JAVA PROGRAMING

비교 연산자, 논리 연산자

비교 연산자
== , != , > , < , >= , =<

a == b : a 와 b가 같으면 1 , 다르면 0
a != b : a 와 b가 같으면 0, 다르면 1
a > b : a 가 b 보다 크면 1, 작으면 0
a >= b : a 가 b 보다 크거나 같으면 1, 작으면 0
a < b : b 가 a 보다 크면 1, 작으면 0
a <= b : b 가 a 보다 크거나 같으면 1, 작으면 0

ex) int i = (3<5); 사실이니까 i = 1
ex) boolean b = (5!=5); 이 경우 boolean 타입은 사실일경우 true(1)를 , 거짓일경우 false(0) 라는 값을 지님

논리 연산자
|| 와 && 그리고 !

'||' : 좌변과 우변 둘중 하나이상이 true 면 true, 아니면 false
'&&' : 좌변과 우변 둘다 true면 true, 아니면 false
'!' : true면 false를, false면 true 를

ex) boolean a = (5==5 || 4<1); <- true
ex) boolean b = (5==5 &&4<1); <- false
ex) boolean c = !(5==5); <- false

캐스트 연산자, 비트 연산자

캐스트 연산자

캐스트 연산자는 변수의 자료형을 바꾸는 연산자이다.
- 사용 방법 : (타입)피연산자
  ex) double d = (double)500;

이렇게 되면 500 은 int 형 정수이지만, 캐스트 연산을 통해 double 형 값을 대입하게 된다.

비트 연산자

데이터의 비트를 조작해서 연산의 속도를 빠르게 할 수 있다는 장점

| (or연산)
&(and연산)
^ (xor 연산)

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
10진수 3	0	0	0	0	0	0	1	1
10진수 2	0	0	0	0	0	0	1	0
3|2	0	0	0	0	0	0	1	1
								결과는 3

if

괄호 안의 조건이 참이면 {}안의 내용을 실행
- 만약 실행할 내용이 한줄이라면 {} 생략 가능

```
class mainClass {
    public static void main(String []args){
        int num = 5;
        if(3 < 5){
            System.out.println("3은 5보다 작다");
        }
        if(num < 5)
            System.out.println("num은 5보다 작다");
    }
}
```

- 조건이 거짓일때는 else뒤의 내용을 실행
- 추가 조건 지정시는 elseif 사용

```
class mainClass {
    public static void main(String []args){
        int num = 4;
        if(num == 1)//num이 1이면 실행
            System.out.println("num은 "+num);
/*실행*/else if(num == 3)//num이 2이면 실행
            System.out.println("num은 "+num);
        else if(num == 2)//num이 3이면 실행
            System.out.println("num은 "+num);
        else  // 아무것도 아니면
            System.out.println("num은 뭐야?");
    }
}
```

switch

단점

- 부등호 연산이 불가능하다.
- 코드에서 알아보기 힘든 부분이 있다.

장점

- 분기의 기준이 명확하다.
- JDK1.7에서는 문자열로 분기가능

반드시 break를 사용해야 함.
Break는 {}를 빠져나가는 용도로

```
public class OddEx2 {

    public static void main(String[] args) {

        int odd = 10%3;

        switch(odd){
        case 0:
            System.out.println("나머지는 0");
        break;
        case 1:
            System.out.println("나머지는 1");
        break;
        default:
            System.out.println("나머지는 2");
        }

    }
}
```

비교문

예시

```
public class IfElesIfEx {
	public static void main(String[] args) {
		int grade = 65;

		if(grade >= 90){
			System.out.println("당신의 학점은 A");
		}else if(grade >= 80){
			System.out.println("당신의 학점은 B");
		}else if(grade >= 70){
			System.out.println("당신의 학점은 C");
		}else if(grade >= 60){
			System.out.println("당신의 학점은 D");
		}else if(grade >= 50){
			System.out.println("당신의 학점은 E");
		}
	}
}
```

```
당신의 학점은 D
```

