



# 03

반복문과 배열 그리고 예외 처리

## 학습 목표

1. 자바의 반복문(for, while, do-while) 이해, 작성
2. continue문과 break문 활용
3. 자바의 배열 선언 및 활용
4. 배열을 리턴하는 메소드 작성
5. 예외 개념과 자바에서의 예외 처리

# 반복문

3

- 자바 반복문 - for 문, while 문, do-while 문
  - ▣ for 문 - 가장 많이 사용하는 반복문

1  
for(초기문; 조건식; 반복 후 작업) {  
    .. 작업문 ..  
}

2  
3  
4

```
// 0에서 9까지 출력  
for(int i=0; i<10; i++) {  
    System.out.print(i);  
}
```

0123456789

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

반복후 작업문에 콤마로 분리하여  
2 문장 작성가능

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

for 문안에서만 사용되는  
변수 i 선언 가능

```
for(초기문; true; 반복 후 작업) { // 무한반복  
    .....  
}
```

```
for(초기문; ; 반복 후 작업) { // 무한 반복  
    .....  
}
```

## 예제 3-1 : for 문을 이용하여 1부터 10까지 합 출력하기

4

for 문을 이용하여 1부터 10까지 덧셈으로 표시하고 합을 출력하라.

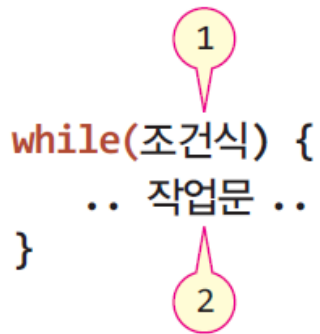
```
public class ForSample {  
    public static void main(String[] args) {  
        int i, sum=0;  
  
        for(i=1; i<=10; i++) { // 1~10까지 반복  
            sum += i;  
            System.out.print(i); // 더하는 수 출력  
  
            if(i<=9) // 1~9까지는 '+' 출력  
                System.out.print("+");  
            else { // i가 10인 경우  
                System.out.print("="); // '=' 출력하고  
                System.out.print(sum); // 덧셈 결과 출력  
            }  
        }  
    }  
}
```

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

# while 문

5

## □ while 문의 구성과 코드 사례



A diagram illustrating the structure of a while loop. It shows the text `while(조건식) {` followed by `.. 작업문 ..` and then a closing brace `}`. A yellow circle with the number '1' is positioned above the opening parenthesis of the condition, and a yellow circle with the number '2' is positioned below the closing brace of the loop body.

```
while(조건식) {  
    .. 작업문 ..  
}
```

```
int i=0;  
while(i<10) { // 0에서 9까지 출력  
    System.out.print(i);  
    i++;  
}
```

0123456789

- 조건식이 '참'인 동안 반복 실행

## 예제 3-2 : while 문을 이용하여 입력된 정수의 평균 구하기

6

while문을 이용하여 정수를 여러 개 입력 받고 평균을 출력하라.  
0이 입력되면 입력을 종료한다.

```
import java.util.Scanner;
public class WhileSample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int count=0, n=0;
        double sum=0;

        System.out.println("정수를 입력하고 마지막에 0을 입력하세요.");
        while((n = scanner.nextInt()) != 0) { // 0이 입력되면 while 문 벗어남
            sum = sum + n;
            count++;
        }
        System.out.print("수의 개수는 " + count + "개이며 ");
        System.out.println("평균은 " + sum/count + "입니다.");

        scanner.close();
    }
}
```

정수를 입력하고 마지막에 0을 입력하세요.

10 30 -20 40 0

수의 개수는 4개이며 평균은 15.0입니다.

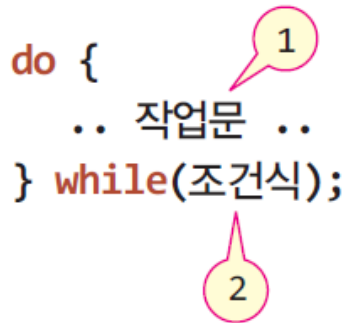
0은 마지막 입력을 뜻함

# do-while 문

7

## □ do-while 문의 구성과 코드 사례

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



```
int i=0;  
do { // 0에서 9까지 출력  
    System.out.print(i);  
    i++;  
} while(i<10);
```

0123456789

- 조건식이 '참'인 동안 반복 실행
- 작업문은 한 번 반드시 실행

## 예제 3-3 : do-while 문을 이용하여 'a'에서 'z'까지 출력하기

8

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

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

abcdefghijklmnopqrstuvwxyz



# 중첩 반복

9

## □ 중첩 반복

- ▣ 반복문이 다른 반복문을 내포하는 구조

```
for(i=0; i<100; i++) { // 100개 학교 성적을 더한다.  
    for(j=0; j<10000; j++) { // 10000명의 학생 성적을 더한다.  
        ....  
        ....  
    }  
    ....  
}
```

