```



## 예제 3-8 : 배열 원소의 평균 구하기

109

배열의 `length` 필드를 이용하여 배열 크기만큼 키보드에서 정수를 입력 받고 평균을 구하는 프로그램을 작성하시오.

```
import java.util.Scanner;
public class ArrayLength {
    public static void main (String[] args) {
        Scanner in = new Scanner(System.in);
        int intArray[] = new int[5];
        double sum = 0;

        for (int i = 0; i < intArray.length; i++)
            intArray[i] = in.nextInt();

        for (int i = 0; i < intArray.length; i++)
            sum += intArray[i];
        System.out.print("배열 원소의 평균은 " + sum/intArray.length + "입니다.");
    }
}
```

10  
20  
30  
40  
50

배열 원소의 평균은 30.0입니다.

# 배열과 for-each 문

110

## ▣ for-each 문

- 배열이나 나열(enumeration)의 각 원소를 순차적으로 접근하는데 유용한 for 문

```
int[] num = { 1,2,3,4,5 };  
int sum = 0;  
for (int k : num) // 반복될 때마다 k는 num[0], num[1], ..., num[4] 값으로 설정  
    sum += k;  
System.out.println("합은 " + sum);
```

합은 15

```
String names[] = { "사과", "배", "바나나", "체리", "딸기", "포도" };  
for (String s : names) // 반복할 때마다 s는 names[0], names[1], ..., names[5] 로 설정  
    System.out.print(s + " ");
```

사과 배 바나나 체리 딸기 포도

```
enum Week { 월, 화, 수, 목, 금, 토, 일 }  
for (Week day : Week.values()) // 반복될 때마다 day는 월, 화, 수, 목, 금, 토, 일로 설정  
    System.out.print(day + "요일 ");
```

월요일 화요일 수요일 목요일 금요일 토요일 일요일

# 예제 3-9 for-each 문을 이용한 반복문 활용

111

for-each 문을  
활용하는  
사례를 보자.

```
public class foreachEx {
    enum Week { 월, 화, 수, 목, 금, 토, 일 }

    public static void main(String[] args) {
        int[] num = { 1,2,3,4,5 };
        String names[] = { "사과", "배", "바나나", "체리", "딸기", "포도" };
        int sum = 0;

        // 아래 for-each에서 k는 num[0], num[1], ..., num[4]로 반복됨
        for (int k : num)
            sum += k;
        System.out.println("합은 " + sum);

        // 아래 for-each에서 s는 names[0], names[1], ..., names[5]로 반복됨
        for (String s : names)
            System.out.print(s + " ");
        System.out.println();

        // 아래 for-each에서 day는 월, 화, 수, 목, 금, 토, 일 값으로 반복됨
        for (Week day : Week.values())
            System.out.print(day + "요일 ");
        System.out.println();
    }
}
```

합은 15

사과 배 바나나 체리 딸기 포도

월요일 화요일 수요일 목요일 금요일 토요일 일요일

# 2차원 배열

112

## □ 2차원 배열 선언

```
int    intArray[][];  
char   charArray[][];  
float  floatArray[][];
```

또는

```
int[][] intArray;  
char[][] charArray;  
float[][] floatArray;
```

## □ 2차원 배열 생성

