**자바 언어의 특징**

* **자바 언어의 역사**

- 선 마이크로시스템즈(Sun MicroSystems)사는 1991년부터 제임스 고슬링을 주축으로 가전 제품을 제어하는 단순하고 버그 없는 프로그래밍 언어 자바를 개발하기 시작하였습니다.

- 제임스 고슬링은 앞으로 가전 제품들이 하나의 거대한 네트워크로 연동될 것이며, 제품의 기능을 담당하는 내장 프로그램들이 하나의 부품처럼 사용될 것이라고 예측하였습니다.

- 사용하던 가전 제품의 기능이 향상되어 내장 프로그램을 업그레이드 할 때마다 네트워크를 통해 서버로부터 새로운 프로그램을 다운로드 받아서 기능을 향상시키고자 하였습니다.

- 초기에는 C++언어로시작하였는데, C++의 여러 문제점들을 개선한 자바 언어가 개발되었음에도 가전제품 회사들은 그다지 관심을 가지지 않았습니다.

- 그러다 1995년 월드와이드웹(WWW)라는 인터넷의 보급 확산으로 핫자바(HotJava)에 의해 인터넷에 응용되면서 '애플릿(Applet)이라는 강력한 출력 형태가 인기를 끌게 되었습니다.

- 애플릿은 브라우저에서 다운로드를 완료하여 실행된다는 점과 실행 속도가 느리다는 점 때문에 외면당하기도 하였습니다.

- 그러나 자바 언어는 작고 단순한 구조로서 효율적으로 변환되어 실행되고, 기존 언어에서의 에러의 원인이 되는 여러 기능들이 보완됨으로써 그 장점을 인정받았습니다.

- 이리하여 자바는 지금과 같은 대중적인 프로그래밍 언어로 자리 잡을 수 있게 되었습니다.

* **자바 언어의 특징**

**- 단순함**

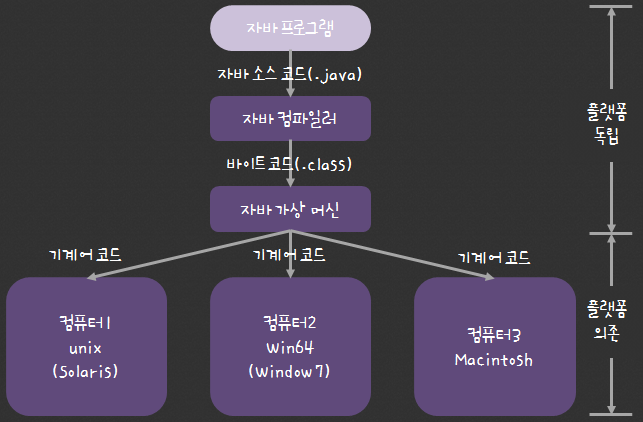
C++언어로부터 나온 자바는 C++/C언어와 문법적으로 유사합니다.

또한 포인터, 다중 상속, 구조체 사용, 전역 변수의 사용, 연산자의 중복 기능 등 C++의 복잡성을 단순화시켰습니다.

**- 객체지향**

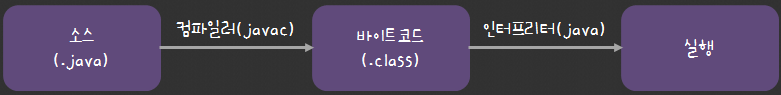
객체(Object)개념을 적용하는 완벽한 객체지향 언어입니다.

**- 독립적인 플랫폼**



플랫폼(Platform)은 프로그램이 실행될 수 있는 하드웨어나 소프트웨어(운영체제)기반 환경으로 자바 프로그램은 자바 가상머신에 의해 실행되므로 플랫폼에 독립적입니다.

* **컴파일러와 인터프리터 언어의 두 가지 특징을 모두**



자바 프로그램은 컴파일러에 의해 중간 코드인 바이트 코드로 번역되고, 바이트 코드는 인터프리터에 의해 기계어로 해석되면서 실행됩니다.

* **분산 처리에 용이**

TCP/IP 네트워크 기능을 내장하고 있으며 프로토콜(HTTP, FTP 등)을 사용할 수 있는 라이브러리를 제공합니다.

* **견고함**

다양한 플랫폼의 컴퓨터에서 실행되기 위해 기존 언어들의 에러 원인이 되는 요소들을 배제하였습니다.

포인터(Pointer)를 사용하지 않습니다.

자동으로 가비지 컬렉션(Garbage Collection기능을 수행합니다.

엄격한 데이터형 검사를 통해 에러를 조기에 발견합니다.

실행 시간에 발생하는 에러를 처리합니다.

* **안전함**

자바는 분산 네트워크 환경에서 실행될 수 있도록 설계된 언어로 이런 환경에서 중요시되는 부분이 보안입니다.

자바는 실행하는 인터프리터는 바이트 코드를 불러오고, 검증하고, 수행하는 작업을 단계적으로 지원합니다.

* **멀티스레드(Multi Thread) 지원**

스레드는 독자적으로 수행하는 하나의 작은 프로그램으로, 자바는 한 번에 여러 스레드가 동시에 수행되는 멀티스레드를 지원하여 수행 속도를 빠르게 합니다.

**- 동적**

자바는 변화에 잘 적응하도록 설계되었습니다.

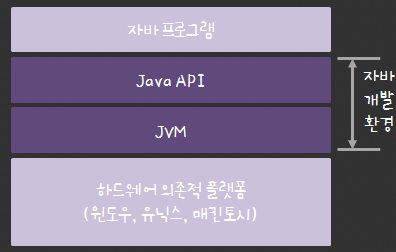
기존의 프로그램에는 영향을 미치지 않고 라이브러리에 새로운 메소드나 속성을 추가할 수 있습니다.

프로그램에 사용되는 라이브러리와의 연결을 실행 시간에 수행하여 라이브러리의 변화를 적용할 수 있습니다.

**자바 개발 환경**

- 자바에는 보통의 프로그래밍 어넝와 달리 하드웨어나 운영체제에 관계없이 실행될 수 있는 자체 플랫폼이 있습니다.

- 이를 자바 개발 환경(JDK : Java Development Kit)이라고 합니다.



**+ 자바 개발 환경 구조**

**- JVM(Java Virtual Machine : 자바 가상 머신)**

자바 가상 머신은 컴파일 후 생성되는 바이트 코드를 다양한 컴퓨터 플랫폼(하드웨어 및 운영체제)에 맞는 기계어로 해석하여 실행합니다.

따라서 자바 프로그램은 각 컴퓨터 플랫폼에 관계없이 자바 가상 머신만 있으면 실행할 수 있습니다.

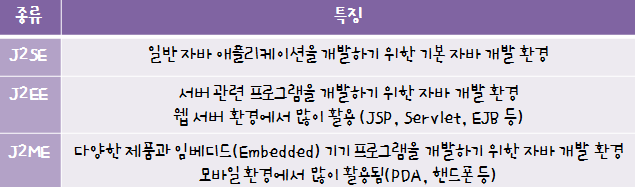
**- Java API(Java Application Programming Interface)**

API는 자박 지원하는 클래스 라이브러리롸, 프로그래밍에 필요한 기능들ㅇ르 클래스로 구현하고 서로 관련있는 클래스를 패키지 단위로 묶어서 제공합니다.

**+ 자바 개발 환경 종류**

- JDK1.2(Java2)부터 개발되는 자바 프로그램의 성격에 따라 나뉩니다.

- J2SE(Java2 Standard Edition), J2EE(Java2 Enterprise Edition), J2ME(Java2 Micro Edition) 3가지 버전의 자바 개발 환경이 있습니다.



**자바 프로그램의 종류**

**+ 일반 프로그램**

- 자바 애플리케이션(Application)

**+ 웹 클라이언트 프로그램 : 애플릿(Applet)**

- 웹 브라우저로 접속한 클라이언트 컴퓨터에 자바 프로그램(애플릿)을 전송하여 클라이언트 쪽에서 프로그램이 실행되는 기술입니다.

**+ 웹 서버 프로그램**

- 웹 브라우저로 클라이언트가 요청하면 웹 서버 쪽에서 자바 프로그램(서블릿, JSP)을 실행하고 그 실행결과만 HTML로 작성하여 클라이언트 컴퓨터에 전송하는 기술입니다.

**+ 웹컴포넌트**

- EJB는 네트워크 상에 필요한 객체를 분산시켜 두고 이 객체를 자바 애플리케이션에서 불러 쓸 수 있도록 하는 분산 객체 기술입니다.

- J2EE는 분산 객체 환경을 제공하여 기업 등의 웹 개발할 때 필요한 자바 플랫폼으로, EJB는 J2EE에 속합니다.

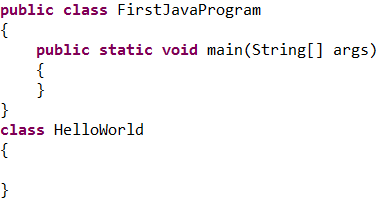
**+ 모바일과 임베디드**

- 휴대폰으로 영화를 보거나 무선 인터넷 서버로부터 게임을 다운로드하여 무선기기에서 사용하는 기술로 무선 환경에서 사용합니다.

- 뿐만 아니라 TV, 자동차 내비게이션 등 임베디드 분야(특정 기능의 프로그램이 내장된 하드웨어)로 개발 분야가 확대되고 있습니다.

**자바 프로그램 기본 구조**

**+ 자바 프로그램 기본 구조**



- 자바 프로그램은 컴파일러(JDK의 javac 명령어)에 의해 바이트 코드로 번역된 후 인터프리터(JDK의 java 명령어)에 의해 실행되는 프로그램입니다.

- 자바 프로그램은 하나 또는 여러 개의 클래스 모임입니다.

- 이 클래스 중 하나는 그 이름이 자바 프로그램 파일 이름과 같고 main() 메소드를 반드시 포함해야 합니다.

- JDK의 java 명령어가 클래스에 접근하여 main() 메소드를 자동 호출함으로써 프로그램이 시작되기 때문입니다.

**변수**

**+ 변수**

- 프로그램은 문제 해결에 필요한 데이터를 사용하기 위해서 메모리의 기억 공간에 이름을 붙이고 이 이름을 통해서 데이터를 저장하거나 가져옵니다.

- 기억 장소에 저장되는 데이터는 필요에 따라 변할 수 있으므로 변수라고 하며, 이 기억 장소의 이름은 변수명이라고 합니다.

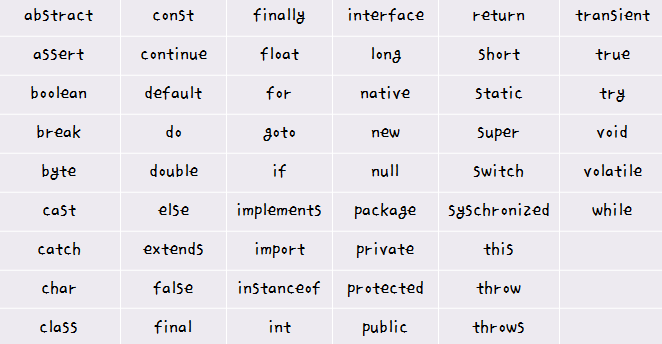
- 프로그램은 기억 장소의 이름인 변수명을 통해 데이터 에 접근할 수 있습니다.

**- 변수 이름 작성 규칙**

알파벳 대소문자, 숫자, \_, $만 사용할 수 있습니다. 첫문자로는 숫자는 올 수 없습니다.

자바 예약어는 사용할 수 없습니다. 대소문자를 구분합니다. 길이 제한은 없습니다.

**+ 자바 예약어**



- 자바 예약어는 문법적인 기능을 수행하기 위해 미리 예약된 단어입니다.

**- 예약어 사용 주의점**

모두 소문자이며 대소문자를 구분합니다.

const, goto는 현재 사용되지 않으나, 변수명으로도 사용할 수 없습니다.

C/C++의 sizeof는 사용되지 않습니다.

**데이터형**

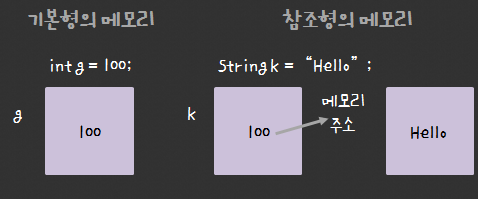
- 데이터형은 프로그램에서 사용되는 여러 종류의 데이터를 분류하여 형으로 규정한 것입니다.

- 자바에서 사용되는 데이터형은 메모리에서 데이터를 접근하는 방법에 따라 나뉩니다.