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

## 예제 3-4 : 2중 중첩을 이용한 구구단 출력하기

10

2중 중첩된 for문을 이용하여 구구단을 출력하는 프로그램을 작성하라.

```
public class NestedLoop {  
    public static void main(String[] args) {  
  
        for(int i=1; i<10; i++) { // 단에 대한 반복. 1단에서 9단  
            for(int j=1; j<10; j++) { // 각 단의 곱셈  
                System.out.print(i + "*" + j + "=" + i*j); // 구구셈 출력  
                System.out.print('\t'); // 하나씩 탭으로 띄기  
            }  
            System.out.println(); // 한 단이 끝나면 다음 줄로 커서 이동  
        }  
    }  
}
```

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

11

## □ continue 문

- 반복문을 빠져 나가지 않고, 다음 반복으로 제어 변경
- 반복문에서 continue; 문에 의한 분기

```
for(초기문; 조건식; 반복 후 작업) {  
    .....  
    continue;  
    .....  
}
```

분기

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

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

## 예제 3-5 : continue 문을 이용하여 양수 합 구하기

12

5개의 정수를 입력 받고 양수 합을 구하여 출력하는 프로그램을 작성하라.

```
import java.util.Scanner;
public class ContinueExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("정수를 5개 입력하세요.");
        int sum=0;

        for(int i=0; i<5; i++) {
            int n=scanner.nextInt();
            if(n<=0) continue; // 0이나 음수인 경우 더하지 않고 다음 반복으로 진행
            else sum += n; // 양수인 경우 덧셈
        }
        System.out.println("양수의 합은 " + sum);

        scanner.close();
    }
}
```

정수를 5개 입력하세요.

5 -2 6 10 -4

양수의 합은 21

# break문

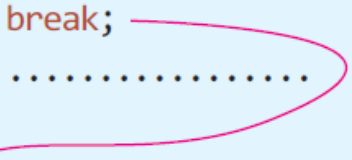
13

## □ break 문

### ▣ 반복문 하나를 즉시 벗어갈 때 사용

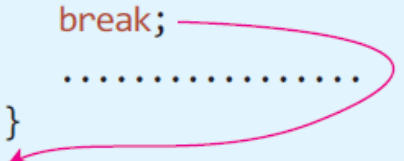
- 하나의 반복문만 벗어남
- 중첩 반복의 경우 안쪽 반복문의 break 문이 실행되면 안쪽 반복문만 벗어남

```
for(초기문; 조건식; 반복 후 작업) {  
    .....  
    break;  
    .....  
}  
.....
```



(a) 하나의 반복문을 벗어나는 경우

```
for(초기문; 조건식; 반복 후 작업) {  
    while(조건식) {  
        .....  
        break;  
        .....  
    }  
    .....  
}  
.....
```



(b) 중첩 반복에서 안쪽 반복문만 벗어나는 경우

## 예제 3-6 : break 문을 이용하여 while 문 벗어나기

14

"exit"이 입력되면 while 문을 벗어나도록 break 문을 활용하는 프로그램을 작성하라.

```
import java.util.Scanner;
public class BreakExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("exit을 입력하면 종료합니다.");

        while(true) {
            System.out.print("> >");
            String text = scanner.nextLine();
            if(text.equals("exit")) // "exit"이 입력되면 반복 종료
                break; // while 문을 벗어남
        }

        System.out.println("종료합니다...");
        scanner.close();
    }
}
```

exit을 입력하면 종료합니다.

>>edit

>>exit

종료합니다...

## □ 배열(array)

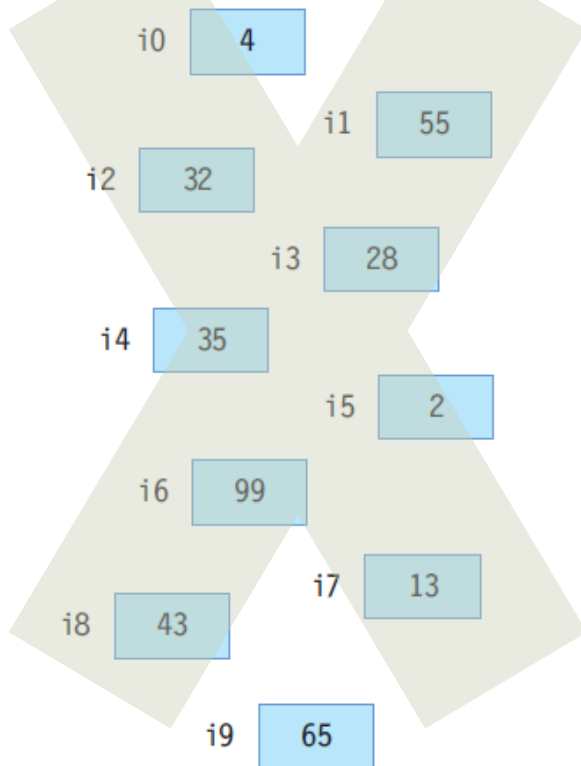
- ▣ 인덱스와 인덱스에 대응하는 데이터들로 이루어진 자료 구조
  - 배열을 이용하면 한 번에 많은 메모리 공간 선언 가능
- ▣ 배열은 같은 타입의 데이터들이 순차적으로 저장되는 공간
  - 원소 데이터들이 순차적으로 저장됨
  - 인덱스를 이용하여 원소 데이터 접근
  - 반복문을 이용하여 처리하기에 적합한 자료 구조
- ▣ 배열 인덱스
  - 0부터 시작
  - 인덱스는 배열의 시작 위치에서부터 데이터가 있는 상대 위치

