

(INTERMEDIATE) JAVA PROGRAMMING

5. If Statement, Repetition and Arrays
Chapter 2 & 3

JAVA: THE IF STATEMENT

Conditional Statements

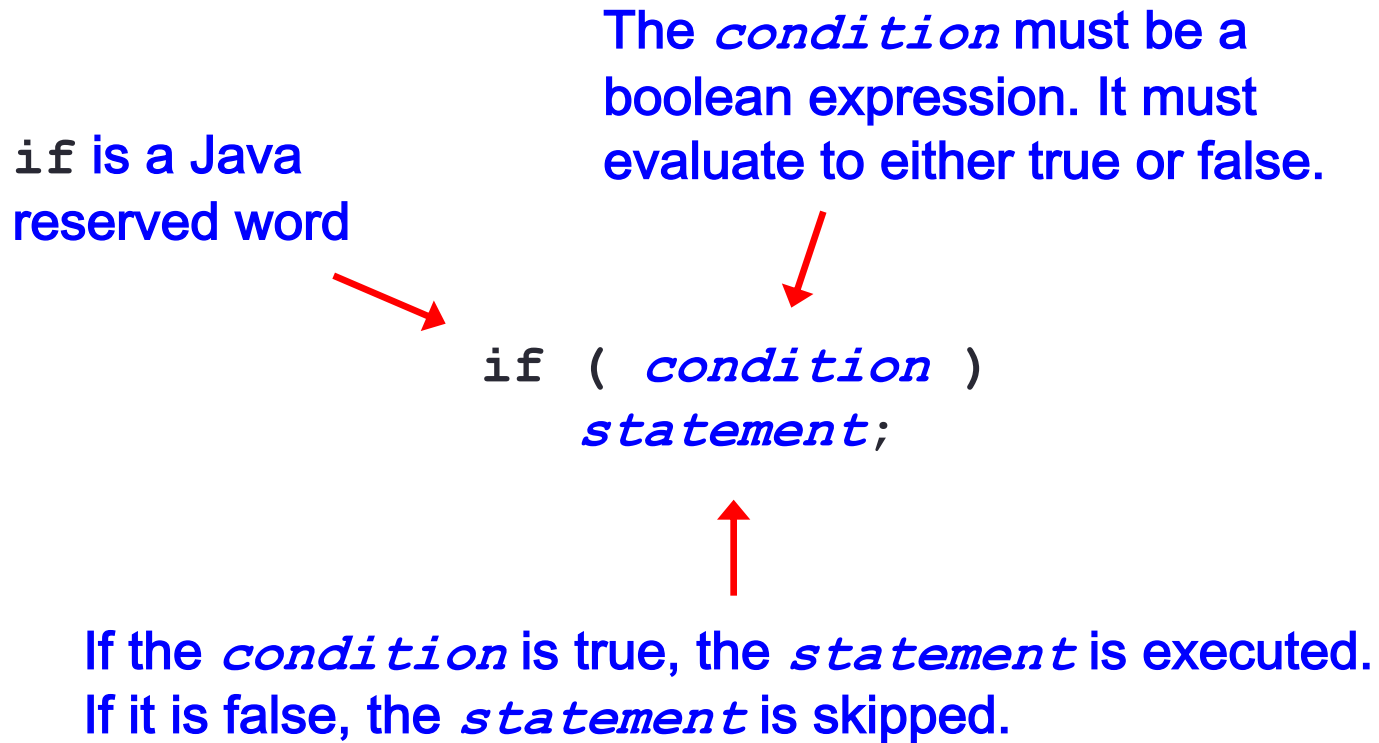
- A *conditional statement* lets us choose which statement will be executed next
- Therefore they are sometimes called *selection statements*
- Conditional statements give us the power to make basic decisions
- The Java conditional statements are the:
 - *if statement*
 - *if-else statement*
 - *switch statement*

The if Statement

- The *if statement* has the following syntax:

`if` is a Java
reserved word

The *condition* must be a
boolean expression. It must
evaluate to either true or false.

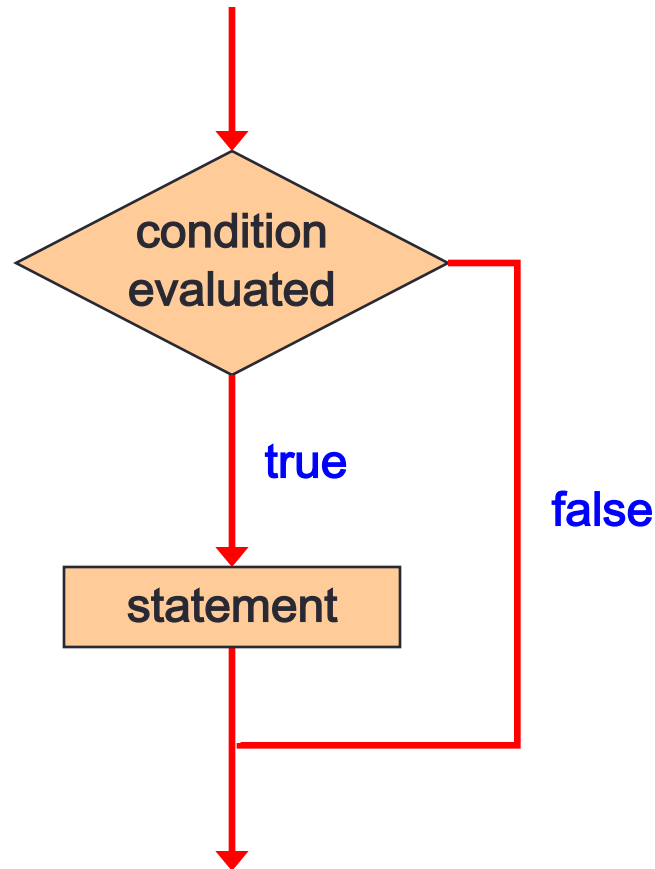


```
if ( condition )  
    statement;
```

The diagram illustrates the syntax of an if statement. It features the code `if (condition)
 statement;` in the center. Three red arrows point to specific parts of the code: one from the text '`if` is a Java reserved word' to the `if` keyword; another from the text 'The *condition* must be a boolean expression...' to the *condition* inside the parentheses; and a third from the text 'If the *condition* is true...' to the *statement* following the closing parenthesis.

If the *condition* is true, the *statement* is executed.
If it is false, the *statement* is skipped.