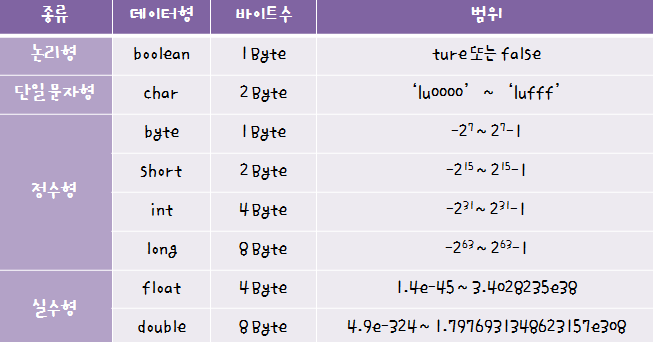
- 변수에 실제 사용되는 데이터가 저장되는 데이터형을 기본형(정수형, 실수형)이라고 합니다.

- 변수에 실제 사용되는 데이터가 저장된 메모리의 주소값이 저장되는 데이터형을 참조형이라고 합니다.

- 참조형 변수는 데이터를 참조할 수 있도록 되어있습니다.



**+ 기본형**



**- 논리형(boolean)**

C/C++에서는 정수값을 이용하여 0일 때는 거짓을 나타내고 0 이외의 값일 때는 참을 나타내도록 하고 있습니다.

자바에서는 true와 false만을 제공합니다.

따라서 다른 데이터형으로 변환될 수 없으며 대소문자를 구분하므로 TRUE와 FALSE는 논리형이 아님에 주의해야합니다.

**- 문자형(char)**

비영어권 국가를 위해서 2 Byte의 유니코드를 기반으로 단일문자를 표현하며, 반드시 작은 따옴표로 감싸줘야합니다.

**ex)** 'G', '|u0041', '|65'

**- 정수형(byte, short, int, long)**

소수점이 없는 숫자를 표현하는 자료형으로 크기에 따라 byte, short, int, long 4개가 있습니다.

C/C++과 달리 자바는 시스템에 관계없이 int형이 4 Byte의 크기를 갖습니다.

정수형 데이터는 기본으로 4 Byte의 int형으로 표현되며 long형으로 표현하려면 숫자 뒤에 'l' 또는 'L'을 붙여야합니다.

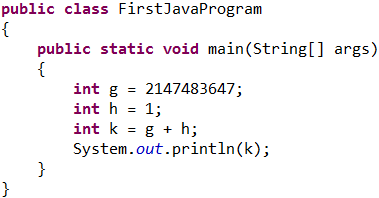
**ex)** 0, -1, -2156, 0745(8진수), 0x4523(16진수), 0L, 8577L

**- 실수형(float, double)**

소수점이 있는 숫자를 표현하는 자료형으로 크기에 따라 float, double형이 있습니다.

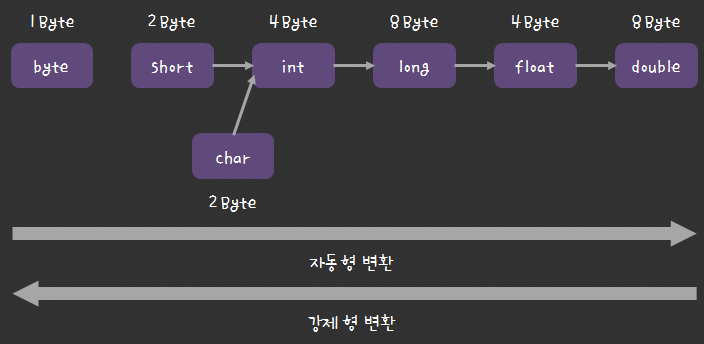
실수형 데이터는 기본으로 8 Byte의 double형으로 표현되며, float형으로 표현하려면 숫자 뒤에 'f' 또는 'F'를 붙여야합니다.

**ex)** 3.141592, 3.141592f, 3.141592e1, 3.141592e1f



http://cfile22.uf.tistory.com/image/251FA83D510664C728A137

**+ 데이터형 변환**



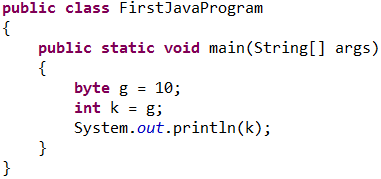
이미 선언된 데이터형을 다른 데이터형으로 변환하는 것을 데이터형 변환(DataType Casting)이라고 합니다.

**- 자동 형 변환(확대 형 변환, 묵시적 형 변환)**

데이터의 손실 없이 안정적으로 형 변환이 가능할 경우에만 내부적으로 자동으로 형 변환을 수행합니다.

데이터의 손실 가능성이 있는 경우네는 에러를 발생시킵니다.

예제



http://cfile1.uf.tistory.com/image/12102F41510664E51AACA1

**- 강제 형 변환(축소 형 변환, 명시적 형 변환)**

강제 형 변환은 더 작은 볌위를 나타내는 데이터형으로 변환하는 것입니다.

데이터형이 축소 변환된 후에도 해당 값을 표현할 수 잇어야 하며, 그렇지 않은 경우에는 형 변환 후에 전혀 다른 값을 가지게 됩니다.

축소 형 변환이라 데이터의 손실 가능성이 있으므로 반드시 형 변환 연산자를 사용해서 명시해야 컴파일 에러가 나지 않습니다.

**배열**

**+ 배열**

- 자바는 같은 데이터형의 데이터를 여러 개 저장하기 위해 배열을 사용합니다.

- 배열은 같은 종류의 데이터를 여러 개 저장하는 기억 장소입니다.

- 배열을 사용하면 메모리도 절약하면서 쉽고 간결한 프로그램을 작성할 수 있습니다.

- 자바에서는 배열을 객체로 처리하며 참조형 변수입니다.

**- 자바에서의 배열 사용 방법**

**1단계 : 배열 선언**

배열객체가 저장된 기억 장소의 주소를 가리키는 변수를 선언합니다.

배열 선언은 실제로 메모리 기억 공간이 확보되는 것이 아니기 떄문에 배열의 크기를 지정할 수 없습니다.

int형 배열객체를 가리키는 객체 참조형 변수 a가 메모리에 확보됩니다.

**ex)** 데이터형 배열변수명[];, 데이터형[] 배열변수명;

**2단계 : 배열 생성**

기억 공간을 할당하는 new 연산자로 배열객체가 저장될 기억 공간을 확보하고 이 기억 공간의 주소를 1단계의 배열 변수에 저장합니다.

반드시 몇 개의 데이터를 저장할 것인지를 지정해야 하며, 정해진 배열의 크기는 변경할 수 없습니다.

배열객체 각 원소의 기억 공간을 접근하는 인덱스는 0부터 시작합니다.

**ex)** 배열변수명 = new 데이터형[배열의 길이];

**3단계 : 배열 초기화**

배열객체 각 원소의 기억 공간에 데이터를 초기화합니다.

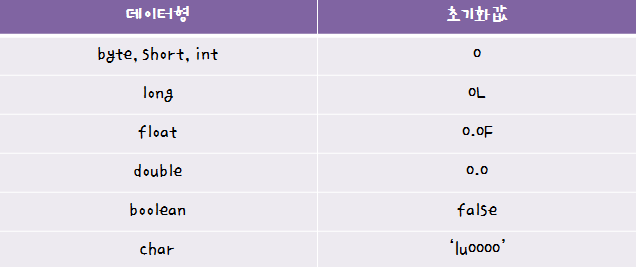
**형식**

데이터형 배열변수명[] = new 데이터형[배열의 길이]; 또는 배열변수명[] 데이터형 = new 데이터형[배열의 길이];

데이터형 [] 배열변수명 = {초기화 값 나열}; 또는 데이터형[] 배열변수명 = new 데이터형[]{초기화 값 나열};

**ex)** int a[] = new int[3];, int[] a = new int[3];, int[] a = {1, 2, 3};, int[] a = new int[]{1, 2, 3};, int a[] = {1, 2, 3};, int a[] = new int[]{1, 2, 3};

**- 데이터형에 따른 배열 초기값**

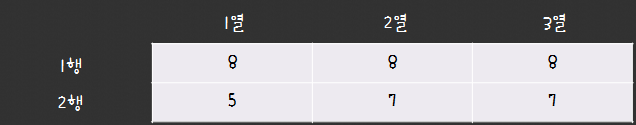


배열객체의 원소 개수인 길이는 배열객체의 멤버 변수인 length를 통해 알 수 있습니다.

**+ 2차원 배열**

- 이차원 배열은 배열객체를 원소로 하는 배열로 배열의 선언, 생성, 초기화하는 방법은 일차원 배열과 같습니다.

**- 자바에서의 2차원 배열 사용 방법**



**1단계 : 배열 선언**

**ex)** int[][] g;, int g[][], int[] g[];

**2단계 : 배열 생성**

**ex) g** = new int[2][3];

**3단계 : 배열 초기화**

**ex) g**[0][0] = 8;, g[0][1] = 8;, g[0][2] = 8;, g[1][0] = 5;, g[1][1] = 7;, g[1][2] = 7;

**형식**

**ex)** int[][] g = new int[2][3];, int g[][] = new int [2][3];, int[][] g = new int[2][]; g[0] = new int[3];, g[1] = new int[3], int[][] g = {{8, 8, 8}, {5, 7, 7}};

**+ 명령형 매개 변수**

- main() 메소드는 자바 어플리케이션에 필수적으로 있어야 하는 특수한 메소드로 자바 애플리케이션이 실행될 때 자동으로 호출됩니다.

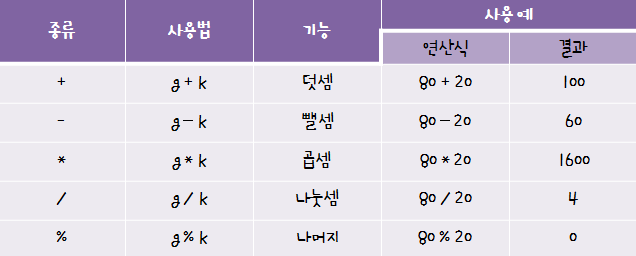
- main() 메소드는 문자열을 매개 변수로 받아 프로그램 실행 시 필요한 정보를 프로그램에 전달하는데, 이러한 매게변수를 명령형 매개 변수라고 합니다.

http://cfile30.uf.tistory.com/image/1307BF38510668A6281123

- main() 메소드의 매개 변수는 String 데이터형을 원소로 하는 배열으로 사용자가 명령형에서 입력한 문자열들을 순서에 따라 차례대로 배열 args에 저장합니다.

**연산자와 우선순위**

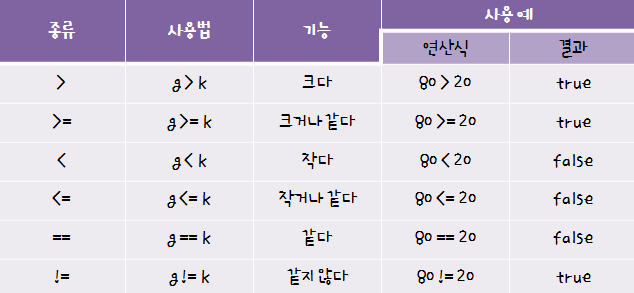
**+ 산술 연산자**



- 가감승제를 표현할 수 있도록 산술연산자를 제공합니다.

- % 연산은 피연산자가 정수형이어야만 하는 C/C++과 달리 실수형도 가능합니다.

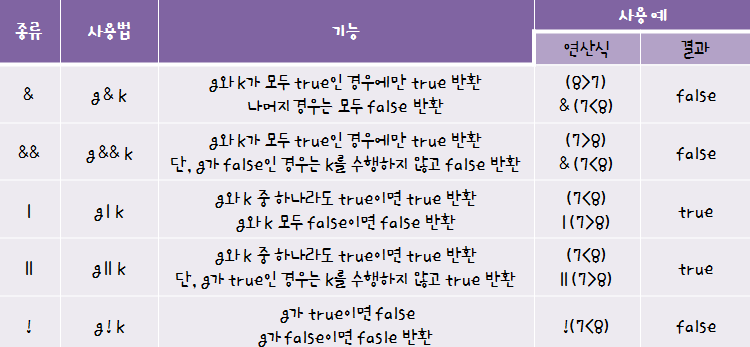
**+ 관계(비교) 연산자**



- 피연산자의 크기나 객체의 타입 비교 등에 사용됩니다.

- 비교한 결과에 따라 true 또는 false의 논리형 결과를 반환합니다.

