1년 초짜의

JAVA 자습서

for 초짜

작성자 김민정(mjkimdelta@gmail.com)

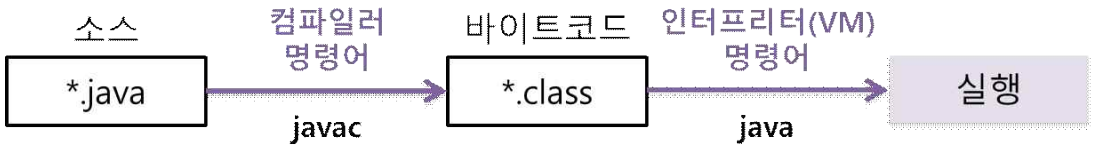
도서 『프로그래밍 JAVA(유순옥 외 2명)』 바탕

1 : 자바의 소개와 특징

‘ 자바(JAVA)’ 는 지구상에 존재하는 프로그래밍 언어 중 다섯 손가락 안에 꼽을 만큼 유명하고 널리 쓰이는 **객체지향 언어**이다. 자바는 썬 마이크로시스템즈의 제임스 고슬링과 연구원들에 의해 1991년부터 개발되었다. 자바라는 이름으로 최초 발표된 것은 1995년이며, 초기에는 가전제품의 내장되는 프로그램을 위해 쉽고 가볍도록 개발되었으나 이 특징으로 인해 대중화에 성공하였다. 20세기에 ‘ C’ 라는 절차지향 프로그래밍 언어가 널리 쓰였다면 21세기에는 객체지향 언어인 자바가 거의 필수라고 볼 수 있다. 물론 자바는 절차지향 언어인 C와 이것으로부터 나온 객체지향 언어인 ‘ C++’ 의 영향을 많이 받아 형태는 비슷하다. 하지만 비교적 훨씬 단순하고 편리하다. 또한 아스키 코드가 아닌 유니코드를 기반으로 한다.

■ 특징

1. VM 위에서 실행하여 플랫폼에 독립적이다.
2. 네트워크에 강해 분산 처리에 용이하다.
3. 보안에 강하여 안전하다.
4. 멀티스레드를 지원하여 속도가 빠르다.
5. 동적이다. 라이브러리의 확장이 용이하기 때문.
6. 견고하다.
   - 포인터를 사용하지 않는다.
   - 자동으로 가비지 컬렉션을 수행한다. (사용하지 않는 객체는 자동으로 사라짐)
   - 엄격한 데이터형 검사를 통해 에러를 조기에 발견한다.
   - 실행시간에 발생하는 에러를 처리한다.
7. 컴파일러와 인터프리터 언어 두 가지 특징을 모두 갖는다.



■ 자바 개발 환경

JDK (Java Development Kit)

자바 개발 환경이란 하드웨어나 운영체제에 관계없이 실행될 수 있는 자체 플랫폼을 말한다. JDK에는 자바 프로그램을 개발하는데 필요한 라이브러리와 플랫폼이 포함되어 있다. 따라서 자바 언어는 운영체제와 플랫폼에 독립적인 특징을 갖게 된다.

- JAVA API : 프로그래밍에 필요한 기능을 구현한 클래스들을 묶은 패키지.
- JVM : 컴파일 후 생기는 바이트 코드를 다양한 플랫폼에 맞는 기계어로 해석하고 실행.

■ 자바 설치

자바 개발 툴로는 ‘ 이클립스 IDE’ 를 많이 사용한다. JDK를 설치와 컴퓨터 내 환경변수 설정 후 이클립스 프로그램을 설치하여 자바 개발을 위한 기본적인 세팅을 끝내면 개발 환경은 준비 완료다.

2 : 변수와 자료형 (변수 1부)

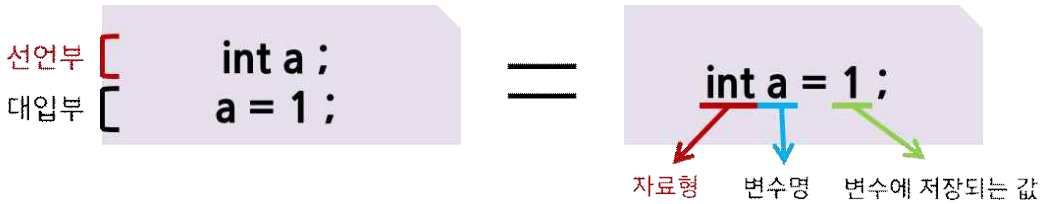
변수란 데이터가 저장된 주기억장치 속 기억공간을 말한다. 데이터는 언제든 바뀔 수 있고 덮어쓰기가 가능하다. 가장 중요한 건 최종으로 덮어쓰기가 된 값이 그 변수의 기억공간에 마지막으로 남는 유일한 데이터가 된다.

■ 변수명의 작성 규칙

1. 알파벳 대소문자와 숫자, 특수기호 중 ' _' 와 ' $' 로만 이루어져야 한다.
   (그러나 암묵적으로 변수명 작성 시 대문자를 거의 사용하지 않는다.)
2. 첫 문자에 숫자는 올 수 없다.
3. 예약어(이미 자바 문법의 일부로 쓰이는 단어/ex: int, return, static etc.)는 사용할 수 없다.
4. 대소문자를 구분한다.
5. 길이 제한이 없다.

☑ **프로그램 코드가 길어질수록 효율적인 코딩을 위해 변수명이 매우 중요하다.**

■ 변수의 형태



변수 선언과 대입 문장의 문법을 지키지 않으면 오류가 난다. 변수 선언 시 변수명은 개발자가 정하지만, 자료형과 변수에 저장하는 값의 관계는 지켜야 할 규칙이 있다.

■ 자료형(데이터 타입)

기본 자료형	논리형	boolean	1 byte	참과 거짓을 나타내는 true/false
	문자형	char	2 byte	유니코드 기반. ' *' 의 형태로 사용.
	정수형	byte	1 byte	$-2^7 \sim 2^7-1$ (-128~127)
		short	2 byte	$-2^{15} \sim 2^{15}-1$ (-32768~32767)
		int	4 byte	기본 정수형. $-2^{31} \sim 2^{31}-1$
		long	8 byte	$-2^{63} \sim 2^{63}-1$. 식별자 l,L.
	실수형	float	4 byte	식별자 f,F.
		double	8 byte	기본 실수형.
참조 자료형 예시	문자열 (객체)	**String**	1 byte	" *" 의 형태로 사용.

▷ **데이터 간의 형변환**이란 이미 선언된 데이터형을 다른 데이터형으로 변환하는 것을 의미하는데, 데이터형의 크기(바이트)와 자료형의 영향을 받는다. 위의 표에서 위에서 아래로의 방향은 **자동형변환**이 허용되는 순서이고, 거꾸로 올라가는 순서는 더 작은 데이터 형으로 변환하는 **강제형변환(타입캐스팅)**이 이루어져야 한다. 강제형변환은 되도록 권장하지 않는다.

- **기본 자료형**: 자바 라이브러리에서 기본으로 제공한다. 사용할 수 있는 메모리의 범위가 정해져 있다. 변수가 데이터의 값을 가진다.
  -논리형: 0과 1은 허용하지 않는다.
  -정수형: 양수, 음수, 0을 나타내는 데 사용되며, 부호가 있는 수를 표현할 땐 맨 앞의 비트가 부호를 나타낸다(양수:0/음수:1). long형의 경우, 대입되는 수가 int형의 크기를 벗어나지 않는다면 식별자가 필요 없다.
- **참조 자료형**: 클래스 자료형이라고도 하며, 자바에서 제공하는 것이나 개발자가 직접 만든 클래스를 참조하는 형태이다. 이 변수는 값에 대한 참조, 즉 주소를 가지며, 종류로는 배열과 인터페이스 등이 더 있다.

■ 데이터를 출력하는 문장 : System.out.print( )

코드에 출력문이 없으면 프로그램을 실행해도 아무런 결과도 출력되지 않는다. 문자열은 쌍따옴표를 사용, 변수와 상수 등은 그 이름이나 제 값 그대로 작성한다. print()가 아닌 println()사용시 자동으로 문장 끝에 줄바꿈이 된다. 줄바꿈을 수행하는 기호는 ‘ \n’이다. println()은 print(“ \n”)와 같다.

예제 [Variables_type.java]

```
public class Variables_type {
     public static void main(String args[]) {
          /*주석은 출력되지 않아요!
            이 형태는 한 줄 이상의 주석에 사용됩니다.*/
          boolean sign = true; //논리형
          int number = 2003; //정수형
          long tel = 2147483647; //int의 범위는 식별자 필요X
          tel = 2147483648L; //변수 tel에 최종으로 저장되는 값
          float radius = 3.14F;
          double radius_2 = 3.14159265; //기본실수형
          String name = “최범규”; //참조형

          System.out.println(“boolean:”+sign);
          System.out.println(“int:”+number);
          System.out.println(“long:”+tel);
          System.out.println(“float:”+radius);
          System.out.println(“double:”+radius_2);
          System.out.println(“String:”+name);
     }
}
```

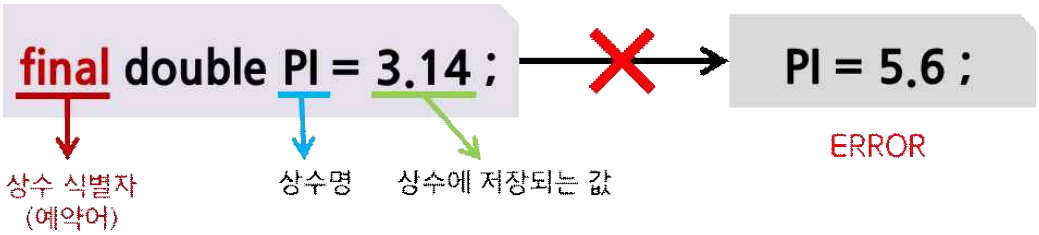
결과

```
boolean:true
int:2003
long:2147483648
float:3.14
double:3.14159265
String:최범규
```

3 : 상수

상수란 프로그램 실행 중 **변하지 않는 값**을 의미하며, 변수와 반대되는 개념이다. 변수처럼 선언을 통해 특정 공간을 가지며 선언된 상수명을 통해 사용되는 것은 다르지 않다. 종류로는 정수형, 실수형, 문자형, 논리형이 있다.

■ 상수의 형태



■ 상수 사용의 유의점

1. 상수의 가장 큰 특징으론 식별자 **final**을 변수 타입 앞에 써야 한다.
2. 상수명은 주로 대문자를 사용하는데, 이것은 변수와 구분하기 위함이니 권장한다.
3. 상수도 변수처럼 선언과 대입(초기화)을 각각 할 수 있지만 가급적 선언과 동시에 초기화하는 것이 좋다. 초기화하지 않은 상수를 사용할 경우 오류가 발생한다.