# 자바 배열의 필요성과 모양

16

(1) 10개의 정수형 변수를 선언하는 경우

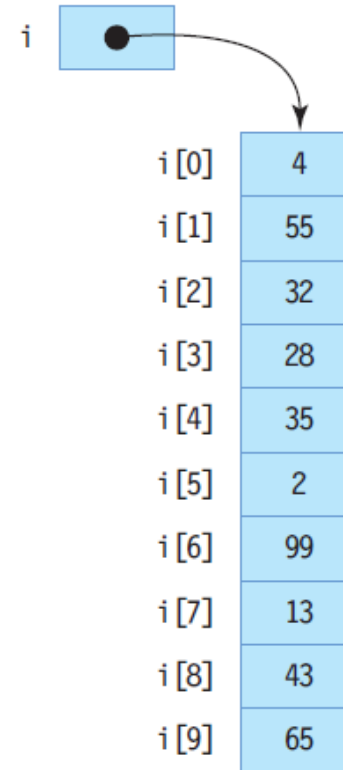
```
int i0, i1, i2, i3, i4, i5, i6, i7, i8, i9;
```



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

(2) 10개의 정수로 구성된 배열을 선언하는 경우

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



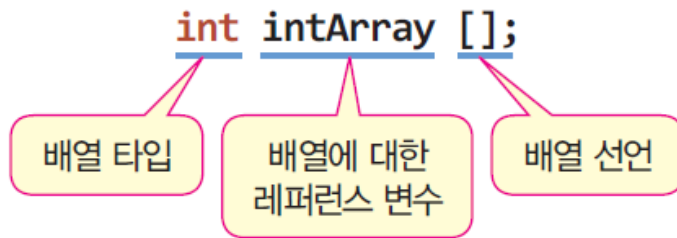
```
for(sum=0, n=0; n<10; n++)  
    sum += i[n];
```



# 배열 선언과 생성

17

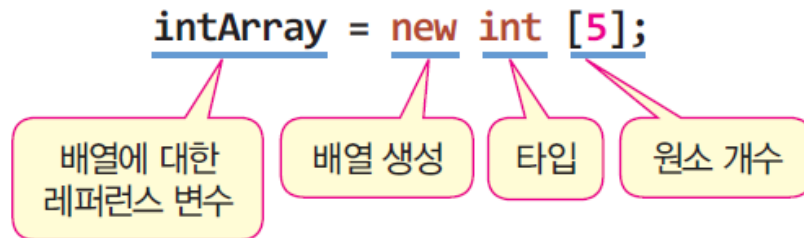
(1) 배열에 대한 레퍼런스 변수 `intArray` 선언



`intArray`



(2) 배열 생성



`intArray`



`intArray[0]`

`intArray[1]`

`intArray[2]`

`intArray[3]`

`intArray[4]`

# 배열 선언 및 생성 디테일

18

- 배열 선언과 배열 생성의 두 단계 필요
- 배열 선언
  - 배열의 이름 선언(배열 레퍼런스 변수 선언)

```
int intArray []; 또는  
int[] intArray;
```



```
int intArray [5]; // 크기 지정 안됨
```

- 배열 생성
  - 배열 공간 할당 받는 과정

```
intArray = new int[5]; 또는  
int intArray[] = new int[5]; // 선언과 동시에 배열 생성
```

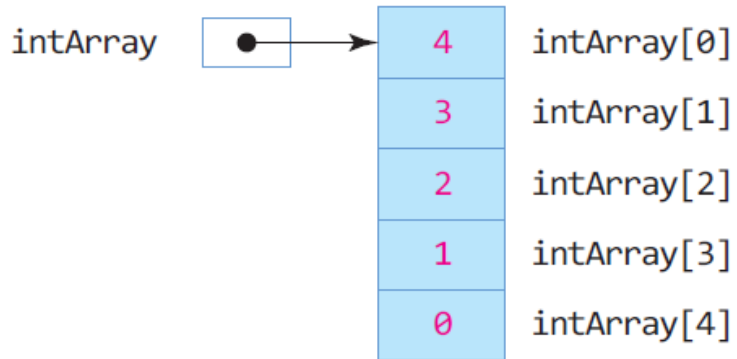
- 배열 초기화
  - 배열 생성과 값 초기화

```
int intArray[] = {4, 3, 2, 1, 0}; // 5개의 정수 배열 생성 및 값 초기화  
double doubleArray[] = {0.01, 0.02, 0.03, 0.04}; // 5개의 실수 배열 생성 및 값 초기화
```