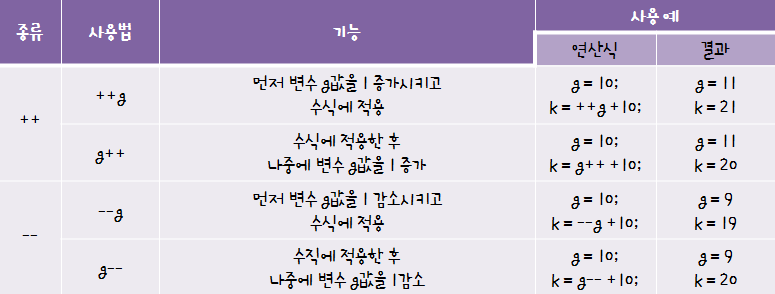
**+ 논리 연산자**



- 주어진 조건식이 참인지 거짓인지를 판단하여 true 또는 false의 논리형 결과를 반환합니다.

- 논리 연산의 대상이 되는 피연산자 역시 논리형이어야 하며 연산 횟수를 축소시키는 단축 논리 연산자인 &&, ||도 제공합니다.

**+ 증감(증가&감소) 연산자**



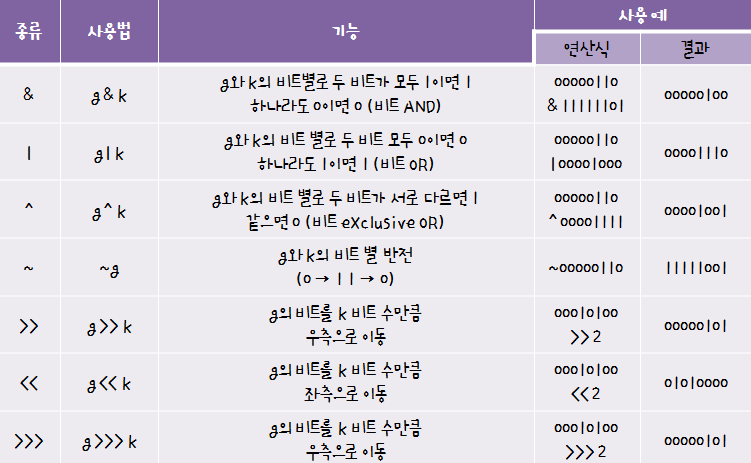
- 증감 연산자는 변수의 값에 1을 증가 또는 감소시킨 후 그 값을 다시 변수에 저장하는 연산자입니다.

- 사용하는 위치에 따라 전위형과 후위형 두 가지 방법이 있습니다.

- 전위형은 먼저 변수값을 1 증감시킨 후 변수가 포함된 수식에 적용하느 방법입니다.

- 후위형은 변수가 포함된 수식에 적용한 후 나중에 변수 값을 1 증감시키는 방법입니다.

**+ 비트 연산자**



- 비트 연산자는 피연산자의 비트 별로 연산을 수행합니다.

- 피연산자로는 실수형을 제외한 산술 데이터를 취합니다.

- & 연산은 특정 비트를 0으로 만들기 위해 사용합니다.

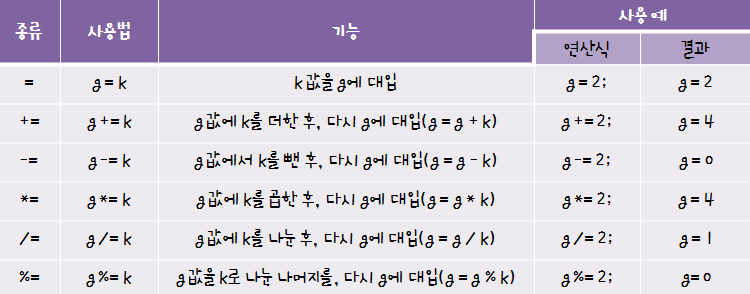
- | 연산은 특정 비트를 1로 만들기 위해 사용합니다.

- ^ 연산은 특정 비트를 비트 반전시키기 위해 사용합니다.

- ~ 연산은 1의 보수를 만들기 위해 사용합니다.

- >>, <<, >>> 연산은 비트를 이동시켜 곱셈, 나눗셈에 응용합니다.

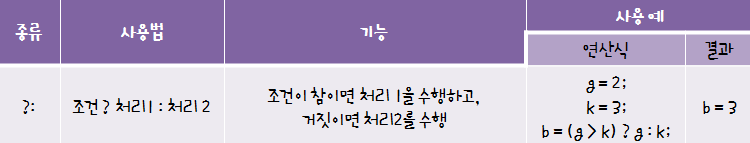
**+ 대입 연산자**



- 대입 연산자는 우변식(변수, 상수, 수식)의 최종 결과를 좌변의 기억장소(변수)에 대입(저장)하는 연산자입니다.

- 대입 연산자는 우변식의 결과를 마지막에 대입하기 때문에 연산자의 우선순위가 낮고, 수학에서의 "같음"의 의미가 아니라 "대입의 의미입니다.

**+ 조건 연산자(삼항 연산자)**



- 조건 연산자는 조건문 if-ele를 축약하여 사용할 수 있는 연산자입니다.

- 세 개의 피연산자를취하는 삼항 여산자로, 첫 번쨰 피연산자는  true 또는 false값을 갖는 논리식이어야 합니다.

- 두 번쨰와 세 번쨰 피연산자는 같은 데이터형을 갖도록 해야 하는데, 서로 다른 데이터형이라도 같은 종류의 데이터형일 경우 형 변환이 가능하면 됩니다.

**+ 연산자 우선순위**



- 수식에 사용되는 연산자 간에는 우선순위가 있습니다.

- 우선순위가 같은 연산자 간에는 왼쪽에 있는 연산자가 오른쪾에 있는 연산자보다 우선순위가 높습니다.

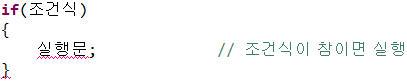
**제어문**

**+ 선택문**

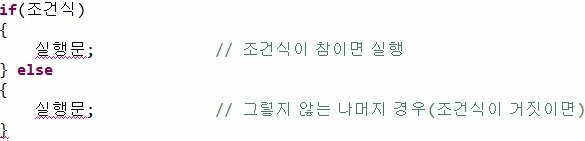
**- if문**

조건에 따라 선택적으로 문장이 수행되며 형식은 다음과 같습니다.

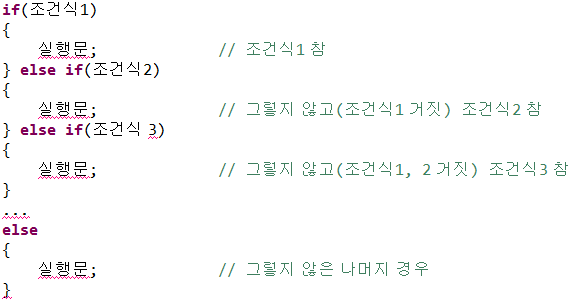
**조건식의 결과에 따라 한 갈래로 분기하는 if문**



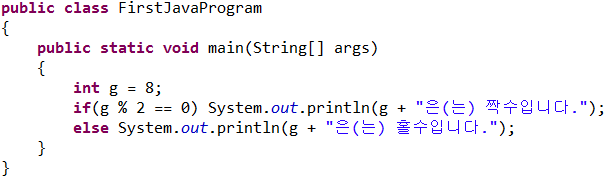
**조건식의 결과에 따라 두 갈래로 분기하는 if문**



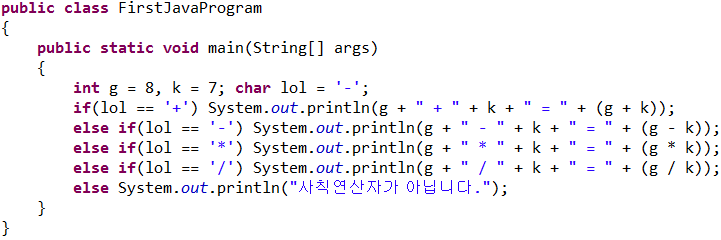
**조건식의 결과에 따라 여러 갈래로 분기하는 if문**



예제



http://cfile21.uf.tistory.com/image/2304BB3451066D1B03C655

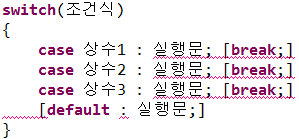


http://cfile5.uf.tistory.com/image/1405C73451066D1C02796D

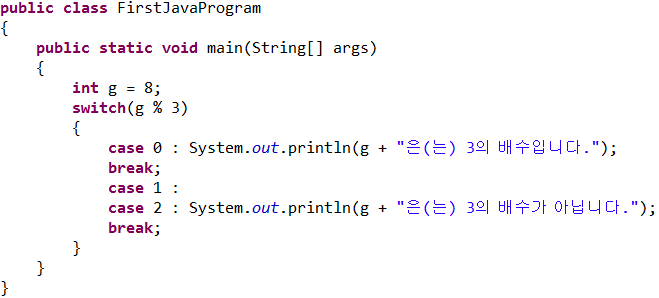
**- switch ~ case문**

switch문은 여러 갈래로 분기되는 if문과 유사한 구조를 갖습니다.

다중 if문보다 문법적으로 제한 사항이 있지만, 좀 더 아름답게 표현할 수 있습니다.



예제

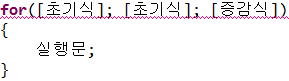


http://cfile5.uf.tistory.com/image/193410335106705C249AFD

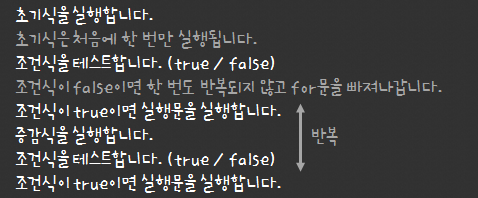
**+ 반복문**

- 반복문은 조건식이 참(true)인 동안에 문장이나 블록을 반복합니다.

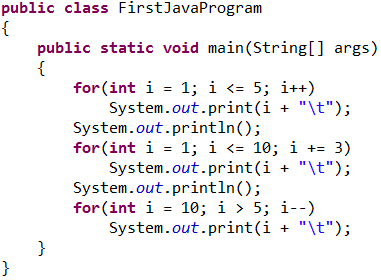
**- for문**

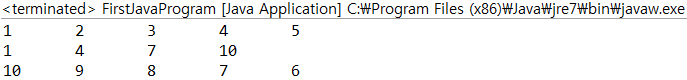


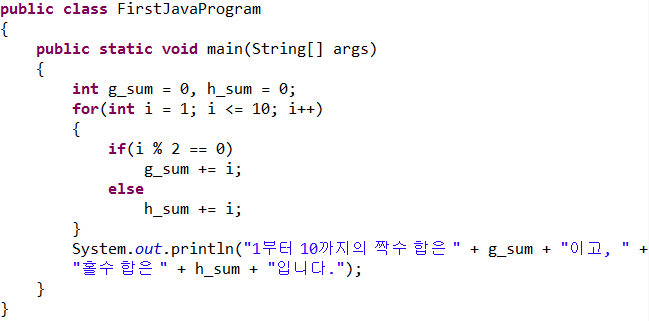
**for문 실행순서**



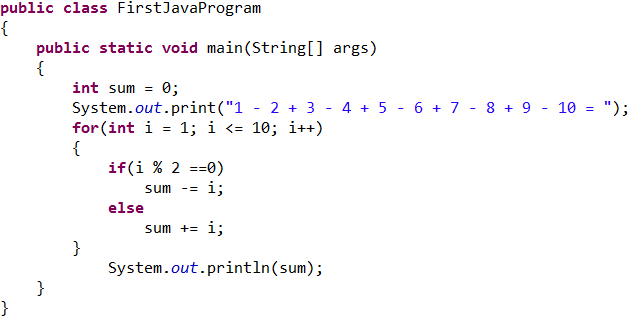
예제



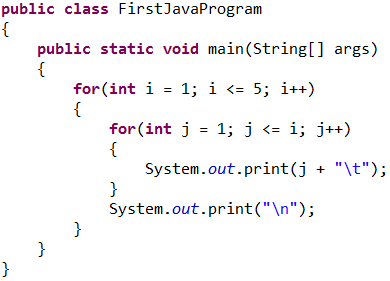


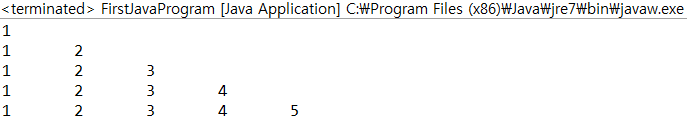


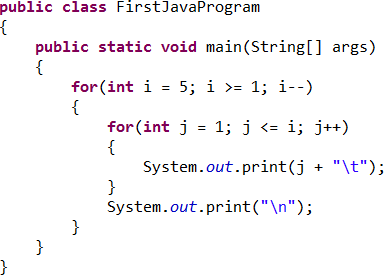
http://cfile23.uf.tistory.com/image/183376385106728B1491D2

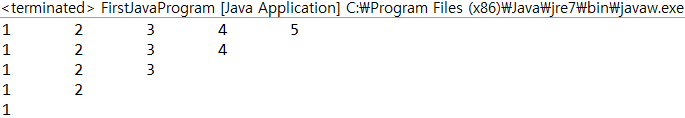


http://cfile5.uf.tistory.com/image/1358183C5106737C32F67B





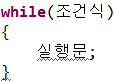




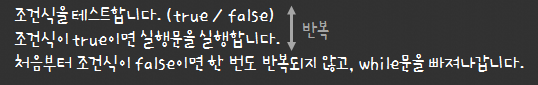
**- while문**

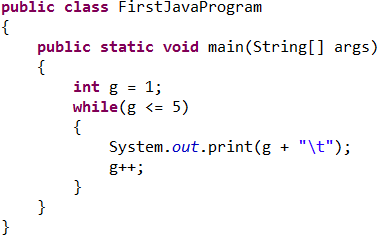
조건식이 참(true)인 동안에 문장이나 블록을 반복 실행합니다.

