



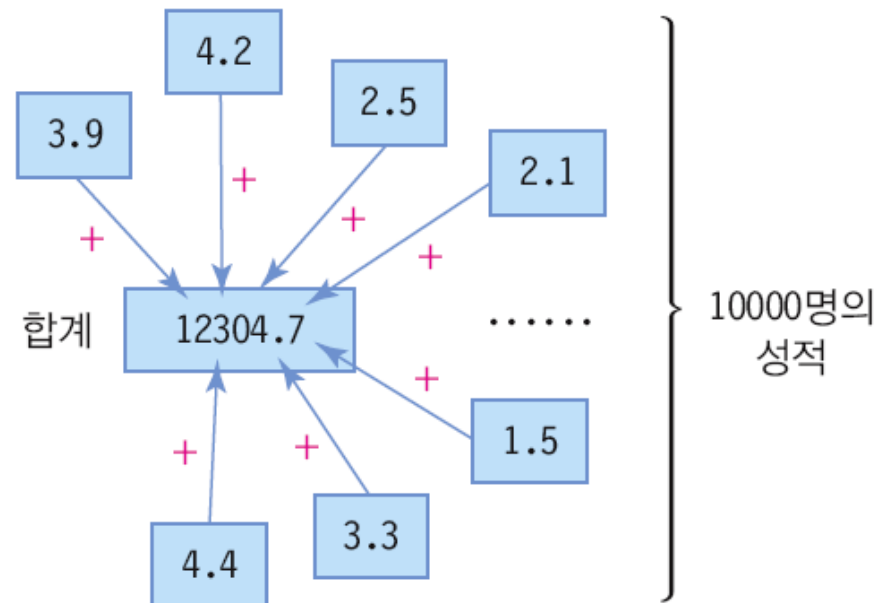
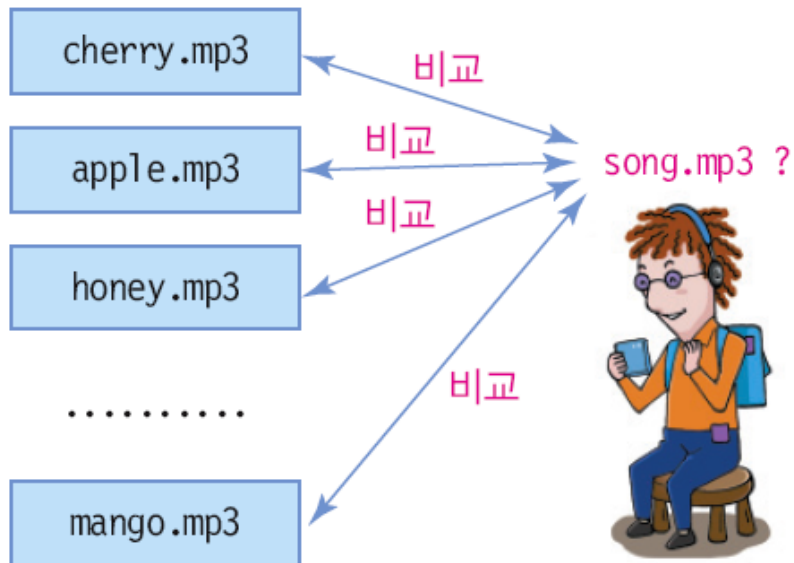
반복문과 배열 그리고 예외 처리