■ 상수 사용의 장점

1. 불분명한 값에 의미를 부여할 수 있다.
2. 코드가 간결해지며 프로그램 수정에 용이하다.
3. 개발자에 의해 잘못 입력되는 것을 방지한다.

■ 리터럴 상수

‘ 리터럴(literal)’ 이란 프로그램에서 사용하는 모든 숫자, 문자, 논리값을 일컫는다. 한마디로 코드 내 직접 표현된 값을 말한다. 이 리터럴은 변수나 상수 값으로 대입할 수 있다. 리터럴은 프로그램이 시작할 때 시스템에 같이 로딩되어 특정 메모리 공간인 상수 풀에 놓이게 된다.

■ 데이터를 입력받는 Scanner문

‘ import Java.util.Scanner’ 라는 문장을 코드 맨 첫 줄에 삽입하여 Scanner클래스를 라이브러리에서 가져온 후, Scanner를 사용할 메소드 내에 정의한다.

정의
```
Scanner 스캐너명 = new Scanner(System.in);
```
사용
```
스캐너명.메서드( ) ;
```
예
```
Scanner scan = new Scanner(System.in);

int age = scan.nextInt();
String city = scan.next();
```

변환 타입	메소드 이름
String	next()
	nextLine()
byte	nextByte()
int	nextInt()
long	nextLong()
short	nextShort()
float	nextFloat()
double	nextDouble()

예제 [Circle.java]

원의 반지름을 입력받아 둘레와 넓이를 구하는 프로그램(원주율: 3.14)

```
import java.util.Scanner; //라이브러리에서 스캐너 가져오기

public class Circle {
     public static void main(String args[]) {
          Scanner scan = new Scanner(System.in); //스캐너 정의

          final double PI = 3.14; //원주율 상수 정의
          System.out.print("반지름 입력:");
          double radius = scan.nextDouble(); //반지름 입력받기

          System.out.println("원의 둘레:"+(2*PI*radius));
          System.out.println("원의 넓이:"+(PI*radius*radius));
     }
}
```

결과

```
반지름 입력:5.5
원의 둘레:34.54
원의 넓이:94.985
```

복습 예제 [ClassType.java]

참조형 데이터 타입은 데이터가 저장되어 있는 메모리 주소를 기억하며,
이 주소를 통해 해당 데이터를 참조할 수 있도록 한다.

```
public class ClassType {
     public static void main(String args[]) {

          int a = 10; //기본 타입
          String str = "Ten"; //참조 타입
          Object refObj = new Object(); //최상위 객체 'Object'

          System.out.println(a);
          System.out.println(str);
          System.out.println(refObj);
     }
}
```

결과

```
10
Ten
java.lang.Object@7852e922 이 주소값은 컴퓨터마다 다르다
```

* 예제의 코드가 정답이라는 생각은 버려야 한다.

4 : 연산자

자바에서도 데이터의 연산을 위해 연산자가 필요하다. 10진수 뿐 아니라 2진수를 다루는 연산도 있으며, 연산자의 종류와 용도를 반드시 알아야 한다. 연산자는 우선 항에 개수에 따라 크게 3가지로 나눌 수 있다.

■ **단항 연산자** 항이 한 개인 연산자

- 부호 연산자 +, − : 이항 연산자이기도 하지만 부호를 나타내는 단항 연산자로도 쓰인다. 리터럴 상수뿐만 아니라 일반 상수와 변수에도 사용 할 수 있다. 주로 값을 음수로 만들기 위해 사용된다.
- 증감 연산자 ++, — : 항(값)의 앞이나 뒤에 붙여 값을 1만큼 늘리거나 감소시킨다. 값의 앞에 붙인 경우엔 해당 문장의 연산이 이루어지기 전 1을 증감하고, 값의 뒤에 붙인 경우엔 해당 문장의 연산이 완료된 후 1을 증감한다. 많이 사용되는 만큼 헷갈리기 쉬우니 주의해야 한다.
- 논리 연산자의 !(부정/NOT) : 항(값)의 앞에 붙이며, 값의 참거짓을 부정한다. 값이 참일 경우엔 부정을 나타내고, 값이 부정일 경우 참을 나타낸다.
- 비트 연산자의 ~(반전) : 2진 형태에서 각 자리의 비트들을 0또는 1로 반전한다.

■ **이항 연산자** 항이 두 개인 연산자

- 산술 연산자 +, -, *, /, % : 차례대로 덧셈, 뺄셈, 곱셈, 나눗셈, 나머지이다. 이때 나머지 연산자란 앞에 있는 항에서 뒤에 있는 항을 나누어 나머지만 구한다. 산술 연산자의 우선순위는 일반 수학연산과 같으며 나머지 연산자의 우선순위는 곱셈과 나눗셈과 같다. 나눗셈 연산 시 정수형으로 반환할 경우 소수점은 무시된다.
- 관계 연산자

>, <	초과와 미만
>=, <=	이상과 이하
==	같다
!=	다르다

해당 식이 참일 때 참을 반환하고, 거짓일 땐 거짓을 반환한다. 이항 연산만 가능하며, 삼항 연산이 불가능하다. 때문에 주로 논리 연산자와 함께 사용된다.

- 논리 연산자의 &&, || : &&는 논리곱(AND)을 나타내며, 두 항이 모두 참이 경우에만 참을 반환한다. ||는 논리합(OR)을 나타내며, 두 항 중 하나만 참이어도 참을 반환한다. 자바에선 0과 양수는 취급하지 않으며, true와 false만 취급한다.

```
boolean flag1, flag2;
flag1 = true;
flag2 = false;
int a = 10;
System.out.println(flag1 && flag2); //결과는 false
System.out.println(!(0<=4)||(a==10)); //결과는 true
```

▪ 비트 연산자 : 두 항의 같은 자리의 비트끼리 계산한다. 따라서 2진수로 계산이 이루어진다.

논리 연산	& (AND)	두 개의 비트 값이 모두 1인 경우에만 결과가 1
	| (OR)	두 개의 비트 값 중 하나라도 1이며 결과가 1
	^ (XOR)	두 비트가 같으면 0, 다르면 1을 결과를 낳는다.
	~ (NOT)	**[단항 연산자]** 비트 값을 반전한다. (0 또는 1)
이동 연산	<<, >>	비트를 화살괄호 방향에 따라 왼쪽 또는 오른쪽으로 이동한다. 왼쪽으로 채워지는 비트값은 기존 값의 부호 비트와 동일하다.
	>>>	비트를 오른쪽으로 이동시킨 후 왼쪽으로 채워지는 비트값은 항상 0이다.

주의할 점은 변수의 비트를 이동하는 문장 자체는 그 변수의 값을 변화시키지 않는다. 변수값을 변화시키려면 해당 연산을 변수에 대입하여야 한다.

```
int num1 = 5; //2진수: 00000101
int num2 = 0B00001010; //정수 10을 2진수로 나타낸 형태
System.out.println( num1 << 2 ); //2비트 좌이동: 00010100(20)
num2 = num >> 1; //1비트 우이동: num2 = 00000101(5)
System.out.println(num2);
```

▪ 대입 연산자 =, +=, -=, *=, /=, %=, <<=, >>=, >>>=, &=, |=, ^= :

변수에 값을 대입하는 연산자이다. 이항 연산자 중 순위가 가장 낮다. 대입 연산자와 다른 연산자를 조합한 형태로, 두 항을 전자의 연산자로 연산한 값을 왼쪽 항의 변수에 대입한다.

a+=b; = **a = a+b;**

■ 삼항 연산자 항이 세 개인 연산자 – 조건 연산자

▪ 조건 연산자

a>b ? a : b ;

조건식 참 거짓

차례대로 조건식, 조건식이 참인 경우 반환하는 값 그리고 조건식이 거짓인 경우 반환하는 값 총 3개의 항으로 이루어진다. 주로 어떤 변수에 값을 대입하기 전 조건을 거를 때 사용된다. **num = a>b ? a : b ;**

■ 연산자 우선순위

괄호는 수학 연산과 같이 모든 연산에 있어서 최고 우위이다. 이를 잘 활용할 수 있어야 한다.

1	일차식	() [] .
2	단항 연산	! ++ -- + -
3	산술 연산	% / + -
4	비트 이동	<< >> >>>
5	관계 연산	< > <= >= == !=
6	비트 연산	& ^ |
7	논리 연산	&& ||
8	삼항(조건) 연산	?:
9	대입 연산	= += -= *= %= /=

5 : 조건문

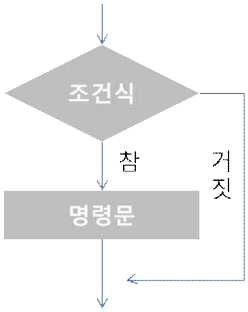
조건문이란 어떠한 조건을 제시한 후, 그 조건이 참 또는 거짓의 결과를 반환하여 그에 따른 문장을 실행할 수 있도록 만든다. 여러 가지의 조건문들이 있다.

■ if문, if-else문

조건문의 기본이 되는 가장 간단한 형식이며, 조건에 따라 선택적으로 문장이 실행되는 구조문이다. ' 만약 ~라면' 라는 뜻을 가지며 일반적으로 중괄호 안에 해당 조건이 참일 때 실행할 문장을 기입한다.

- if

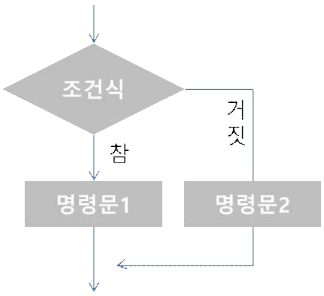
```
If(조건식){
        참일때 실행할 수행문;
}
```



조건식이 거짓인 경우, 중괄호 안의 명령문을 실행하지 않는다.

- if-else

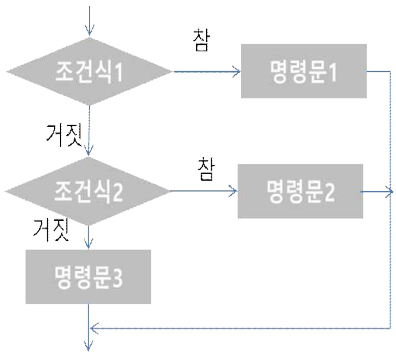
```
If(조건식){
        참일때 실행할 수행문;
}else{
        거짓일때 실행할 수행문;
}
```



예외일 경우의 명령을 가지는 if문이다.

- if-else if...else

```
If(조건식1){
        조건식1이 참일때 실행할 수행문;
}else if(조건식2){
        조건식1이 거짓이고 조건식2가 참일때 실행할 수행문;
}else{
        조건식1과 조건식2가 거짓일때 실행할 수행문;
}
```



조건이 여러개 일 경우 사용하는 조건문이다. 단계적으로 명령이 걸러진다.