처음 실행부터 조건식이 참인지를 테스트하므로 반복문이 한 번도 실행되지 않을 수도 있습니다.

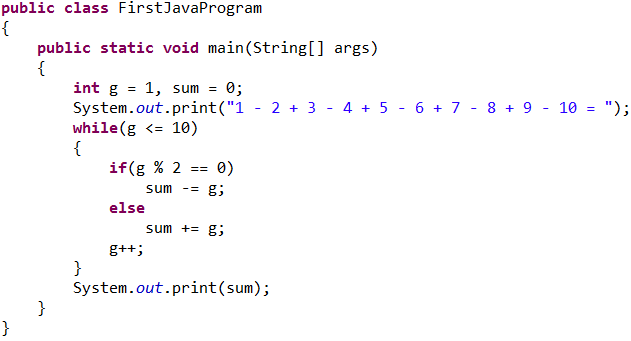


**while문 실행순서**



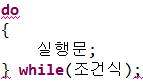


http://cfile10.uf.tistory.com/image/0115EC495106756F010ED7

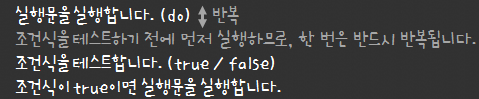


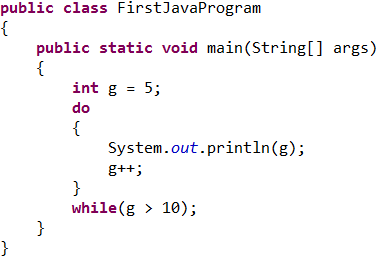
http://cfile26.uf.tistory.com/image/205DB3485106772C1B4E07

**- do ~ while문**  
while문처럼 조건식이 참(true)인 동안에 문장이나 블록을 반복하지만, 처음에 조건식을 테스트하지 않고 무조건 실행하므로 한 번은 반드시 반복됩니다.



**do ~ while문 실행순서**





http://cfile2.uf.tistory.com/image/18652C4D5106779F27D3B7

**+ 분기문**

- 프로그램의 수행 흐름을 강제로 특정 위치로 이동시키기 위해 break문과 continue문을 제공합니다.

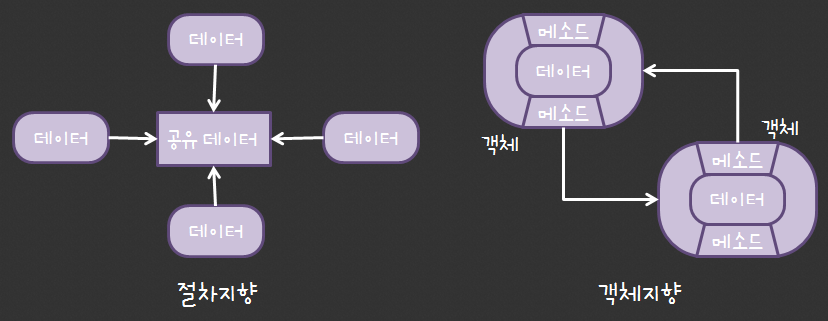
**- break문**

switch문을 빠져 나오거나 반복문(for, while)에서 가장 가까운 반복 블록을 빠져 나올 때 사용합니다.

**- continue문**

반복문에서 continue문은 이후의 문장은 실행하지 않고 그 다음 반복을 계속할 때 사용합니다.

**객체지향의 개념**



- 객체지향(Object-oriented)은 실세계의 모의실험(Simulation)을 목적으로 만들어진 시뮬라(Simula)라는 언어에소 소개되었습니다.

- 시뮬라 언어는 모의 실험 대상인 실세계 사물의 성질과 특징을 최대한 유지하면서 시스템에 적용하려는 목적으로 객체지향 이론을 소개합니다.

- 객체지향 개념은 70, 80년대를 거치면서 프로그램 언어 뿐만 아니라 프로그램 개발 방법론 분야 등 만흔 분야에서 발전하고 있습니다.

- 90년대 들어 소프트웨어 품질 향상과 관련하여 대두된 객체지향 기술은 소프트웨어 부품화, 소프트웨어 컴포넌트의 재사용을 주요 목적으로 합니다.

- 프로그래밍 분야에서의 객체지향 언어는 시뮬라에서 시작하여 스몰톡(Smalltalk), 이펠(Effel) 등 순수 객체지향 언어로 발전해 왔습니다.

- 그러나 이러한 순수 객체지향 언어는 범용적인 언어로 널리 활용되지 못 하였고, C언어에 객체지향 개념을 도입한 C++이 객체지향 언오로는 널리 이용되고 있습니다.

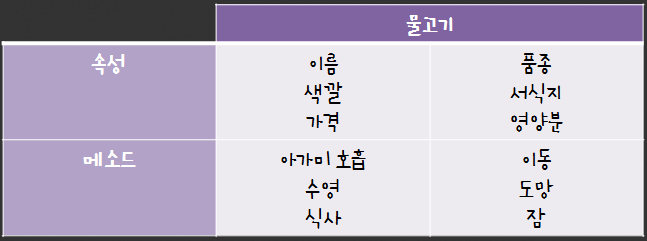
- C++를 기반으로 해서 탄생하게 된 객체지향 언어가 바로 JAVA입니다.

- 객체지향 언어와 대비되는 언어로서 기존의 프로그램 개발 방법인 절차지향(Procedual-oriented) 언어가 있습니다.

- 절차지향 언어가 공유된 데이터와 문제 해결 절차를 중시하는 반면, 객체지향 언어는 프로그램을 객체들의 집합으로 보며 이러한 객체들이 프로그램의 중심이 됩니다.

- 여기서 객체는 자신이 가지는 데이터와 그 데이터를 처리할 수 있는 메소드로 구성되는 하나의 단위입니다.

**객체와 클래스**



- 객체는 효율적인 정보를 관리하기 위해서 사람들이 의미를 부여하고 분류하는 논리적인 단위입니다.

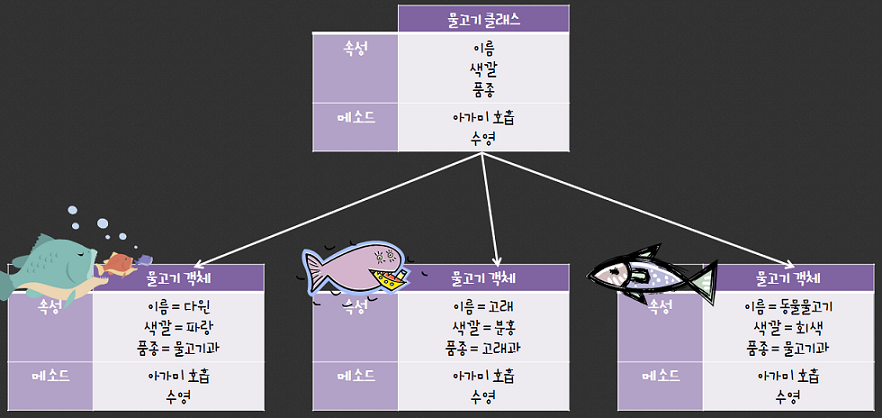
- 객체는 실세계에 존재하는 모든 것이 될 수 있으며 명사로서의 성질을 갖춘 모든 것은 객체로 만들어질 수 있습니다.

- 실세계의 객체가 갖는 특징을 모두 표현하기에는 한계가 있고 문제 해결과 직접적인 관련이 없는 불필요한 특성도 존재합니다.

- 그러므로 객체는 실제 프로그램에서 필요한 특성을 중심으로 모델링 됩니다.

- 실세계에 존재하는 객체는 두 가지 관점으로 모델링 됩니다.

- 하나는 객체의 정적인 특성을 표현하는 속성(Attribute)이고, 또 하나는 객체의 동적인 특성인 행위를 표현하는 메소드(Method)입니다.



- 객체지향 프로그래밍을 하다 보면 같은 특성을 갖는 여러 개의 객체가 필요할 때가 있습니다.

- 예를 들어, 같은 특성을 갖는 물고기가 100마리가 필요하다면 동일한 물고기 객체를 생성하는 과정을 100번 반복해야 합니다.

- 이를 위해 객체지햐에서는 동일한 속성과 메소드를 가진 여러 개의 객체를 생성하는 형판을 제공하는데 이 형판을 클래스(Class)라고 합니다.

- 객체와 클래스의 관계를 붕어빵에 비유하면, 찍어낸 수 많은 붕어빵들은 객체에, 붕어빵을 만들어내는 틀은 클래스에 해당한다고 볼 수 있습니다.

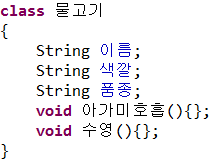
- 객체지향 프로그래밍은 객체를 사용하기 위해서 반드시 객체의 틀인 클래스를 정의하고 이 클래스로부터 객체를 생성해야 합니다.

**+ 실세계 객체 모델링**

- 실세계에 존재하는 객체로부터 프로그램에 필요한 특성(속성, 메소드)을 모델링합니다.

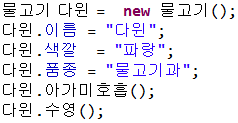
- 물고기의 객체의 속성(이름, 색깔, 품종)과 메소드(아기미 호흡, 수영)가 실세계를 모델링한 예가 될 수 있습니다.

**+ 클래스의 정의**



- 앞에서 모델링한 속성과 메소드를 클래스로 정의합니다.

**+ 객체의 생성 및 사용**



- 2단계에서 정의한 클래스로부터 실제로 존재하는 객체를 메모리에 생성하고 사용합니다.

**3. 객체지향 언어의 특징**

**+ 캡슐화**

- 클래스는 캡슐과 같은 형태로 만들어집니다.

- 객체는 속성과 속성을 처리하는 메소드가 캡슐화되어 있어서 갹채를 사용하는 쪽에서는 캡슐화된 속성과 메소드가 실제 어떻게 처리되는지는 알 필요가 없습니다.

- 단지 객체를 사용할 수 있는 인터페이스를 통해 사용만 하면 됩니다.

- 이를 캡슐화(Encapsulation)라고 합니다.

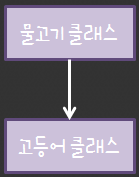
- 이 때, 외부에서 접근할 수 있는 객체의 속성과 메소드는 접근 권한을 외부에서도 사용할 수 있게 해줘야 합니다.

- 그리고 객체 내부에서만 알고 있어야 하는 속성과 메소드는 외부에서 접근할 수 없게 해야 합니다.

- 이를 위해 객체를 구성하는 속성과 메소드는 접근 제어자(ex : private, public 등)를 사용하여 접근 권한을 분명히 표시하고 있습니다.

- 이를 통해 객체지향 언어의 특징인 캡슐화와 정보 은닉(Information Hiding)이 가능해집니다.

**+ 상속**



- 객체를 생성하려면 먼저 속성과 메소드를 갖는 클래스를 정의해야 합니다.

- 만약 정의하려는 클래스가 이미 정의된 클래스가 갖는 속성과 메소드에 약간의 속성과 메소드가 추가되어야한다면 기존의 클래스의 모든 속성과 메소드를 상속받습니다.

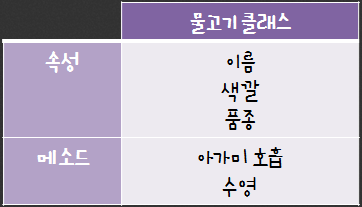
- 그리고 여기에 새로 필요한 속성과 메소드만 추가하면 될 것입니다.