# 반복문의 특징

2

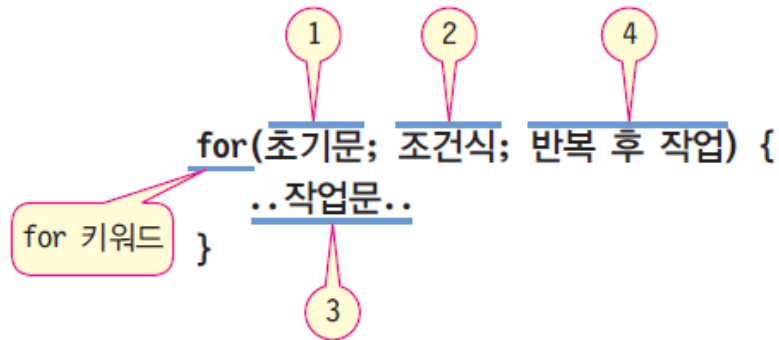
## □ 자바 반복문의 종류

- ▣ for 문
- ▣ while 문
- ▣ do while 문



# for 문의 구성

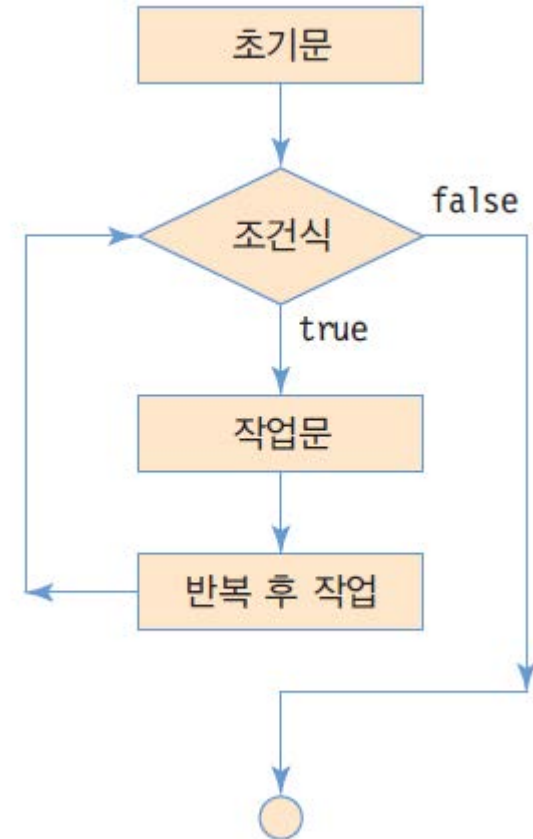
3



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

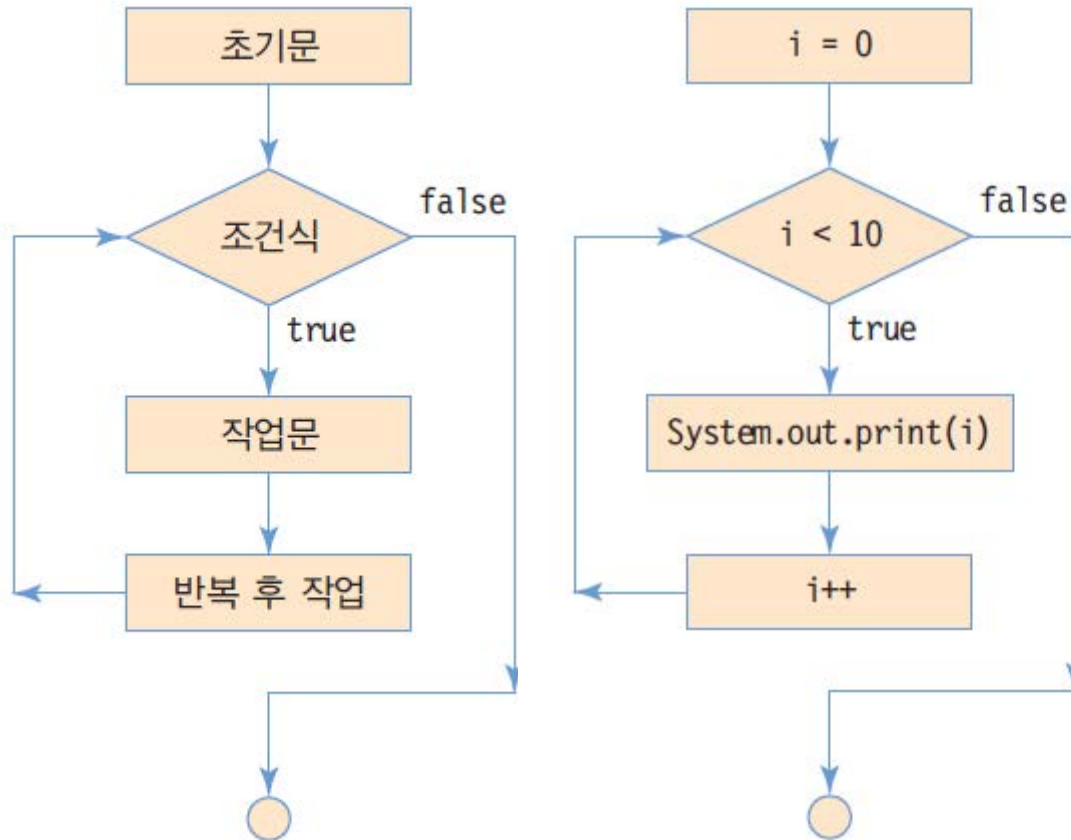
- 2 • 논리형 변수나 논리 연산만 가능
- 반복 조건이 true이면 반복 계속, false이면 반복 종료
- 반복 조건이 true 상수인 경우, 무한 반복
- 반복 조건이 비어 있으면 true로 간주

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



# for문의 실행 과정을 나타내는 순서도

4



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

# for문의 예시

5

- 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

6

```
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

7

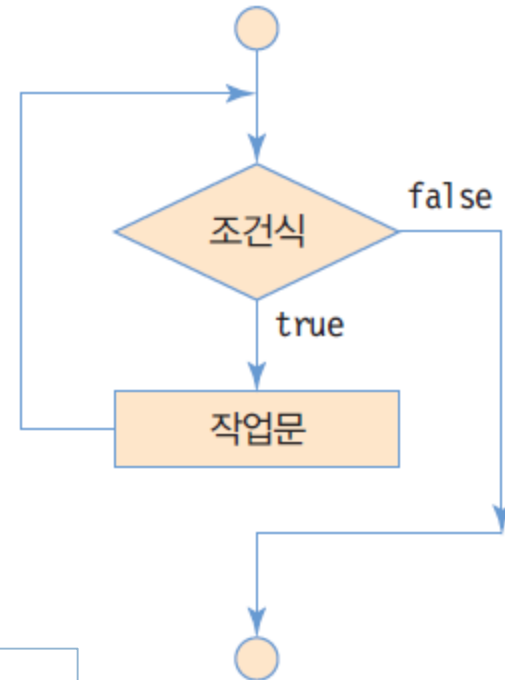
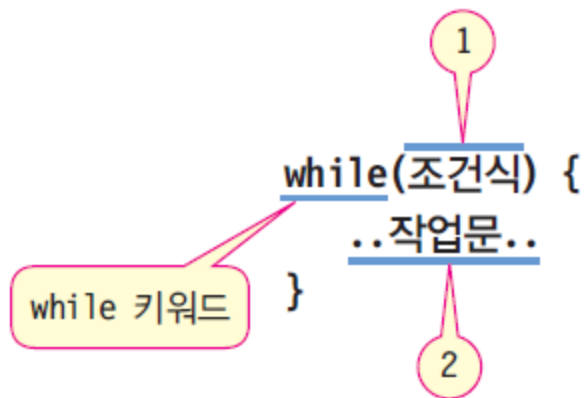
for문을 이용하여 1부터 10까지 덧셈을 표시하고 합을 구하시오.