비교문

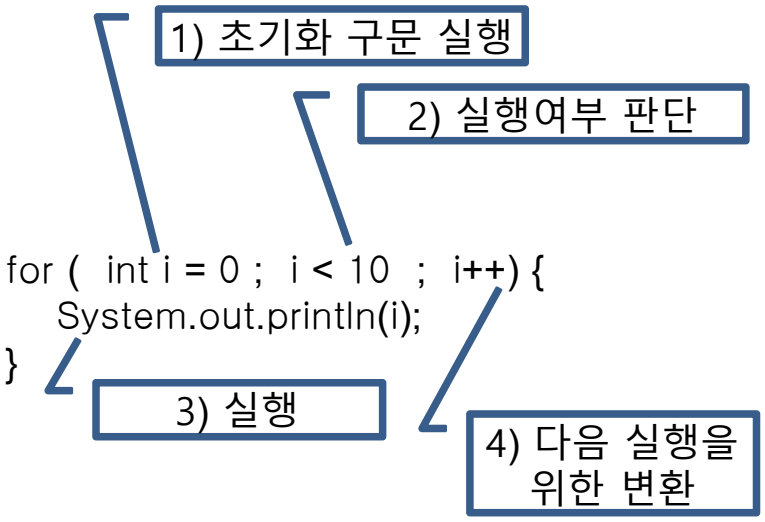
주사위(6면체) 숫자를 랜덤하게 발생시켜서

주사위 1이 나오면 “멍멍”
주사위 2가 나오면 “야옹”
주사위 3이 나오면 “3”
주사위 4가 나오면 “4”
주사위 5가 나오면 “5”
주사위 6이 나오면 “6” 을 출력하는 프로그램을 만드시오.

단, 프로그램은 2종류로서 1종류는 if문을 사용, 한종류는 switch문을 사용하시오.

순환문

For루프



초기화 조건은 루프 내에서만 쓰이는가?
- 안쪽의 제어문과 변수의 충돌 가능성
- 결과를 누적하는 변수

판단조건은 언젠가 false가 될 수 있는가?

변환 조건 때문에 불필요하게 루프를 돌지는 않는가?

순환문

```
public class ForEx1 {
	public static void main(String[] args) {
		int a = 10;
		for( System.out.println("AAAA"); a < 20; a++ ){
			System.out.println("BBBB");
		}
	}
}
```

```
AAAA
BBBB
BBBB
이하 생략
```

순환문

while 루프

실행횟수를 정확히 판단하기 어려운 경우에 많이 사용

일반적으로 while(true) {... }의 형태로 작성되는 경우가 가장 많다.

초기 조건이나, 변환 조건은 자유로운 형태로 지정할 수 있다.

do~ while: 일단 실행하고 판단

do{ ... 적어도 한번은 실행할 구문 }

while(조건문): 두 번째 이후의 실행을 판단할 판단조건

return, break, continue

코드의 실행의 제어권을 반납하는 return

순환문에서 순환을 중지하는 break

순환문에서 다음 코드로의 순환을 지시하는 continue

순환문

예제

```
public class ContinueEx {

        public static void main(String[] args) {
                for(int i = 0;  i < 10; i++){
                        if(i == 5){
                                System.out.println("i 값이 5이므로 continue");
                                continue;
                        }//end if
                        System.out.println(i);
                }//end for
        }
}
```

i의 값이 5일 때는 무조건 위로 갑니다.

```
0
1
2
3
4
i 값이 5이므로 continue
6
7
```

순환문

1~100까지 홀수의 합, 짝수의 합을 구하는 프로그램을 만드시오.

단, if문은 한번만 사용해야 합니다. (else, else if 사용 불가)

컴퓨터와 대전하는 주사위 게임을 만드시오.

1_ "화면에 주사위를 굴릴까요?"라는 메시지가 보입니다.
2_ 사용자는 Enter 를 누릅니다.
3_ 화면에 사용자가 뽑은 번호가 출력됩니다.
4_ "컴퓨터가 주사위를 굴립니다. 실행할까요?"라는 메시지가 출력됩니다.
5_ 사용자는 Enter 를 누릅니다.
6_ 컴퓨터가 만들어 낸 주사위 숫자가 출력되고, 사용자가 더 높은 수이면 "You Win!",
은 수이면 "You Lose!", 비기면 "Draw"라는 메시지를 출력합니다.