예제 [If_else01.java]

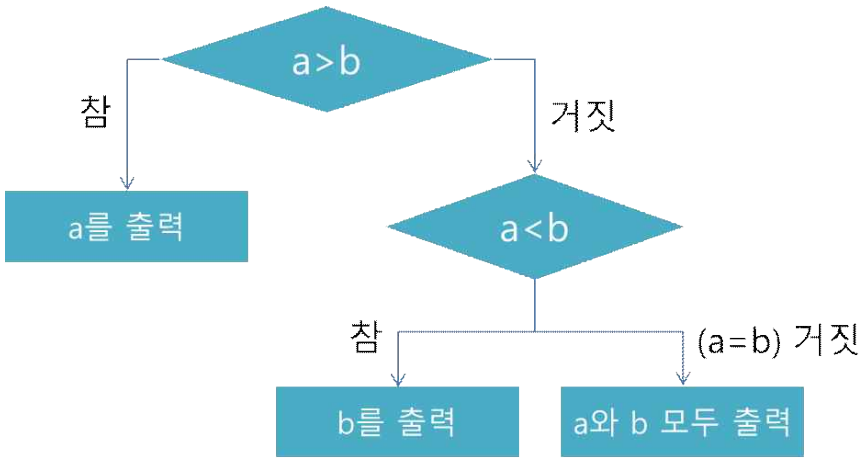
입력받은 두 정수 a, b 중 a가 b보다 크면 a를 출력하고, b가 a보다 크면 b를 출력하며, a와 b가 같을 경우 a, b 모두 출력하는 조건문을 만드시오.

```
public class If_else01 {
     public static void main(String args[]) {
          Scanner scan = new Scanner(System.in);
          System.out.print("a = ");
          int a = scan.nextInt();
          System.out.print("b = ");
          int b = scan.nextInt();
          System.out.println("---------");
          if(a>b){
               System.out.println(a);
          }else if(a<b){
               System.out.println(b);
          }else{
               System.out.println(a+", "+b);
          }
     }
}
```

결과

```
a = 17
b = 20
---------
20
```

위 예제의 조건 도형도(아래)



예제 [If_else02.java]

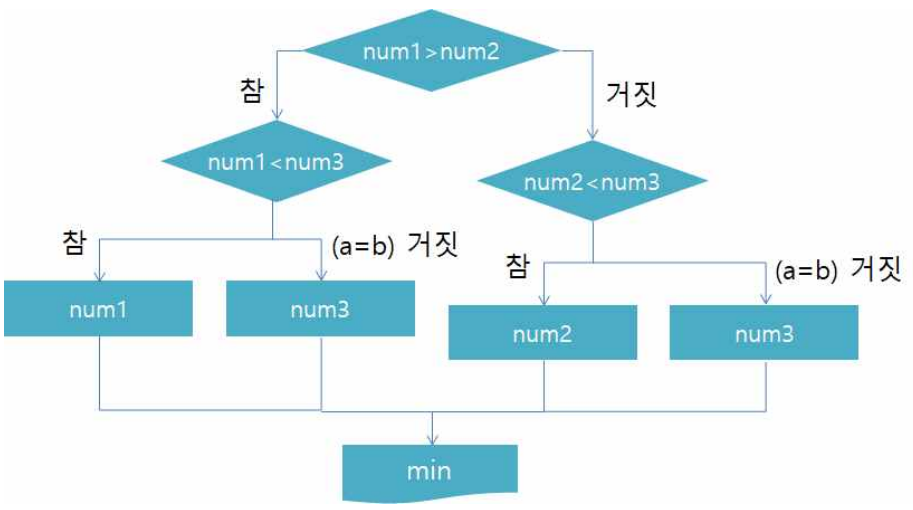
세 수를 입력 받아 가장 작은 값을 구하시오.

```
public class If_else02 {
     public static void main(String args[]) {
          Scanner scan = new Scanner(System.in);

          int min;
          System.out.print("첫번째 수: ");
          int num1 = scan.nextInt();
          System.out.print("두번째 수: ");
          int num2 = ("---------");
          System.out.print("세번째 수: ");
          int num3 = scan.nextInt();

          if(num1<num2){
              if(num1<num3) min = num1;
              else min = num3;
          }else if(num1>num2){
              if(num2<num3) min = num2;
              else min = num3;
          }
          System.out.println("가장 작은 수: "+min);
     }
}
```

결과

```
첫번째 수: 6
두번째 수: 3
세번째 수: 11
가장 작은 수: 3
```

위 예제의 조건 도형도(아래)



예제 [If_else03.java]

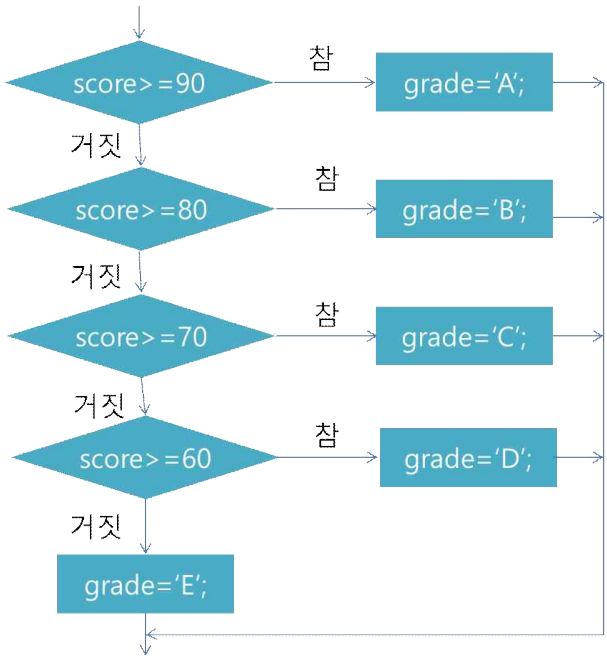
if문을 이용해 입력받은 점수에 대한 성취도를 매기는 프로그램. 100점 만점일 때, 90점 이상이면 A, 80점 이상이면 B, 70점 이상이면 C, 60점 이상이면 D, 나머지는 E를 반환한다.

```
public class If_else03 {
     public static void main(String args[]) {
          Scanner scan = new Scanner(System.in);
          int score = scan.nextInt(); //입력받을 점수
          char grade; //성취도 등급

          if(score>=90) grade = 'A';
          else if(score>=80) grade = 'B';
          else if(score>=70) grade = 'C';
          else if(score>=60) grade = 'D';
          else grade = 'E';

          System.out.println("점수:"+score+"/성취도:"+grade);
     }
}
```

결과

```
85
점수:85/성취도:B
```

위 예제의 조건 도형도(아래)



■ switch~case문

switch문이란 조건식에 대한 상수값에 따라 여러 갈래로 명령이 분기되는 조건문이다. 다음 예시를 보자.

형태

```
switch(조건식 또는 변수){
case 1: 명령문1;  break;
case 2: 명령문2;  break;
case 3: 명령문3;  break;
default: 명령문4;
}
```

예시

```
switch(session){
case 1: name_color='파란색'; break;
case 2: name_color='초록색'; break;
case 3: name_color='노란색';
default: System.out.println("잘못된 입력입니다.");
}
```

switch문은 상황에 따라서 if문보다 간결하고 효율적이다. 반드시 조건문의 결과를 먼저 작성하여야 하며, 결과값은 모두 리터럴 상수이다. 결과값으로 변수나 선언된 상수가 올 수 없다. break문이 뒤따라 오지 않을 시 한 결과값에 대한 명령문을 수행한 후 그 아래의 결과값들에 대한 명령문들을 모두 수행한 후 switch문을 빠져나오는게 기본이다. default 키워드를 이용해 if문의 else와 같은 예외 결과 처리 또한 가능하다.

예제 [SwitchEx01.java]

키보드로부터 두 수와 사용할 연산을 입력받아 두 수에 대한 연산을 실행한 결과를 출력하는 프로그램을 switch문을 이용해 작성하라.

```
public class SwitchEx01 {
    public static void main(String args[]) {
        Scanner scan = new Scanner(System.in);

        int result = 0; //오류 방지를 위한 초기값 부여

        System.out.print("두 수 입력: ");
        int num1 = scan.nextInt(); int num2 = scan.nextInt();
        System.out.println("1.덧셈 2.뺄셈 3.곱셈 4.나눗셈");
        System.out.print("사용할 연산: ");
        int sol = scan.nextInt();

        switch(sol) {
        case 1: result=num1+num2; break;
        case 2: result=num1-num2; break;
        case 3: result=num1*num2; break;
        case 4: result=num1/num2; break;
        default: System.out.println("잘못된 입력입니다.");
        }
        System.out.println("연산 결과: "+result);
    }
}
```

결과

```
두 수 입력: 4 6
1.덧셈 2.뺄셈 3.곱셈 4.나눗셈
사용할 연산: 3
연산 결과: 24
```

예제 [SwitchEx02.java]

키보드로부터 초등학년을 입력 받아 학년과 그에 맞는 깃발 색을 출력하시오.
깃발 색: 1학년-노랑, 2학년-주황, 3학년-파랑, 4학년-분홍, 5학년-초록, 6학년-보라

```
public class SwitchEx02 {
    public static void main(String args[]) {
        Scanner scan = new Scanner(System.in);

        int session = 0;
        String flag_color = NULL; //오류 방지를 위한 초기값 부여

        System.out.print(“학년입력:");
        session = scan.nextInt();

        switch(sol) {
        case 1: session=“노랑”; break;
        case 2: session=“주황”; break;
        case 3: session=“파랑”; break;
        case 4: session=“분홍”; break;
        case 5: session=“초록”; break;
        case 6: session=“보라”; break;
        default: System.out.println("잘못된 입력입니다.");
        }
        System.out.println(“학년:"+session+“\n깃발색:”+flag_color);
    }
}
```

결과

```
학년입력:5
깃발색:초록
```

6 : 반복문

반복문이란 비슷하거나 같은 특정 명령문을 반복하는 것을 말한다. 이것은 같거나 비슷한 명령을 반복해야 할 때 코드의 간결화를 위해 사용한다. 프로그램 내에선 비슷하거나 같은 특정 동작들을 반복해야 할 경우가 많은데, 반복문이 유용하게 쓰인다.

■ for문

반복문 중에 가장 많이 사용된다. 조건식이 참인 동안 문장이나 블록을 반복한다. 형태는 다음과 같다.

실행문이 한 줄 이상일 때

```
for( 초기식 ; 조건식 ; 증감식 ) {
    실행문;
}
```

실행문이 한 줄 일 때

```
for( 초기식 ; 조건식 ; 증감식 ) 실행문;
```

```
for( 초기식 ; 조건식 ; 증감식 )
    실행문;
```

초기식은 반복문에 진입하는 초기에만 실행되는 초기화 등의 내용이다. 반복할 변수를 초기화하는 역할을 한다. 조건식, 증감식 그리고 실행문이 차례로 반복된다. 증감식에서 반복 변수 값이 증감되면서 for문이 반복되고, 조건식에 맞지 않는 경우 반복문에서 빠져나온다.

* 첫 조건이 거짓이면 반복문이 한번도 실행되지 않을 수도 있다.

