

# 프로그래밍기초

---

# 차 례

---

## 학습 1. 개발환경 구축하기

1-1. 자바 개발도구(JDK, Eclipse) 설치 .....	1
-------------------------------------	---

## 학습 2. 자바 기본 문법

2-1. 변수 .....	9
2-2. 연산자 .....	26
2-3. 명령문 .....	48
2-4. 배열 .....	67

## 학습 3. 객체지향 프로그래밍

3-1. 클래스와 객체 .....	88
3-2. 변수와 메서드 .....	96
3-3. 오버로딩 .....	114
3-4 생성자 .....	117
3-5. 상속 .....	127
3-6. 오버라이딩 .....	141
3-7. 제어자 .....	144
3-8. 다형성 .....	151
3-9. 추상클래스 .....	162
3-10. 인터페이스 .....	166

## 학습 4 예외 처리

4-1. 프로그램 오류 .....	171
--------------------	-----

## 학습 5 자바 주요 클래스

5-1. java.lang 패키지 ----- 181

5-2. 정규식 ----- 193

## 학습 6 Collection Framework

6-1. 컬렉션 프레임워크 ----- 196

## 학습 7 DB프로그래밍

7-1. JDBC 프로그래밍 ----- 213

## 학습 1 개발환경 구축하기

### 1-1. 자바 개발도구(JDK, Eclipse) 설치

#### 학습목표

- 응용소프트웨어 개발에 필요한 하드웨어 및 소프트웨어의 필요 사항을 검토하고 이에 따라, 개발환경에 필요한 준비를 수행할 수 있다.

#### 필요 지식 /

#### ① 자바 개발도구 JDK설치 및 환경변수설정

##### 1. 자바 개발도구 (JDK) 설치 개요

자바로 프로그래밍을 하기 위해서는 먼저 JDK(Java Development Kit)를 설치해야 한다. JDK를 설치하면 자바 가상머신(Java Virtual Machine, JVM)과 자바클래스 라이브러리(Java API)와 자바를 개발하는데 필요한 프로그램들이 설치된다.

이 책을 학습하기 위해서는 JDK7.0이상의 버전이 필요하며 'http://java.sun.com/'에서 다운로드 받을 수 있다.

(1) JDK의 bin디렉토리에 있는 주요 실행파일은 다음과 같다.

- javac.exe : 자바 컴파일러, 자바소스코드를 바이트코드로 컴파일 한다.
- java.exe : 자바 인터프리터, 컴파일러가 생성한 바이트코드를 해석하고 실행한다.
- javap.exe : 역어셈블러, 컴파일된 클래스파일을 원래의 소스로 변환한다.

