

자바 기초문법

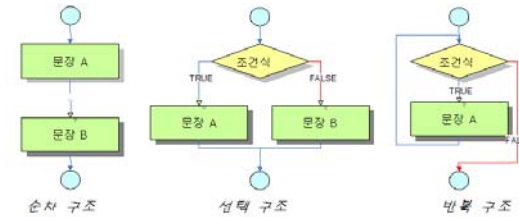
선택, 반복, 배열, 예외처리

514760-1
2019년 봄학기
3/19/2019
박경신

Control Statement

제어문의 종류

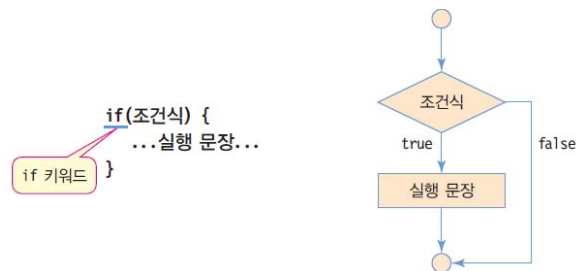
- 제어문이란 프로그램을 실행할 때는 논리적인 흐름이 필요한데, 이러한 문장의 논리적인 흐름을 통제해 주는것.
- 조건문 - 조건식의 값에 따라 각각에 해당되는 명령문을 수행한다. 예) if 문, switch 문
- 반복문 - 조건이 만족하는 동안 특정 명령문을 반복적으로 수행한다. 예) while 문, do 문, for 문, foreach 문
- 점프문 - 제어권을 이동시킬 때 점프문을 사용한다. 예) label 문, break 문, continue 문



If Statement

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



예제 : if 구문 사용

점수가 80점이 이상이면 합격 여부를 판별

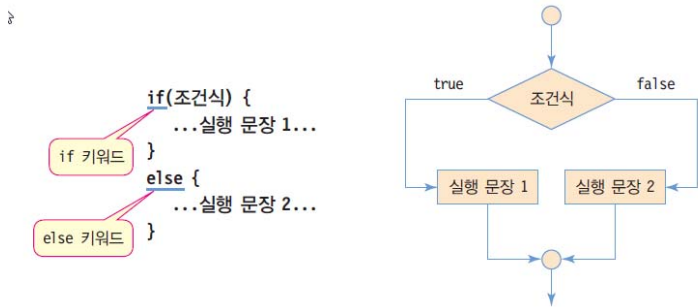
```
import java.util.Scanner;
public class SuccessOrFail {
    public static void main (String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("점수를 입력하시오: ");
        int score = in.nextInt();
        if (score >= 80)
            System.out.println("축하합니다! 합격입니다.");
    }
}
```

점수를 입력하시오: 95
축하합니다! 합격입니다.

If-Else Statement

if-else 문

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



예제 : if-else 구문 사용

입력된 수가 3의 배수인지 판별

```
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.

If-Else Statement

다중 if-else 문

- 실행문장이 다시 if문 또는 if-else문을 포함
- else 문은 바로 전의 if문과 짝을 이룬다.
- 조건문이 너무 많은 경우, switch 문 사용 권장



예제 : 다중 if-else 사용

키보드 입력된 성적에 학점을 부여하는 프로그램 작성

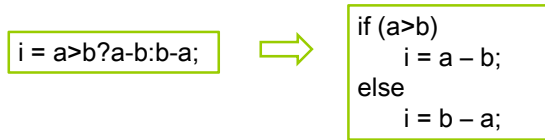
```
import java.util.Scanner;
public class Grading {
    public static void main (String[] args) {
        char grade;
        Scanner a = new Scanner(System.in);
        while (a.hasNext()) {
            int score = a.nextInt();
            if(score >= 90) // score가 90 이상인 경우
                grade = 'A';
            else if(score >= 80) // score가 80 이상이면서 90 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