- 이와 같이 기존의 클래스에서 모든 속성과 메소드를 상속받고 더 필요한 속성과 메소드를 추가하여 새로운 클래스를 생성하는 것을 상속(Inheritance)이라고 합니다.

- 기존의 클래스는 상위 클래스, 새로 만든 클래스를 하위 클래스라고 하며 두 개의 클래스는 상속 관계에 있다고 합니다.

- 예를 들어, 물고기 클래스가 이미 정의되어 있고 금붕어 클래스를 새롭게 정의해야 한다면, 금붕어는 물고이에 포함되므로 금붕어 클래스는 물고기 클래스를 상속 받고 여기에 승용차가 갖는 속성과 메소드를 더 추가하면 됩니다.

**+ 다형성**



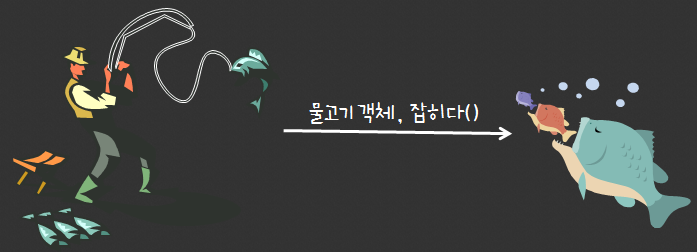
- 다형성(Polymorphism)은 객체의 메소드 형태가 다양한 것을 의미합니다.

- 대표적인 예로 메소드 오버로딩(Overloading)과 오버라이딩(Overriding)이 있습니다.

- 메소드 오버로딩은 하나의 클래스에서 동일한 이름의 메소드가 매개 변수의 개수나 타입을 달리해서 여러 가지 형태로 존재하는 것 입니다.

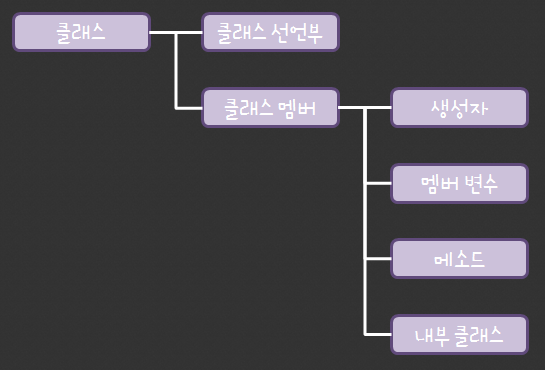
- 하나의 클래스에서 동일한 이름의 메소드가 다형성을 가질 수 있게 됩니다.

**+ 메세지**



- 메시지(Message)는 객체 간에 서로 통신하는 방법으로 객체들은 메시지를 주고받으면서 일을 수행합니다.

**클래스의 정의**

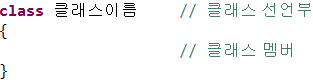


- 클래스(Class)는 선언부와 멤버로 나뉩니다.

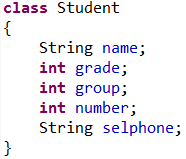
- 클래스 멤버에는 객체 생성 및 초기화를 담당하는 생성자, 속성을 정의하는 멤버 변수, 데이터를 조작하는 메소드가 포함됩니다.

- 경우에 따라서는 클래스 안에 또 다른 클래스가 선언되기도 하는데 이를 내부 클래스(Nested Class)라고 합니다.

**- 형식**



- 학교에서 학생을 관리하는 프로그램을 만든다고  가정해 보면, 학생별 이름, 학년, 반, 번호, 나이, 전화번호 등의 기본 정보가 필요할 것입니다.



- 프로그램의 요구 사항에 따라 다양한 형태의 학생 클래스가 만들어질 수 있습니다.

**2. 객체 생성**

http://cfile1.uf.tistory.com/image/116F00425162B786171CD8

- 클래스로부터 객체를 생성하는 방법은 다음과 같습니다.

**+ 1단계 : 객체참조형 변수 선언**

http://cfile26.uf.tistory.com/image/0338E8415162B7450E26D4

- 클래스형의 객체가 저장되는 기억 장소의 주소를 가리키는 객체참조형 변수를 선언합니다.

**+ 2단계 : 객체 생성**

http://cfile25.uf.tistory.com/image/1134543F5162B74F2771EA

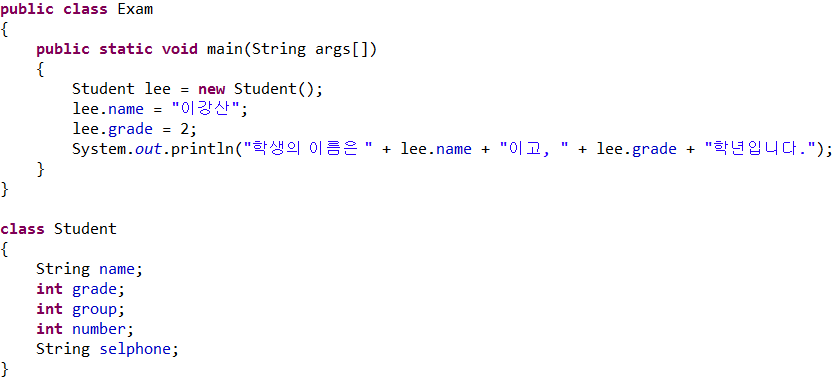
- new 연산자를 이용하여 객체를 저장할 기억 공간을 확보하는 과정입니다.

- 생성된 객체의 기억 공간 주소는 1단계에서 선언된 객체참조형 변수에 저장됩니다.

- 생성된 객체의 속성이나 메소드에 접근하려면 .(Dot) 연산자를 이용합니다.

- 예를 들어, 객체 lee의 멤버 변수인 name은 lee.name을 통해 접근 가능합니다.

예제



http://cfile26.uf.tistory.com/image/156722425162B8692CD23C

**3. 멤버 변수**

- 멤버 변수는 클래스를 정의할 때 메소드 밖에 선언하는 변수로서 객체가 가지는 정적인 특성인 속성을 나타내는 데 사용합니다.

**- 형식**

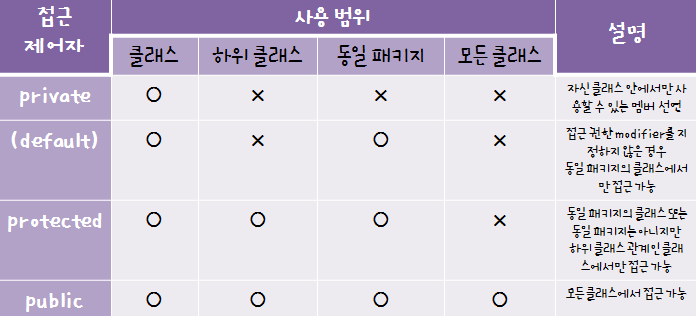
http://cfile9.uf.tistory.com/image/201F8B3D5162B8A726C0E5

- 멤버 변수에 대한 접근 권한이나 활용 방법을 표현하는 접근 제어자를 사용합니다.

**접근 권한 :** public, protected, private

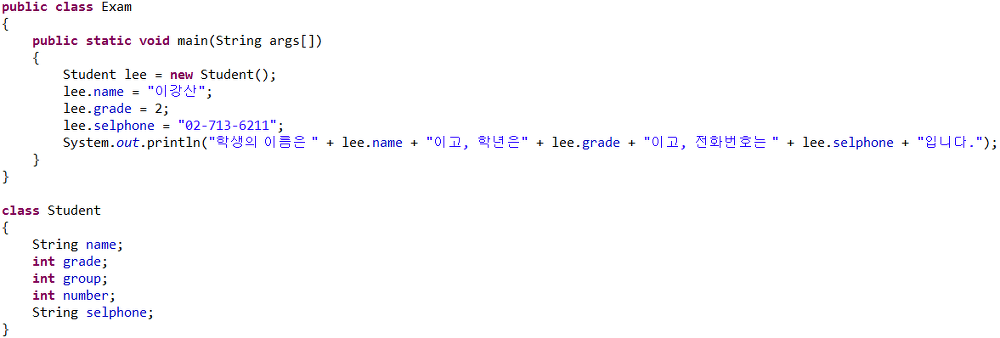
**활용 방법 :** static, final

**+ 접근 권한**



- 클래스 내의 멤버 변수에 대한 접근 권한을 제어하기 위해 사용되며 객체지향 언어의 특징인 캡슐화와 정보 은닉을 구현할 수 있습니다.

예제



http://cfile9.uf.tistory.com/image/201E68465162BB0413DE64

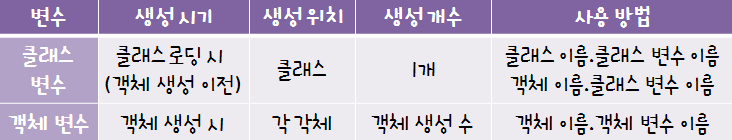
**+ 클래스 변수(static)**

- 멤버 변수에는 변수 이름 앞에 static이 붙은 클래스 변수와 그렇지 않은 객체 변수가 있습니다.

- 객체 변수는 객체가 생성될 때마다 기억 공간이 각각 확보됩니다.

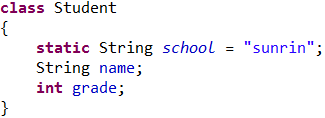
- 하지만 클래스 변수는 클래스가 로딩되는 과정에서 기억 공간이 한 번만 확보되고 해당 클래스 타입의 모든 객체들은 클래스 변수를 공유하게 됩니다.

- 그러므로, 클래스 변수는 하나의 클래스로부터 생성된 객체들 사이의 공통된 속성을 표현하는 데 주로 사용됩니다.

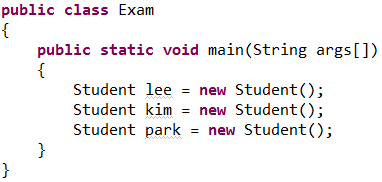


- 클래스 변수는 클래스 이름과 객체 이름으로 모두 접근 가능하나 클래스 이름을 사용하는 것이 객체 이름보다는 코드의 가독성을 높이기 때문에 많이 사용됩니다.

- 예를들어 Student는 클래스 변수인 school, 객체 변수인 name, grade를 멤버 변수로 갖는 클래스입니다.



- Student 클래스 타입의 객체 lee, kim, park을 선언 및 생성합니다.



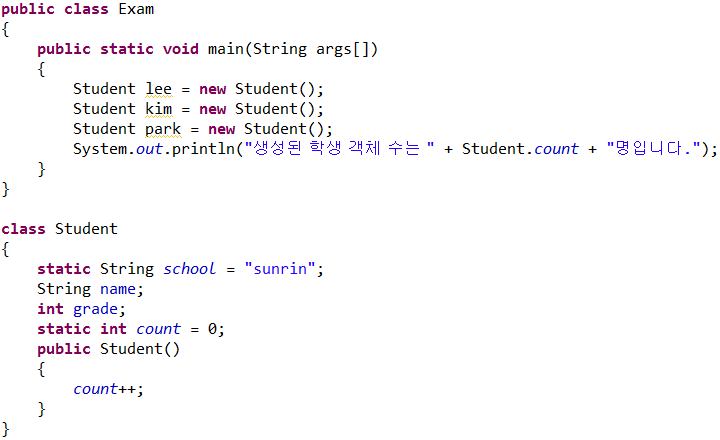
- Student 클래스가 메모리로 로딩되면 클래스 변수(Static 변수)인 school의 메모리가 확보되고 문자열 "sunrin"이 저장됩니다.

- 클래스가 로딩된 후 main() 메소드가 실행되면, lee, kim, park의 객체 변수 메모리가 각각 확보되고 클래스 변수인 school을 공유하게 됩니다.

- 클래스 변수인 school은 Student.school, lee.school, kim.school, park.school로 접근할 수 있습니다.

- 그리고 객체 lee의 객체 변수인 name은 lee.name으로만 접근할 수 있습니다.

예제



http://cfile28.uf.tistory.com/image/152762335162BD0D2F8BFB

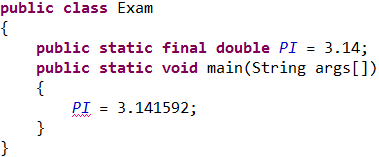
**+ 상수(final)**

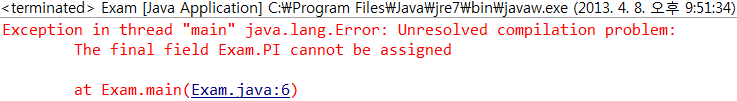
- 멤버 변수에 final이 붙으면 해당 멤버 변수는 더 이상 값을 변경할 수 없다는 의미를 갖습니다.