예제 [ForEx01.java]

for문을 이용해 1부터 10까지와 10부터 1까지의 수를 출력하라.

```
public class ForEx01 {
     public static void main(String args[]) {
          for(int i=0; i<10; i++){
              System.out.print((i+1)+" ");
          }
          System.out.println("");
          int j;
          for(j=10; j>=1; j--){
              System.out.print(i+" ");
          }
     }
}
```

결과

```
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
```

* 반복하는 변수명은 일반적으로 i나 j를 주로 사용한다.

예제 [ForEx02.java]

for문을 이용해 1부터 100까지의 수의 합을 출력하시오.

```
public class ForEx02 {
     public static void main(String args[]) {
          int sum;
          for(int i=1; i<100; i++){
              sum+=i;
          }
          System.out.print(sum);
     }
}
```

결과

```
5050
```

예제 [ForEx03.java]

for문을 이용해 1부터 100까지의 홀수를 모두 출력하는 프로그램을 작성하시오.

```
public class ForEx03 {
     public static void main(String args[]) {
          for(int i=1; i<100; i++){
              if(i%2==1) System.out.println(i);
          }
     }
}
```

결과

```
1 3 5 7 9 11 13 15 … 95 97 99
```

예제 [ForEx04.java]

중첩 for문을 이용해 구구단을 출력하시오.

```
public class ForEx04 {
     public static void main(String args[]) {
          for(int i=2; i<=9; i++)
              for(int j=1; j<=9; j++)
                  System.out.println(i+“*”+j+“=”+i*j);
     }
}
```

결과

```
2*1=2
2*2=4 …
9*9=81
```

■ while문

조건문이 참인 동안만 문장이나 블록을 반복 실행한다. 조건식을 먼저 테스트하므로 반복문이 한번도 실행되지 않을 수도 있다. 초기식과 증감식이 필수는 아니다.

형태

```
[초기식;]
while( 조건식 ) {
        실행문;
        [증감식;]
}
```

예제 [WhileEx1.java]

while문을 이용하여 1부터 100까지의 수 중 3의 배수만 모두 출력하시오.

```
public class WhileEx1 {
     public static void main(String args[]) {
          int i=1;
          while(i%3==0) {
                System.out.print(i+" ");
                i++;
          }
     }
}
```

결과

```
3 6 9 12 15 18 21 ··· 96 99
```

■ do-while문

조건식이 참인 동안만 문장이나 블록을 반복실행하는데, while문과 달리 조건식을 실행문 다음에 수행하므로 반복문이 무조건 1회 이상 실행된다.

형태

```
do{
        실행문;
} while( 조건식 );
```

예제 [ForEx01.java]

0이 입력될 때까지 점수를 입력 받은 후 점수의 합과 평균을 출력하시오.

```
public class ForEx01 {
     public static void main(String args[]) {
          Scanner scan = new Scanner(System.in);
          int score, sum = 0, i=-1;
          do{
               i++;
               score=scan.nextInt();
               sum+=score;
          }while(score!=0);
          System.out.print("총합:"+sum+"\t평균:"+sum/i);
     }
}
```

결과

```
100
90
0
총합:190    평균:95
```

■ 분기문

- break 키워드: 조건문 또는 반복문을 빠져나오고자 할 때 사용한다. break문을 만나는 동시에 해당 조건문 또는 반복문의 바로 다음 명령을 실행한다. 조건문의 경우 switch문에서 주로 사용되는데, 이것은 특정 결과값에 대한 명령을 수행한 후 그 아래의 명령문들이 실행되지 않고 switch문을 빠져나오기 위함이다. 반복문 속에서는 해당 회차에서 특정 조건에 맞는 경우 반복문을 빠져나오게 하도록 사용하곤 한다.
- continue 키워드: 반목문에서 continue 이후의 문장은 실행하지 않고 그 다음 반복을 계속할 때 사용된다.

예제 [ContinueEx.java]

분기문 continue를 사용하여 1부터 100까지의 수 중에서 17의 배수를 제외한 모든 수의 합을 출력하시오.

```
for(int i=1; i<=100; i++) {
        if(i%17==) continue;
        System.out.print(i+“ ”);
}
```

결과

```
1 2 3 … 16 18 19 … 33 35 36 … 50 52 53 … 98 99 100
```

7 : 메서드 1부

메서드란 특정 작업을 수행하는 명령어들의 모음이다. 특정한 일을 수행하여 변수와 더불어 클래스를 구성하는 주요 요소이다. 자바 프로그램은 다양하고 많은 클래스들로 구성되는데, 클래스 내에서 변수는 정적 속성이라면 메서드는 동적인 속성이다(클래스에 대해선 뒤에서 자세히 다룬다). 변수처럼 선언 부분이 있으며 사용하고자 하는 부분에서 메서드 이름으로 호출한다. 메서드 실행이 종료되면 호출 다음 부분으로 돌아와 그 다음 명령을 이어간다. 다음은 그 형태와 예시이다. 일반적으로 메서드이름은 소문자로 시작한다.

형태

```
접근제어자 반환값유형 메소드이름 ( 형식매개변수목록 … ) {
        실행문;
        …;
        return 반환값; //반환유형이 void인 경우 반환값(return)이 없다.
}
```

예1

```
public int getAge( ) { //매개변수가 없다.
        int age = 18;
        return age; //반환유형을 int로 선언했으므로 int형만 반환 가능.
}
```

예2

```
public void setTitle( String title ) {
        bookTitle = title;
}
```

■ 매개 변수

매개 변수란 메서드 호출과 실행 과정에서 넘겨지는 변수를 말한다. 그 개수에는 제한이 없다. 메서드 이름 옆에 붙는 괄호 안에 메서드를 실행하기 전 넘겨줄 매개변수를 기입하는데, 메서드가 받을 매개변수의 정보는 메서드 선언과정에서 반드시 작성해야한다. 때문에 호출시 넘겨주는 매개변수의 이름과 메서드 실행시 받는 매개변수의 이름은 달라도 되나 자료형과 매개변수의 개수는 반드시 같아야 한다. 넘겨받은 매개변수 이름이 호출에 사용된 변수명과 달라도 두 값은 같으며 메서드 내에선 메서드 선언에도 사용된 매개변수 이름으로 사용하면 된다.

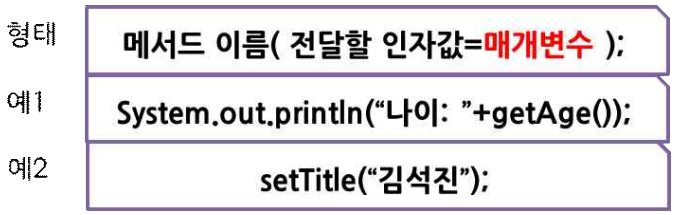
■ 지역 변수(local variable)

먼저 예1을 보자. ‘ int age’ 는 메소드 내에서만 정의되어 사용하는 변수라는 점에서 ‘ 지역 변수’ 라고 부른다. 이 변수는 해당 메서드 실행이 끝나면 형식매개변수와 함께 메모리에서 사라진다. 하지만 지역 변수 값을 return 명령으로 메서드 밖으로 반환하는 것이 가능하다.

■ 반환값이 없는 유형 void

예2처럼 return할 반환값이 없는 메서드를 선언할 땐 선언부의 반환값유형 자리에 void를 작성한다. void가 아닌 반환유형이 있는 경우엔 return문이 반드시 필요하다. void타입의 메서드를 사용하는 이유 중 하나는 반환값이 있는 메서드는 메서드 실행 후 반드시 하나의 값만 반환이 가능하지만 void타입은 반환과 관계없이 어떤 명령문의 결과든 자유롭게 낼 수 있기 때문이다.

그렇다면 이제 메서드의 호출 방법을 알아보자. 호출 방법은 간단하다. 위의 선언된 예시 메서드를 호출해보았다.



- 주의할 점

1. 메서드는 호출 전에 반드시 객체 내에 선언되어 있어야 한다. 호출에 대해 상하 위치는 상관 없다.
2. void타입의 메서드는 자체 내에서 출력문 실행을 포함한 모든 명령이 가능하지만 반환형 메서드의 경우 결과적으로 한 개의 값만 반환이 가능하므로 메서드 호출과 동시에 출력하거나 반환값을 메서드 밖의 변수에 대입하여야만 반환값의 결과를 얻을 수 있다.

예제1 [MethodEx01.java]

전달받은 물건의 가격의 30%를 할인시킨 금액을 반환시키는 메서드를 작성하고 테스트하는 클래스를 작성하시오.

```
public class MethodEx01 {
    public static void main(String args[]) {
        Scanner scan = new Scanner(System.in);
        System.out.print("상품의 정가: ");
        int goodsPrice = scan.nextInt();
        System.out.println("상품의 할인가: "
            +discountPrice(goodsPrice)+"원");
    }

    public int discountPrice(int price) {
        return price*0.7;
    }
}
```

결과

```
상품의 정가: 85000
상품의 할인가: 59500원
```

8 : 객체와 클래스

객체 지향 기술은 소프트웨어 부품화와 소프트웨어 컴퍼넌트의 재사용을 목적으로 한다. 자바는 객체지향 언어이며, 그렇기 때문에 다양한 클래스들의 집합으로 프로그램이 이루어질 수 있다. 클래스를 공부하기 전 객체지향의 이론을 짚고 넘어가보자. 자바가 나오기까지 역대 객체 지향 언어를 소개해보았다.

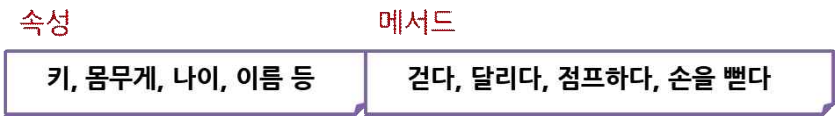
1. 1960년대 시뮬라 : 실용적 언어로는 발전하지 못하였으나 객체 지향 이론을 소개하였다.
2. 1970년대 스몰토크 : 순수 객체 지향 언어로 인정받는 최초의 언어이며, 이후 에이다, 이펠 등의 객체지향언어에 영향을 미쳤다. 스몰토크는 프로그램 언어뿐만 아니라 프로그램 개발 방법론 등에 영향을 미쳤다.
3. 1990년대 C++ : 자바의 어머니라고 할 수 있다. 범용적으로 사용되기 시작한 객체 지향 언어로 GUI 발전에 따라 기능이 향상되었다.