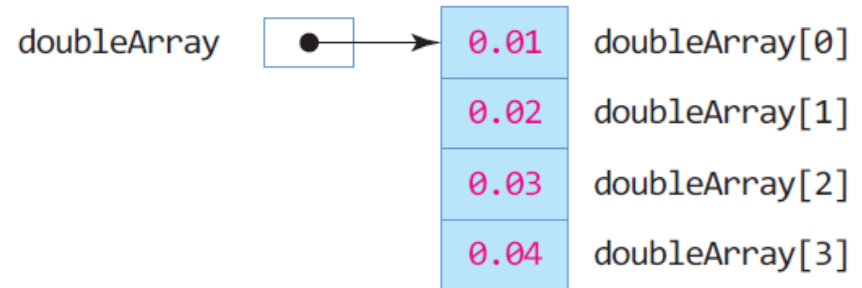
# 배열을 초기화하면서 생성한 결과

19

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



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



# 배열 인덱스와 배열 원소 접근

20

## □ 배열 인덱스

- ▣ 배열의 인덱스는 0 ~ (배열 크기 - 1)

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

- ▣ 인덱스를 잘못 사용한 경우

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

- ▣ 반드시 배열 생성 후 접근

```
int intArray []; // 레퍼런스만 선언함
```

```
오류 intArray[1] = 8; // 오류. 배열이 생성되어 있지 않음
```

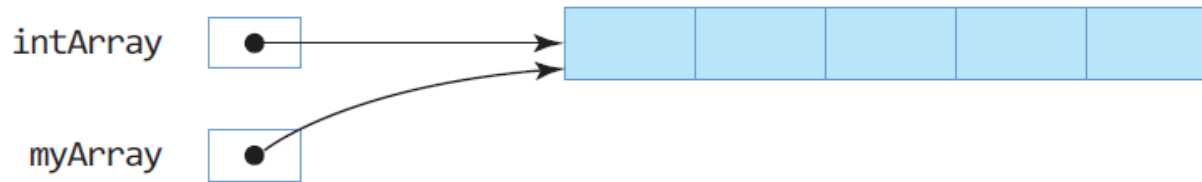
# 레퍼런스 치환과 배열 공유

21

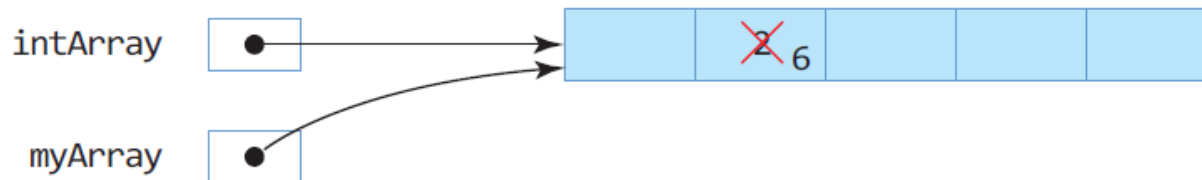
- 레퍼런스 치환으로 두 레퍼런스가 하나의 배열 공유

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

레퍼런스 치환으로 배열 공유



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



## 예제 3-7 : 배열 선언 및 생성

22

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[];
        intArray = new int[5];
        int max=0; // 현재 가장 큰 수
        System.out.println("양수 5개를 입력하세요.");

        for(int i=0; i<5; i++) {
            intArray[i] = scanner.nextInt();    // 입력 받은 정수를 배열에 저장
            if(intArray[i] > max)
                max = intArray[i]; // max 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

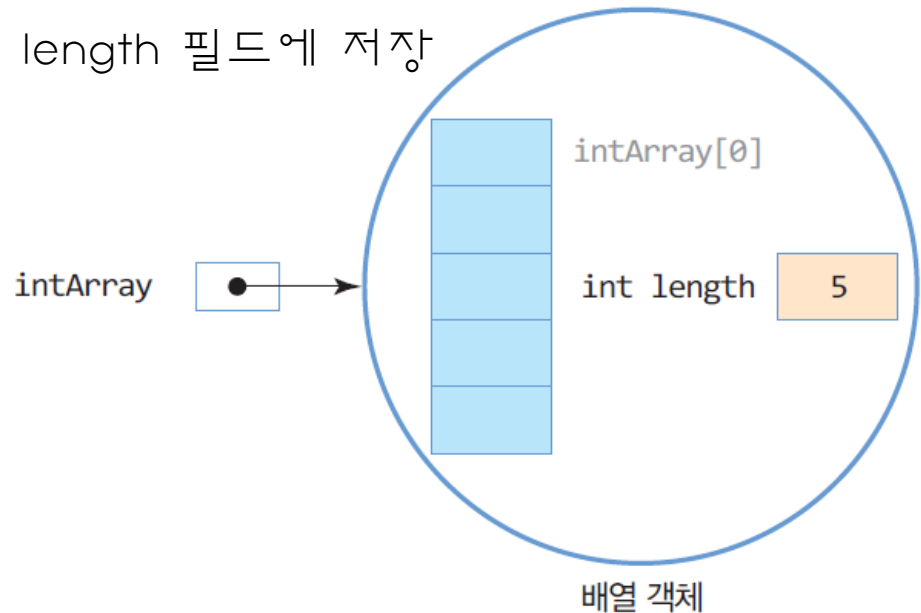
23

## □ 자바의 배열은 객체로 처리

### ▣ 배열 객체의 length 필드

- 배열의 크기는 배열 객체의 length 필드에 저장

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



- length 필드를 이용하여 배열의 모든 값을 출력하는 사례

```
for(int i=0; i<intArray.length; i++) // intArray 배열 크기만큼 루프를 돈다.  
    System.out.println(intArray[i]);
```