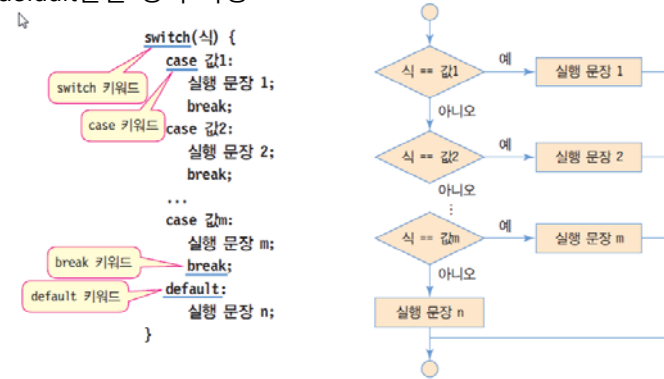
Ternary Conditional Operator ?:

- 조건 연산자 ?는 if-else로 바꿀 수 있음



Switch Statement

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



Switch Statement

- switch문 내의 break문
 - break 문장을 만나면 switch문을 벗어남
 - 만일 case 문에 break문이 없다면, 다음 case문의 실행 문장으로 실행을 계속하게 되며, 언젠가 break를 만날 때까지 계속 내려감

```

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

90 ~ 100점입니다.
80 ~ 89점입니다.

예제 : switch 사용

- switch구문을 사용하여 학점에 따른 출력

```

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

Switch Statement

□ case 문의 값의 특징

- switch 문은 식의 결과 값을 case 문과 비교
- 사용 가능한 case문의 비교 값
 - 정수 타입 리터럴, JDK 1.7부터는 문자열 리터럴도 허용

```
int a = 0;
int b = 1;
int c = 25;
switch(c%2) {
  case 1 : // 정수 리터럴
    ...;
  break;
  case 2: // 정수 리터럴
}
}
```

```
String s = "예";
switch(s) {
  case "예" : // 문자열 리터럴
    ...;
  break;
  case "아니오" : // 문자열 리터럴
    ...;
  break;
}
```

Switch Statement

□ 잘못된 case 문

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

예제 : switch 사용한 학점 분류

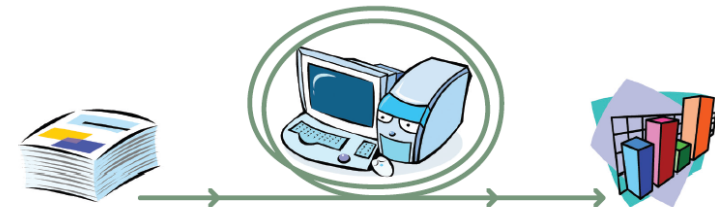
```
import java.util.Scanner;
public class Grading2 {
  public static void main (String[] args) {
    char grade;
    Scanner a = new Scanner(System.in);
    while (a.hasNext()) {
      int score = a.nextInt();
      switch (score/10) {
        case 10:
        case 9:
          grade = 'A';
          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");
    }
  }
}
```

```
100
학점은 A입니다
55
학점은 F입니다
76
학점은 C입니다
```

Loop

Q) 반복 구조는 왜 필요한가?

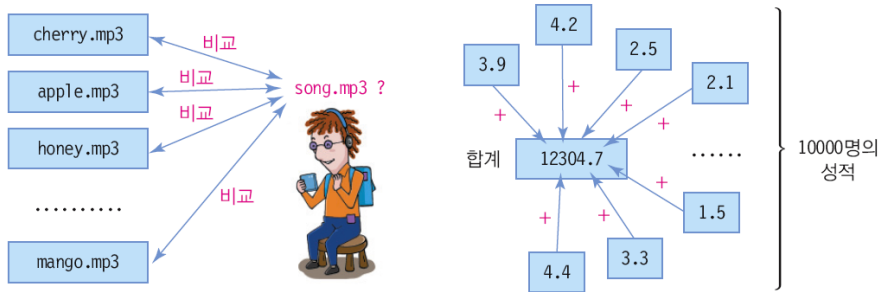
A) 같은 처리 과정을 되풀이하는 것이 필요하기 때문이다.
학생 30명의 평균 성적을 구하려면 같은 과정을 30번 반복하여야 한다.