```
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

8

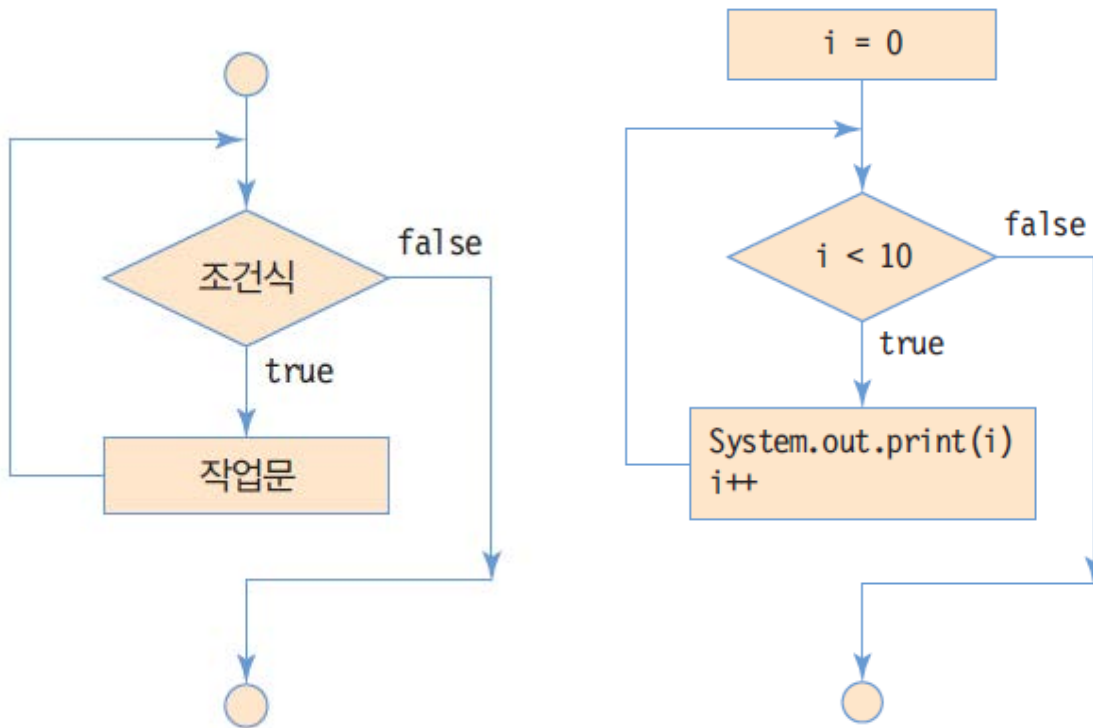


- 반복 조건이 true이면 반복, false이면 반복 종료
- 반복 조건이 없으면 컴파일 오류
- 처음부터 반복조건을 통과한 후 작업문 수행



# while문의 실행 과정을 나타내는 순서도

9



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

## 예제 3-2 : 입력된 수의 평균 구하기

10

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.");
    }
}
```

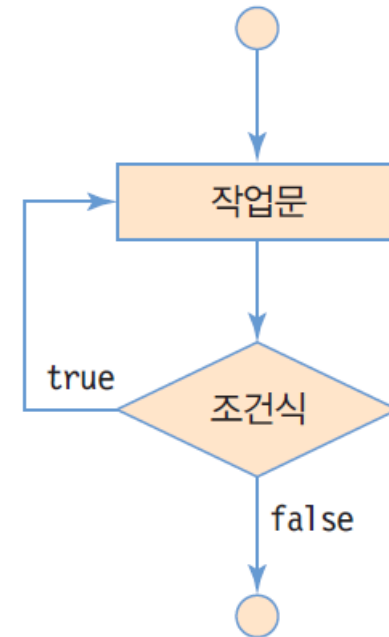
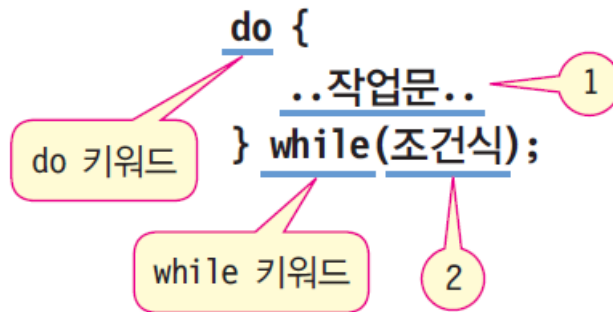
10  
20  
30  
40  
0

마지막 입력을 뜻함

입력된 수의 개수는 4개이며 평균은 25.0입니다.

# do-while 문의 구성

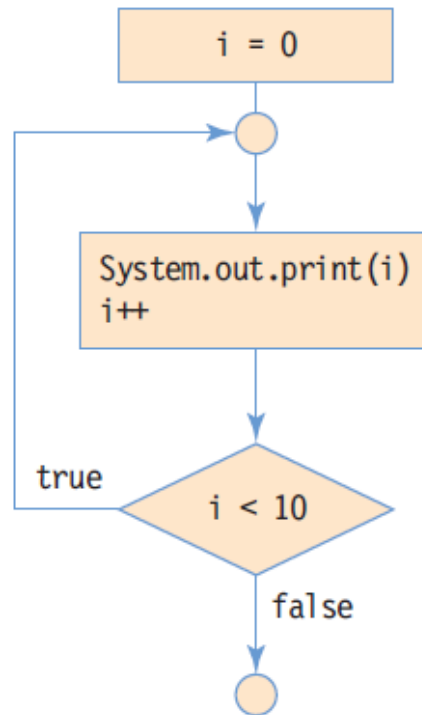
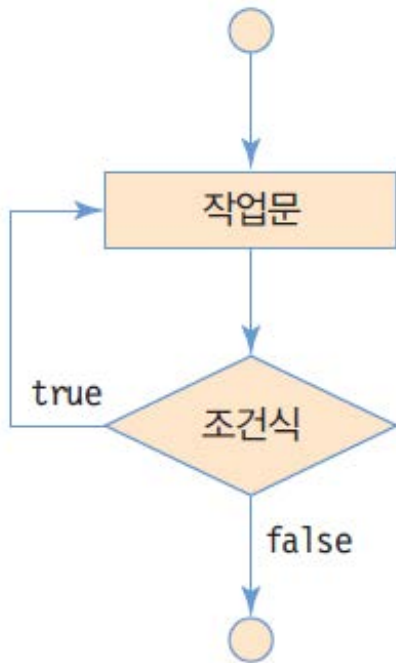
11



- ①
  - 무조건 최소 한번은 실행
- ②
  - 반복 조건이 true이면 반복, false이면 반복 종료
  - 반복 조건이 없으며 컴파일 오류

# do-while문의 실행 과정을 나타내는 순서도

12



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

## 예제 3-3 : a-z까지 출력

13

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

14

## □ 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

15

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

16

## □ 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

17

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

18

## □ 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

19

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

20

## □ 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?

21

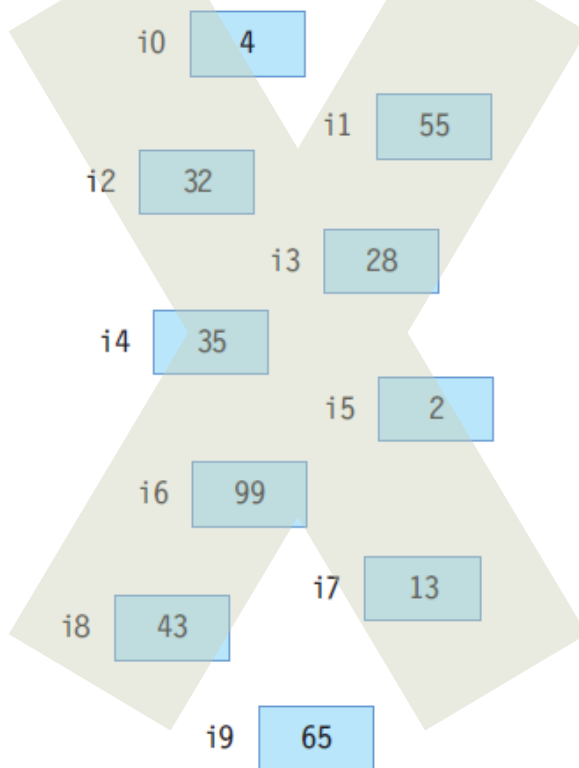
- 배열(array)
  - ▣ 인덱스와 인덱스에 대응하는 데이터들로 이루어진 자료 구조
    - 배열을 이용하면 한 번에 많은 메모리 공간 선언 가능
  - ▣ 배열은 같은 타입의 데이터들이 순차적으로 저장되는 공간
    - 원소 데이터들이 순차적으로 저장됨
    - 인덱스를 이용하여 원소 데이터 접근
    - 반복문을 이용하여 처리하기에 적합한 자료 구조
  - ▣ 배열 인덱스
    - 0부터 시작
    - 인덱스는 배열의 시작 위치에서부터 데이터가 있는 상대 위치

# 자바 배열의 필요성과 모양

22

(1) 10개의 정수형 변수를 선언하는 경우

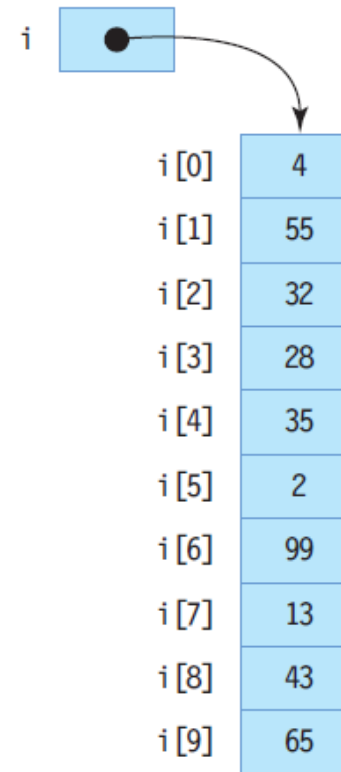
```
int i0, i1, i2, i3, i4, i5, i6, i7, i8, i9;
```