Scanner s = new Scanner(System.in);
Scanner에는 여러 기능이 있다. 그 중 nextInt()는 입력된 값을 숫자로 바꿔주는 기능이다.
이번에는 nextLine()을 사용한다. 이것은 입력된 값을 문자로 만들어 준다.

배열

배열의 초기화

배열 초기화의 두 단계
- 메모리상의 공간 확보
- 공간에 데이터 추가

배열 선언과 동시에 내용물을 초기화 하는 방식
- 배열 선언시에는 공간만 확보하고 나중에 내용물을 넣는 방식

배열의 접근

Java에서의 모든 배열의 접근은 index번호를 통해서만 접근
index 번호는 0에서 시작
배열의 길이를 알아내려고 할 경우에는 '.length'를 이용
루프에서 인덱스 번호 활용

배열의 제약

배열을 생성할 때 크기를 고정한다.
배열의 크기는 int의 범위를 넘지 못한다.
배열에는 동일한 타입의 데이터만 들어갈 수 있다.

배열

실제 사용 예

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

int[] arr2 = arr;

int[] arr3 = arr2;

arr3[0] = 10;

System.out.println(arr[0]);
System.out.println(arr2[0]);
System.out.println(arr3[0]);
```

배열 복사의 원리

자동적으로 크기가 늘어나는 배열 만들기

System.arraycopy( )를 이용한 복사

배열

배열 변수의 의미

```
public class Ex2 {
	public static void main(String[] args) {
		int[] arr = {10,20,30};
		System.out.println(arr);
	}
}
```

```
[I@de6ced ← 실행된 결과는 매번 달라질 수 있습니다.
```

실제 값이 저장된 것이 아니라, 데이터가 들어있는 공간의 주소값을 저장(레퍼런스)
참고) 자바에서 @가 포함된 값이 출력된다면, 그것은 레퍼런스 값이라는 의미

배열

배열 안에 있는 내용을 확인하고 싶을 때

```
import java.util.Arrays; // 반드시 추가해주어야 합니다.
import java.util.Scanner;

public class GradeEx {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);
		int[] grades = new int[5];

		for(int i = 0; i < grades.length; i++){
			System.out.println(i  + " 성적 점수를 넣어주세요.");
			int userInput = scanner.nextInt();
			grades[i] = userInput;
		}//end for

		System.out.println("처리할 점수들은 다음과 같습니다.");
		System.out.println(Arrays.toString(grades));
	}
}
```

배열

0~100까지 임의의 수를 30개 발생시켜서 배열에 넣은 후 정렬하시오.

조건1) 정렬 전/후의 값을 화면에 출력
조건2) 소트 알고리즘 중 사용한 알고리즘을 설명한 레포트 (형식 무관, 내용만 제출) 첨부

비고. 정렬 알고리즘 및 이중 루프문에 대한 공부는 따로 하셔야 합니다.
(서로간의 정보 교환 가능)

사용자에게 9개의 숫자를 입력받아서 3*3의 이중 배열에 넣은 후 각행의 덧셈 합을 출력하시오.

조건1) 출력 시 4*4의 형식으로 출력 (4행째는 덧셈 합)
조건2) 데이터 출력시 정렬된 형식으로 출력

비고. 이중배열 사용법 int[][] arr = new int[4][4];
비고. 데이터를 정렬된 형식으로 출력하는 방법을 찾아보시오.

배열

배열 복사시 (레퍼런스 복사가 아닌 실제 데이터 복사)

System.arraycopy(원본 배열, 원본의 복사 시작 인덱스 번호, 대상 배열, 대상 배열의 복사 시작 인덱스 번호, 개수);

의 형식으로 사용.

```
int[] arr = {1,2,3,4,5};
int[] temp = {1,2,0,0,0};

System.arraycopy(arr, 2, temp, 2,3);
System.out.println(Arrays.toString(arr));
System.out.println(Arrays.toString(temp));
```

JDK 1.5이상에서는 배열 처리시 foreach방식의 처리 가능

```
int[] arr = {10,20,30,40,50};

for(int value : arr){
      System.out.println(value);
}
```

Arrays.sort(배열이름); 을 사용하여 간단하게 소트도 가능