■ 객체지향 프로그래밍의 특징

1. 대규모 프로젝트 개발에 유연하다.
2. **캡슐화** : **자료 추상화**라고도 하는데, 객체 내부에서만 알고 있어야 하는 정보는 숨기고 외부에 보여야 할 정보만을 표현함으로써 프로그램을 간단히 만드는 것이다. 객체를 사용하는 측에서는 캡슐화된 속성과 메서드가 실제로 어떻게 처리되는지는 알 필요 없이 단지 객체를 사용할 수 있는 인터페이스를 통해 사용만 하면 된다. 캡슐화의 방법으론 접근제어자나 인터페이스를 사용하는 방법 등이 있다.(접근제어자에 관해선 후에 다룸)
3. 상속 : 새로운 클래스가 기존의 클래스의 자료와 연산을 이용할 수 있게 하는 기능이다. 만약 정의하려는 클래스가 이미 정의된 클래스가 갖는 속성과 메서드에 약간의 속성과 메서드가 추가되어야 한다면 기존의 클래스를 상속받아 새로 필요한 속성과 메서드를 추가한다.
4. 다형성 : 여러개의 형태를 갖는다는 의미로, 메서드의 형태가 다양하거나 동일한 이름의 메서드가 약간씩 다른 의미를 가질 수 있다는 것을 뜻한다. 종류로는 대표적으로 생성자 메서드처럼 메서드를 중복정의할 수 있는 메서드 **오버로딩**과 상속관계 등에서 메서드 재정의가 가능한 메서드 **오버라이딩**이 있다.

■ 객체(Object)

객체는 사람들이 의미를 부여하고 분류하는 논리적 단위로, 실세계에 존재하는 명사로서의 성질을 갖는 모든 것이라고 정의할 수 있다. 객체는 실제 프로그램에서 필요한 특성만을 중심으로 모델링되며, 정적인 특성(attribute)과 동적인 특성(method)으로 구성된다. 정적인 특성은 주로 변수가 되고, 동적인 특성은 메서드라고 할 수 있다. 너무 흔하지만 사람에 비유해보겠다.



■ 클래스

그렇다면 클래스란 무엇일까? 바로 객체를 프로그래밍 언어로 정의한 것이다. 클래스는 동일한 속성과 메서드를 가진 여러개의 객체를 생성하는 형판의 역할을 한다. 자바 파일 이름은 해당 파일의 대표 클래스의 이름이기도 하다. 위 문단에서 비유한 사람 객체를 클래스로 정의해보았다.

```
public class Person {
          int height, weight, age;
          String name;

          public void walkH(int step){
                    weight -= step*0.05;
          }
          public void runH(int step){
                    weight -= step*0.1;
          }
          public void jump(){
                    weight -= 0.05;
                    height += 0.05;
          }
          public void putHand(){
                    System.out.println("My name is "+name+"!");
          }
}
```

객체지향 프로그래밍은 3단계로 나누어볼 수 있다.

1. **실세계 객체의 모델링** : 현실의 객체로부터 프로그램에서 사용하고자 하는 요소들을 가져오는 것. 학생 객체를 만들고자 할 때 ' 학번', ' 이름', ' 학년', ' 공부를 하다' 등 프로그램에서 필요한 부분만 속성으로 사용하는 것이다.
2. **클래스 정의** : 모델링한 객체의 요소들을 프로그래밍 언어로 구현하는 과정이다. 정의된 클래스 코드에선 위 Person 클래스처럼 미리 모델링한 정적 속성과 동적 속성을 가지고 있다.
3. **인스턴스 객체의 생성 및 사용** : 앞서 클래스를 형판이라고 설명했는데, 어떤 클래스를 틀로 하여 그로부터 다양한 속성값을 갖도록 생성한 것들을 인스턴스(instance) 객체라고 한다. 아래는 위 Person 클래스로부터 만든 인스턴스 객체이다.

형태	예시

클래스명 인스턴스객체명 = new 클래스명(매개변수);
인스턴스객체명.정적속성
인스턴스객체명.동적속성(매개변수);
	Person baek = new Person( );
	baek.height = 60;
	baek.walkH(15);

Person은 부모 객체, baek은 Person의 자식인 인스턴스 객체가 된다. 인스턴스 객체의 속성을 사용하는 방법으론 인스턴스 객체명 옆에 ' .' 을 붙이고 속성 변수명 또는 메서드명을 쓴다. 하나의 클래스로부터 나온 인스턴스들은 속성 구성은 모두 같아도 그 값은 다를 수 있다. 메서드의 경우 괄호와 매개변수를 잊지 말자.

예제1 [Triangle.java]

삼각형의 넓이와 둘레를 구하고자 한다. 삼각형을 모델링하여 클래스를 정의하시오.

```
public class Triangle {
    double bottomside;
    double side1;
    double side2;
    double height;
    double area;
    double sideSum;

    public double getAarea() {
           area = bottomside * height;
           return area;
    }
    public double getSideSum() {
           sideSum = bottomside+ side1+ side2;
           return sideSum;
    }
}
```

예제2 [Point.java]

삼각형의 넓이와 둘레를 구하고자 한다. 삼각형을 모델링하여 클래스를 정의하시오.

```
public class Point {
           private double x, y;


           public Point(double x, double y) {
                      this.x = x;
                      this.y = y;
           }

           public void printPoint() {
                      System.out.print("[점의 현재위치: ("+x+", "+y+")]\n");
           }
           public void raisePoint(double x_add, double y_add) {
                      x+=x_add;
                      y+=y_add;
           }
}
```

■ 클래스의 접근제어자

접근제어자	사용범위
public	모든 클래스에서 접근이 가능하다.
final	해당 접근제어자로 선언된 클래스로부턴 새로운 클래스가 상속되어 생성될 수 없다. 즉, 하위 클래스를 가질 수 없다.
abstract	추상 클래스로, 객체를 생성할 수 없는 클래스를 의미한다,.

* 클래스의 접근제어자의 경우 main메서드를 가진 클래스에만 public을 사용한다.

■ 자바의 최상위 클래스 Object

Object란 자바의 최상위 클래스로서 사용자가 생성한 클래스는 모두 Object를 자동으로 상속한다. 몇가지 Object는 메서드는 클래스 생성시 오버라이딩(재정의)해서 사용하기도 하는데, 다음은 그 중 몇가지이다.

- equals() : 두 인자를 비교한다.
- clone() : 객체의 복제에 이용한다. 이 메소드를 사용할 수 있는 라이브러리 객체가 정해져 있다. 메서드 사용시 값이 그대로 복제된다.
- toString() : 객체의 클래스 이름과 메모리 참조 내용(속성)을 문자열로 반환하며, print 메서드에 참조형 변수 이름을 사용하면 자동 호출된다.

  -자동 생성 방법: 마우스 우클릭 > source > Generate toString()

예제 [Sample_class.java]

Object의 toString메서드 테스트 예제

```
class Account{
    public String account_name, openDate, account_num;
    private int deposit;
    public int getDeposit() { return deposit; }
    public void setDeposit(int amount) { this.deposit = amount; }
    @Override
    public String toString() {
        return "Account [account_name="+account_name+", openDate="
         + openDate+", account_num="+account_num+", getDeposit()="+getDeposit()+"]";
    }
}
public class Sample_class {
    public static void main(String args[]) {
        Account vjin = new Account();
        vjin.account_name = "태형아머리좀잘라라"; vjin.openDate = "2020년";
        vjin.account_num = "5252609";
        vjin.setDeposit(300000);
        System.out.println(vjin);
    }
}
```

결과

```
Account [account_name=태형아머리좀잘라라, openDate=2020년, account_num=5252609, getDeposit()=300000]
```

* toString 메서드 없이 인스턴스를 출력할 경우 인스턴스의 주소값이 출력된다.

9-1 : 변수 2부

앞서 변수 1부에선 변수의 기본 개념 종류, 활용을 배웠다. 객체지향과 클래스에 대해 자세히 공부했으니 이제 본격적으로 객체지향 언어 자바의 클래스 속 변수의 다양한 활용에 대해 다루어보려고 한다.

■ 멤버변수와 접근제어자

앞에서 멤버변수에 대한 명확한 정의가 없었다. 멤버변수란 클래스를 정의할 때 메서드 밖에 선언하는 변수이다. 클래스 안에서는 객체가 가질 수 있는 정적인 속성을 나타낸다. 메서드가 종료되면 메모리에서 사라지는 지역변수와 달리 프로그램 실행동안 메모리에 계속 남아있다.

클래스 내에서 멤버변수 또는 메서드의 접근 권한을 지정할 수 있다.

접근제어자	사용범위	
private	오직 해당 클래스 내 (동일 패키지 내 타 클래스 접근 불가능)	
(default)	동일 패키지 내 클래스	EX) String school; int age;
protected	동일 패키지 내 클래스 또는 하위(상속) 클래스	
public	모든 타 클래스 (*보안성 낮음)	

접근권한을 나타내는 접근제어자(식별자)는 선언시 맨 앞에 위치한다.

```
private int deposit; protected String name; public String account_num;
```

예제 [Sample_class.java]

접근제어자 테스트

```
class Account{
            String account_name;
            private int deposit;
}

public class Sample_class {
      public static void main(String args[]) {
            Account vjin = new Account();
            vjin.account_name = "태형아머리좀잘라라";
            vjin.deposit = 525252;
            System.out.println("예금주:"+vjin.account_name+"\n잔액:"+vjin.deposit);
      }
}
```

결과

```
Exception in thread "main" java.lang.Error: Unresolved compilation problen
The field Account.deposit is not visible
The field Account.deposit is not visible ...
```

- 접근제어의 필요성

기존 절차지향 언어 C에선 프로그램의 전역변수가 변경된 경우 해당 전역변수가 사용된 나머지 모든 프로그램이 변경되어야 한다. 때문에 일관된 수정이 어렵고 오류를 범하기 쉬운데, 자바와 같은 객체지향 언어의 경우는 캡슐화 개념에 기초하여 객체의 상태를 나타내는 데이터는 그 객체의 메소드에 의해서만 변경되어야 한다. 프로그램의 안전성과 체계화를 위해선 다른 객체가 직접 접근하여 상태를 바꾸는 것은 불가능하거나 어렵게 하는 것이 좋다.

■ 정적(static) 변수 – 클래스 변수

인스턴스 객체를 생성하지 않고도 사용할 수 있는 변수가 있는데 이런 변수를 정적 변수 또는 클래스 변수라고 한다. 생성시기는 클래스 로딩시이며, 생성 위치는 클래스이다. 때문에 인스턴스 개수와 상관없이 오직 1개이다.

- 특징

```
static String school;
```

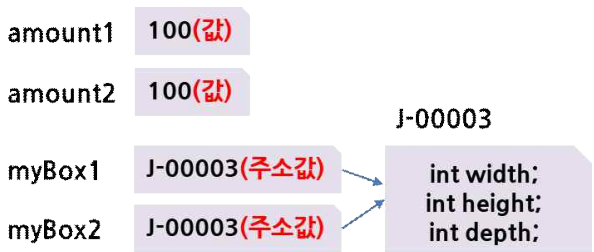
1. 전역변수(global object)의 개념이다.
2. 클래스가 로딩되는 과정에서 기억공간이 한번만 확보된다.
3. 해당 클래스의 모든 인스턴스가 공유한다.
4. 객체들 사이의 통신이나 공통되는 속성을 표현하는데 사용한다.
5. 일반 변수의 경우 인스턴스 이름으로 접근하는데 반해 클래스 변수는 클래스 이름으로 접근한다(클래스 변수를 사용하기 위해 인스턴스를 따로 정의할 필요가 없다).
   ex) 클래스명.클래스변수명
6. 클래스 선언시 유형자/반환유형자 앞에 식별자‘ static’ 을 붙인다.