```
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

23

- 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
```

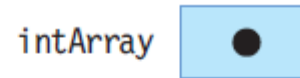
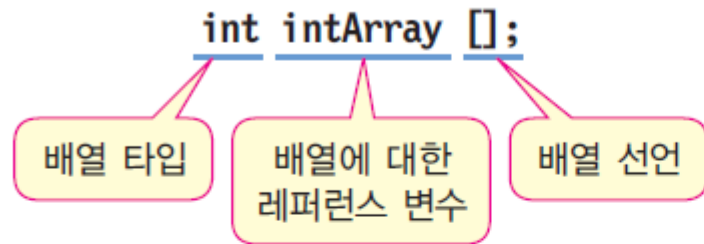
- 잘못된 배열 선언

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

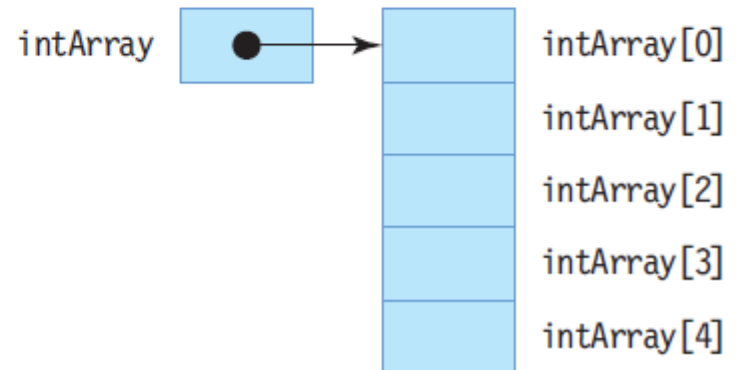
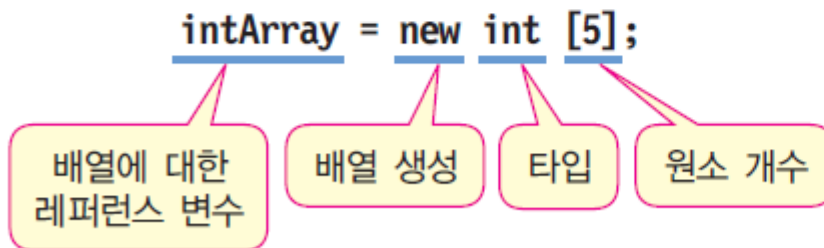
# 배열 선언과 생성

24

## (1) 배열에 대한 레퍼런스 변수 intArray 선언



## (2) 배열 생성

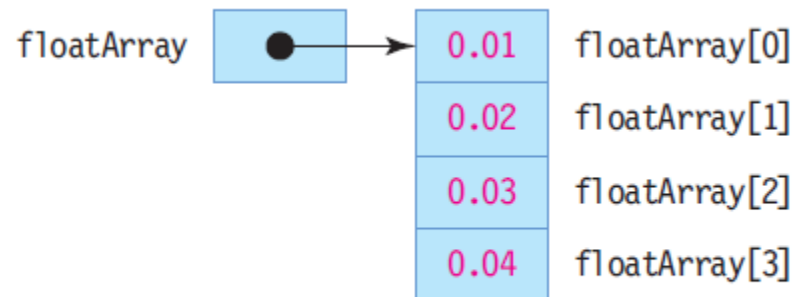
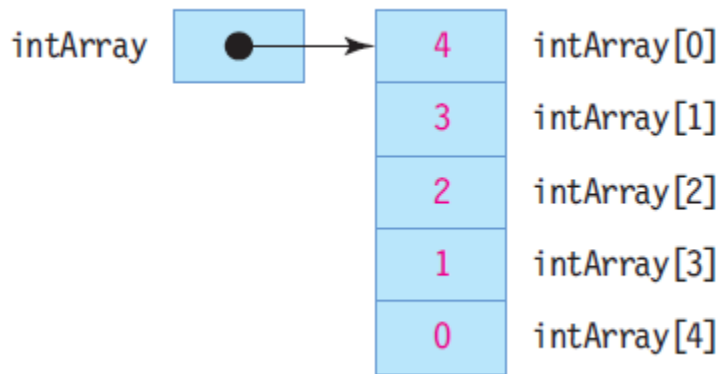




# 배열을 초기화하면서 생성한 결과

25

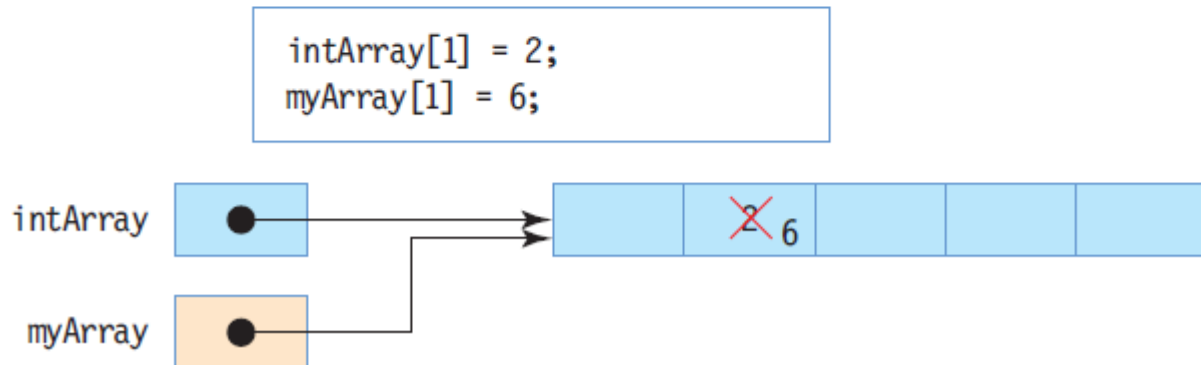
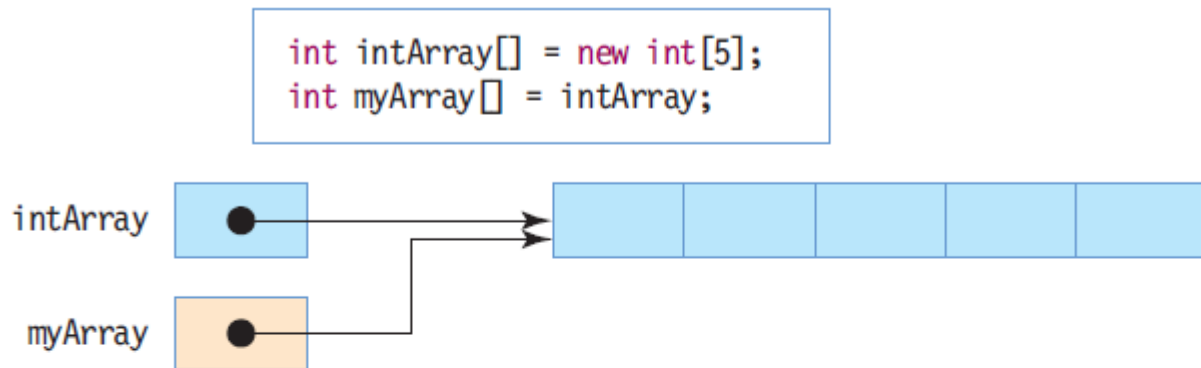
```
int intArray[] = {4, 3, 2, 1, 0};  
float floatArray[] = {0.01, 0.02, 0.03, 0.04};
```



# 배열 참조

26

- 생성된 하나의 배열을 다수의 레퍼런스가 참조 가능



# 배열 접근 방법

27

## □ 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

28

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

29

## □ 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

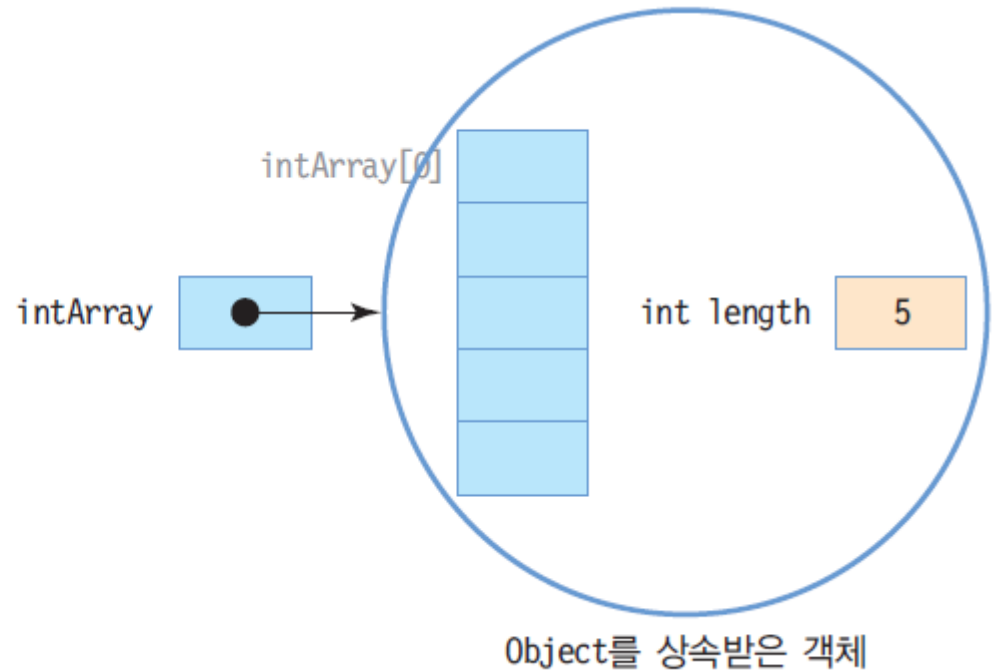
- ▣ 배열의 크기는 배열 레퍼런스 변수를 선언할 때 결정되지 않음
  - 배열의 크기는 배열 생성 시에 결정되며, 나중에 바꿀 수 없음
- ▣ 배열의 크기는 배열의 length 필드에 저장

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

# 배열은 객체로 관리

30

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



## 예제 3-8 : 배열 원소의 평균 구하기

31

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

32

## ▣ 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

33

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

34

## □ 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

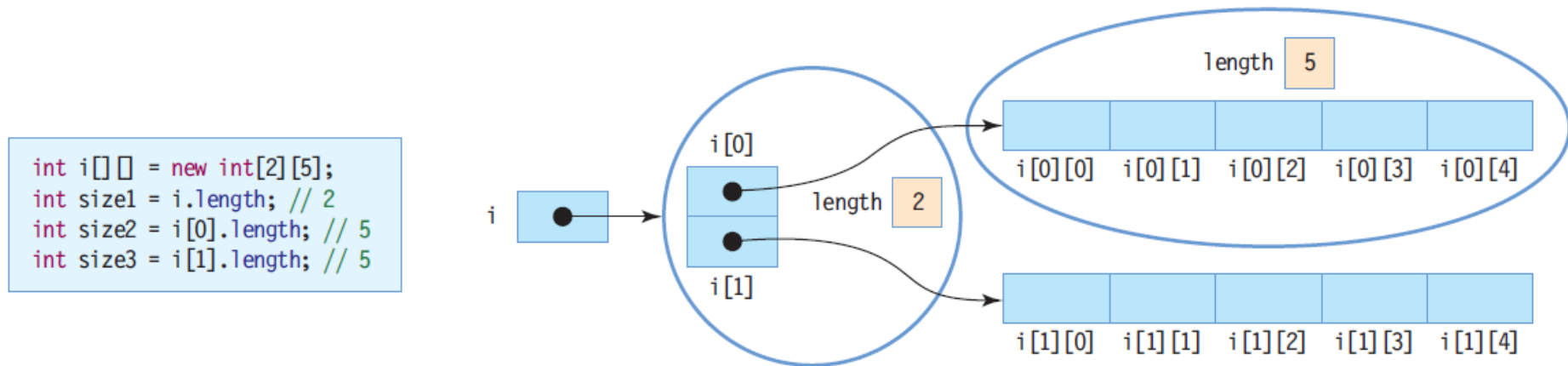
```
int    intArray[] = new int[2][5];  
char   charArray[] = new char[5][5];  
float  floatArray[] = new float[5][2];
```

## □ 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

35



## □ 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

36

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.");  
    }  
}
```