(2) 자바개발도구 참고 용어

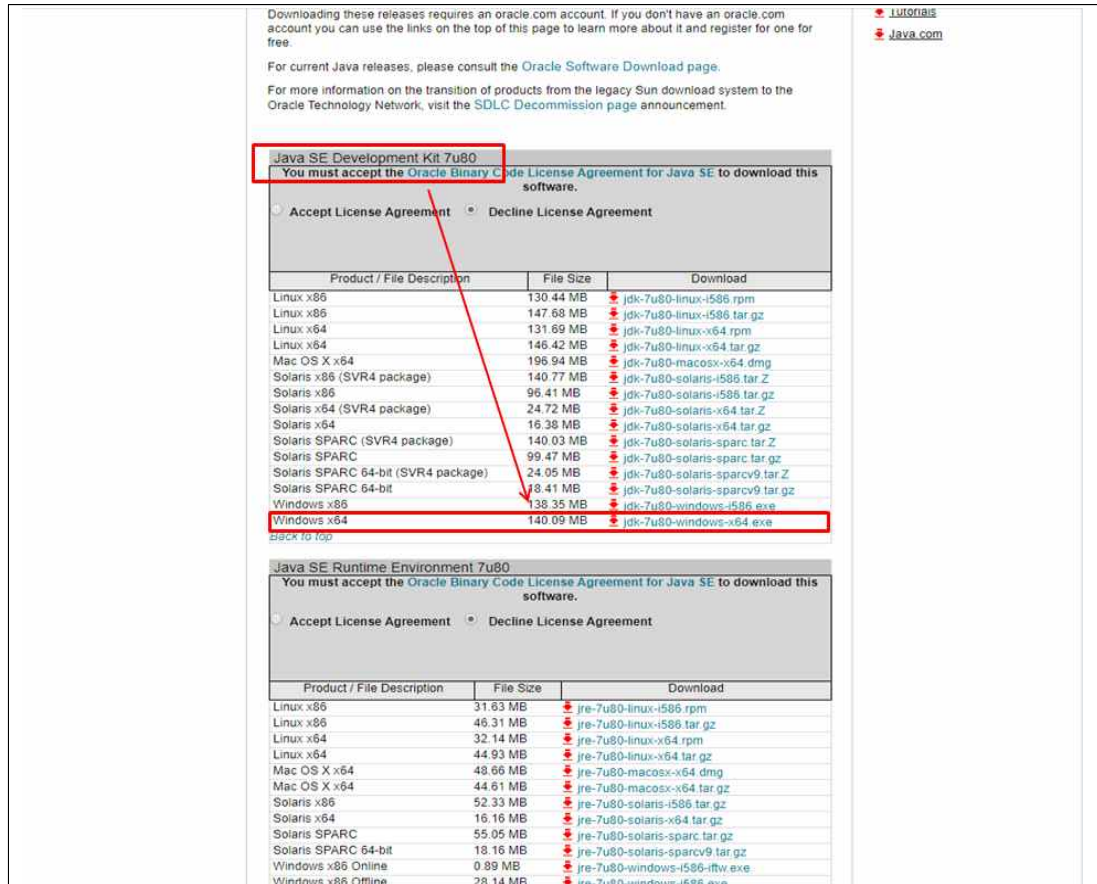
- JRE - 자바실행환경 (Java Runtime Environment), 자바로 작성된 응용프로그램이 실행되기 위한 최소 환경.
- JDK - 자바개발도구 (java Development Kit)
- JRE = JVM + 클래스라이브러리 (Java API)
- JDK = JRE + 개발에 필요한 실행파일 (javac.exe 등)

##### 2. 자바 개발도구 (JDK) 설치

(1) JDK다운로드

- <https://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase7-521261.html> 웹사이트로 이동한다.

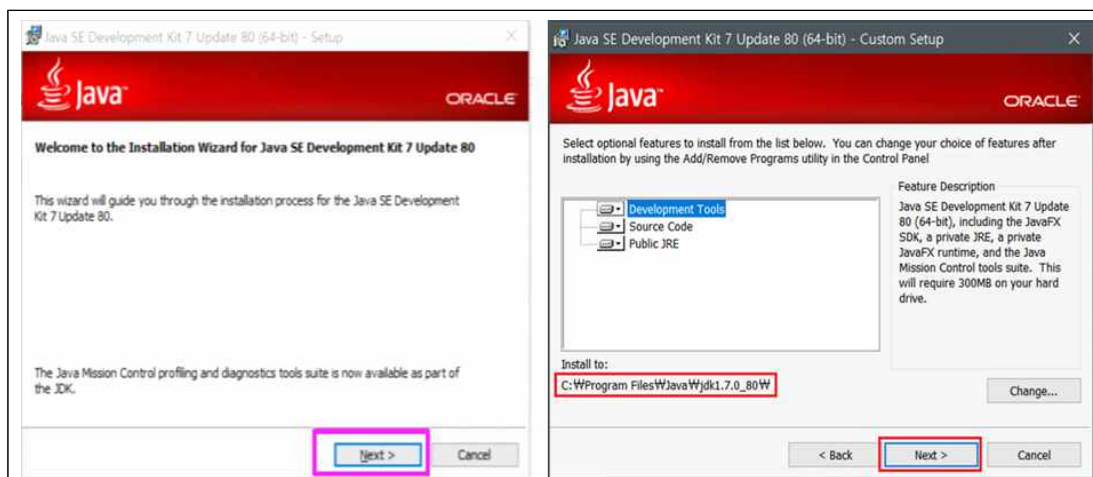
- 여러 운영체제에서 자신의 컴퓨터 운영체제에 맞는 JDK를 다운로드 한다.
- 공통으로 설치된 Windows x64 버전의 JDK를 다운로드 한다.



[그림 1-1] JDK 다운로드

## (2) JDK설치

- 다운로드한 프로그램을 실행한다.
- 실행화면에서 Next를 선택 후 설치경로(Install to)를 change버튼을 이용하여 변경한다.