```
intArray = new int[2][5];  
charArray = new char[5][5];  
floatArray = new float[5][2];
```

또는

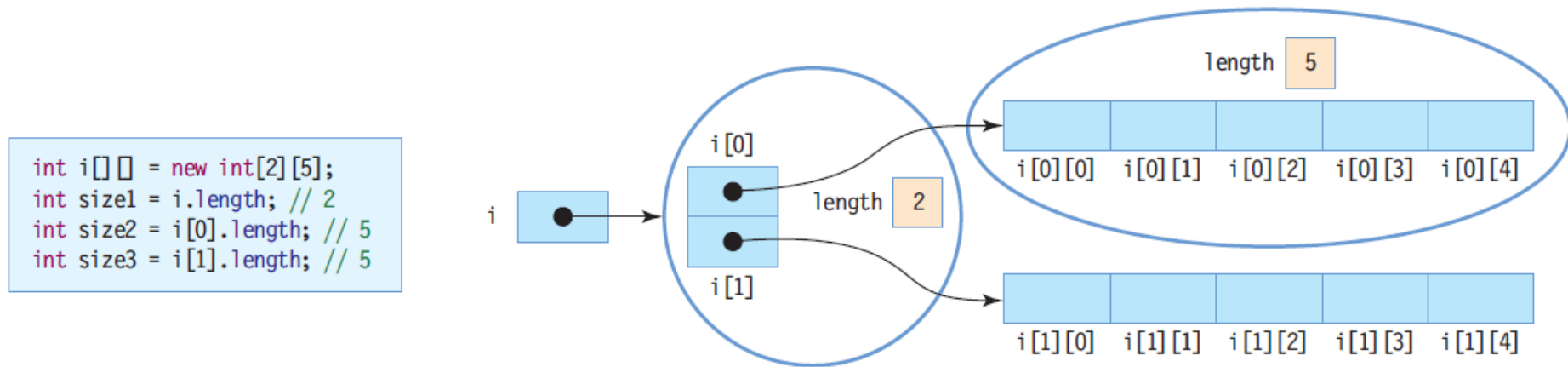
```
int    intArray[] = new int[2][5];  
char   charArray[] = new char[5][5];  
float  floatArray[] = new float[5][2];
```

## □ 2차원 배열 선언, 생성, 초기화

```
int intArray[][] = {{0,1,2},{3,4,5},{6,7,8}};  
char charArray[][] = {{'a', 'b', 'c'},{'d', 'e', 'f'}};  
float floatArray[][] = {{0.01, 0.02}, {0.03, 0.04}};
```

# 2차원 배열의 length 필드

113



## □ 2차원 배열의 length

- ▣ `i.length` -> 2차원 배열의 행의 개수로서 2
- ▣ `i[n].length`는 `n`번째 행의 열의 개수
  - `i[0].length` -> 0번째 행의 열의 개수로서 5
  - `i[1].length` -> 1번째 행의 열의 개수로서 역시 5



## 예제 3-10 : 3년간 매출 총액과 평균 구하기

114

한 회사의 지난 3년간 분기별 매출의 총액과 연평균 매출을 구하는 프로그램을 작성하시오.

```
public class SalesRevenue {  
    public static void main (String[] args) {  
        int intArray[][] = {{90, 90, 110, 110},  
                             {120, 110, 100, 110},  
                             {120, 140, 130, 150}} ;  
        double sum = 0;  
  
        for (int i = 0; i < intArray.length; i++) // intArray.length=3  
            for (int j = 0; j < intArray[i].length; j++) // intArray[i].length=4  
                sum += intArray[i][j];  
  