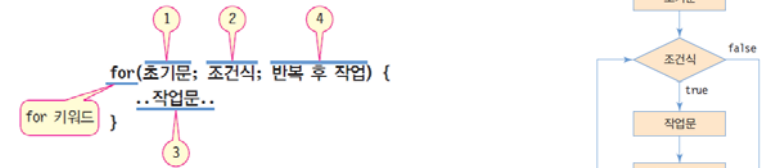
Loop

자바 반복문의 종류

- for 문
- while 문
- do while 문



For Statement

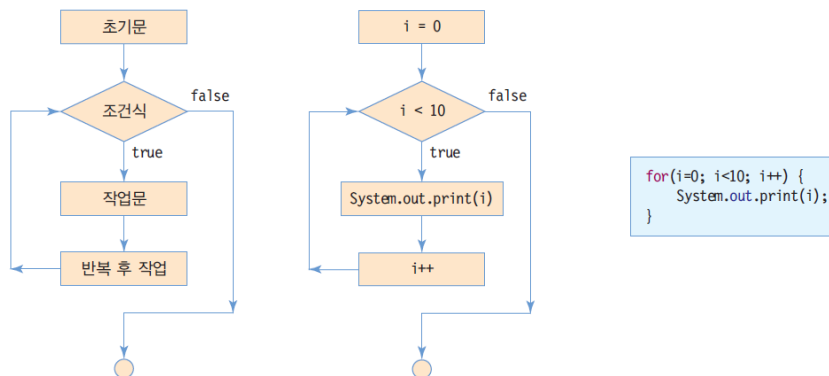


- for 문이 실행한 후 오직 한번만 실행되는 초기화 작업
 • 콤마(,)로 구분하여 여러 문장 나열 가능
 • 초기화할 일 없으면 비어둘 수 있음
- 논리형 변수나 논리 연산만 가능
 • 반복 조건이 true이면 반복 계속, false이면 반복 종료
 • 반복 조건이 true 상수인 경우, 무한 반복
 • 반복 조건이 비어 있으면 true로 간주
- 반복 작업 문장들의 실행 후 처리 작업
 • 콤마(,)로 구분하여 여러 문장 나열 가능

For Statement

For 문의 실행 과정을 나타내는 순서도

- 주로 정해진 횟수만큼 반복할 때 사용됨
- 또는 정해진 범위를 반복할 때 활용됨



For Statement

0~9 까지 정수 출력

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

For-loop 안에 변수 선언 가능

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

0~100까지 합 계산

```
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;
```

For Statement

```
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 : for 구문 사용

- 1부터 10까지 덧셈을 표시하고 합 계산

```
public class ForSample {
    public static void main (String[] args) {
        int i, j;
        for (j=0,i=1; i <= 10; 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

예제 : for 구문 사용

- 사용자로부터 정수를 입력 받아 factorial 계산

```
public class Factorial {
    public static void main (String[] args) {
        long fact = 1;
        int n;
        System.out.printf("정수를 입력하십시오: ");
        Scanner scan = new Scanner(System.in);
        n = scan.nextInt();
        for (int i=1; i <= n; i++) {
            fact = fact * i;
        }
        System.out.printf("%d!는 %d입니다. ", n, fact);
    }
}
```

정수를 입력하십시오: 5
5!는 120입니다.

예제 : for 구문 사용

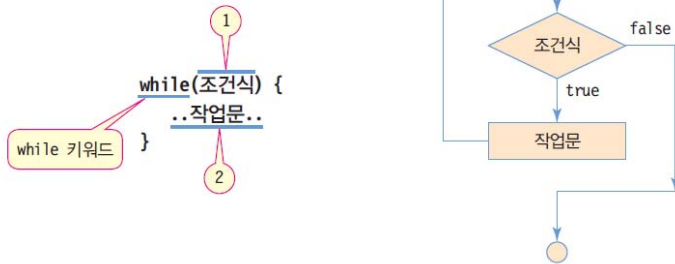
- 사용자로부터 양의 정수를 입력 받아 약수를 계산

```
public class Divisor {
    public static void main (String[] args) {
        System.out.printf("양의 정수를 입력하십시오: ");
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        System.out.println(n + "의 약수는 다음과 같습니다.");
        for (int i=1; i <= n; ++i) {
            if (n % i == 0)
                System.out.print(" " + i);
        }
    }
}
```