[그림 1-2] JDK 설치

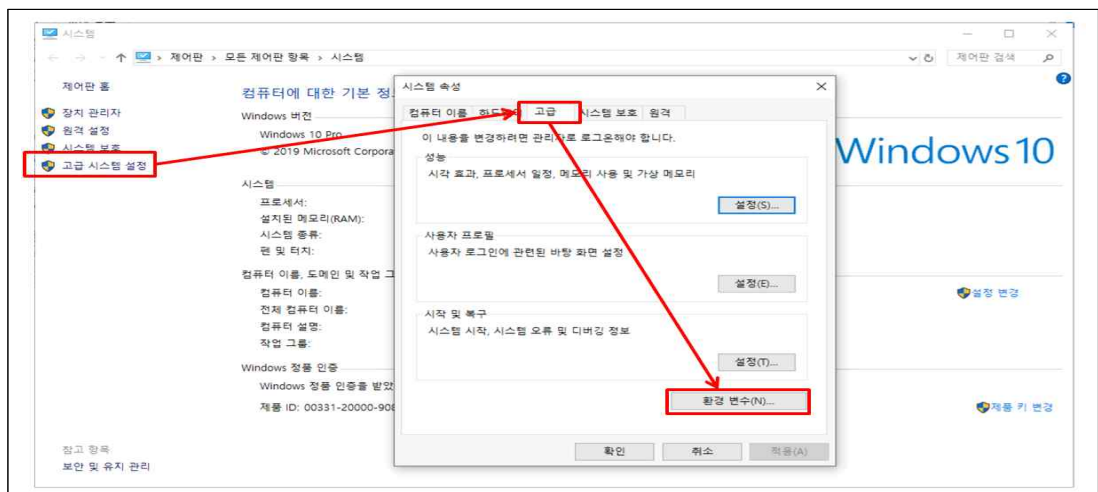
### 3. 환경변수 설정

#### (1) 환경변수 설정 개요

- JDK설치 이후 환경변수를 등록하는 이유는 환경변수에 경로는 컴퓨터의 어떤 경로에서라도 접근(=실행)이 가능하기 때문이다.  
즉, 파일의 접근이 쉽고 편하게 실행하기 위한 환경변수 등록을 한다고 생각하면 된다.
- 해당 설정 방법은 운영체제에 따라 상이하다.

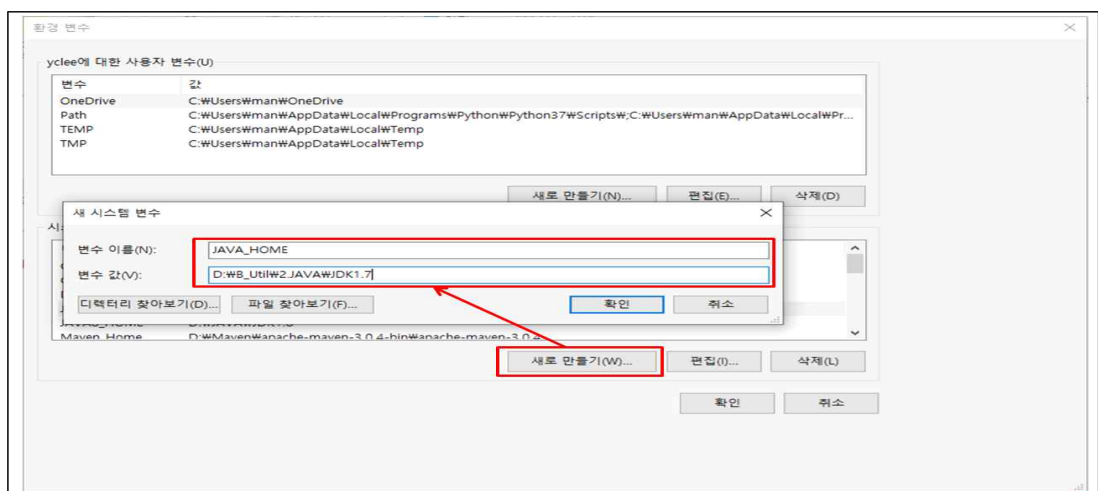
#### (2) 환경변수 설정하기

- 제어판 -> 모든 제어판 항목 -> 시스템으로 이동한다.
- 좌측 상단의 고급 시스템 설정 탭으로 이동한다.
- 시스템속성 탭의 고급 탭의 환경변수 버튼을 클릭한다.