        System.out.println("지난 3년간 매출 총액은 " + sum + "이며 연평균 매출은 "  
                           + sum/intArray.length + "입니다.");  
    }  
}
```

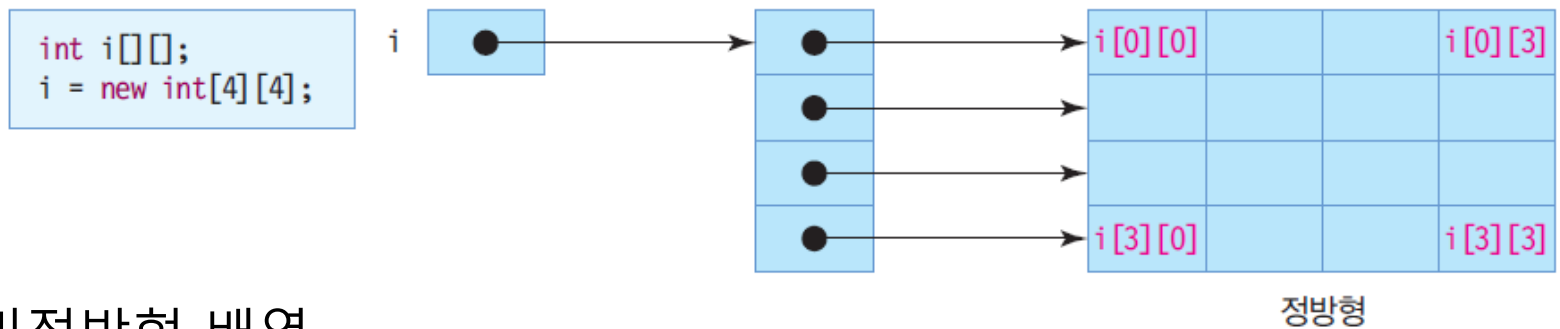
지난 3년간 매출 총액은 1380.0이며 연평균 매출은 460.0입니다.

# 비정방형 배열

115

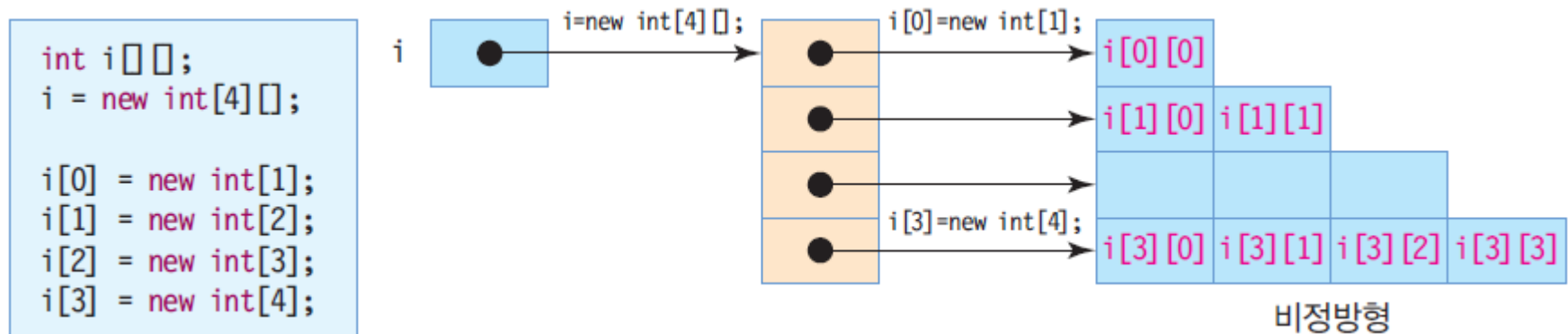
## □ 정방형 배열

- 각 행의 열의 개수가 같은 배열



## □ 비정방형 배열

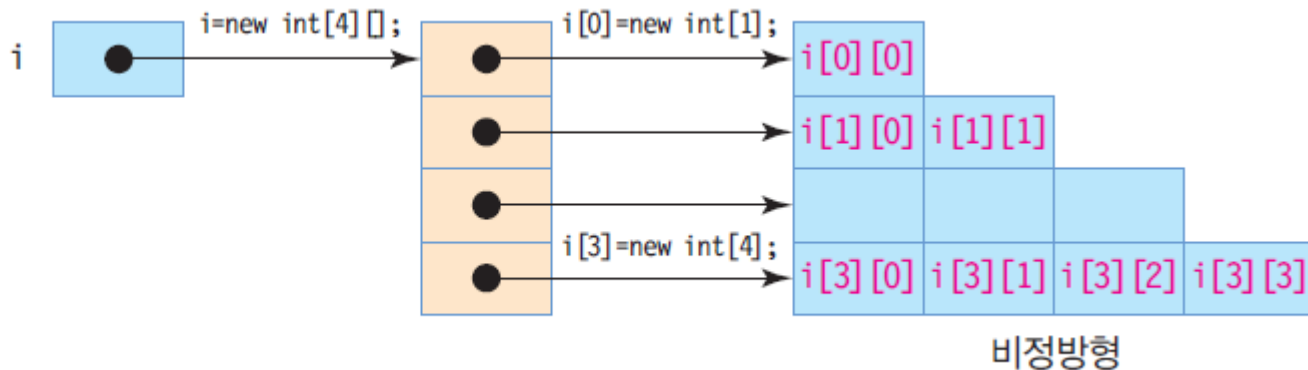
- 각 행의 열의 개수가 다른 배열
- 비정방형 배열의 생성



# 비정방형 배열의 length

116