■ 객체 참조 변수

객체 참조 변수란 생성된 객체, 즉 인스턴스를 지정하는 변수이다. int나 double 같은 기본자료형이 아닌, String과 배열 등의 참조 자료형에 속한다. 이 참조 변수들은 값에 대한 참조 즉, 주소 값을 가진다.‘ 상수’ 단원의 복습 예제에서 인스턴스 변수는 그 주소값을 가지고 있다는 걸 확인하였다. 그에 따라 인스턴스가 가진 주소값만 바꾸어 참조하는 객체를 쉽게 바꿀 수도 있다.

코드

```
int amount1 = 100;
int amount2 = amount1;

//클래스 'Box' 의 존재
Box myBox1 = new Box();
Box myBox2 = myBox1;
```



9-2 : 메서드와 오버로딩 (메서드 2부)

앞서 메서드 1부에선 메서드의 기본 개념과 활용을 배웠다. 객체지향과 클래스에 대해 자세히 공부했으니 이제 본격적으로 객체지향 언어 자바의 클래스 속 메서드의 다양한 활용에 대해 다루어보려고 한다. 또한 객체지향의 특징 중 하나인 오버로딩에 대해서도 다뤄보려고 한다.

■ this

이 식별자는 메서드 내에서 주로 사용되는데, 현재 생성되어 사용중인 인스턴스 객체 자신을 의미한다.

이 식별자를 사용함으로써 의미를 명확하게 하여 가독성을 높일 수 있으며, 객체 변수나 매개 변수의 이름을 같게 사용할 수 있다는 장점이 생긴다. 다음은 사용되는 경우이다.

[접근제어자] 생성자이름 (매개변수...):

1. 생성자 메서드나 메서드의 매개변수 이름이 클래스의 멤버 변수 이름과 동일할 경우
2. 현재 같은 클래스 내의 다른 생성자 메서드를 호출하는 경우

1.
```
Class Student{
  String name;
  int grade;
  public void setAtt(String name, int grade){
        this.name = name;
        this.grade = grade;
  }
}
```

* 생성자 내에서 this()문은 반드시 첫 줄에 위치해야 한다.(에러방지)

2.
```
Class  Box{
  int width;
  int  height;
  int depth;
  public Box(){ //1번생성자
    this(1,1,1);  //3번호출
  }
  public Box(int w, int h){ //2번생성자
    this(w,h,1); //3번호출
  }
  public Box(int w, int h, int d){ //3번생성자
    width = w;
    height = h;
    depth = d;
  }
}
```

■ 생성자 메서드

생성자 메서드란 클래스로부터 객체를 생성하고 객체의 초기화를 담당하는 특수한 메서드로, 객체가 생성될 때 무조건 수행된다. 외형적으로 일반 메서드와 비슷하지만 용도와 형태는 다른점이 많다. 다음은 그 형태이다.

접근제어자는 public, proctected 등을 사용할 수 있다.

클래스 'Account'

1.
```
public Account(String name, int amount) {
        this.name = name;
        this.amount = amount;
}
```

2.
```
Account acct = new Account("전정삼", 20000);
```

▪ 특징

1. 호출시 new 연산자 함께 사용한다.
2. 객체 생성시 멤버 변수의 초기화를 담당하기도 한다.
3. 이름은 클래스 이름과 동일하다(첫 문자는 대문자).
4. 반환 유형이 없다(접근제어자 public 뒤에 곧바로 메서드명이 붙는 형태).
5. 생성자는 객체 생성시 반드시 호출되어야 하는 메서드이다. 그러나 반드시 생성자 메서드를 갖고 있을 필요는 없다. 때문에 생성자 메서드가 없는 경우 JVM이 자동으로 기본 생성자를 삽입한다(이렇게 생성된 생성자 메서드는 코드 내에서 보이지 않아도 프로그램 상에는 존재한다).
6. 사용자가 정의한 생성자 메서드가 있는 경우 JVM이 삽입한 기본 생성자 메서드는 사라진다.

▪ 사용 방법

1. 생성자 메서드를 통해 객체 변수들의 초기 값들을 줄 수 있다. 부여할 초기값을 매개변수로 받는다.
2. 객체를 가리키는 참조 변수를 정의하고 생성자 함수를 호출하여 클래스의 객체를 만든다. 이때 부여할 초기값을 생성자 함수의 매개변수로 넘겨준다.

■ 생성자 오버로딩

생성자 메서드는 매개변수의 종류나 개수를 달리하여 여러 개가 존재할 수 있다. 이것은 객체지향에서 다형성의 특징을 갖는 오버 로딩의 대표적인 경우이다. 생성자 메서드명은 모두 클래스명과 같기 때문에 구분을 위해서 호출 시 사용할 생성자에 맞게 매개변수를 대입하는 것이 중요하다. 오버로딩에 관해선 후에 자세히 다룬다.

예제 [ConstructorExam.java]

```
public class ConstructorExam {
     public static void main(String[] args){
          Student kim = new Student(); //1번 생성자메서드가 사용
          Student jang = new Student("장민재"); //2번 생성자메서드 사용
          System.out.println("학생의 이름은 "+kim.name+"입니다.");
          System.out.println("학생의 이름은 "+jang.name+"입니다.");
     }
}
class Student{
     String name; int grade; int clas; int number; String telephone;
     public Student(){}; //매개변수가 없는 생성자메서드1
     public Student(String n){ name = n }; //매개변수가 하나있는 오버로딩된 생성자메서드2
}
```

예제 [Square.java]

클래스 변수(static) 예제. Square 클래스를 만든 후 그로부터 생성되는 객체들에 일련번호(isNum)을 부여하고 인스턴스가 생성될 때마다 자동으로 일련번호를 부여하는 생성자메서드를 만드시오.

```
class Square {
          int width;
          int height;
          int idNum;
          static int SqId=0;

          public Square() { //클래스와 동일한 이름의 생성자함수
             idNum = SqId++;
            //Square 생성자 함수를 이용하여 객체를 생성 후 SqId를 증가
          }
}
```

■ 메서드의 오버로딩

메서드 오버로딩은 생성자의 오버로딩과 동일한 개념이며, **같은 클래스 안에 매개변수의 개수, 타입, 순서를 달리하는 동일한 이름의 메서드가 여러개 존재하는 것**이다. 오버로딩을 통해 다형성을 구현할 수 있다. 활용시 리턴타입과 접근제어자가 다른 것은 상관이 없다. 메서드의 접근제어자는 멤버변수의 접근제어자와 같다.

형태 **[접근제어자] [활용방법] 리턴타입 메서드이름 (매개변수)**

▪ 활용방법

식별자	활용
final	하위 클래스에서 오버라이딩될 수 없다.
abstract	추상 메소드로, 추상클래스는 선언부분만 가지고 몸체는 가질 수 없다. 몸체는 서브클래스에서 오버라이딩된다 (추상 클래스를 통해 인스턴스 객체를 생성할 수 없다).
synchronized	스레드를 동기화할 수 있는 기법을 제공하기 위해 사용된다,
static	메서드가 정적의 속성을 가진다.

■ 메서드를 활용한 캡슐화

객체지향 프로그램에선 프로그램의 캡슐화가 중요하다. 대표적으로 클래스의 멤버변수는 접근권한을 private로 외부에서는 숨겨진 형태로 만들고 public으로 지정한 메서드를 통해서만 멤버변수에 접근하도록 하는 것이 안전하다. 일반적으로 private로 지정된 속성의 반환이나 변경을 도와주는 메서드를 getter와 setter라고 한다. 이전 예제에서도 이것을 사용해왔었다.

뿐만 아니라 toString 메서드에서도 속성값을 그대로 사용하기보단 해당 속성값을 반환하는 getter메서드를 사용하는 것이 좋다. 다음은 예시이다.

```
public class Account { //계좌 클래스
          private String acc_num, password; //예금주명, 계좌의비밀번호
          private int deposit; //계좌잔고
          public int getDeposit() { return deposit; }
          public void setDeposit(int deposit) { this.deposit = deposit; }
          public String getPassword() { return password; }
          public void setPassword(String password) { this.password = password; }
          public String getAcc_num() { return acc_num; }
          public void setAcc_num(String acc_num) { this.acc_num = acc_num; }
          @Override
          public String toString() {
          return "Account [getDeposit()=" + getDeposit()+", getPassword()=“
                    +getPassword()+", getAcc_num()=“+getAcc_num()+"]";
          }
}
```

-이클립스에서 getter와 setter 자동생성 : 마우스우클릭 > source > Generate getters and setters

■ 클래스 메서드

변수 2부에서 다룬 클래스 변수와 같은 종류이다. 메서드에도 클래스 메서드가 있다. 대표적으로 main메서드가 있으며, 다음은 클래스 메서드의 특징이다.

1. 클래스를 로딩할 때 생긴다. 정적이다.
2. 클래스 이름을 통해 접근한다(인스턴스 객체 이름으로도 접근이 가능하다).
3. 클래스로부터 생성된 모든 객체가 공유한다.
4. 클래스 메서드 내에선 일반 객체 변수를 사용할 수 없고 클래스 변수만 사용이 가능하다.

예제1 [Person.java] [PersonDriver.java]

사람의 성명을 모델하는 Person 클래스를 설계하고 작성하시오. 그 클래스는 사람의 성과 이름을 나타내고 사람의 성과 이름을 넘겨받아 초기화하는 생성자 메서드, 성을 반환하는 메서드, 이름을 반환하는 메서드 그리고 성과 이름 안에 포함된 문자들의 총수를 반환하는 메서드를 가집니다.
Person클래스를 시험하기 위한 PersonDriver 클래스에선 사용자로부터 성과 이름을 입력받은 후 각 성명에 대해 Person인스턴스를 만들고 성과 이름을 반환한 후 성명의 길이를 출력하는 프로그램을 작성하시오.

```
class Person{
    String lastName, firstName;
    public Person(String last, String first){
        this.lastName = last; this.firstName = first;
    }
    public String getLastName(){ return lastName; }
    public String getFirstName(){ return firstName; }
    public int getLength() { return (lastName+firstName).length(); }
}
public class PersonDriver{
    public static void main(String args[]){
        Scanner scan = new Scanner(System.in);
        String tmp1, tmp2;
        System.out.print("성 입력:"); tmp1 = scan.next();
        System.out.print("이름 입력:"); tmp2 = scan.next();
        Person j_000003 = new Person(tmp1, tmp2);
        System.out.println("이름:"j_000003.getLastName()+ j_000003.getFirstName());
        System.out.print("글자수: "+j_000003.getLength());
    }
}
```

결과

```
성 입력:전
이름 입력:정삼
이름:전정삼
글자수:3
```

예제2 [MakePoint.java]

점을 모델링하는 클래스를 만드시오.