- 즉, final이 붙은 변수는 단 한번 초기화할 수 있으며 그 뒤로는 값을 변경할 수 없는 상수입니다.

- 상수의 이름은 관례상 대문자를 사용합니다.

예제





**4. 생성자**

**+ 생성자 정의**

- 생성자(Constructor)는 클래스로부터 객체를 생성할 때 객체의 초기화 과정을 기술하는 특수한 메소드입니다.

- 외향적으로 일반 메소드와 비슷합니다.

- new 연산자와 함께 사용되며 객체 생성 시에 호출합니다.

- 객체 생성시에 객체의 멤버 변수(객체 변수)를 초기화하는 일을 주로 합니다.

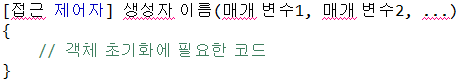
- 생성자의 이름은 클래스 이름과 동일하며, 일반적으로 첫 문자는 대문자입니다.

- return 유형이 없습니다.

- 생성자가 없는 클래스는 자바 가상 머신(JVM)이 자동으로 삽입하는 기본 생성자를 갖습니다.

- 사용자가 정의한 생성자가 있을 경우 자바 가상 머신(JVM)이 삽입한 기본 생성자는 자동으로 없어집니다.

**- 형식**



- 생성자 안에는 객체 초기화에 필요한 코드가 들어가며 필요할 경우 매개 변수를 사용할 수 있습니다.

- 생성자에도 접근 권한과 관련된 접근 제어자인 public, protected, private를 사용할 수 있습니다.

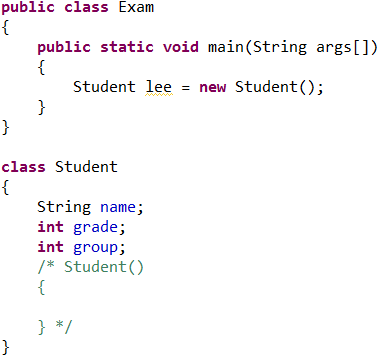
**+ 생성자와 객체**

- 생성자는 객체 생성 시 반드시 호출되어야 하는 메소드입니다.

- 그러나, 생성자가 없는 클래스의 경우에는 기본 생성자를 갖기 때문에 클래스가 반드시 생성자를 가지고 있어야 하는 것은 아닙니다.

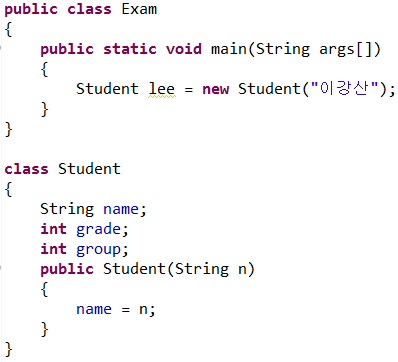
- 생성자가 없는 클래스

예제



- 생성자가 있는 클래스

예제

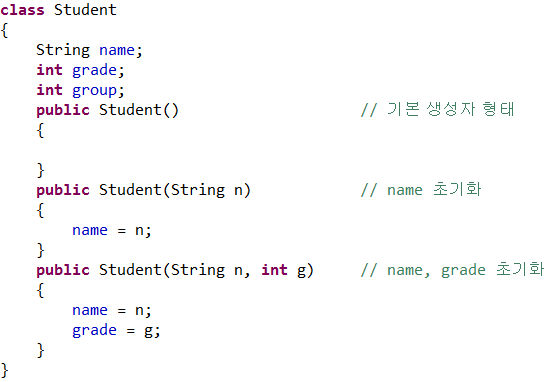


**+ 생성자 오버로딩**

- 클래스는 매개 벼수의 개수, 타입, 순서를 달리하는 같은 이름의 생성자를 여러 개 정의할 수 있으며 이를 생성자 오버로딩(Overloading)이라고 합니다.

- 이러한 다양한 형태의 생성자를 통해 다양한 형태의 객체를 생성함으로써 다형성이 구현될 수 있습니다.

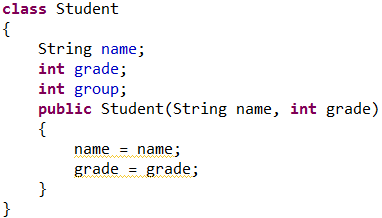
예제



**+ this**

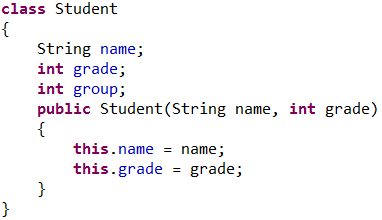
- 클래스에 사용되는 this는 현재 생성되어 사용 중인 객체의자신을 의미합니다.

**- this**



자바에서 this는 객체 자신을 의미하며 this.속성, this.메소드 형태로 사용됩니다.

보통 생성자나 메소드의 매개 변수 이름이 멤버 변수 이름과 같을 경우에 많이 사용됩니다.



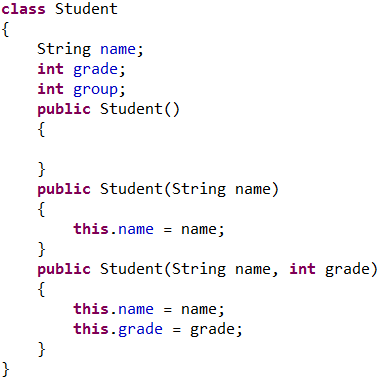
생성자 Student()에서 좌변의 name, grade 변수는 개체의 멤버 변수이고 우변의 name, grade 변수는 생성자의 매개 변수를 의미합니다.

하지만 변수 이름이 같아 프로그램의 가독성이 떨어집니다.

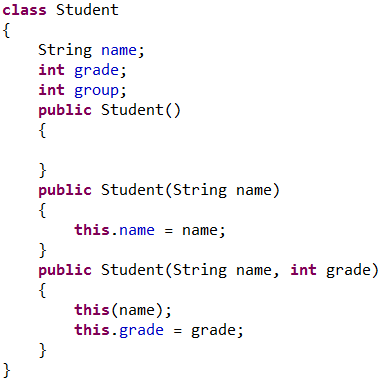
이러한 경우 좌변이 멤버변수 임을 분명하게 하기 위해 다음과 같이 자신의 객체를 의미하는 this를 사용하여 좌변의 name, grade 변수를 this.name, this.grade로 표현합니다.

**- this()**

객체의 생성자를 의미합니다.



보통 생성자 오버로딩에서 생성자 안에서 다른 생성자를 호출하는 경우에 많이 사용됩니다.



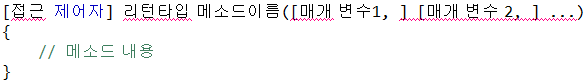
밑의 생성자가 위의 생성자를 포함하므로 객체 생성자를 의미하는 this()를 사용하여 수정하면 다음과 같습니다.

**5. 메소드와 오버로딩**

**+ 메소드 선언**

- 메소드는 클래스가 가지는 동적인 특성인 행위를 정의합니다.

**- 형식**

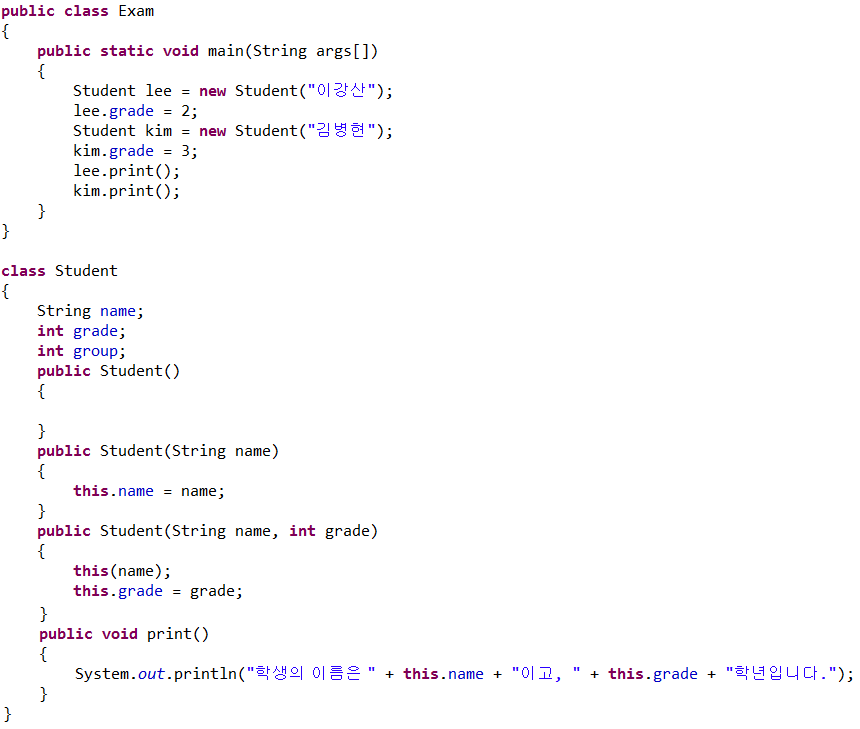


- 생성자에는 리턴 타입이 없으나 메소드에는 반드시 리턴 타입이 있어야 합니다.

- 메소드에 대한 접근 권한이나 활용 방법을 표현하는 접근 제어자를 사용합니다.

**+ 메소드 구현**

- 만일 학생 클래스가 학생의 정보를 출력해야 하는 동적인 특성이 필요하다면 이를 메소드로 구현하면 됩니다.



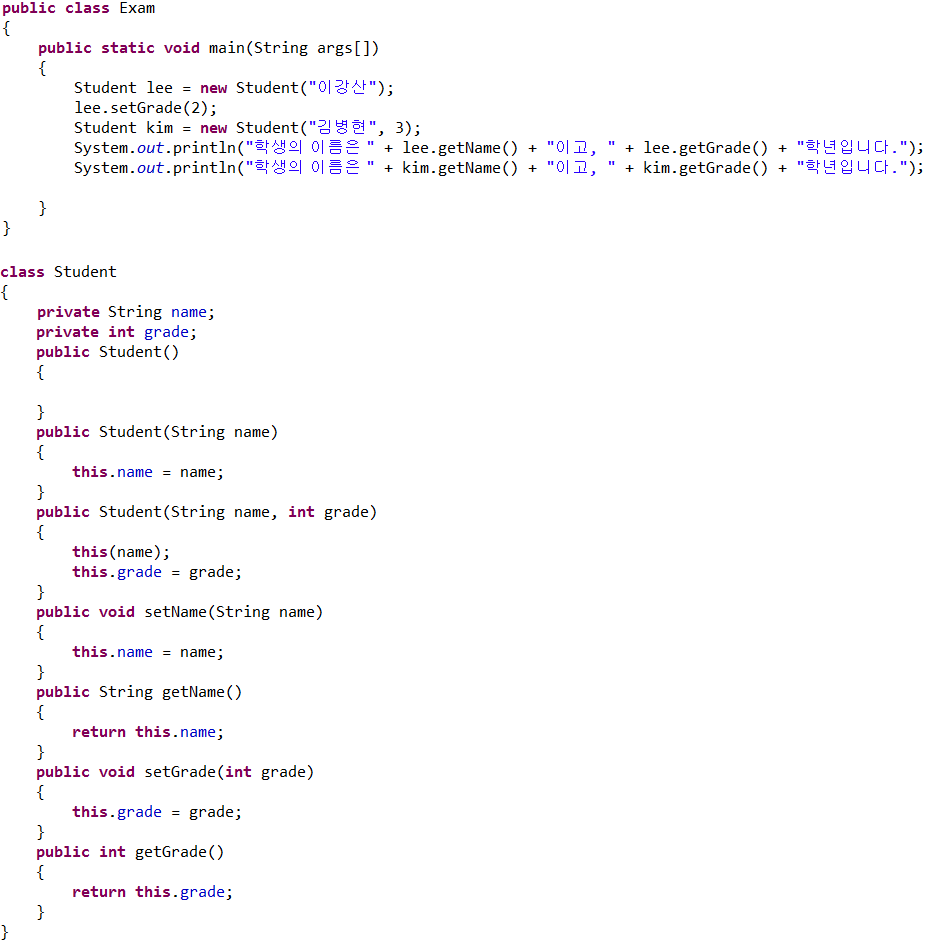
- Student 클래스에 출력 메소드를 추가한 예입니다.

- 위의 예에서 Student 클래스의 멤버 변수인 name, grade는 동일 패키지에서 사용 가능한 비교적 자유로운 접근 권한을 가집니다.