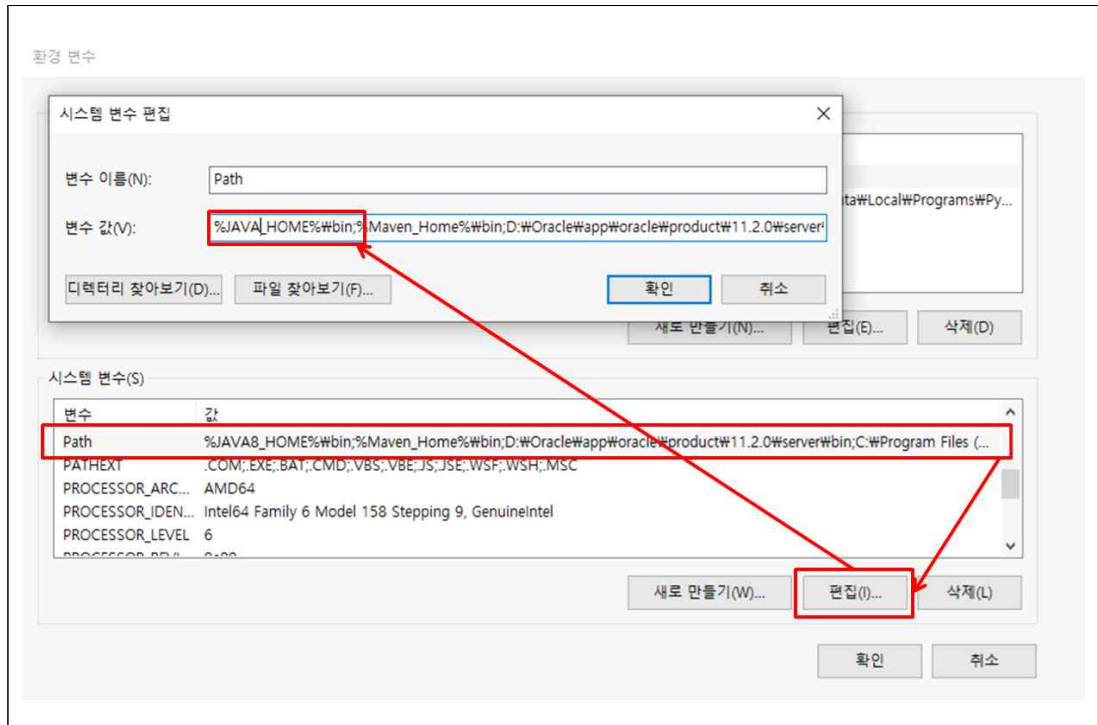
[그림 1-3] 환경변수 찾기

- 시스템 변수(S)에 새로 만들기를 선택한다.
- 변수 이름은 JAVA\_HOME으로 기입한다.
- 변수 값에는 경로 찾기를 이용하여 JAVA가 설치된 폴더를 선택한다.



[그림 1-4] 시스템변수 설정

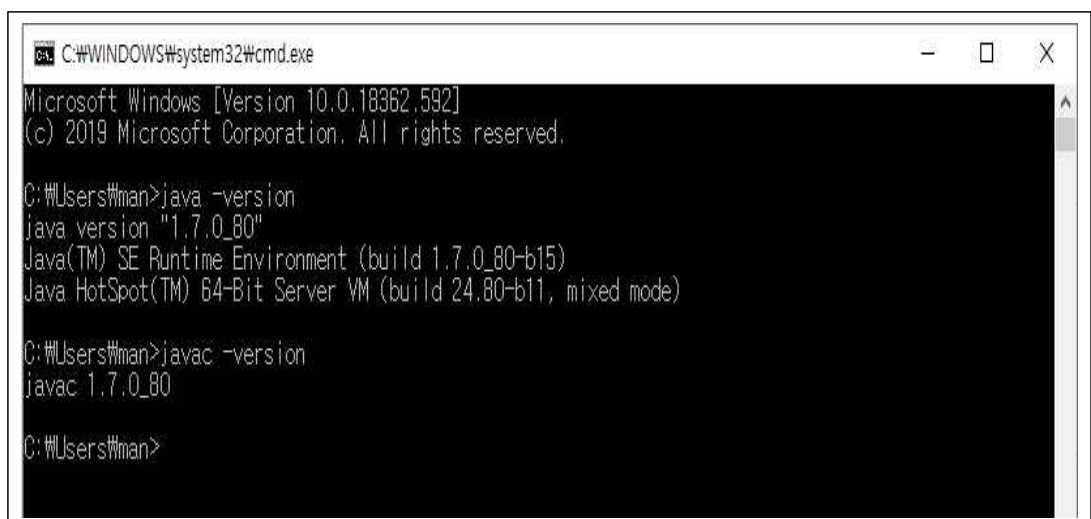
- 시스템 변수(S)의 변수에서 Path를 찾아 선택하여 준다.
- 편집버튼을 클릭하여 시스템변수 편집탭을 활성화한다.
- 변수 값 가장 앞부분에 '%JAVA\_HOME%\bin;'을 입력 후 확인 버튼을 클릭한다.