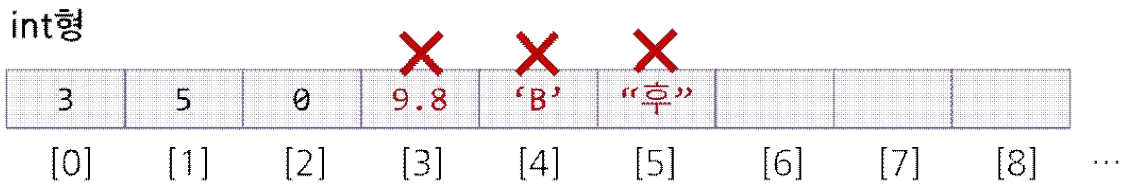
```
public class MakePoint {
          private double x, y;
          public MakePoint_wt(double x, double y) {
                    this.x = x;
                    this.y = y;
          }
          public void printPoint() {
                   System.out.println("점의 위치 :("+x+", "+y+")");
          }
          @Override //object의 toString()을 오버라이딩 =>
          public String toString() {
                    return "P좌표 [x=" + x + ", y=" + y + "]";
          }
          public void move(double dx, double dy) {
                    x+=dx;
                    y+=dy;
          }
}
```

■ [참고] main메서드에 static을 붙이는 이유

일반적으로 static이 붙지 않은 메서드의 경우 메서드 호출이 실행될 때 메모리를 할당받고 실행이 종료되면 메모리를 반환한다. 그에 반해 static이 붙은 변수나 메서드는 호출 전에 메모리를 먼저 할당받고 프로그램이 실행되는 동안 계속 상주한다. 따라서 main메서드는 자바 프로그램이 실행되는 동안, 호출하지 않더라도 메모리를 할당받고 항상 상주하고 있어야 하므로 static을 붙인다.

10 : 배열

배열이란 같은 타입의 데이터를 연속으로 나열한 자료 구조이다. 같은 자료형의 데이터 여러개를 한번에 순차적으로 관리할 수 있다는 것이 가장 큰 장점이다. 또한 메모리를 절약하고 간결한 프로그램을 만들 수 있다. 배열(Array)은 객체로 처리하며 **참조형 변수**기도 하다.



■ 배열 선언

선언과 생성 ⟶ +초기화

자료형[ ] 배열명 = new 자료형[개수];

||

자료형 배열명[ ] = new 자료형[개수];

초기화

배열명[n] = 데이터1;
배열명[n+1] = 데이터2;
…

자료형[ ] 배열명 = new 자료형[ ] { 데이터 };

자료형[ ] 배열명 = { 데이터 };

자료형[ ] 배열명;
배열명 = new 자료형[ ] { 데이터 };

- **선언** : 기억 장소의 주소를 가리키는 변수를 선언하며, 크기를 지정할 수 없다.
- **생성** : new연산자로 기억공간을 확보하고 그 주소를 선언한 배열변수에 저장하며, 크기를 지정한다.
- **초기화** : 배열객체에 초기화하는 경우 배열의 길이를 지정할 수 없다. 초기화 과정이 없을 시 배열은 자료형에 따른 일정한 초기값을 갖는다.
  (초기값 : 정수· 실수형: 0 / 논리형: false / 문자형: ' ₩0000')

예제 [Array_sum.java]
1-2+3-4+5-6…+99-100의 합계를 1차원 배열을 이용하여 구하시오. 조건은 int형 배열 요소 개수가 100개인 배열을 선언하시오.

```
public class Array_sum {
        public static void main(String[] args) {
                int[] a= new int[100];
                for(int j=0; j<a.length; j++) {
                        a[j]= j+1;
                }
                int num = 0;
                for(int i=0; i<a.length; i++) {
                        if(a[i]%2==1) { num+=(a[i]); }
                        else { num-=(a[i]); }
                }//end of for
                System.out.println(num);
        }//end of main
}//end of class
```

예제 [Array_sum.java]

10명의 점수를 입력받아 오름차순으로 점수를 정렬하는 프로그램을 작성하시오.

```
public class Array_sort {
          public static void main(String[] args) {
                    Scanner scan=new Scanner(System.in);
                    System.out.print("점수 10개 입력(띄어쓰기로 구분): ");
                    int score[]=new int[10];
                    for(int i=0;i<score.length;i++) {
                              score[i]=scan.nextInt();
                    }
                    int sort;
                    for(int j=0; j<score.length-1;j++) {
                              for(int k=j+1; k<score.length;k++) {
                                        if(score[j]>score[k]) {
                                                  sort=score[j];
                                                  score[j]=score[k];
                                                  score[k]=sort ;
                                        }
                              }
                    }System.out.println(" ");
                    for(int l=0; l<score.length; l++) {
                    System.out.print(score[l]+"  ");
                    }
          }
}
```

■ 다차원 배열

일차원 배열 뿐만 아니라 이차원 배열도 자주 사용된다. 삼차원 이상도 사용할 순 있지만 많이 쓰이지는 않는다. 일차원 배열이 행의 존재라면, 이차원배열은 행과 열의 갖는다. 일차원의 배열 인덱스(대괄호)가 하나였다면, 이차원은 배열 인덱스가 2개이다. 배열 선언과 생성은 일차원 배열과 같다. 다만 초기화 과정이 조금 다르다.

형태 **int 배열명[ ][ ] = new int[행개수][열개수];**

선언과 생성 **int a[ ][ ] = new int[2][3];**

초기화 경우1

```
a[0][0] = 1; a[0][1] = 2; a[0][2] = 3;
a[1][0] = 4; a[1][1] = 5; a[1][2] = 6;
```

배열 a	열[0]	열[1]	열[2]
행[0]	a[0][0] 1	a[0][1] 2	a[0][2] 3
행[1]	a[1][0] 4	a[1][1] 5	a[1][2] 6

초기화 경우2 : 선언+생성과 동시에 초기화 **int a[ ][ ] = new int[][]{{1, 2,3},{4,5,6}};**

아래는 이차원 배열 a를 출력하는 예제이다.

```
for(int i=0; i<2; i++){
     for(int j=0; j<3; j++){
         System.out.println("a["+i+"]["+j+"] = "+ a[i][j]);
     }
}
```

결과

```
a[0][0] = 1
a[0][1] = 2
a[0][2] = 3
a[1][0] = 4
a[1][1] = 5
a[1][2] = 6
```

예제1 [ArrayExam1.java]

2차원 배열을 이용하여 아래와 같이 출력하는 프로그램을 for문을 2개 이용하여 작성하시오.

```
public class ArrayExam1 {

          public static void main(String[] args) {
                    int[][] a= new int[4][4];
                    ar(a);
          }
          public static void ar(int a[][]) {
                    for(int i=0; i<a.length; i++) {
                              for(int j=0; j<=i; j++) {
                                        a[i][j] = j+1;
                                        System.out.print(a[i][j]+"  ");
                              }
                              System.out.println("");
                    }
          }
}
```

결과

```
1
2 3
4 5 6
7 8 9 10
```

예제2 [ArrayExam2.java]

2차원 배열을 이용하여 아래와 같이 출력하는 프로그램을 for문을 2개 이용하여 작성하시오.

```
public class ArrayExam2 {
          public static void main(String[] args) {
                    int a[][] = new int[4][4];
                    int n=1;
                    for(int i=0; i<a.length; i++) {
                              for(int j=0; j<a[i].length; j++) {
                                        a[i][j]= n++ ;

                                        System.out.print(a[i][j]+"  ");
                              }
                              System.out.println("");
                    }
          }
}
```

결과

```
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 16
```

■ 명령행 매개변수 배열

명령행 매개변수란 main() 메서드에서 매개변수로 받아오는 문자열을 말한다. main() 메서드는 항상 String args[] 라는 문자열의 배열을 매개 변수로 가져온다. 그 한도는 넘어오는 변수 개수 만큼이다(한도가 없다). 명령 프롬포트에서 실행시 ' java+파일명+인자들' 로 작성하고, 이클립스에서 실행시 run configuration에서 인자를 넘긴다. 문자열의 경우, 쌍따옴표 유무는 상관없다.

실행

```
C:>java TestFile 클로닝 1
```

소스

```
public static void main ( String args[] ) {
        System.out.println(args[0]+" 읽는 중 "+args[1]);
}
```

결과

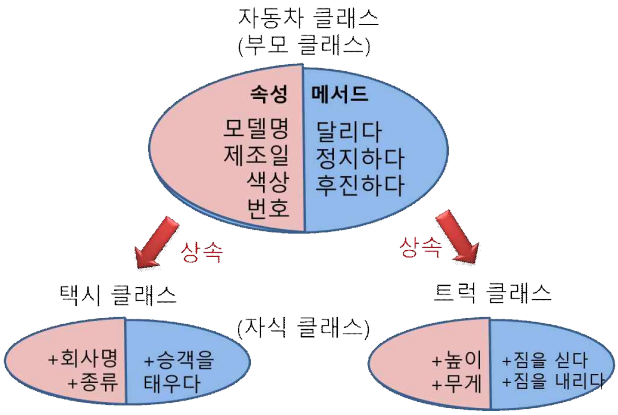
```
클로닝 읽는 중 1
```

11 : 상속

자바의 상속이란, 기존에 있는 클래스의 멤버 변수나 메서드를 물려받아 새로운 클래스를 만든느 것이다. 이것은 객체지향의 이점 중 하나인 코드의 재사용을 실현하는 방법 중 하나이다. 코드 간결화에도 이롭다. 워후!(거의다와가서매우기쁘다.)

상속은 부모클래스-자식클래스 또는 상위클래스-하위클래스, 수퍼클래스-서브클래스라는 관계의 클래스로 이루어진다. 이것을 IS-A 관계로 생각하면 쉽다.

	IS (has)	A (a)
명칭	자식클래스 하위클래스 서브클래스	부모클래스 상위클래스 수퍼클래스
예 -매개	택시, 포도	자동차, 과일
예 -관계	택시는 포도는	자동차이다. 과일이다.



하위클래스들은 상위클래스로부터 상속받은 속성들에 추가로 다른 속성을 추가할 수 있다. 상속받은 상위클래스의 속성들은 하위클래스의 코드에선 보이지 않지만 프로그램 상엔 존재한다. 따라서 하위 클래스의 객체를 생성 후 상위클래스의 속성을 사용이 문제가 없다(정적 속성 호출 시엔 [하위클래스.상위클래스의속성]의 형태). 다음은 상속 방법이다.