- 그러나 많은 경우 멤버 변수는 접근 권한을 private로 해서 외부에서는 숨겨진 형태로 만들고 public으로 지정한 메소드를 통해서만 멤버 변수에 접근하도록 클래스를 구현하는 것이 더 바람직합니다.

- 외부에 허용되는 메소드를 통해서만 접근 가능하고 실제 클래스에 어떤 변수가 있는지는 몰라도 되며 알 수도 없습니다.

- 이를 통해 객체지향 언어의 특징인 캡슐화를 구현할 수 있습니다.



- private한 멤버 변수를 public한 메소드를 통해 외부에서 사용하는 예 입니다.

- private 접근 권한을 갖는 Student 클래스의 멤버 변수 name, grade는 Exam 클래스에서는 사용할 수 없습니다.

- 따라서 외부에 공개된 메소드인 setName(), getName(), setGrade(), getGrade() 메소드를 통해 멤버 변수의 값을 수정 또는 확인할 수 있습니다.

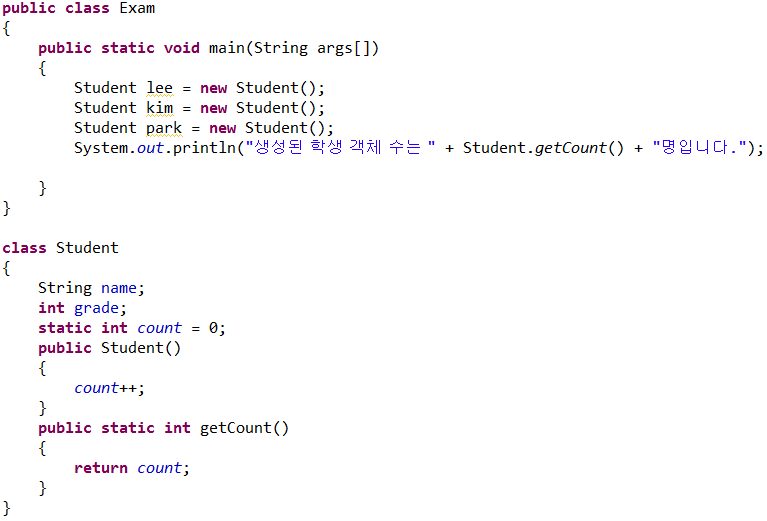
**+ 클래스 메소드(static)**

- 멤버 변수에 static으로 선언되는 클래스 변수가 있는 것처럼 메소드에도 static으로 선언되는 클래스 메소드가 있습니다.

- 클래스 메소드도 클래스를 로딩할 때 생성되며 객체 변수가 아닌 클래스 변수만 사용할 수 있습니다.

- 클래스 메소드도 보통 클래스 이름으로 접근합니다.

예제



http://cfile24.uf.tistory.com/image/20693F3D5162CAFC194A63

**+ 메소드 오버로딩**

- 메소드 오버로딩(Overloading)은 생성자 오버로딩과 비슷합니다.

- 같은 클래스 안에 매개 변수의 개수, 타입, 순서를 달리하는 동일한 이름의 메소드가 여러 개 존재하는 것입니다.

- 이를 통해서 객체지향 언어의 특징인 다형성을 구현할 수 있습니다.

- 메소드 오버로딩은 리턴타입, 접근제어자가 다른 것은 상관 없습니다.

**1. 상속의 개요**

- 현실 세계에서 부모와 자식은 상속 관계에 있습니다.

- 이때 부모가 가진 모든 것을 자식이 상속 받는다면 자식은 부모가 가진 모든 것을 사용할 수 있습니다.

- 이와 같이 자바에서도 상속을 통하여 기존에 있는 클래스의 멤버 변수나 메소드를 물려받아 새로운 클래스를 만들 수 있습니다.

- 따라서 새로운 프로그램을 작성하고자 할 때 매번 처음부터 클래스를 새로 만들지 않아도 이미 만들어진 클래스를 재사용하여 만들 수 있습니다.

- 상속을 이용하여 새로운 클래스를 정의할 경우 상속받은 클래스에서 기존 클래스의 멤버 변수와 메소드를 수정 없이 사용할 수 있기 때문입니다.

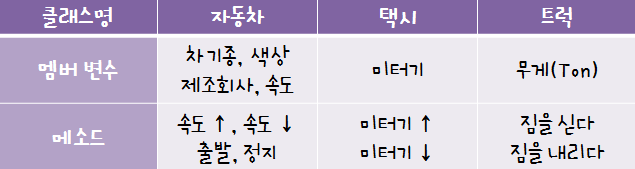
- 상속(Inheritance)이란 기존에 있는 클래스의 멤버 변수나 메소드를 물려받아 새로운 클래스를 만드는 것입니다.

- 상속을 통하여 객체지향 프로그램의 주요 특징인 코드의 재사용성을 구현할 수 있습니다.

- 상속 과정에서 상속받는 클래스를 서브 클래스, 하위 클래스, 자식 클래스라고 하고, 상속하는 클래스를 수퍼 클래스, 상위 클래스, 부모 클래스라고 합니다.

- 또한 상속 관계에 있는 클래스간의 관계는 "is-a 관계"로 즉, "서브 클래스는 수퍼 클래스이다" 관계가 성립합니다.

- 예를 들어 "택시는 자동차이다"라고 말할 수 있을 때 택시는 상속 받는 클래스이고, 자동차는 상속하는 클래스입니다.



- 택시 클래스와 트럭 클래스를 정의할 때, 멤버 변수 "차 기종, 색상, 제조회사, 속도"과 메소드 "속도를 올리다, 속도를 내리다, 출발하다, 정지하다"는 정의한 필요 없이 자동차 클래스의 것을 상속받아 사용하면 되기 때문에 간단히 정의할 수 있습니다.

**+ 상속 방법**

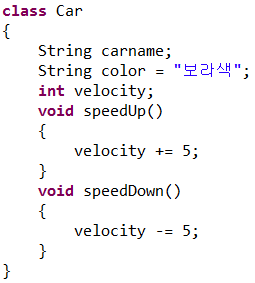
- 상속은 extends 키워드를 사용하고, 단일 상속만을 지원합니다.

- 단일 상속이기 때문에 extends 뒤에 하나의 클래스명만 쓸 수 있습니다.

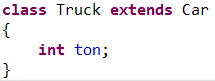
**- 형식**

http://cfile4.uf.tistory.com/image/176E88425162DF44261D0B

- 그러나 모든 클래스는 extends Object를 기술하지 않아도 Object 클래스를 자동으로 상속받습니다.

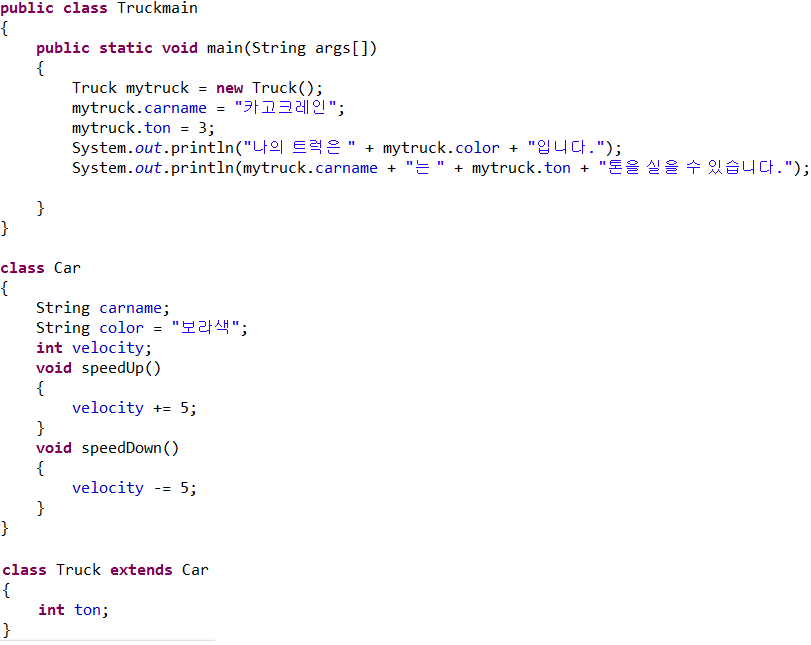


- Car 클래스를 정의합니다.



- Car 클래스를 상속받아 Truck 클래스를 생성한다면 위와 같은 형식으로 선언합니다.

예제



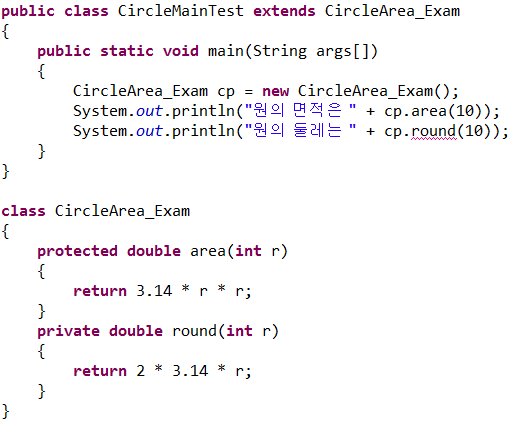
http://cfile25.uf.tistory.com/image/12653D435162E47F34A65D

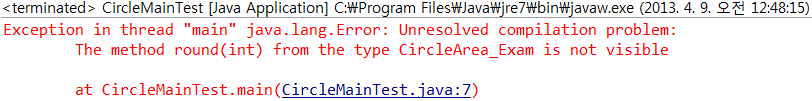
**+ 상속과 접근 제어자**

- 접근 제어자는 외부의 객체가 객체 내부의 멤버에 접근할 수 있는지 없는지를 결정합니다.

- 특히 상속 관계에서 상위 클래스의 접근 제어자가 protected일 경우 하위 클래스에서 접근할 수 있지만 여전히 private일 경우 상속 관계에서도 접근이 불가능합니다.

예제





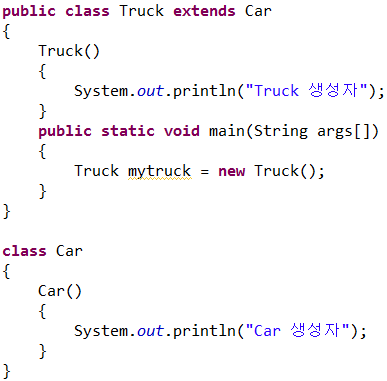
**2. 상속과 생성자**

- 상속 과정에서 생성자는 상속되지 않고, 하위 클래스의 객체 생성 시 자동으로 상위 클래스의 인자 없는 생성자가 호출됩니다.

- 즉, 상위 클래스를 상속받는 하위 클래스를 객체 생성할 때 자동으로 상위 클래스의 인자 없는 생성자를 호출합니다.

- 때문에 상속하는 상위 클래스에 인자 없는 생성자가 없으면 에러가 발생합니다.

- 특별히 생성자가 없을 경우는 디폴트 생성자인 생성자명(){}을 실행합니다.

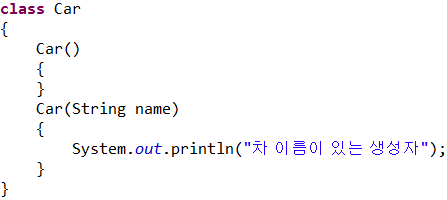


- 상속받는 클래스는 상위 클래스의 인자가 없는 생성자를 자동으로 실행하므로 인자 없는 생성자가 없으면 에러가 발생합니다.

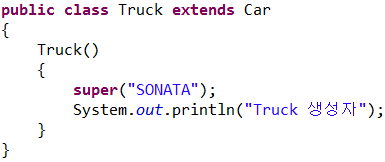
- 이때, 생성자가 있을 경우 디폴트 생성자를 인식하지 못 합니다.

- 에러 수정 방법은 디폴트 생성자인 Car(){} 클래스를 추가하거나 하위 클래스에서 super() 메소드를 이용하여 명시적으로 호출하는 것입니다.

**- 상위 클래스에 인자 없는 생성자를 넣는 방법**



**- 상위 클래스의 생성자를 super() 메소드를 이용하여 명시적으로 호출하는 방법**



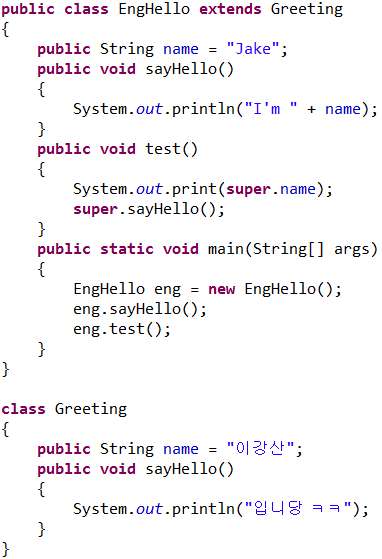
**+ super 키워드와 super() 메소드**

- 상속 관계에 있는 클래스에서 상위 클래스의 멤버 변수나 생성자를 호출하기 위해 super키워드와 super() 메소드를 사용합니다.