Logic of an if statement



Boolean Expressions

- A condition often uses one of Java's *equality operators* or *relational operators*, which all return boolean results:

==	equal to
!=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

- Note the difference between the equality operator (==) and the assignment operator (=)

The if Statement

- What do the following statements do?

```
if (top >= MAXIMUM)
    top = 0;
```

Sets top to zero if the current value of top is greater than or equal to the value of MAXIMUM

```
if (total != stock + warehouse)
    inventoryError = true;
```

Sets a flag to true if the value of total is not equal to the sum of stock and warehouse

- The precedence of the arithmetic operators is higher than the precedence of the equality and relational operators

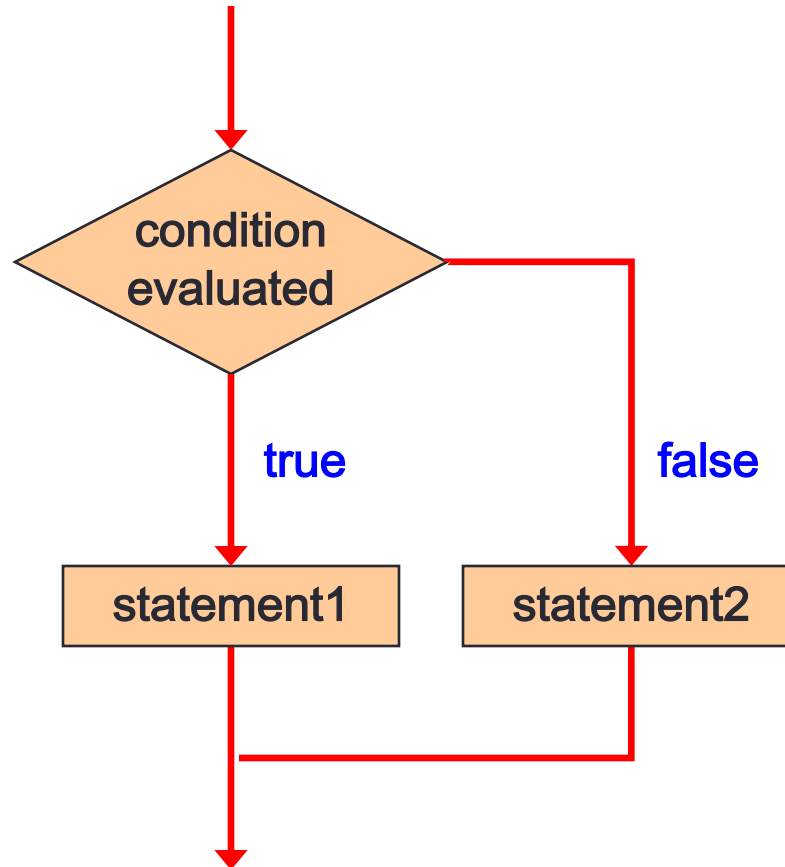
The if-else Statement

- An *else clause* can be added to an `if` statement to make an *if-else statement*

```
if ( condition )  
    statement1;  
else  
    statement2;
```

- If the *condition* is true, *statement1* is executed; if the condition is false, *statement2* is executed
- One or the other will be executed, but not both

Logic of an if-else statement



Logical Operators

- Boolean expressions can also use the following *logical operators*:

!	Logical NOT
& &	Logical AND
	Logical OR

- They all take boolean operands and produce boolean results
- Logical NOT is a unary operator (it operates on one operand)
- Logical AND and logical OR are binary operators (each operates on two operands)

Logical NOT

- The *logical NOT* operation is also called *logical negation* or *logical complement*
- If some boolean condition a is true, then $!a$ is false; if a is false, then $!a$ is true
- Logical expressions can be shown using a *truth table*

a	$!a$
true	false
false	true

Logical AND and Logical OR

- The *logical AND* expression

`a && b`

is true if both `a` and `b` are true, and false otherwise

- The *logical OR* expression

`a || b`

is true if `a` or `b` or both are true, and false otherwise

Logical Operators

- Expressions that use logical operators can form complex conditions

```
if (total < MAX+5 && !found)
    System.out.println ("Processing..") ;
```

- **All logical operators have lower precedence than the relational operators**
- **Logical NOT has higher precedence than logical AND and logical OR**

Short-Circuited Operators

- The processing of logical AND and logical OR is “short-circuited”
- If the left operand is sufficient to determine the result, the right operand is not evaluated

```
if (count != 0 || (avg=total/count) < MAX)
    System.out.println (avg);
```

- This type of processing must be used carefully

Indentation

- The statement controlled by the `if` statement is indented to **indicate that relationship**
- The use of a **consistent indentation style** makes a program easier to read and understand
- Although it makes no difference to the compiler, proper indentation is crucial


"Always code as if the person who ends up maintaining your code will be a violent psychopath who knows where you live."

-- Martin Golding

Indentation Revisited

- Remember that indentation is for the human reader, and is ignored by the computer

```
if (total > MAX)
    System.out.println ("Error!!");
    errorCount++;
```



Despite what is implied by the indentation, the increment will occur whether the condition is true or not

Block Statements

- In an `if-else` statement, the `if` portion, or the `else` portion, or both, could be block statements

```
if (total > MAX)
{
    System.out.println ("Error!!");
    errorCount++;
}
else
{
    System.out.println ("Total: " + total);
    current = total*2;
}
```

The Conditional Operator

- Java has a *conditional operator* that uses a boolean condition to determine which of two expressions is evaluated
- Its syntax is:

condition ? *expression1* : *expression2*

- If the *condition* is true, *expression1* is evaluated; if it is false, *expression2* is evaluated
- The value of the entire conditional operator is the value of the selected expression

The Conditional Operator

- The conditional operator is similar to an `if-else` statement, except that it is an expression that returns a value
- For example:

```
larger = ((num1 > num2) ? num1 : num2);
```

- If `num1` is greater than `num2`, then `num1` is assigned to `larger`; otherwise, `num2` is assigned to `larger`
- The conditional operator is *ternary* because it requires three operands

The Conditional Operator

- Another example:

```
System.out.println ("Your change is " + count +  
    ((count == 1) ? "Dime" : "Dimes"));
```

- If count equals 1, then "Dime" is printed
- If count is anything other than 1, then "Dimes" is printed

Nested if Statements

- The statement executed as a result of an `if` statement or `else` clause could be another `if` statement
- These are called *nested if statements*
- An `else` clause is matched to the last unmatched `if` (no matter what the indentation implies)
- Braces can be used to specify the `if` statement to which an `else` clause belongs