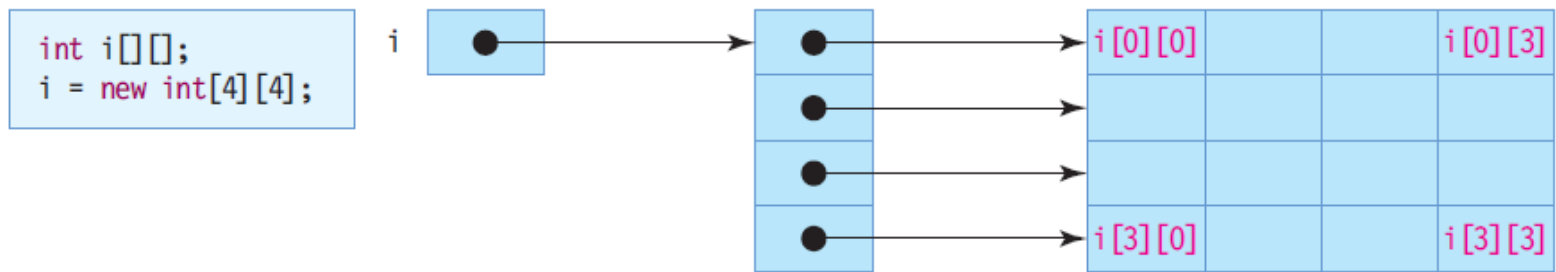
지난 3년간 매출 총액은 1380.0이며 연평균 매출은 460.0입니다.