**- super 키워드**

http://cfile5.uf.tistory.com/image/176783345162E9C51E9C35

상위 클래스를 참조할 수 있는 가상의 참조변수로 상위 클래스의 멤버 변수나 메소드를 하위 클래스에서 명시적으로 호출할 때 사용합니다.



http://cfile9.uf.tistory.com/image/2261763F5162EB8A1F9A76

**- super() 메소드**

http://cfile2.uf.tistory.com/image/025DCB3B5162E9EC24B6F4

상위 클래스의 생성자를 명시적으로 호출할 때 사용하는 것으로 하위 클래스의 생성자에서 제일 먼저 호출해야 합니다.

**3. 상속과 메소드 오버라이딩**

- 상속받은 클래스는 상위 클래스의 멤버 변수나 메소드를 새로 정의하지 않고 사용할 수 있습니다.

- 그런데 상위 클래스의 메소드에 약간 기능을 추가하ㅓ나 같은 이름으로 새로운 기능을 정의할 필요가 있을 수 있습니다.

- 예를 들어 현실 세계에서 부모님이 컴퓨터가 있을 경우 자식들은 이 컴퓨터를 사용할 수 있지만, 자식이 부모님이 가지고 있는 컴퓨터 사양이 자신이 사용하기에는 부적합할 경우 별도로 컴퓨터를 구매하여 사용할 수 있습니다.

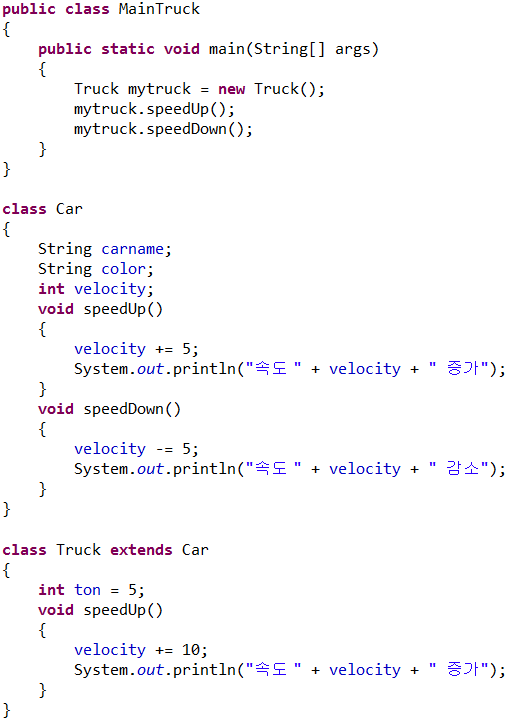
- 이때, 우리는 부모님이 가지고 있는 컴퓨터와 구분하기 위해 학생이 산 컴퓨터를 자동 계산기 등과 같이 다른 이름으로 부르지 않고 두 개 모두 컴퓨터라는 이름을 사용합니다.

**+ 메소드 오버라이딩(Overriding) 개념**

- 상위 클래스에 있는 메소드와 똑같은 메소드를 하위 클래스에서 다시 정의하여 사용하는 것으로 상위 클래스와 하위 클래스에서 메소드의 기능을 바꾸는 것을 말합니다.

- 이제 메소드 오버라이딩 도니 경우에는 특별한 지시가 없는 한 하위 클래스의 메소드가 호출됩니다.

**- 메소드 오버라이딩의 조건**



상위 클래스 메소드의 이름, 인자, 반환형에 대해서 완전히 같아야 합니다.

static, final, private 메소드의 경우에는 오버라이딩 할 수 없습니다.

**4. 형변환**

**+ 참조 데이터 형변환**

- 참조 데이터형은 배열, 클래스 등을 말하는 것으로 다른 참조 데이터 할당이나 다른 클래스로 객체를 생성하고자 할 때 참조 데이터 형변환이 일어납니다.

- 참조 데이터 형변환에는 묵시적인 형변환(UpCasting)과 명시적 형병환(DownCasting)이 있습니다.

**- 참조 데이터 형변환의 조건**

상속 관계에 있는 클래스 간에만 형변환이 가능합니다.

서로 다른 종류의 클래스 간에는 형변환이일어나지 않습니다.

상위 클래스 참조 데이터 변수는 하위 클래스 객체를 참조할 수 있지만, 그 반대느 안 됩니다.

**- 업 캐스팅**

http://cfile8.uf.tistory.com/image/2004413F5162F05603EE61

업 캐스팅(Up Casting)은 특정 객체가 하위 클래스의 형에서 상위 클래스의 형으로 형변환되는 것으로 형만 정확하다면 묵시적으로 형변환됩니다.

위와 같이 객체 생성할 경우 자동으로 업 캐스팅이 일어납니다.

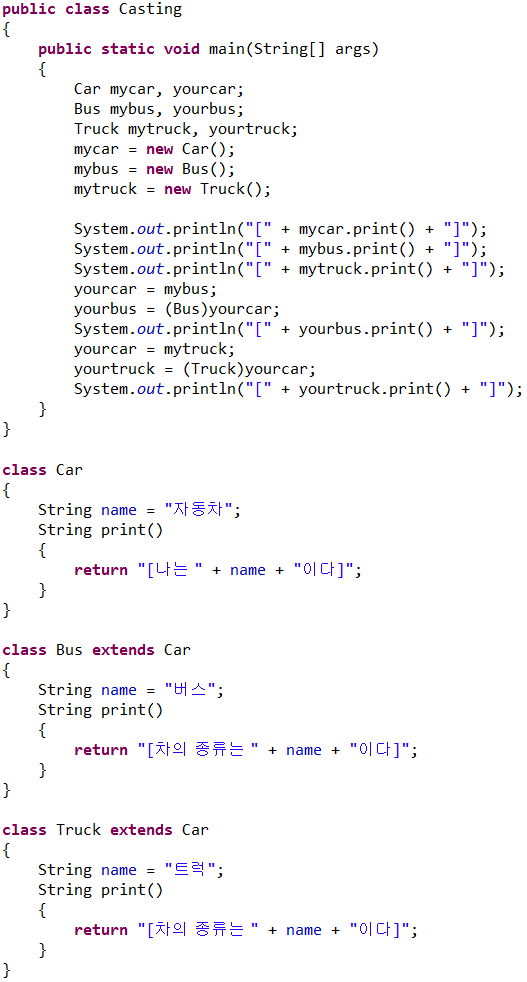
**- 다운 캐스팅**

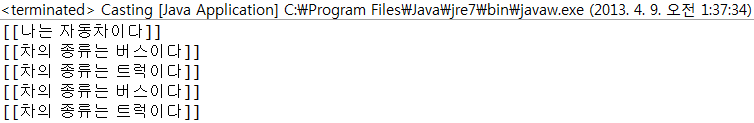
http://cfile24.uf.tistory.com/image/145BFF3D5162F07F12F112

다운 캐스팅(Down Casting)은 업 캐스팅한 것을 다시 원래의 형으로 복구시켜 주는 것으로 업 캐스팅된 것만 다운 캐스팅시킬 수 있습니다.

다운 캐스팅은 강제 캐스팅을 원칙으로 합니다.

예제





**+ 가상 메소드**

- 상위 클래스의 이름으로 하위 클래스의 메소드를 호출할 수 있도록 하는 기법으로 업 캐스팅했을 때 상위 클래스의 이름으로 하위 클래스의 메소드를 호출하도록 합니다.

- 원래 상속관계에 있을 때 상위 클래스의 객체 변수는 하위 클래스의 메소드에 접근할 수 없으나 업 캐스팅을 했을 경우 하위 클래스 메소드를 접근할 수 있도록 하는 것을 말합니다.

**1. 추상 클래스**

- 상속 관계에 있는 클래스 중에서 상위 클래스에서는 특별한 구현 없이 사용하고자 하는 메소드만 기술하고 구체적인 구현은 하위 클래스에서 하도록 할 수 있습니다.

- 이때 사용하는 것이 추상 클래스로 구체적인 내용 기술 없이 모양만 갖춘 클래스입니다.

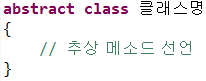
- 추상 클래스는 abstract 키워드를 사용하여 표시하고 추상 클래스 안에는 추상 메소드를 가집니다.

**+ 추상 클래스**

- 추상 클래스(Abstract Class)는 추상 메소드를 가진 클래스로 추상 클래스는 new 연산자로 객체 생성할 수 없습니다.

- 따라서 추상 클래스의 추상 메소드는 자신이 직접 이용하지 못 하고 반드시 하위 클래스에서 이 메소드를 상속받아 구현하여 사용해야합니다.

**- 형식**



- 추상 클래스의 추상 메소드는 반드시 오버라이딩되어야 하기 때문에 하위 클래스들이 특정 메소드를 반드시 구현하도록 강제할 수 있습니다.

- 그리고 만약 추상 메소드를 오버라이딩하지 않으면 상속받는 클래스는 자동으로 추상 클래스가 됩니다.

- 추상 클래스는 일반 메소드와 추상 메소드를 모두 포함할 수 있습니다.

**+ 추상 메소드**

- 추상 메소드는 메소드의 이름 앞에 abstract가 있으며, 구현 내용이 없는 메소드로 이를 상속받는 하위 클래스들이 반드시 재정의해서 사용해야 하는 메소드들을 지정하고자 할 때 사용합니다.

- 메소드의 구현 내용({})이 없고, 메소드 다음에 세미콜론(;)을 붙입니다.

**- 형식**

http://cfile9.uf.tistory.com/image/1449B0435162FBE40178B6

**2. 인터페이스**

- 자바는 다중 상속을 지원하지 않는 반면에 다중 상속과 유사한 기능이 있는 인터페이스를 제공합니다.

- 만약 어린이 클래스를 정의하는데 아버지 클래스와 어머니 클래스를 동시에 상속받을 필요가 있을 때 자바는 다중 상속이안 되기 때문에 아버지와 어머니 클래스로부터 동시에 상속받을 수 없습니다.

- 이때, 아버지 클래스로부터 상속 받고 어머니를 인터페이스로 정의하여 어린이 클래스에서 구현하면 다양한 기능을 부가적으로 사용할 수 있습니다.

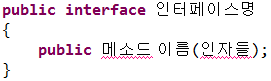
- 인터페이스는 모든 메소드가 추상 메소드이며, 데이터는 final static 변수만을 가지는 특별한 형태의 클래스입니다.

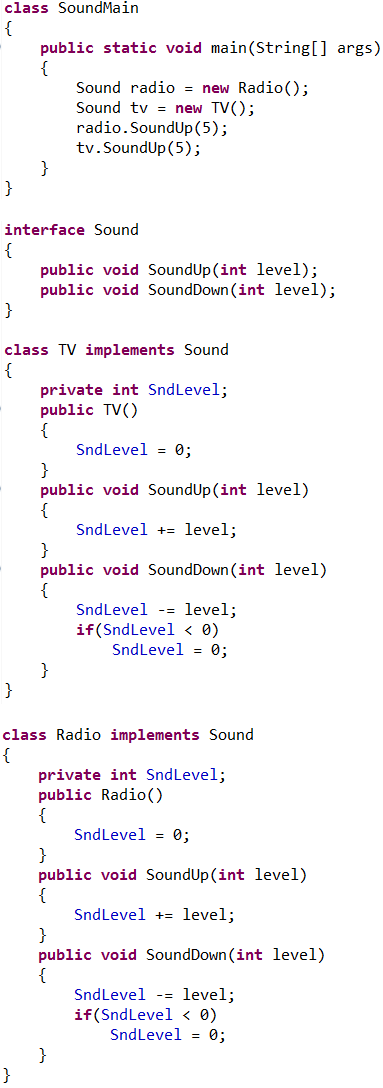
- 인터페이스는 클래스가 아니기 때문에 객체가 가질 수 있는 메소드 이름만을 명시하고 실제 기능은 인터페이스를 상속받아 구현하는 클래스에서 기술해야 합니다.

- 즉, 인터페이스를 적용한 클래스는 인터페이스에서 선언된 메소드를 모두 구현해 주어야합니다.

- 만일 해당 메소드를 구현하지 않은 클래스는 추상 클래스가 됩니다.

**- 형식**



**+ 인터페이스 상속**

- 인터페이스는 일반 클래스처럼 또 다른 인터페이스를 상속받을 수 있습니다.

**- 형식**

http://cfile28.uf.tistory.com/image/127975455162FDE002BF08