# 함수 호출시 C/C++와 자바의 배열 전달 비교

24

C/C++ 경우,  
배열과 크기를 각각 전달 받음

```
int sum(int x[], int size) {  
    int n, s=0;  
    for(n=0; n<size; n++)  
        s += x[n];  
    return s;  
}
```

```
int a[] = {1,2,3,4,5};  
int n = sum(a, 5);
```

자바 경우,  
배열만 전달받음

```
int sum(int x[]) {  
    int n, s=0;  
    for(n=0; n<x.length; n++)  
        s += x[n];  
    return s;  
}
```

```
int a[] = {1,2,3,4,5};  
int n = sum(a);
```

자바가 C/C++에 비해  
배열을 다루기 10배  
편한 구조임



## 예제 3-8 : 배열의 length 필드 활용

25

배열의 length 필드를 이용하여 배열 크기만큼 정수를 입력 받고 평균을 출력하라.

```
import java.util.Scanner;
public class ArrayLength {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("5개의 정수를 입력하세요.");
        int intArray[] = new int[5];

        double sum=0.0;
        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 " + sum/intArray.length);
        scanner.close();
    }
}
```

5개의 정수를 입력하세요.

2 3 4 5 9

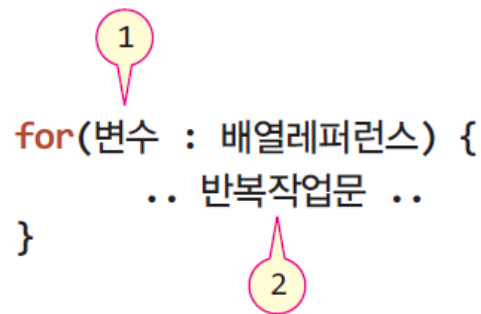
평균은 4.6.

# 배열과 for-each 문

26

## □ for-each 문

- ▣ 배열이나 나열(enumeration)의 원소를 순차 접근하는데 유용한 for 문

  
**for**(변수 : 배열레퍼런스) {  
 .. 반복작업문 ..  
}

## ▣ for-each 문으로 정수 배열의 합을 구하는 사례

```
int [] n = { 1,2,3,4,5 };  
int sum = 0;  
for (int k : n) {  
    sum += k;  
}
```

반복될 때마다 k는 n[0], n[1], ..., n[4]로 번갈아 설정

for 문으로 구성하면 다음과 같다.

```
for(int i=0; i<n.length; i++) {  
    int k = n[i];  
    sum += k;  
}
```

## 예제 3-9 for-each 문 활용

27

for-each 문을 활용하여 int [] 배열의 합을 구하고, String [] 배열의 문자열을 출력하는 사례를 보인다.

```
public class foreachEx {  
    public static void main(String[] args) {  
        int [] n = { 1,2,3,4,5 };  
        int sum=0;  
        for(int k : n) { // k는 n[0], n[1], ..., n[4]로 반복  
            System.out.print(k + " "); // 반복되는 k 값 출력  
            sum += k;  
        }  
        System.out.println("합은 " + sum);  
  
        String f[] = { "사과", "배", "바나나", "체리", "딸기", "포도" };  
        for(String s : f) // s는 f[0], f[1], ..., f[5]로 반복  
            System.out.print(s + " ");  
    }  
}
```

1 2 3 4 5 합은 15  
사과 배 바나나 체리 딸기 포도

# 2차원 배열

28

## ▣ 2차원 배열 선언

```
int intArray[][];    또는   int[][] intArray;
```

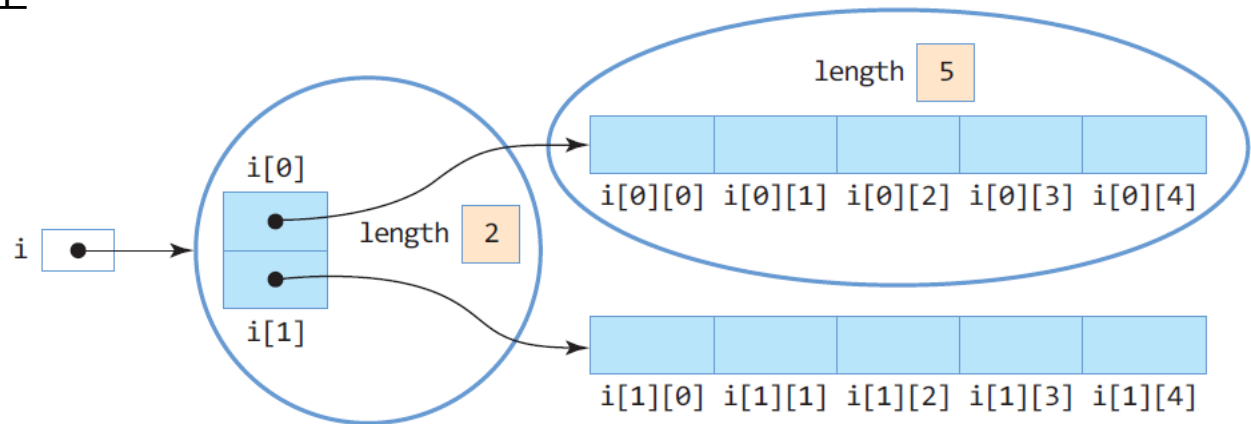
## ▣ 2차원 배열 생성

```
intArray = new int[2][5];
```

```
int intArray[] = new int[2][5]; // 배열 선언과 생성 동시
```