중첩 if

```
if( grade >= 80 )    // ❶  
    if( grade >= 90 ) // ❷  
        System.out.println("당신의 학점은 A입니다.");
```

if 문안의 문장자리에 if문이
들어간 경우

```
if( grade >= 80 )    // ❶  
    if( grade >= 90 ) // ❷  
        System.out.println("당신의 학점은 A입니다.");  
else  
    System.out.println("당신의 학점은 B입니다.");
```

if 문안의 문장자리에 if-else
문이 들어간 경우

if와 else의 매칭 문제

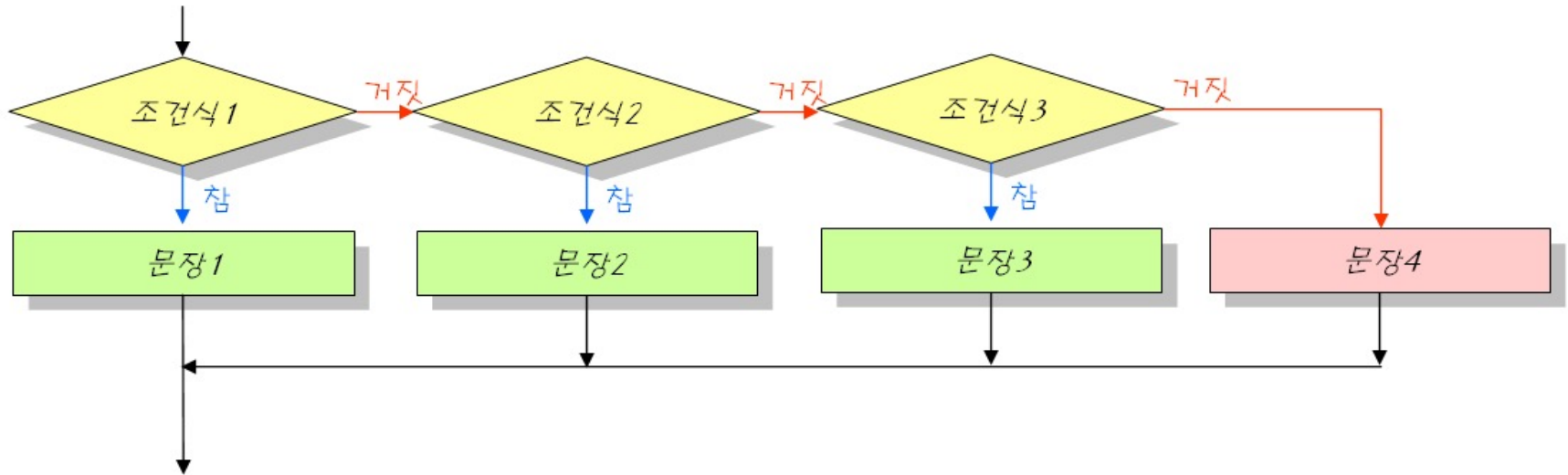
else 절은 가장 가까운
if절과 매치된다.

```
if(grade >= 80)
    if( grade >= 90)
        System.out.println("당신의 학점은 A입니다");
    else
        System.out.println("당신의 학점은 B입니다");
```

```
if( grade >= 80 ) {
    if(grade >= 90 )
        System.out.println("당신의 학점은 A입니다.");
}
else
    System.out.println("당신의 학점은 A나 B가 아닙니다.");
```

만약 다른 if절과 else
절을 매치 시키려면 중
괄호를 사용하여 블록
으로 묶는다.

연속적인 if



```
if( 조건식1 )  
    문장1;  
else if( 조건식2 )  
    문장2;  
else if( 조건식3 )  
    문장3;  
else  
    문장4;
```


JAVA: REPETITION STATEMENTS

Repetition Statements

- Java has three kinds of repetition statements:
 - the *while loop*
 - the *do loop*
 - the *for loop*
- The programmer should choose the right kind of loop for the situation

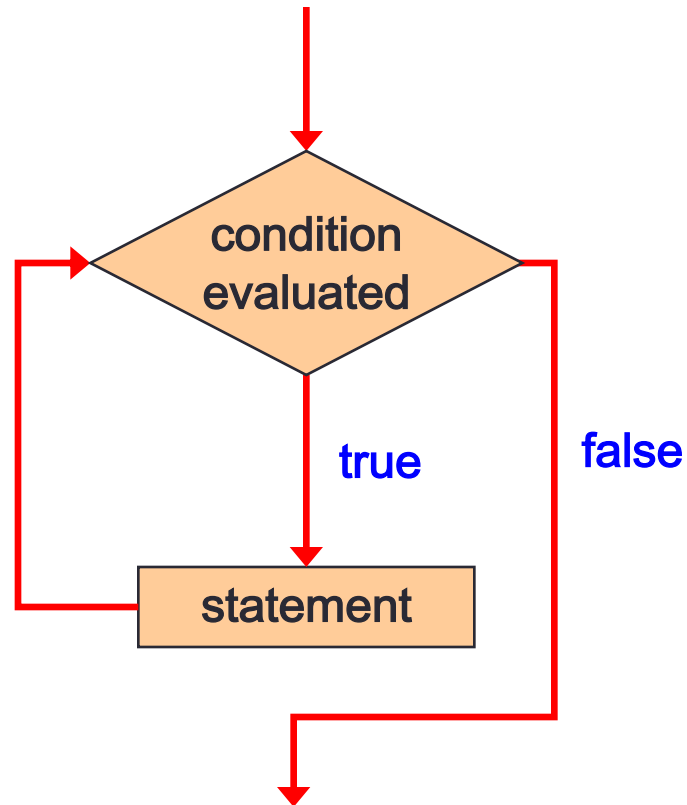
The while Statement

- A *while statement* has the following syntax:

```
while ( condition )  
    statement;
```

- The statement is executed repeatedly until the condition becomes false

Logic of a while Loop



The while Statement

- An example of a while statement:

```
int count = 1;
while (count <= 5)
{
    System.out.println (count);
    count++;
}
```

- If the condition of a `while` loop is false initially, the statement is never executed
- Therefore, the body of a `while` loop will execute zero or more times


Infinite Loops

- The body of a **while** loop eventually must make the condition false
- *an infinite loop* will execute until the user interrupts the program

Infinite Loops

- An example of an infinite loop:

```
int count = 1;
while (count <= 25)
{
    System.out.println (count);
    count = count - 1;
}
```

- This loop will continue executing until interrupted (**terminate** ) or until an underflow error occurs

예제

- 두수의 최대 공약수 구하기
- 유클리드 호제 알고리즘

- https://ko.wikipedia.org/wiki/%EC%9C%A0%ED%81%B4%EB%A6%AC%EB%93%9C_%ED%98%B8%EC%A0%9C%EB%B2%95
- http://mathbees2.blogspot.kr/2014/09/4_24.html

1. 두 수 가운데 큰 수를 x , 작은 수를 y 라 한다.
2. $r \leftarrow x \% y$
3. r 이 0이면 공약수는 y 이다.
4. $x \leftarrow y$
5. $y \leftarrow r$
6. 단계 2로 돌아 간다.

예제

Gcd.java

```
01 import java.util.*;
02
03 public class Gcd {
04     public static void main(String[] args) {
05         int x, y, r;
06         System.out.print("두 개의 정수를 입력하시오(큰 수, 작은 수): ");
07         Scanner scan = new Scanner(System.in);
08         x = scan.nextInt();
09         y = scan.nextInt();
10
11         while (y != 0) {
12             r = x % y;
13             x = y;
14             y = r;
15         }
16         System.out.printf("최대 공약수는 %d입니다.\n", x);
17     }
18 }
```

← y가 0이 아니면 반복을 계속한다.

실행결과

두 개의 정수를 입력하시오(큰 수, 작은 수): 24 36
최대 공약수는 12입니다.