# 비정방형 배열

37

## 정방형 배열

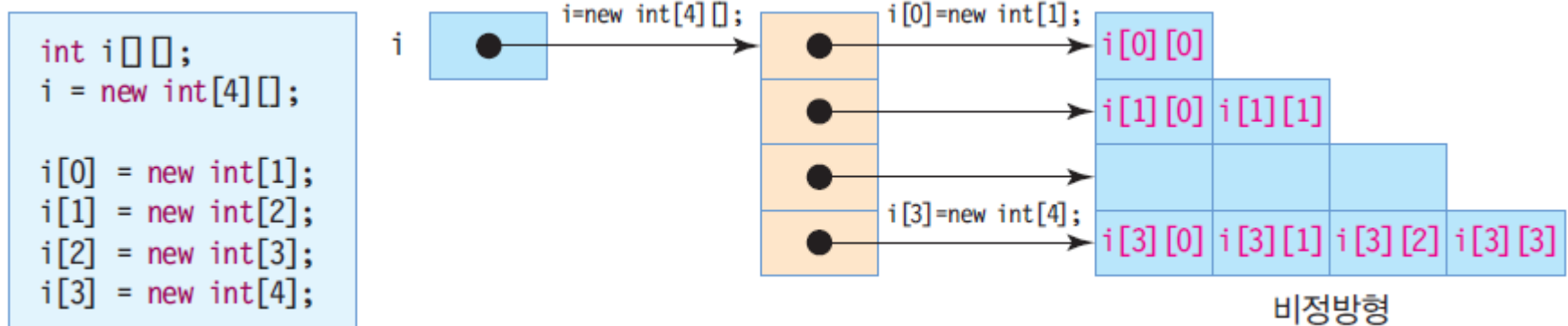
- 각 행의 열의 개수가 같은 배열



정방형

## 비정방형 배열

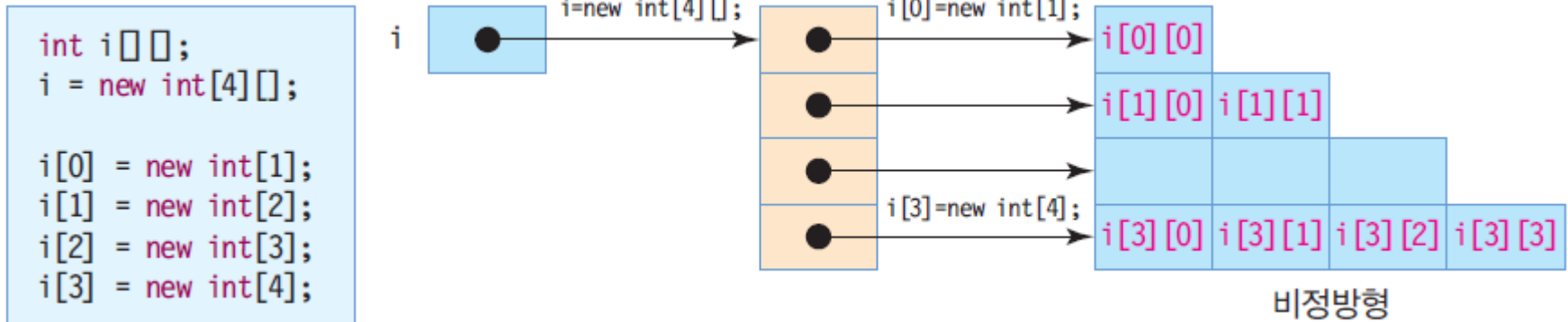
- 각 행의 열의 개수가 다른 배열
- 비정방형 배열의 생성



비정방형

# 비정방형 배열의 length

38



## □ 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

39

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

40

- ▣ 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

41

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

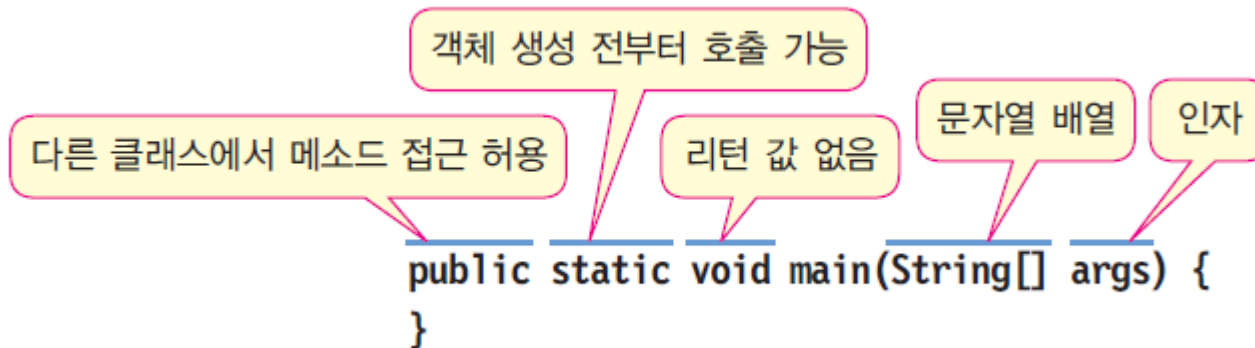
```
public class ReturnArray {  
    static int[] makeArray() {  
        int temp[] = new int[4];  
        for (int i=0;i<temp.length;i++)  
            temp[i] = i;  
        return temp;  
    }  
    public static void main (String[] args) {  
        int intArray [];  
        intArray = makeArray();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

0  
1  
2  
3

# main() 메소드

42

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

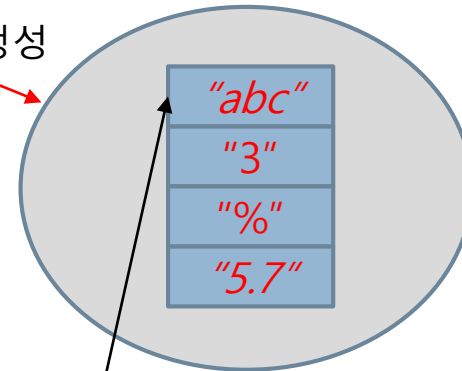


# main(string [] args) 메소드의 인자 전달

43

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

생성



class Hello

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

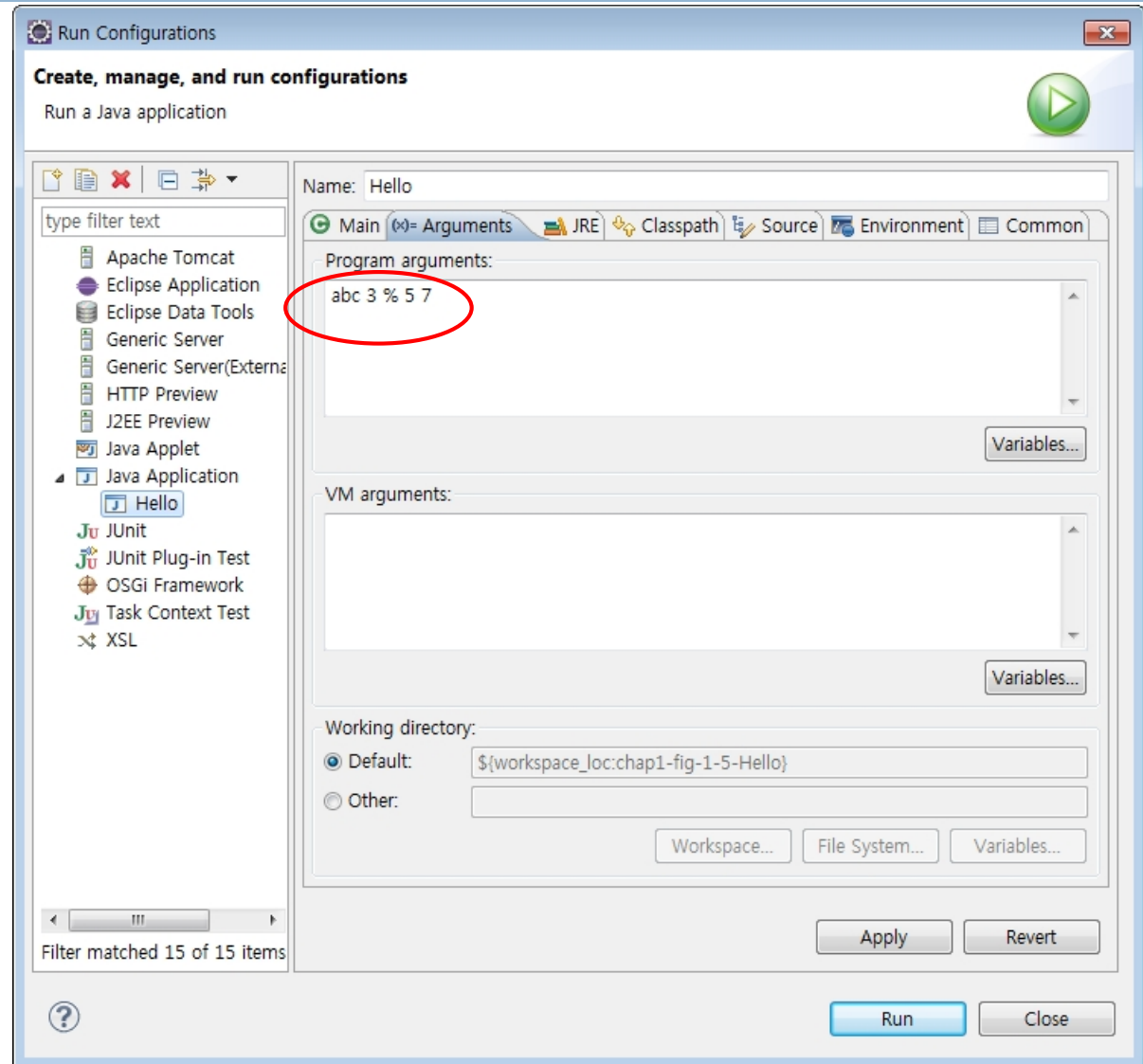
```
{  
    String a = args[0]; // a는 "abc"  
    String b = args[1]; // b는 "3"  
    String c = args[2]; // c는 "%"  