## ▣ 2차원 배열의 구조

```
int i[][] = new int[2][5];  
int size1 = i.length; // 2  
int size2 = i[0].length; // 5  
int size3 = i[1].length; // 5
```



## ▣ 2차원 배열의 length 필드

- `i.length` -> 2차원 배열의 행의 개수로, 2
- `i[n].length` -> n번째 행의 열의 개수
- `i[1].length` -> 1번째 행의 열의 개수, 5

# 2차원 배열의 초기화

29

## □ 배열 선언과 동시에 초기화

```
int intArray[][] = { { 0, 1, 2},  
                    { 3, 4, 5},  
                    { 6, 7, 8} }; // 3x3 배열 생성
```

```
char charArray[][] = { {'a', 'b', 'c'}, {'d', 'e', 'f'} }; // 2x3 배열 생성
```

```
double doubleArray[][] = { {0.01, 0.02}, {0.03, 0.04} }; // 2x2 배열 생성
```

# 예제 3-10 : 2차원 배열 생성 및 활용하기

30

2차원 배열에 학년별로 1, 2학기 성적을 저장하고, 4년 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

# 메소드의 입력 파라미터에 배열을 사용하기

31

## 배열을 매개변수로 하여 메소드 정의

```
void printArray( int array[] ){  
    for( int item : array )  
        System.out.println( item );  
}
```

## Java 메소드의 입력 파라미터는 항상 값을 복사(전달)받음

- 배열처럼 Reference Type의 변수에서는 참조값이 복사되는 값임
- 메소드 내에 새로운 배열이 복제되어 생기지 않음!

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

메모리에는 5개의 int 요소를 저장하는 배열 하나만 존재함.  
하지만, array와 arr의 참조값(메모리를 찾아가기 위한 정보)이  
같아졌기 때문에 두 개의 변수가 같은 배열에 접근하게 됨

## 메소드에서 배열의 내용을 바꾸면?

- 메소드를 호출한 쪽에서도 배열 요소의
- 값이 변경되어 있음을 확인할 수 있음

```
int[] arr = {1,2,3,4,5};  
changeArray( arr );  
System.out.println( arr[0] );
```

```
void changeArray( int array[] ) {  
    array[0] += 10;  
}
```

# 메소드의 배열 리턴

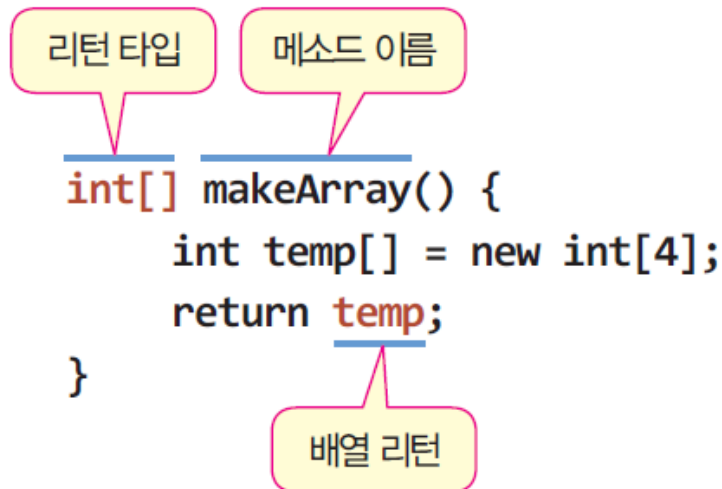
32

## ▣ 배열 리턴

- 배열의 레퍼런스만 리턴(배열 전체가 리턴되는 것이 아님)

## ▣ 메소드의 리턴 타입

- 리턴하는 배열 타입과 리턴 받는 배열 타입 일치
- 리턴 타입에 배열의 크기를 지정하지 않음



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

```
int [] intArray;  
intArray = makeArray();
```




# 배열을 리턴 받아 사용하는 과정

33

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



(1) `int[] intArray;`

intArray 

(2) `intArray = makeArray();`

intArray  

makeArray() 메소드

temp  

new int [4]

(3) `for(int i=0; i<intArray.length; i++)  
 intArray[i] = i;`

intArray  

## 예제 3-11 : 배열 리턴

34

일차원 정수 배열을 생성하여 리턴하는 `makeArray()`를 작성하고, 이 메소드로부터 배열을 전달받는 프로그램을 작성하라.

```
public class ReturnArray {  
    static int[] makeArray() {  
        int temp[] = new int[4];  
        for(int i=0; i<temp.length; i++)  
            temp[i] = i; // 배열 초기화, 0, 1, 2, 3  
        return temp; // 배열 리턴  
    }  
  
    public static void main(String[] args) {  
        int intArray[];  
        intArray = makeArray(); // 메소드가 리턴한 배열 참조  
        for(int i=0; i<intArray.length; i++)  
            System.out.print(intArray[i] + " ");  
    }  
}
```