Nested Loops

- Similar to nested `if` statements, loops can be nested as well
- That is, the body of a loop can contain another loop
- For each iteration of the outer loop, the inner loop iterates completely

Nested Loops

- How many times will the string "Here" be printed?

```
count1 = 1;
while (count1 <= 20)
{
    count2 = 1;
    while (count2 <= count1)
    {
        System.out.println ("Here");
        count2++;
    }
    count1++;
}
```

$$20 * (20-1) / 2 = 190$$

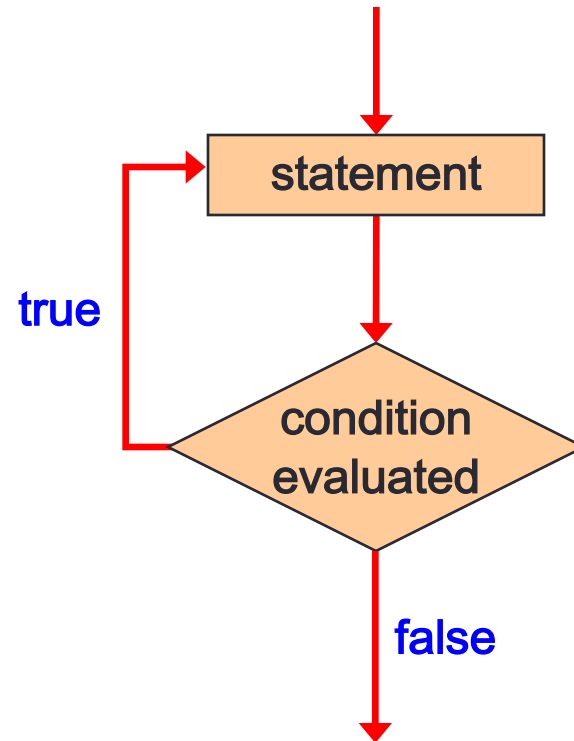
The do Statement

- A *do statement* has the following syntax:

```
do
{
    statement;
}
while ( condition )
```

- The ***statement*** is executed once initially, and then the ***condition*** is evaluated
- The statement is executed repeatedly until the condition becomes false

Logic of a do Loop



The do Statement

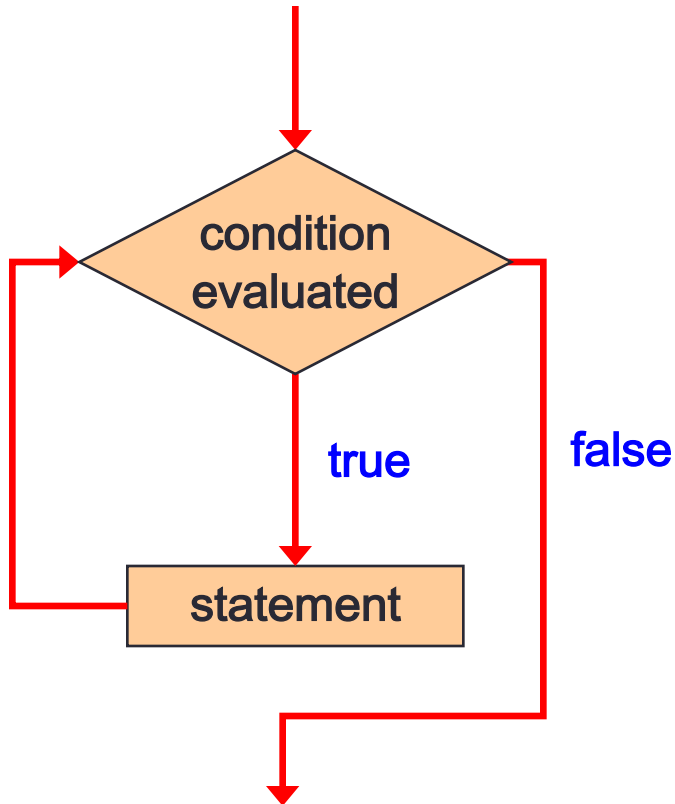
- An example of a **do** loop:

```
int count = 0;  
do  
{  
    count++;  
    System.out.println (count);  
} while (count < 5);
```

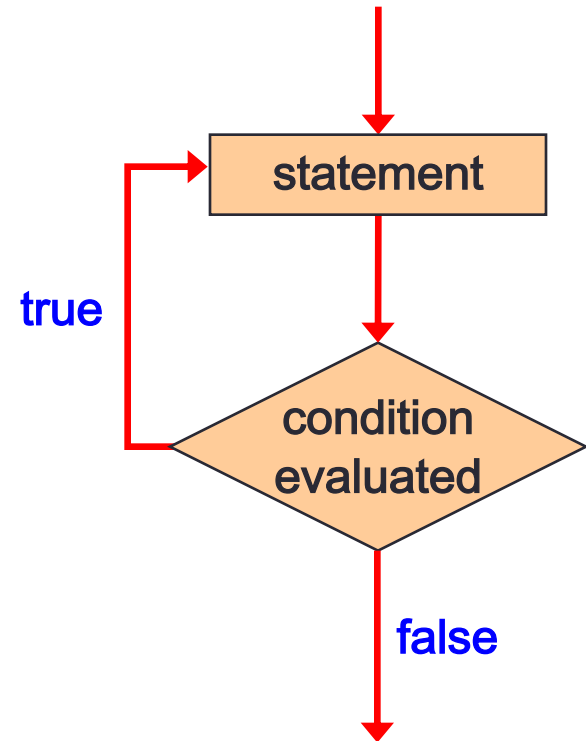
- **The body of a do loop executes at least once**

Comparing while and do

The while Loop



The do Loop



The for Statement

- A *for statement* has the following syntax:

The *initialization*
is executed once
before the loop begins



The *statement* is
executed until the
condition becomes false

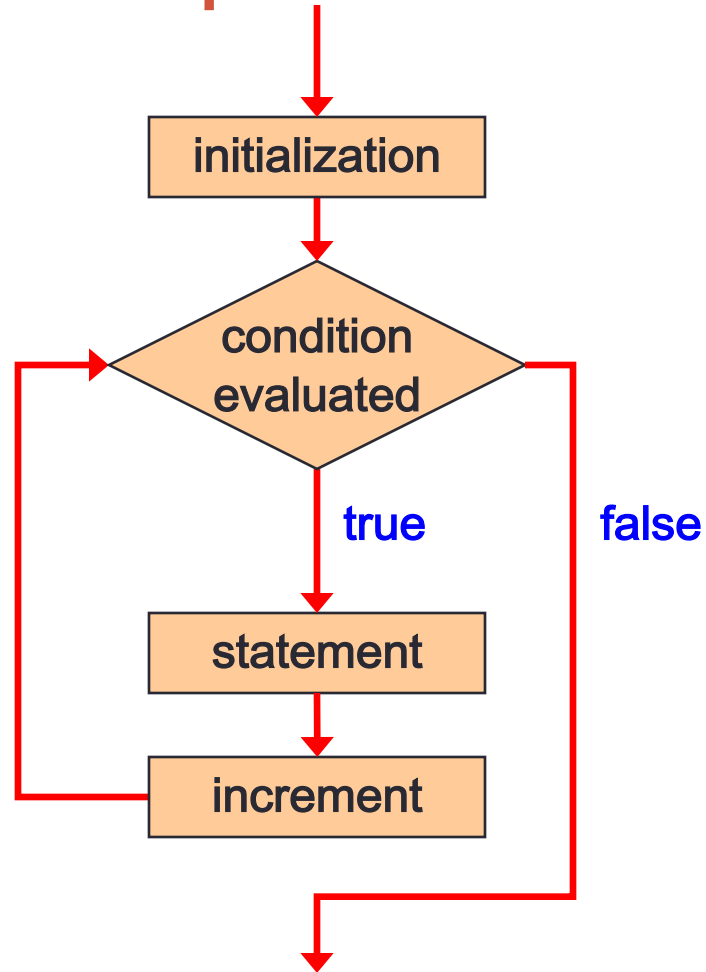


```
for ( initialization ; condition ; increment )  
    statement;
```