형태 **[접근제어자] class 클래스명 extends 상위클래스명**

예

```
class Car{
          String carname;
          String color="검정색 " ;
          int velocity;
          void speedUp( ){  velocity += 5; }
          void speedDown( ){  velocity -= 5; }
}
class Truck extends Car{
          int ton;
}
```

- 주의점

1. 단일 상속만 지원하기 때문에 하위클래스는 상위클래스를 한 개만 둘 수 있다.
2. 모든 클래스는 extends Object를 기술하지 않아도 최상위 클래스 Object 클래스를 자동으로 상속받는다.
3. 멤버변수 상속시, 상위 클래스의 변수와 같은 이름의 변수를 하위클래스에서 선언하면 하위 클래스의 변수에 의해 가려진다.

```
class A { int x; }
class B extends A { String x; }

...
main(String args[]){
          B b = new B();
          b.x = 100; //에러 발생 : b.x는 재정의된 String x.
```

다음은 앞서 처음에 모델링한 자동차 클래스와 그를 상속받은 택시 클래스, 트럭 클래스를 코딩한 후 테스트하는 예제이다.

```
class Car {
     String model, date, color, number; //모델명,제조일,색상,번호
     public Car(String model, String date, String color, String number){
          this.model = model; this.date = date;
          this.color = color; this.number = number;
     }
     public void race(); //달리다
     public void stop(); //정지하다
     public void backTo(); //후진하다
}
class Taxi extends Car {
     String agency, type; //회사명,종류
     //생성자 메서드
     public Taxi(String model, String date, String color, String number, String agency, String type){
          this(model, date, color, number); //상위클래스의 생성자 호출
          this.agency = agency; this.type = type;
     }
     public void takeClient( super.stop; /*상위클래스의 메서드 호출*/); //승객을 태우다
     @Override
     public String toString() {
          return "Taxi [agency="+agency+", type="+type+", model="+model+", date="+date+",
               color="+color+", number="+number+"]"; //상위클래스의 멤버변수는 super식별자 필요X
     }
}
class Truck extends Car {
     double height_meter, weight_ton; //높이,무게
     public Taxi(String model, String date, String color, String number, double height, double weight){
          this(model, date, color, number); //상위클래스의 생성자 호출
          this.height_meter = height; this.weight_ton = weight;
     }
     public void loadUp(); //짐을 싣다
     public void unLoad(); //짐을 내리다
}
public class Car_test {
     public static void main(String args[]) {
          Taxi taxi1 = new Taxi("sonata", "2018-05", "orange", "소4067", "Seoul", "standard");
          System.out.print(taxi1);
     }
}
```

결과 `Taxi [agency=Seoul, type=standard, model=sonata, date=2018-05, color=orange, number=소4067]`

■ 상속 관계의 접근제어자

상위 클래스의 접근제어자가 public 인 경우 하위 클래스가 상속받는 데는 아무 문제가 없다. protected인 경우는 같은 패키지가 아니더라도 상속된 하위클래스에서 상속클래스에 접근이 가능하다.

그러나 private인 경우 상속 관계라 해도 하위클래스에서 상위클래스에 접근이 불가하다. 즉, 하위클래스에서 상위클래스의 속성에 접근할 수 없다.

■ 상속과 생성자메서드 – super

상속 관계의 클래스에선 상위클래스의 메서드를 하위클래스가 자동으로 상속받을 수 있다.
다음은 메서드 호출의 기본 원칙이다. 클래스에서 메서드가 호출될 경우 아래 순서를 따른다.

1. 클래스 내에 정의된 메서드인지 확인한다. 그렇다면 클래스 내의 것을 실행한다.
2. 클래스 내에 정의된 메서드가 아니라면 상위 메서드에 정의된 메서드인지 확인한다. 그렇다면 상위메서드 내의 것을 실행한다.
3. 최상위에 도달할 때까지 반복한다.

그러나 일반 메서드와 달리 상위 클래스의 생성자 메서드는 하위 클래스에 자동 상속되지 않는다.

어떤 클래스에 생성자 메서드를 정의하지 않는다면 컴파일러는 기본 생성자 메서드를 추가하는데, 그 형태는 오른쪽과 같다.

```
생성자 메서드명 {
    super();
}
```

super()메서드는 상위 클래스의 생성자 메서드를 호출하며,
상위 클래스의 생성자 메서드가 없다면 최상위인 Object 클래스의 메서드를 호출한다.
때문에 하위 클래스에서 객체가 생성되면 자동으로 상위 클래스의 인자가 없는 생성자메서드(default)를 실행한다. 이때 상위 클래스에 인자없는 생성자가 없으면 에러가 발생한다. 하위클래스를 생성하면 수행과 초기화 단계가 상위 클래스 다음 하위클래스 순서이다.

```
class Car {
    Car() { System.out.println("Car 생성자"); }
}
class Truck extends Car {
    Truck() { System.out.println("Truck 생성자"); }
    //테스트
    public static void main(String[] args) {
         Truck mytruck = new Truck();
}
```

결과

```
Car 생성자
Truck 생성자
```

```
class Car {
    Car(String name) {
         System.out.println("Car 생성자메서드");
    }
}
class Truck extends Car {
    Truck() { System.out.println("Truck 생성자"); }
    //테스트
    public static void main(String[] args) {
         Truck mytruck = new Truck();
}
```

결과 에러 발생 이유: 사용자정의 Car생성자가 만들어짐으로써 JVM이 만든 default생성자가 사라짐.
=>사용자가 직접 정의해야함.

예제 [Greeting.java] [EngGreeting.java] [SuperTest.java]

다음은 상속관계에서의 super식별자를 활용하는 예제이다.

```
class Greeting{
          public String name = "나미림";
          public void sayHello() {
                    System.out.println("씨 안녕하세요");
          }
}
class EngGreeting extends Greeting{
          public String name = "Na";
          public void sayHello() {
                    super.sayHello();
                    System.out.println("Nice to meet you");
          }
          public void test() {
                    System.out.print(super.name);
                    super.sayHello();
          }
          public void test2() {
                    System.out.print(this.name);
                    super.sayHello();
          }
}
public class SuperTest {
          public static void main(String[] args) {
                    EngGreeting eng = new EngGreeting();
                    eng.sayHello();
                    eng.test();
                    eng.test2();
          }
}
```

결과

```
씨 안녕하세요
Nice to meet you
나미림씨 안녕하세요
Na씨 안녕하세요
```

- super

앞에 예제 코드들에서 사용되었던‘ super’ 예약어란, 상위 클래스의 변수나 메서드를 참조하기 위해 사용하는 예약어이다. 상위클래스의 변수를 나타낼 땐 [**super.변수명**], 상위클래스의 메서드를 호출할 땐 [**super.메서드명()**]으로 사용한다.

생성자메서드의 경우 super를 메서드 형태로 사용하는데, [**super(매개변수)**]의 형태이다. super메서드는 상위 클래스의 생성자메서드를 명시적으로 호출할 때 사용하는 것으로 하위 클래스의 생성자 메서드에서 제일 먼저 호출한다.

■ 메서드 오버라이딩

오버라이딩이란 객체지향의 특징을 보여주는 경우 중 하나로, 상위 클래스의 메서드를 하위 클래스에서 재정의하여 사용하는 것을 말한다. 유념할 점은 상위클래스 메서드의 이름, 인자, 반환형이 완전히 같아야 한다. 또한 static, final, private로 지정된 메서드의 경우 오버라이딩 할 수 없다.

```
class A{
		void eat() {
			System.out.println("밥국반찬을 먹습니다.");
		}
		void run() {
			System.out.println("달립니다.");
		}
}
class B extends A{
		void eat() { //재정의
			System.out.println("햄버거를 먹습니다.");
		}
}
public class OverRidingTest {
		public static void main(String[] args) {
			A a1 = new A();
			a1.eat();
			B b1 = new B();
			b1.run();
			b1.eat();
		}
}
```

결과

```
밥국반찬을 먹습니다.
달립니다.
햄버거를 먹습니다.
```

12 : 추상클래스

추상클래스란 클래스 계층구조에서 일반적인 개념을 나타내는 클래스이다. 하나 이상의 추상 메서드를 포함한다. 추상클래스로부터 new연산자를 이용해 객체를 만들 수 있고, 보통의 메서드와 변수를 포함할 수 있다.

형태 **접근 제어자 abstract 리턴타입 메서드이름( ); //{ }가 없다.**

■ 추상 메서드

추상 메서드는 메서드 머리부분만 있고 몸체는 없는 메서드이다. 추상 클래스의 추상메서드는 반드시 오버라이딩되어야 한다. 때문에 하위 클래스들이 특정 메서드를 반드시 구현하도록 강제할 수 있다. 추상메서드도 오버라이딩의 일반적인 규칙을 따른다. 만약 추상메서드를 오버라이딩하지 않으면 상속받는 클래스도 자동으로 추상 클래스가 된다.

예제1 [Shape.java] [Rectangle.java]

도형을 모델링한 추상클래스를 만든 후 그를 상속받는 사각형에 대한 클래스를 만드시오.

```
abstract class Shape {
   public String name;
   public String getName(){ //일반메서드
       return name;
   }
   //추상메서드로 몸체가 없다.
   public abstract int getArea();
   public abstract int getCircum();
}

public class Rectangle extends Shape {
    private int Length;
    private int Width;
    public Rectangle (String name, int Length, int Width){
         this.name = name;
         this.Length =Length;
         this.Width = Width;
    }

    public int getArea(){ //추상메서드 오버라이딩
         return (Length*Width);
     }
     public static void main(String[] args) {
       Rectangle myRect = new Rectangle("My R",5,3);
        System.out.println("사각형의 넓이 : " + myRect.getArea() );
     }
}
```

예제2 [Shape.java] [Circle.java] [Square.java] [Triangle.java] [AbstractTest.java]
도형을 모델링한 추상클래스를 만든 후 그를 상속받는 클래스들을 만드시오.

```
abstract class Shape{ //상위클래스(추상클래스)
	String name; //도형 이름
	public Shape(String name) {
		super();
		this.name = name;
	}
	public abstract void getArea(); //추상메서드
}
class Circle extends Shape{ //원 클래스
	double radius;
	final double PI = 3.14;
	public Circle(String name, double r) {
		super(name);
		this.radius = r;
	}
	public void getArea() { //추상메서드 상속
		System.out.println("원의 넓이:"+(radius*radius*PI));
	}
}
class Triangle extends Shape{ //삼각형 클래스
	int height, width;
	public Triangle(String name, int height, int width) {
		super(name);
		this.height = height;
		this.width = width;
	}
	@Override //오버라이딩
	public void getArea() { //추상메서드 상속
		System.out.println("면적:"+(double)(height*width*0.5));
	}
}
class Square extends Shape{//사각형 클래스
	int width, height;
	public Square(String name, int w, int h) {
		super(name);
		this.width = w;
		this.height = h;
	}
	@Override
	public void getArea() {
		System.out.println("사각형의 넓이: "+(width*height));
	}
}
public class AbstractTest {
	public static void main(String[] args) {
		Circle c1 = new Circle("원", 5);
		Triangle t1 = new Triangle("삼각형", 4, 3);
		Square s1 = new Square("사각형", 6, 7);
		c1.getArea(); t1.getArea(); s1.getArea();
	}
}
```

실행

```
원의 넓이:78.5
면적:6.0
사각형의 넓이: 42
```

와다다다닷 끝냈다!
여러분도 자바 힘내세요