`makeArray()`가 종료해도 생성된 배열은 소멸되지 않음

0 1 2 3

# 자바의 예외 처리

35

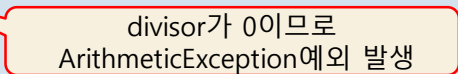
- 예외(Exception)
  - ▣ 실행 중 오동작이나 결과에 악영향을 미치는 예상치 못한 상황 발생
    - 자바에서는 실행 중 발생하는 에러를 예외로 처리
- 실행 중 예외가 발생하면
  - ▣ 자바 플랫폼은 응용프로그램이 예외를 처리하도록 호출
    - 응용프로그램이 예외를 처리하지 않으면 프로그램 강제 종료 시킴
- 예외 발생 경우
  - ▣ 정수를 0으로 나누는 경우
  - ▣ 배열의 크기보다 큰 인덱스로 배열의 원소를 접근하는 경우
  - ▣ 정수를 읽는 코드가 실행되고 있을 때 사용자가 문자를 입력한 경우

## 예제 3-12 : 0으로 나누기 시 예외 발생으로 응용프로그램이 강제 종료되는 경우

36

두 정수를 입력받아 나눗셈을 하고 몫을 구하는 프로그램 코드이다. 사용자가 나누는 수에 0을 입력하면 자바 플랫폼에 의해 예외가 발생하여 프로그램이 강제 종료된다

```
1 import java.util.Scanner;
2 public class DivideByZero {
3     public static void main(String[] args) {
4         Scanner scanner = new Scanner(System.in);
5         int dividend; // 나뉘는수
6         int divisor; // 나눗수
7
8         System.out.print("나뉘는수를 입력하십시오:");
9         dividend = scanner.nextInt(); // 나뉘는수 입력
10        System.out.print("나눗수를 입력하십시오:");
11        divisor = scanner.nextInt(); // 나눗수 입력
12        System.out.println(dividend+"를 " + divisor + "로 나누면 몫은 "
13            + dividend/divisor + "입니다.");
14    }
15 }
```



나뉘는수를 입력하십시오:100

나눗수를 입력하십시오:0

Exception in thread "main" java.lang.ArithmeticException: / by zero  
at DivideByZero.main(DivideByZero.java:13)

# 자바의 예외 처리, try-catch-finally문

37

## □ 예외 처리

- ▣ 발생한 예외에 대해 개발자가 작성한 프로그램 코드에서 대응하는 것
- ▣ try-catch-finally문 사용
  - finally 블록은 생략 가능

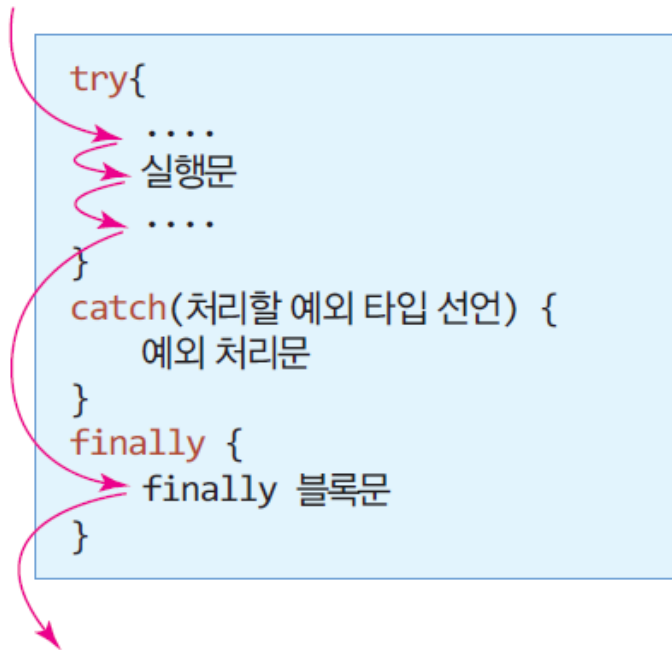
```
try {  
    예외가 발생할 가능성이 있는 실행문(try 블록)  
}  
catch (처리할 예외 타입 선언) {  
    예외 처리문(catch 블록)  
}  
finally {  
    예외 발생 여부와 상관없이 무조건 실행되는 문장(finally 블록)  
}
```