The *increment* portion is executed at
the end of each iteration

Logic of a for loop



The for Statement

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;  
while ( condition )  
{  
    statement;  
    increment;  
}
```

The for Statement

- An example of a `for` loop:



```
for (int count=1; count <= 5; count++)  
    System.out.println (count);
```

- The initialization section can be used to declare a variable
- Like a `while` loop, the condition of a `for` loop is tested prior to executing the loop body
- Be careful to the scope of the variable 'count'!

The for Statement

- The increment section can perform any calculation

```
for (int num=100; num > 0; num -= 5)  
    System.out.println (num);
```

- A `for` loop is well suited for executing statements a specific number of times that can be calculated or determined in advance

예제

NestedLoop.java

```
01 import java.util.*;
02
03 public class NestedLoop {
04     public static void main(String[] args) {
05
06         for (int y = 0; y < 5; y++) {
07             for (int x = 0; x < 10; x++)
08                 System.out.print("*");
09
10             System.out.println("");
11         }
12
13     }
14 }
```

프로그램을 실행하면 50개의 *가 화면에 5×10 의 정사각형 모양으로 출력된다. *를 출력하는 문장의 외부에는 두개의 for 루프가 중첩되어 있다. 외부의 for 루프는 변수 y를 0에서 4까지 증가시키면서 내부의 for 루프를 실행시킨다. 내부의 for 루프는 변수 x를 0에서 9까지 증가시키면서 print() 메소드를 호출한다. 내부 for 루프가 한번 실행될 때마다 화면에는 한 줄의 *가 그려진다. 내부 for 루프가 한 번씩 종료될 때마다 줄바꿈 문자가 화면에 출력되어 다음 줄로 넘어가게 된다.

실행결과

```
*****
*****
*****
*****
*****
```

for문의 특이한 형태

```
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)) {
.....
}
```

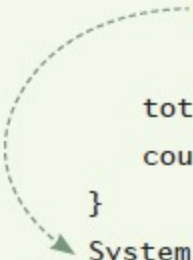
```
// for문 내에 변수 선언
for(int i=0; i<10; i++) { // 변수 i는 for문 내에서만 사용 가능
.....
}
```

break 문 : Loop의 중단

```
import java.util.*;

public class Averager {
    public static void main(String[] args) {
        int total = 0;
        int count = 0;
        Scanner scan = new Scanner(System.in);
        while (true) {
            System.out.print("점수를 입력하시오: ");
            int grade = scan.nextInt();
            if (grade < 0)
                break;

            total += grade;
            count++;
        }
        System.out.println("평균은 " + total / count);
    }
}
```



continue 문 : 현재 iteration의 중단