```
int i[][];  
i = new int[4][];  
  
i[0] = new int[1];  
i[1] = new int[2];  
i[2] = new int[3];  
i[3] = new int[4];
```



## □ 비정방형 배열의 length

- ▣ `i.length` -> 2차원 배열의 행의 개수로서 4
- ▣ `i[n].length`는 `n`번째 행의 열의 개수
  - `i[0].length` -> 0번째 행의 열의 개수로서 1
  - `i[1].length` -> 1번째 행의 열의 개수로서 2
  - `i[2].length` -> 2번째 행의 열의 개수로서 3
  - `i[3].length` -> 3번째 행의 열의 개수로서 4

# 예제 3-11 : 비 정방형 배열의 생성과 접근

117

다음 그림과 같은 비정방형 배열을 만들어 값을 초기화하고 출력하시오.

10	11	12
20	21	
30	31	32
40	41	

```
public class IrregularArray {  
    public static void main (String[] args) {  
        int intArray[][] = new int[4][];  
        intArray[0] = new int[3];  
        intArray[1] = new int[2];  
        intArray[2] = new int[3];  
        intArray[3] = new int[2];  
  
        for (int i = 0; i < intArray.length; i++)  
            for (int j = 0; j < intArray[i].length; j++)  
                intArray[i][j] = (i+1)*10 + j;  
  
        for (int i = 0; i < intArray.length; i++) {  
            for (int j = 0; j < intArray[i].length; j++)  
                System.out.print(intArray[i][j] + " ");  
            System.out.println();  
        }  
    }  
}
```

```
10 11 12  
20 21  
30 31 32  
40 41
```

# 메소드에서 배열 리턴

118

- ▣ 메소드의 배열 리턴
  - 배열의 레퍼런스만 리턴
- ▣ 메소드의 리턴 타입
  - 메소드가 리턴하는 배열의 타입은 리턴 받는 배열 타입과 일치
  - 리턴 타입에 배열의 크기를 지정하지 않음

The diagram shows a Java method signature and its body. The signature is `int[] makeArray()`. The body contains `int temp[] = new int[4];` and `return temp;`. Annotations are provided for each part: `int[]` is labeled '리턴 타입' (Return Type), `makeArray()` is labeled '메소드 이름' (Method Name), and `temp` is labeled '배열 리턴' (Array Return).

```
int[] makeArray() {  
    int temp[] = new int[4];  
    return temp;  
}
```

## 예제 3-12 : 배열 리턴 예제

119

배열을 생성하고 각 원소 값을 출력하는 프로그램을 작성하시오. 배열 생성은 배열을 생성하여 각 원소의 인덱스 값으로 초기화하여 반환하는 메소드를 이용한다.

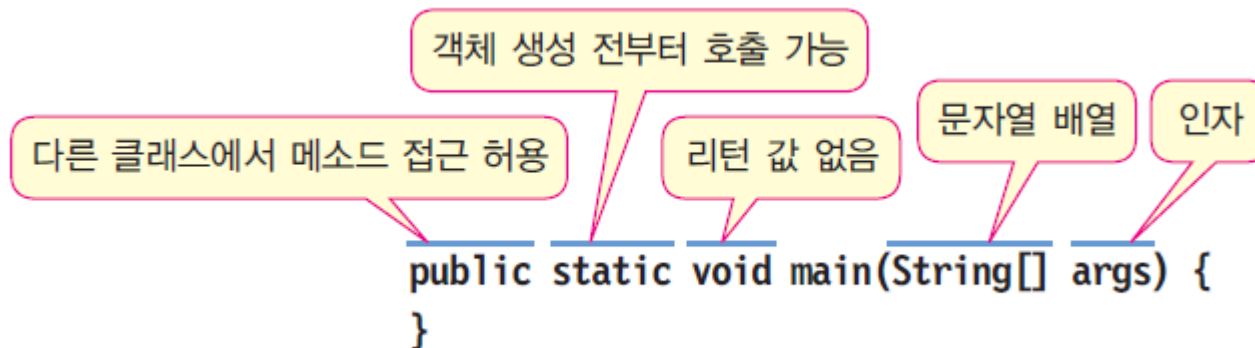
```
public class ReturnArray {  
    static int[] makeArray() {  
        int temp[] = new int[4];  
        for (int i=0;i<temp.length;i++)  
            temp[i] = i;  
        return temp;  
    }  
    public static void main (String[] args) {  
        int intArray [];  
        intArray = makeArray();  
        for (int i = 0; i < intArray.length; i++)  
            System.out.println(intArray[i]);  
    }  
}
```

0  
1  
2  
3

# main() 메소드

120

- main()은 자바 응용프로그램의 실행 시작 메소드
- main()의 원형
  - 반드시 static
  - 반드시 public
  - 반드시 void
  - 반드시 매개 변수 타입은 문자열 배열

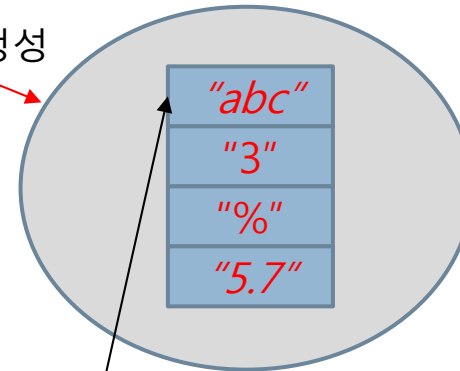


# main(string [] args) 메소드의 인자 전달

121

C:\W>java Hello **abc 3 % 5.7**

생성



class Hello