양의 정수를 입력하십시오: 100
100의 약수는 다음과 같습니다.
1 2 4 5 10 25 50 100

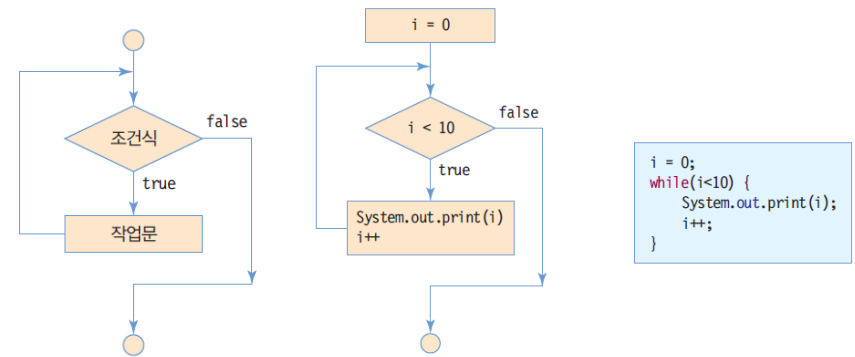
While Statement

- 반복 조건이 true이면 반복, false 이면 반복 종료
- 반복 조건이 없으면 컴파일 오류
- 처음부터 반복 조건을 통과한 후 작업문 수행



While Statement

- While 문의 실행과정을 나타내는 순서도



예제 : while 구문 사용

- 사용자로부터 숫자를 입력 받아 입력 받은 모든 수의 평균을 출력. 0이 입력되면 입력이 종료되고 평균 계산.

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

10
20
30
40
0
마지막 입력
입력된 수의 개수는 4개이며 평균은 25.0입니다.

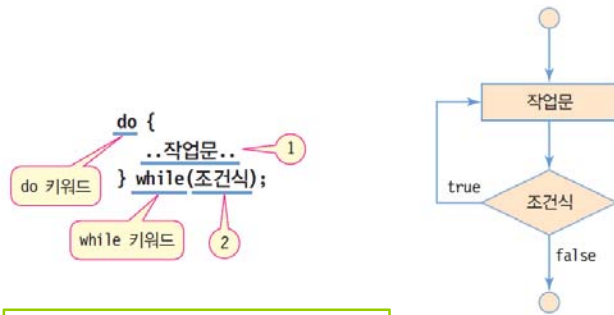
예제 : while 구문 사용

- 사용자로부터 두 수를 입력 받아 최대 공약수를 계산

```
public class Gcd {
    public static void main (String[] args) {
        int x, y, r;
        System.out.printf("두개의 정수를 입력하십시오 (큰수, 작은수): ");
        Scanner scan = new Scanner(System.in);
        x = scan.nextInt();
        y = scan.nextInt();
        while (y !=0) {
            r = x % y;
            y = r;
        }
        System.out.println("최대공약수는 " + x);
    }
}
```