```
01 public class ContinueTest {
02     public static void main(String[] args) {
03
04         String s = "no news is good news";
05         int n = 0;
06
07         for (int i = 0; i < s.length(); i++) {
08             // n이 나오는 회수를 센다.
09             if(s.charAt(i) != 'n')
10                 continue;
11
12             // n의 개수를 하나 증가한다.
13             n++;
14         }
15         System.out.println("문장에서 발견된 n의 개수 " + n);
16     }
17 }
```

s에 들어있는 i번째 문자가 'n'이 아니면 반복을 다시 시작한다.

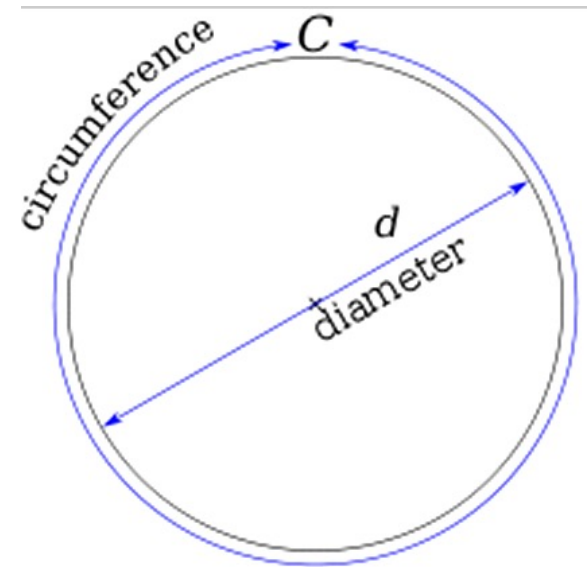
실행결과

문장에서 발견된 n의 개수 3

예제: 파이 구하기

- 파이를 계산하는 가장 고전적인 방법은 Gregory-Leibniz 무한 수열을 이용하는 것

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \dots$$



실행 결과



실행결과

반복횟수:100000

$\text{Pi} = 3.141583$

계속하려면 아무 키나 누르십시오 . . .

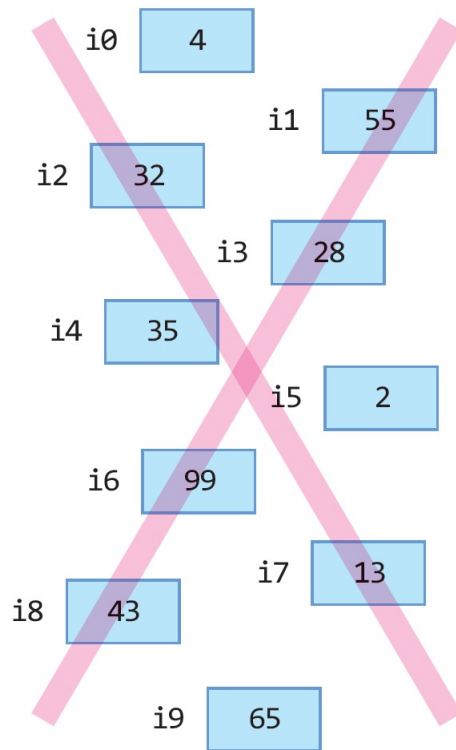
배열이란?

- 배열(array)
 - 인덱스와 인덱스에 대응하는 데이터들로 이루어진 자료 구조
 - 배열을 이용하면 한 번에 많은 메모리 공간 할당 가능
 - 같은 타입의 데이터들이 순차적으로 저장
 - 인덱스를 이용하여 원소 데이터 접근
 - 반복문을 이용하여 처리하기에 적합
 - 배열 인덱스
 - 0부터 시작
 - 인덱스는 배열의 시작 위치에서부터 데이터가 있는 상대 위치

자바 배열의 필요성과 모양

(1) 10개의 정수형 변수를 사용하는 경우

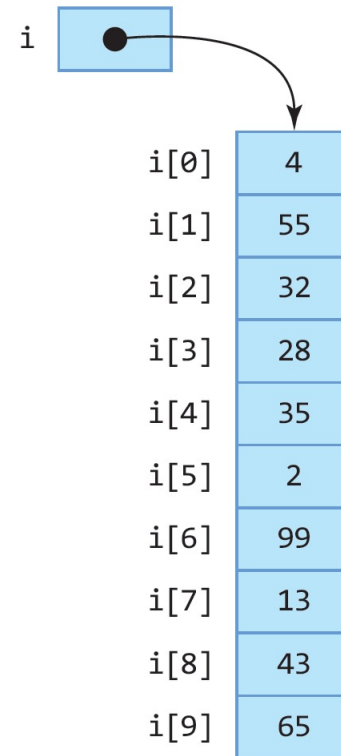
```
int i0, i1, i2, i3, i4, i5, i6, i7, i8, i9;
```



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

(2) 10개의 정수로 구성된 배열을 사용하는 경우

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



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

일차원 배열 만들기

- 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 함께 초기화

- 배열 선언 시 값 초기화

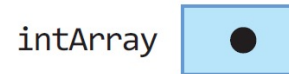
```
int intArray[] = {0,1,2,3,4,5,6,7,8,9}; // 초기화된 값의 개수(10)만큼의 배열 생성
```

- 잘못된 배열 선언

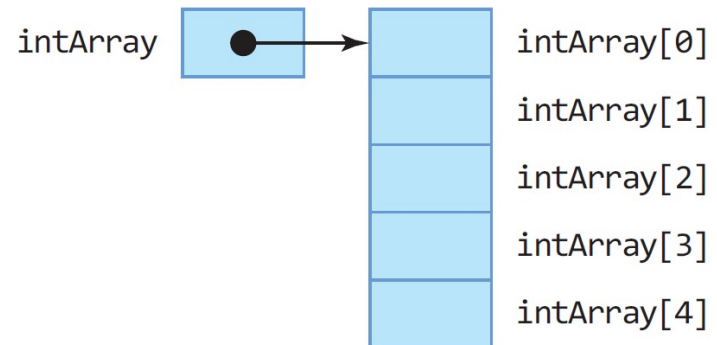
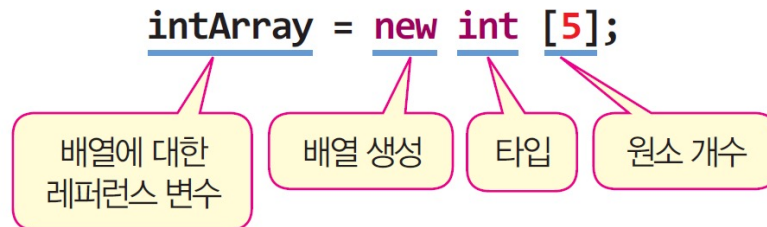
```
int intArray[10]; // 컴파일 오류. 배열의 크기를 지정하면 안됨
```

레퍼런스 변수와 배열

(1) 배열에 대한 레퍼런스 변수 `intArray` 선언

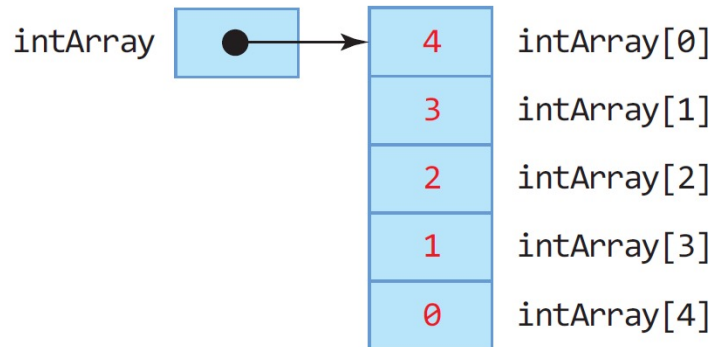


(2) 배열 생성

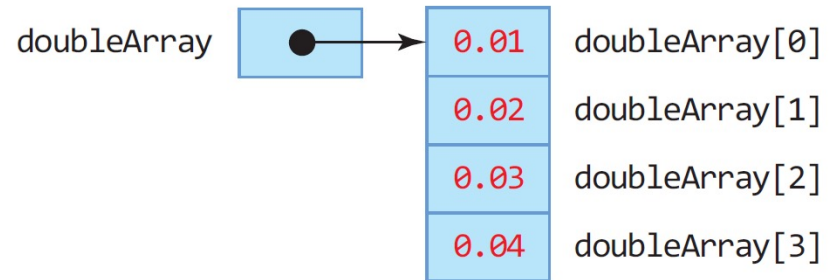


배열을 초기화하면서 생성한 결과

```
int intArray[] = {4, 3, 2, 1, 0};
```



```
double doubleArray[] = {0.01, 0.02, 0.03, 0.04};
```



배열 인덱스와 원소 접근

• 배열 원소 접근

- 배열 변수명과 [] 사이에 원소의 인덱스를 적어 접근
 - 배열의 인덱스는 0부터 시작
 - 배열의 마지막 항목의 인덱스는 (배열 크기 - 1)

```
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
```

• 인덱스의 범위

오류

```
n = intArray[-2]; // 실행 오류. 인덱스로 음수 사용 불가
n = intArray[5]; // 실행 오류. 5는 인덱스의 범위(0~4)를 넘었음
```

• 반드시 배열 생성 후 접근

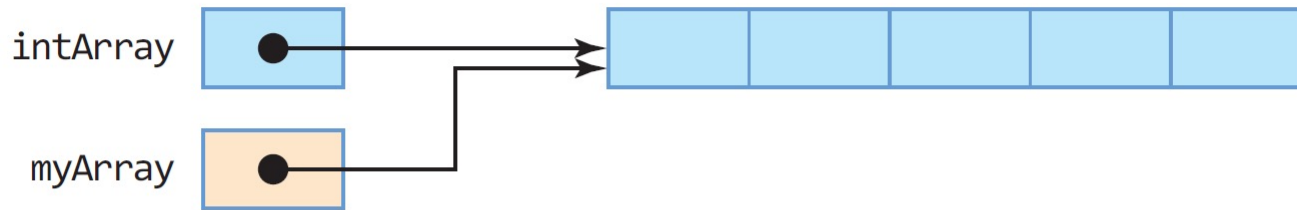
오류