```
public static void main(String[] args) args
```

```
{  
    String a = args[0]; // a는 "abc"  
    String b = args[1]; // b는 "3"  
    String c = args[2]; // c는 "%"  
    String d = args[3]; // d는 "5.7"  
}
```

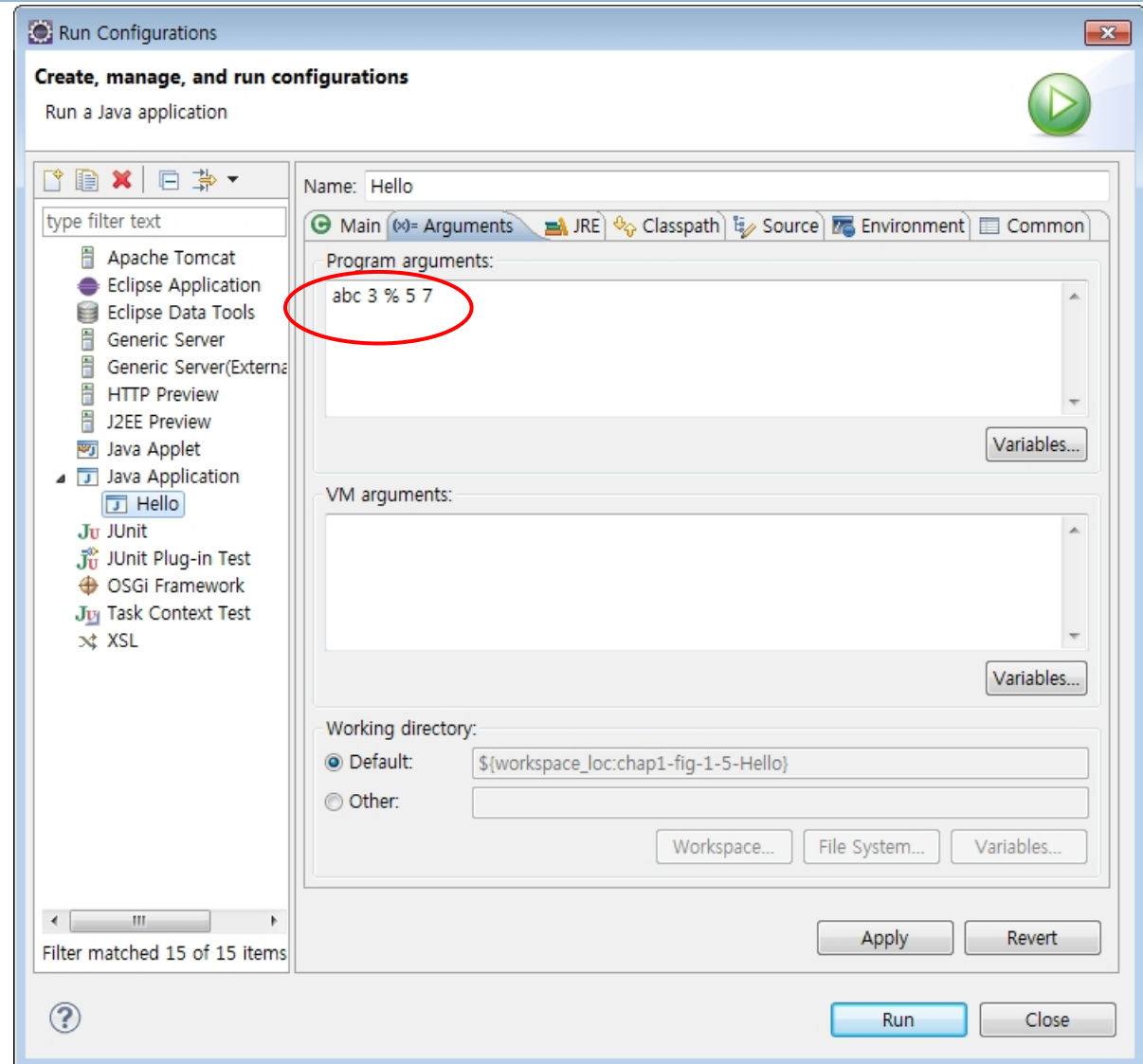
```
args.length => 4  
args[0] => "abc"  
args[1] => "3"  
args[2] => "%"  
args[3] => "5.7"
```



# 이클립스에서 main() 메소드의 인자전달

122

Run 메뉴의  
Run Configurations  
항목에서  
main() 메소드의  
인자 나열

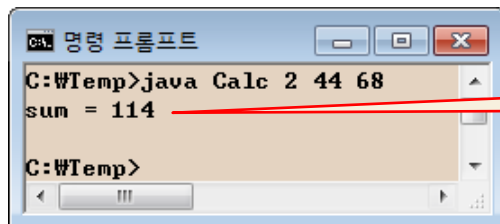


# main()의 인자 이용 예

123