두개의 정수를 입력하십시오 (큰수, 작은수) : 24 36
최대 공약수는 12

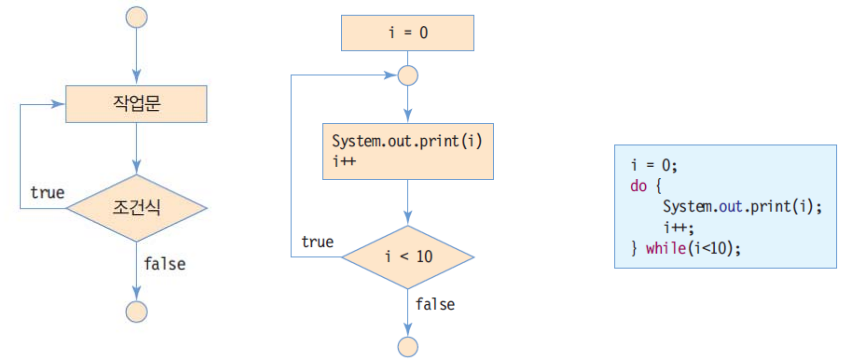
Do-While Statement



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

Do-While Statement

□ do-while 문의 실행 과정을 나타내는 순서도



예제 : do-while 구문 사용

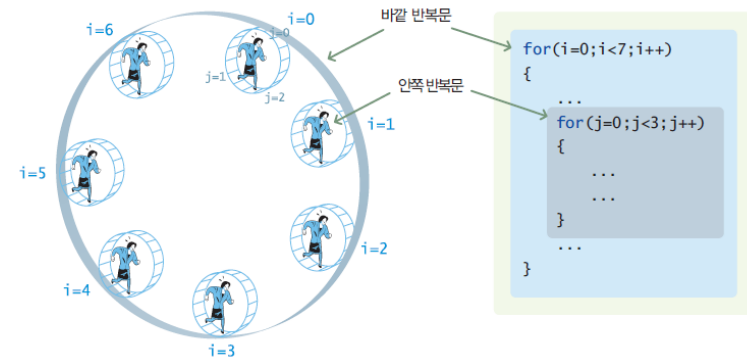
□ 'a'부터 'z'까지 출력

```
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

Nested Loop

□ 중첩 반복문(nested loop)은 반복문 안에 다른 반복문이 위치



Nested Loop

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 : Nested-Loop 사용

중첩 for문을 사용하여 구구단을 한 줄에 한 단씩 출력

```
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
```

Continue Statement

continue 문

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

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

분기

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

조건식으로
분기

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

조건식으로분기

예제 : for와 continue 문 사용

For와 continue문을 사용하여 1부터 100까지 짝수의 합을 계산

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

1부터 100까지 짝수의 합은 2550

Break Statement

□ 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 : while과 break문 사용

- while과 break문을 사용하여 -1이 입력될 때까지 입력된 숫자의 개수를 출력

```
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 입력

Array

□ 배열(array)

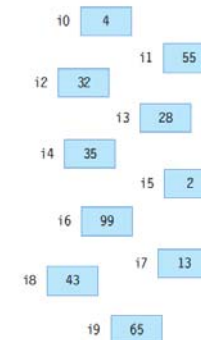
- 여러 개의 데이터를 같은 이름으로 활용할 수 있도록 해주는 자료 구조
 - 인덱스(Index, 순서 번호)와 인덱스에 대응하는 데이터들로 이루어진 자료 구조
 - 배열을 이용하면 한 번에 많은 메모리 공간 선언 가능
- 배열은 같은 타입의 데이터들이 순차적으로 저장되는 공간
 - 원소 데이터들이 순차적으로 저장됨
 - 인덱스를 이용하여 원소 데이터 접근
 - 반복문을 이용하여 처리하기에 적합한 자료 구조(주로 for 또는 for-each 반복문과 많이 사용됨)
- 배열 인덱스
 - 0부터 시작
 - 인덱스는 배열의 시작 위치에서부터 데이터가 있는 상대 위치

Array

□ 자바 배열의 필요성과 모양

(1) 10개의 정수형 변수를 선언하는 경우

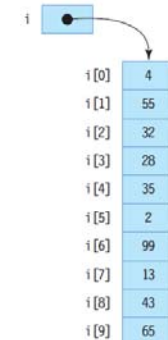
```
int i0, i1, i2, i3, i4, i5, i6, i7, i8, i9;
```



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];