```
int intArray [];
intArray[1] = 8; // 오류, 생성 되지 않은 배열 사용
```

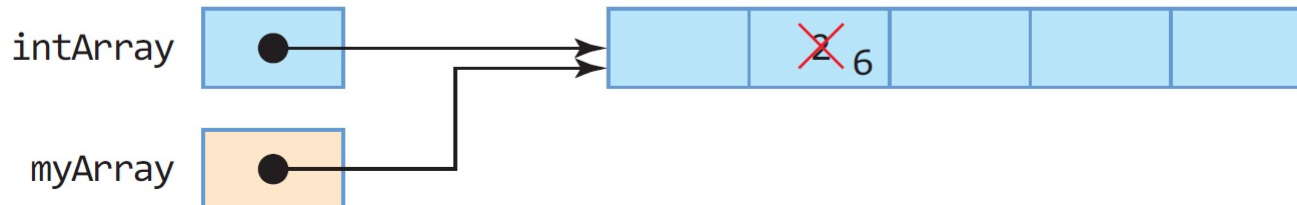

레퍼런스 치환과 배열 공유

- 하나의 배열을 다수의 레퍼런스가 참조 가능

```
int intArray[] = new int[5];  
int myArray[] = intArray;
```



```
intArray[1] = 2;  
myArray[1] = 6;
```



예제 3-7 : 배열에 입력받은 수 중 제일큰수 찾기

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

```
import java.util.Scanner;

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

        int intArray[] = new int[5]; // 배열 생성

        int max=0;    // 현재 가장 큰 수
        System.out.println("양수 5개를 입력하세요.");
        for(int i=0; i<5; i++) {
            intArray[i] = scanner.nextInt(); // 입력받은 정수를 배열에 저장
            if(intArray[i] > max) // intArray[i]가 현재 가장 큰 수보다 크면
                max = intArray[i]; // intArray[i]를 max로 변경
        }
        System.out.print("가장 큰 수는 " + max + "입니다.");

        scanner.close();
    }
}
```

양수 5개를 입력하세요.

1

39

78

100

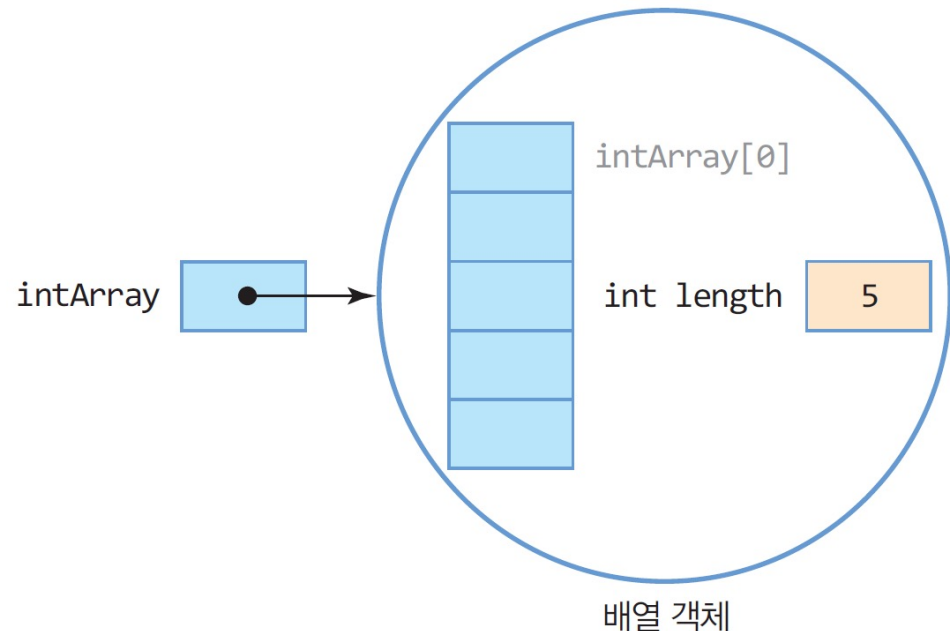
99

가장 큰 수는 100입니다.

배열의 크기, length 필드

- 배열은 자바에서 객체로 관리
 - 배열 객체 내에 length 필드는 배열의 크기를 나타냄

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



예제 3-8 : 배열 원소의 평균 구하기

배열의 length 필드를 이용하여 배열 크기만큼 정수를 입력 받고 평균을 구하는 프로그램을 작성하라.

```
import java.util.Scanner;

public class ArrayLength {
    public static void main(String[] args) {
        int intArray[] = new int[5]; // 배열의 선언과 생성
        int sum=0;

        Scanner scanner = new Scanner(System.in);
        System.out.print(intArray.length + "개의 정수를 입력하세요>>");
        for(int i=0; i<intArray.length; i++)
            intArray[i] = scanner.nextInt(); // 키보드에서 입력받은 정수 저장

        for(int i=0; i<intArray.length; i++)
            sum += intArray[i]; // 배열에 저장된 정수 값을 더하기

        System.out.print("평균은 " + (double)sum/intArray.length);
        scanner.close();
    }
}
```

5개의 정수를 입력하세요>> 2 3 4 5 9
평균은 4.6

배열과 for-each 문

- 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 문 활용

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 [] n = { 1,2,3,4,5 };
        String names[] = { "사과", "배", "바나나", "체리", "딸기", "포도" };

        int sum = 0;
        // 아래 for-each에서 k는 n[0], n[1], ..., n[4]로 반복
        for (int k : n) {
            System.out.print(k + " "); // 반복되는 k 값 출력
            sum += k;
        }
        System.out.println("합은" + sum);

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

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

1 2 3 4 5 합은 15

사과 배 바나나 체리 딸기 포도

월요일 화요일 수요일 목요일 금요일 토요일 일요일

2차원 배열

□ 2차원 배열 선언

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

또는

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

□ 2차원 배열 생성