```
public class Calc {  
    public static void main(String[] args) {  
        int sum = 0;  
        for(int i=0; i<args.length; i++) { // 명령행 인자의 개수만큼 반복  
            int n = Integer.parseInt(args[i]); // 명령행 인자인 문자열을 정수로 변환  
            sum += n; // 숫자를 합한다.  
        }  
        System.out.println("sum = " + sum);  
    }  
}
```

Integer.parseInt()는 매개변수로  
주어진 문자열을 정수로 변환.  
Integer.parseInt("68") 은 정수 68  
리턴



명령행 인자  
2, 44, 68을  
모두 합하여 114 출력

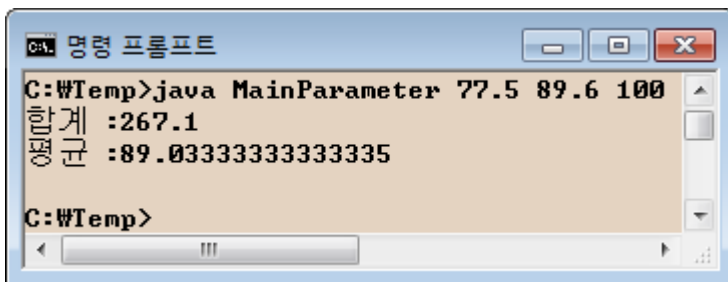
## 예제 3-13 : main()의 인자들을 받아서 평균값을 계산하는 예제

124

여러 개의 실수를 main() 메소드 인자로 전달받아 평균값을 구하는 프로그램을 작성하시오.

```
public class MainParameter {  
    public static void main (String[] args) {  
        double sum = 0.0;  
  
        for (int i=0; i<args.length; i++)  
            sum += Double.parseDouble(args[i]);  
        System.out.println("합계 :" + sum);  
        System.out.println("평균 :" + sum/args.length);  
    }  
}
```

Double.parseDouble()는 매개변수로 주어진 문자열을 실수로 변환.  
Double.parseDouble("77.5") 은 실수 77.5 리턴



The screenshot shows a Windows Command Prompt window titled "명령 프롬프트". The command entered is `C:\WTemp>java MainParameter 77.5 89.6 100`. The output displayed is:  
`합계 :267.1`  
`평균 :89.03333333333335`  
The prompt then returns to `C:\WTemp>`.