Array

배열 선언과 배열 생성의 두 단계 필요

배열 선언

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

배열 생성

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

선언과 초기화

배열 생성과 값 초기화

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

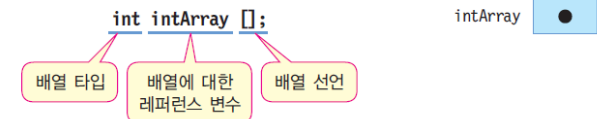
잘못된 배열 선언

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

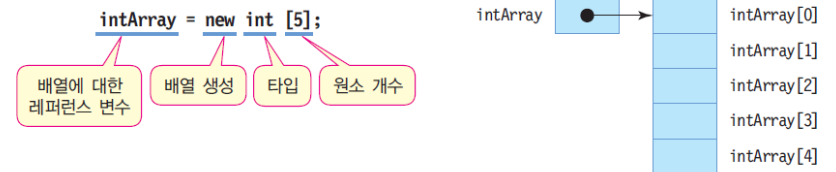
Array

배열 선언과 생성

(1) 배열에 대한 레퍼런스 변수 intArray 선언



(2) 배열 생성



Array

배열을 초기화하면서 생성한 결과

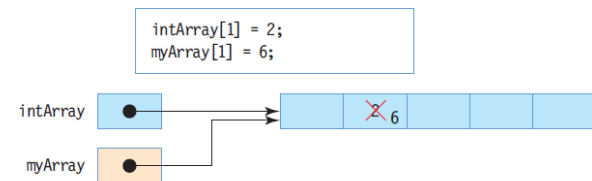
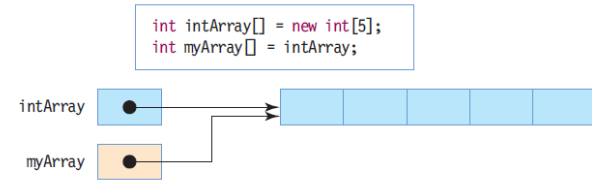
```
int intArray[] = {4, 3, 2, 1, 0};  
float floatArray[] = {0.01, 0.02, 0.03, 0.04};
```



Array

배열 참조

생성된 1개의 배열을 다수의 레퍼런스가 참조 가능



Array

배열 원소 접근

- 반드시 배열 생성 후 접근

```
int[] intArray; // 배열 선언
intArray[4] = 8; // 오류, 아직 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; // 배열 선언
intArray = new int[10]; // 배열 생성

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

예제: Array 사용

- array를 사용하여 숫자를 입력받아 저장하고 입력된 숫자 중에 제일 큰 수를 화면에 출력

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

```
숫자를 입력하시오:1
숫자를 입력하시오:39
숫자를 입력하시오:78
숫자를 입력하시오:100
숫자를 입력하시오:99
입력된 수에서 가장 큰 수는
100입니다.
```

예제 : 문자열 Array 사용

- 문자열 array를 사용하여 5가지 피자 토핑을 출력

```
public class PizzaTopping {
    public static void main(String[] args) {
        String[] toppings = {"Pepperoni", "Mushrooms", "Onions", "Sausage", "Bacon"};
        for (int i = 0; i < toppings.length; i++) {
            System.out.print(toppings[i] + " ");
        }
    }
}
```

Pepperoni Mushrooms Onions Sausage Bacon

Array

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
```

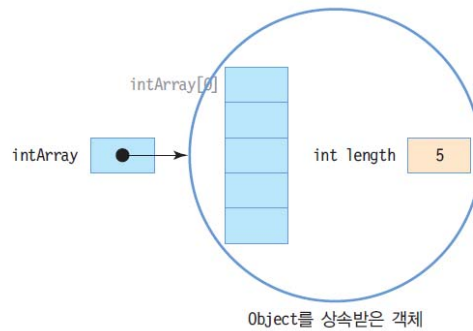
배열의 크기

- 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;
```

Array

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



예제 : Array를 사용한 원소의 평균 구하기

▣ 성적을 5개 입력 받아 배열에 저장하고 평균 성적을 계산

```
import java.util.Scanner;  
public class ArrayTest2 {  
    public static void main(String[] args) {  
        final int STUDENTS = 5;  
        int total = 0;  
        Scanner scan = new Scanner(System.in);  
        int[] scores = new int[STUDENTS];  
        for (int i = 0; i < scores.length; i++) {  
            System.out.print("성적을 입력하  
            scores[i] = scan.nextInt();  
        }  
        for (int i = 0; i < scores.length; i++) {  
            total += scores[i];  
            System.out.println("평균 성적은" + total / STUDENTS + "입니다");  
        }  
    }  
}
```

성적을 입력하시오:10
성적을 입력하시오:20
성적을 입력하시오:30
성적을 입력하시오:40
성적을 입력하시오:50
평균 성적은 30입니다.

예제 : 순차 탐색 (Sequential Search)