    String d = args[3]; // d는 "5.7"  
}
```

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

# 이클립스에서 main() 메소드의 인자전달

44

Run 메뉴의  
Run Configurations  
항목에서  
main() 메소드의  
인자 나열

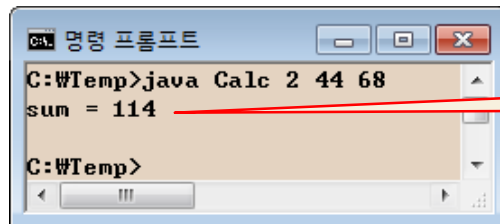


# main()의 인자 이용 예

45

```
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

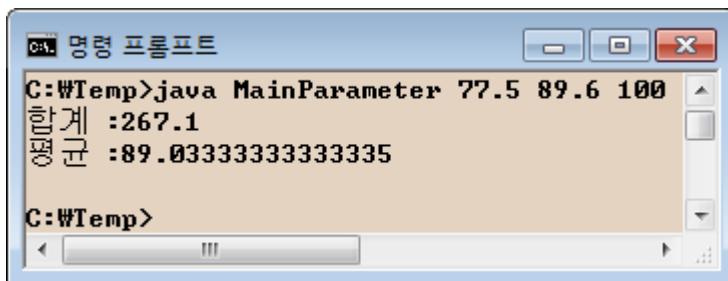
# 예제 3-13 : main()의 인자들을 받아서 평균값을 계산하는 예제

46

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



```
C:\Temp>java MainParameter 77.5 89.6 100  
합계 :267.1  
평균 :89.03333333333335  
  
C:\Temp>
```

# 자바의 예외 처리

47

## □ 예외(Exception)

- 실행 중 발생하는 에러는 컴파일러가 알 수 없음
- 자바에서는 실행 중 발생하는 에러를 예외로 처리
  - 응용프로그램에서 예외를 처리하지 않으면, 예외가 발생한 프로그램은 강제 종료

```
import java.util.Scanner;
public class ExceptionExample1 {
    public static void main (String[] args) {
        Scanner rd = new Scanner(System.in);
        int divisor = 0;
        int dividend = 0;

        System.out.print("나뉘는수를 입력하시오:");
        dividend = rd.nextInt();
        System.out.print("나눌수를 입력하시오:");
        divisor = rd.nextInt();
        System.out.println(dividend+"를 "+divisor+"로 나누면 몫은 "+dividend/divisor+"입니다.");
    }
}
```

divisor가 0이므로  
예외 발생

나뉘는수를 입력하시오:100

나눌수를 입력하시오:0

Exception in thread "main" java.lang.ArithmeticException: / by zero  
at ExceptionExample1.main(ExceptionExample1.java:12)