} 생략 가능

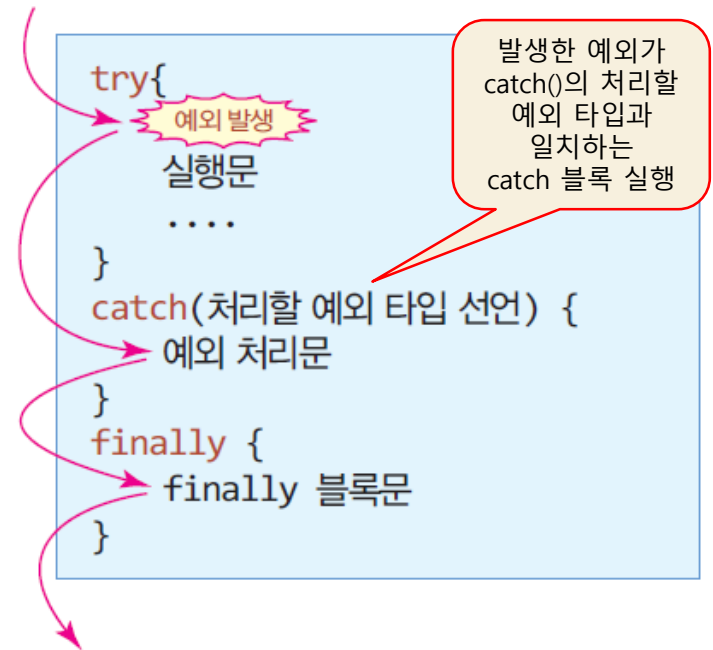
# 예외가 발생/발생하지 않은 경우 제어의 흐름

38

try 블록에서 예외가 발생하지 않은 정상적인 경우



try 블록에서 예외가 발생한 경우



# 자바의 예외 클래스

39

- 자바 플랫폼은 응용프로그램이 실행 중 오류를 탐지할 수 있도록 많은 예외를 클래스 형태로 제공

예외 타입(예외 클래스)	예외 발생 경우	패키지
ArithmeticException	정수를 0으로 나눌 때 발생	java.lang
NullPointerException	null 레퍼런스를 참조할 때 발생	java.lang
ClassCastException	변환할 수 없는 타입으로 객체를 변환할 때 발생	java.lang
OutOfMemoryError	메모리가 부족한 경우 발생	java.lang
ArrayIndexOutOfBoundsException	배열의 범위를 벗어난 접근 시 발생	java.lang
IllegalArgumentException	잘못된 인자 전달 시 발생	java.lang
IOException	입출력 동작 실패 또는 인터럽트 시 발생	java.io
NumberFormatException	문자열이 나타내는 숫자와 일치하지 않는 타입의 숫자로 변환 시 발생	java.lang
InputMismatchException	Scanner 클래스의 nextInt()를 호출하여 정수로 입력받고자 하였지만, 사용자가 'a' 등과 같이 문자를 입력한 경우	java.util

# 예외 클래스 사례

40

- 배열의 범위를 벗어나 원소를 접근하는 예외 처리
  - ▣ `ArrayIndexOutOfBoundsException` 예외

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

```
try {  
    intArray[3] = 10; // 예외 발생하지 않음  
    intArray[6] = 5; // 예외 발생  
}
```

이 문장 실행 시  
`ArrayIndexOutOfBoundsException`  
예외 발생

```
catch(ArrayIndexOutOfBoundsException e) { // 객체 e에 예외 정보가 넘어옴  
    System.out.println("배열의 범위를 초과하여 원소를 접근하였습니다.");  
}
```



## 예제 3-13 : 0으로 나누는 예외에 대처하는 try-catch 블록 만들기

41

try-catch-finally 블록을 이용하여 예제 3-12를 수정하여, 정수를 0으로 나누는 경우에 "0으로 나눌 수 없습니다!"를 출력하는 프로그램을 작성하라.

```
import java.util.Scanner;
public class DevideByZeroHandling {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int dividend; // 나뉘는수
        int divisor; // 나눗수

        System.out.print("나뉘는수를 입력하시오:");
        dividend = scanner.nextInt(); // 나뉘는수 입력
        System.out.print("나눗수를 입력하시오:");
        divisor = scanner.nextInt(); // 나눗수 입력
        try {
            System.out.println(dividend+"를 " + divisor + "로 나누면 몫은 " + dividend/divisor + "입니다.");
        }
        catch(ArithmeticException e) { // ArithmeticException 예외 처리 코드
            System.out.println("0으로 나눌 수 없습니다!");
        }
        finally {
            scanner.close(); // 정상적이든 예외가 발생하든 최종적으로 scanner를 닫는다.
        }
    }
}
```

divisor가 0인 경우  
ArithmeticException 예외 발생

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

ArithmeticException 예외가 발생해도  
프로그램이 강제 종료되지 않고 정상 실행됨

## 예제 3-14 : 입력오류시 발생하는 예외(InputMismatchException)

42

Scanner 클래스를 이용하여 3개의 정수를 입력받아 합을 구하는 프로그램을 작성하라. 사용자가 정수가 아닌 문자를 입력할 때 발생하는 InputMismatchException 예외를 처리하여 다시 입력받도록 하라.

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

public class InputException {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("정수 3개를 입력하세요");
        int sum=0, n=0;
        for(int i=0; i<3; i++) {
            System.out.print(i+">>");
            try {
                n = scanner.nextInt(); // 정수 입력
            }
            catch(InputMismatchException e) {
                System.out.println("정수가 아닙니다. 다시 입력하세요!");
                scanner.next(); // 입력 스트림에 있는 정수가 아닌 토큰을 버린다.
                i--; // 인덱스가 증가하지 않도록 미리 감소
                continue; // 다음 루프
            }
            sum += n; // 합하기
        }
        System.out.println("합은 " + sum);
        scanner.close();
    }
}
```

사용자가 문자를 입력하면  
InputMismatchException 예외 발생

정수 3개를 입력하세요

0>>5

1>>R

정수가 아닙니다. 다시 입력하세요!

1>>4

2>>6

합은 15