```
intArray = new int[2][5];
charArray = new char[5][5];
doubleArray = new double[5][2];
```

또는

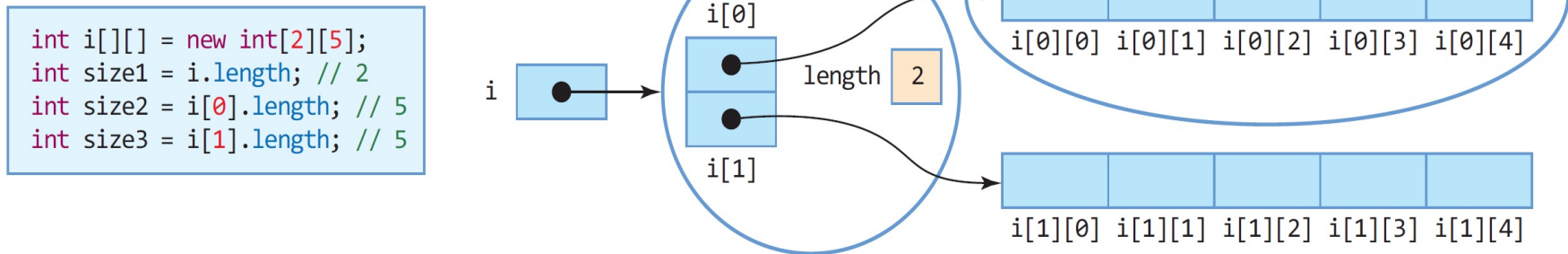
```
int    intArray[] = new int[2][5];
char   charArray[] = new char[5][5];
double doubleArray[] = new double[5][2];
```

□ 2차원 배열 선언, 생성, 초기화

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

2차원 배열의 모양과 length 필드

• 2차원 배열의 모양



• 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 5

예제 3-10 : 2차원 배열로 4년 평점 구하기

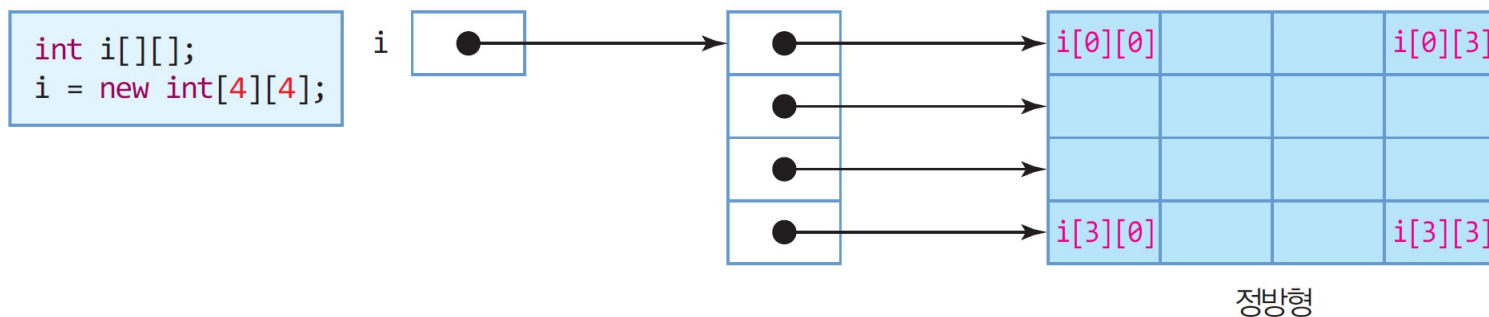
2차원 배열에 학년별로 1,2학기 성적으로 저장하고, 4년간 전체 평점 평균을 출력하라.

```
public class ScoreAverage {  
    public static void main(String[] args) {  
        double score[][] = {{3.3, 3.4},    // 1학년 1, 2학기 평점  
                             {3.5, 3.6},    // 2학년 1, 2학기 평점  
                             {3.7, 4.0},    // 3학년 1, 2학기 평점  
                             {4.1, 4.2}};    // 4학년 1, 2학기 평점        double sum=0;  
        for(int year=0; year<score.length; year++) // 각 학년별로 반복  
            for(int term=0; term<score[year].length; term++) // 각 학년의 학기별로 반복  
                sum += score[year][term]; // 전체 평점 합        int n=score.length; // 배열의 행 개수, 4  
        int m=score[0].length; // 배열의 열 개수, 2  
        System.out.println("4년 전체 평점 평균은 " + sum/(n*m));  
    }  
}
```

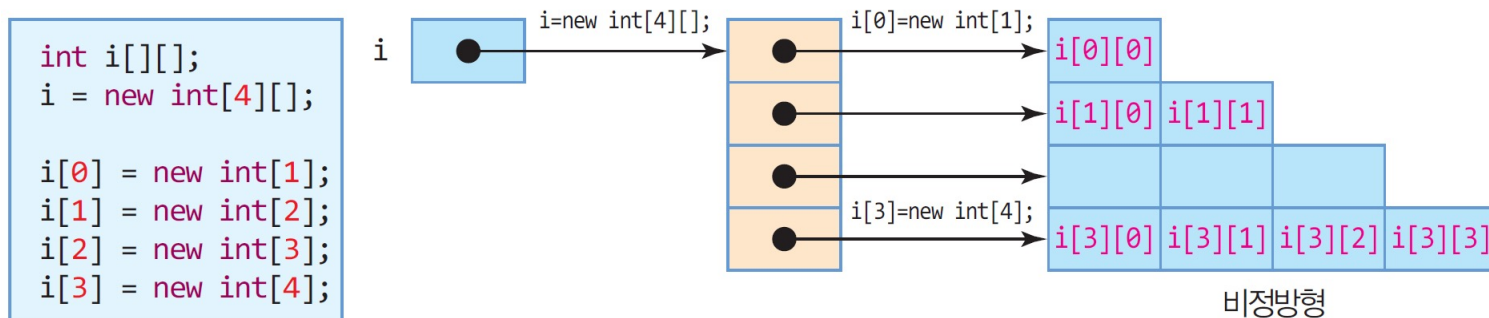
4년 전체 평점 평균은 3.725

비정방형 배열

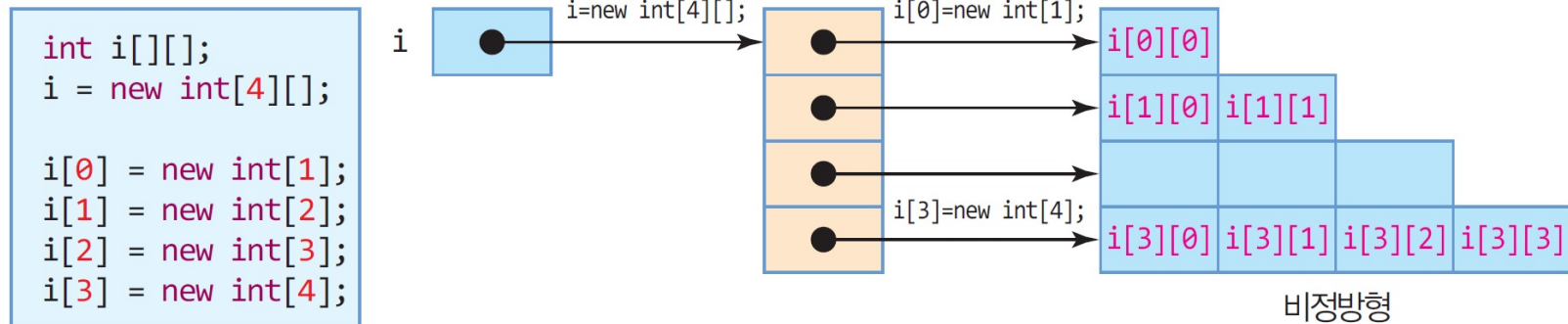
- 정방형 배열
 - 각 행의 열의 개수가 같은 배열



- 비정방형 배열
 - 각 행의 열의 개수가 다른 배열
 - 비정방형 배열의 생성



비정방형 배열의 length



• 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 : 비정방형 배열의 생성과 접근

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
```