# try-catch-finally문

48

## □ 예외 처리문

- ▣ try-catch-finally문 사용
- ▣ finally 블록은 생략 가능

생략  
가능

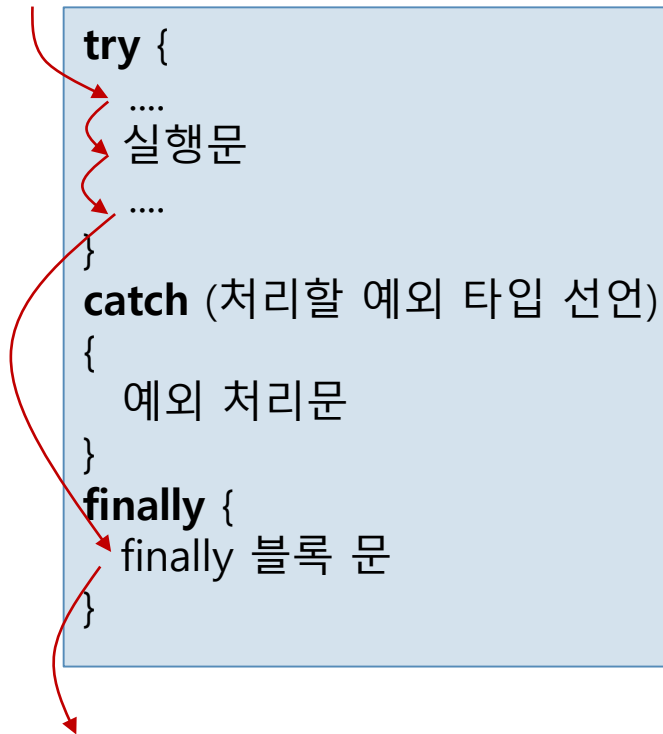
```
try {  
    예외가 발생할 가능성이 있는 실행문 (try 블록)  
}  
catch (처리할 예외 타입 선언) {  
    예외 처리문 (catch 블록)  
}  
finally {  
    예외 발생 여부와 상관없이 무조건 실행되는 문장  
    (finally 블록)  
}
```



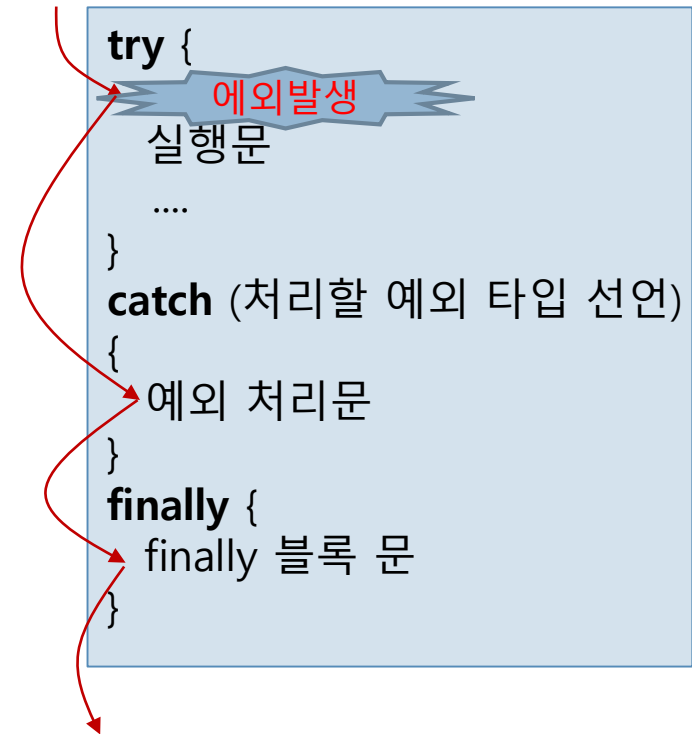
# 예외가 발생한 경우와 예외가 발생하지 않은 경우 제어의 흐름

49

try블록에서 예외가 발생하지 않은 정상적인 경우



try블록에서 예외가 발생한 경우



# 자주 발생하는 예외

50

예외 종류	예외 발생 경우
ArithmeticException	정수를 0으로 나눌 때 발생
NullPointerException	null 레퍼런스를 참조할 때 발생
ClassCastException	변환할 수 없는 타입으로 객체를 변환할 때 발생
OutOfMemoryError	메모리가 부족한 경우 발생
ArrayIndexOutOfBoundsException	배열의 범위를 벗어난 접근 시 발생
IllegalArgumentException	잘못된 인자 전달 시 발생
IOException	입출력 동작 실패 또는 인터럽트 시 발생
NumberFormatException	문자열이 나타내는 숫자와 일치하지 않는 타입의 숫자로 변환 시 발생

# 예제 3-14 : ArithmeticException 예외 처리

51

try-catch문을 이용하여 정수를 0으로 나누려고 할 때 "0으로 나눌 수 없습니다."라는 경고 메시지를 출력하도록 프로그램을 작성하시오.

```
import java.util.Scanner;
public class ExceptionExample2 {
    public static void main (String[] args) {
        Scanner rd = new Scanner(System.in);
        int divisor = 0;
        int dividend = 0;

        System.out.print("나뉘수를 입력하시오:");
        dividend = rd.nextInt();
        System.out.print("나눗수를 입력하시오:");
        divisor = rd.nextInt();
        try {
            System.out.println(dividend+"를 "+divisor+"로 나누면 몫은 "+ dividend/divisor+"입니다.");
        } catch (ArithmeticException e) {
            System.out.println("0으로 나눌 수 없습니다.");
        }
    }
}
```

ArithmeticException  
예외 발생

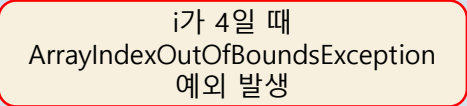
나뉘수를 입력하시오:100  
나눗수를 입력하시오:0  
0으로 나눌 수 없습니다.

# 예제 3-15 : 범위를 벗어난 배열의 접근

52

배열의 인덱스가 범위를 벗어날 때 발생하는  
`ArrayIndexOutOfBoundsException`을 처리하는 프로그램을 작성하시오.

```
public class ArrayException {  
    public static void main (String[] args) {  
        int[] intArray = new int[5];  
        intArray[0] = 0;  
        try {  
            for (int i = 0; i < 5; i++) {  
                intArray[i+1] = i+1 + intArray[i];  
                System.out.println("intArray["+i+"]"+"="+intArray[i]);  
            }  
        }  
        catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("배열의 인덱스가 범위를 벗어났습니다.");  
        }  
    }  
}
```



```
intArray[0]=0  
intArray[1]=1  
intArray[2]=3  
intArray[3]=6  
배열의 인덱스가 범위를 벗어났습니다.
```

# 예제 3-16 : 정수가 아닌 문자열을 정수로 변환할 때 예외 발생

53

문자열을 정수로 변환할 때 발생하는 `NumberFormatException`을 처리하는 프로그램을 작성하라.

```
public class NumException {  
    public static void main (String[] args) {  
        String[] stringNumber = {"23", "12", "998", "3.141592"};  
        try {  
            for (int i = 0; i < stringNumber.length; i++) {  
                int j = Integer.parseInt(stringNumber[i]);  
                System.out.println("숫자로 변환된 값은 " + j);  
            }  
        }  
        catch (NumberFormatException e) {  
            System.out.println("정수로 변환할 수 없습니다.");  
        }  
    }  
}
```

"3.141592"를 정수로 변환할 때  
`NumberFormatException`  
예외 발생

숫자로 변환된 값은 23  
숫자로 변환된 값은 12  
숫자로 변환된 값은 998  
정수로 변환할 수 없습니다.