```
import java.util.Scanner;  
public class SeqSearch {  
    public static void main(String[] args) {  
        int[] s = { 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 };  
        int value, index = -1;  
        Scanner scan = new Scanner(System.in);  
        System.out.print("탐색할 값을 입력하시오: ");  
        value = scan.nextInt();  
        for (int i = 0; i < s.length; i++) {  
            if (s[i] == value)  
                index = i;  
        }  
        if (index < s.length && index >= 0)  
            System.out.println(" " + value + "값은 " + index + "위치에 있습니다.");  
    }  
}
```

탐색할 값을 입력하시오:50
50값은 5위치에 있습니다.

Array & For-each

▣ For-each 문

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

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

합은 15

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

사과 배 바나나 체리 딸기 포도

Anonymous Array

- 자바에서는 배열의 이름을 지정하지 않고 단순히 초기값만으로 배열을 생성시킬 수 있음
- 무명 배열 (Anonymous array)는 즉시 배열을 만들어서 함수의 인수로 전달하고자 할 때 많이 사용됨

```
new int[] { 1, 2, 3, 4, 5 }; // 배열의 이름이 없다
// 초기값을 가지는 무명 배열 생성
```

```
public static int sum(int[] numbers) {
    int total = 0;
    for (int i=0; i<numbers.length; i++) {
        total = total + numbers[i];
    }
    return total;
}
public static void main(String[] args) {
    System.out.println("합은 : " + sum(new int[] {1, 2, 3, 4}));
}
```

합은 10

2D Array

- 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

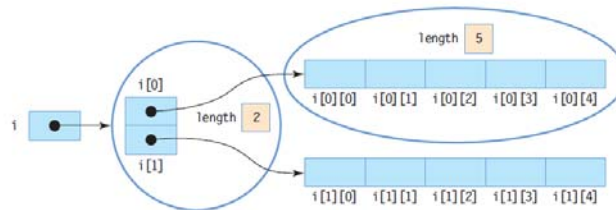
```
int[]    intArray = new int[2][5];
char[]   charArray = new char[5][5];
float[]  floatArray = new float[5][2];
```

- 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}};
```

2D Array

```
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

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

예제 : 2D Array 사용

지난 3년간 매출 총액과 연평균 매출을 계산

```
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];

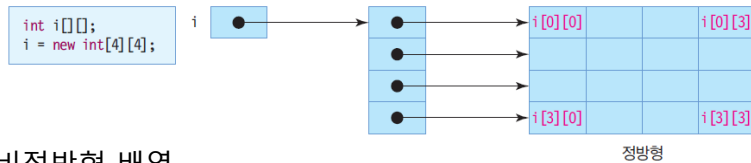
        System.out.println("지난 3년간 매출 총액은 " + sum +
            "이며 연평균 매출은 " + sum/intArray.length + "입니다.");
    }
}
```

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

비정방형 배열

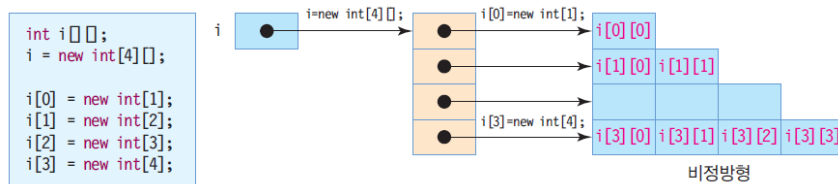
□ 정방형 배열

- 각 행의 열의 개수가 같은 배열

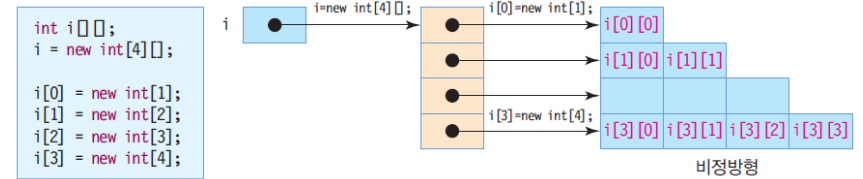


□ 비정방형 배열

- 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 n번째 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 : 비 정방형 배열의 생성과 접근

다음 그림과 같은 비정방형 배열을 만들어 값을 초기화하고 출력하십시오.

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

□ 메소드의 배열 리턴

- 배열의 레퍼런스만 리턴

□ 메소드의 리턴 타입

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

```
리턴 타입   메소드 이름
int[] makeArray() {
    int temp[] = new int[4];
    return temp;
}
배열 리턴
```

예제 : 배열 리턴 사용

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