[그림 1-5] 환경변수 추가

### (3) 환경변수 설정 확인

- command창을 활성화한다. ( 윈도우 + R을 누른 후 실행 창에 cmd를 입력 한다.)
- 'java -version'을 입력한다.
- 'javac -version'을 입력한다.
- 아래와 같이 출력되는지 확인한다.



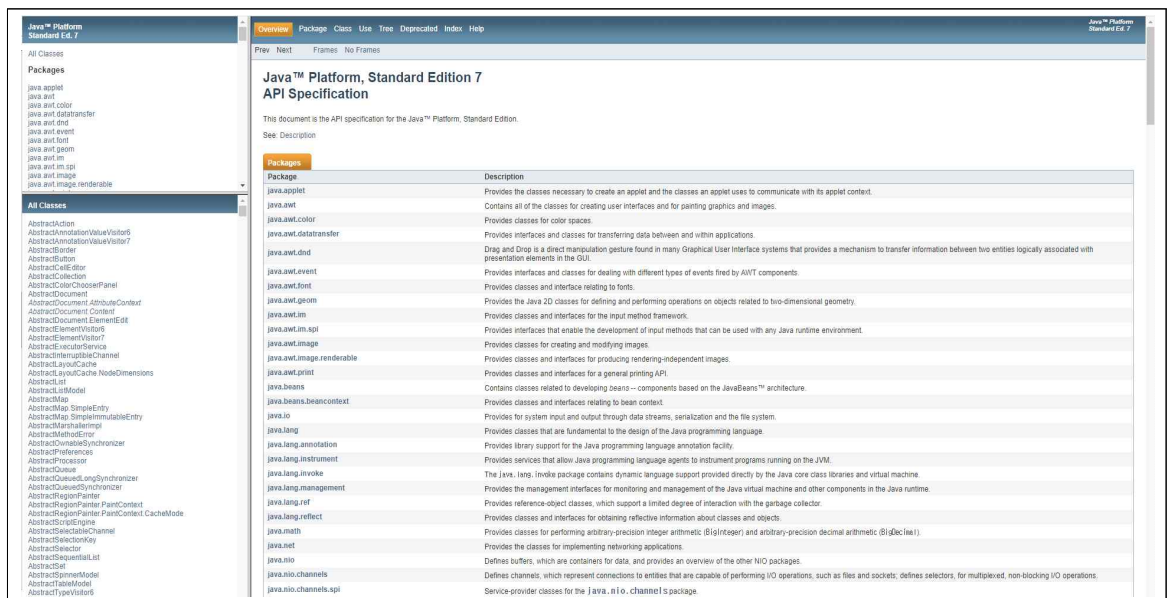
[그림 1-6] 환경변수 설정 확인

## ② 자바 API

### 1. 자바 API 개요

자바에서 제공하는 클래스 라이브러리(Java API)를 잘 사용하기 위해서는 Java API문서가 필수적이다. 이 문서에는 클래스 라이브러리의 모든 클래스에 대한 설명이 자세하게 나와 있다. 많은 양에 걱정도 들겠지만 모두 공부할 필요는 없고 자주 사용되는 것을 공부한 다음 나머지는 필요할 때 사전처럼 사용하면 된다.

Java API문서는 브라우저에 '<https://docs.oracle.com/javase/7/docs/api/>'의 주소를 입력하면 웹에서 확인이 가능하다.



[그림 1-7] 자바 API

- 좌측상단 : package
- 좌측하단 : class
- 우측전체 : class내 method 및 변수에 대한 설명

## ③ 자바 개발도구 Eclipse 설치

### 1. 자바 개발도구 Eclipse 설치 개요

#### (1) 통합 개발환경의 종류 및 특징

다양한 프로그래밍 언어를 지원하는 통합 개발 환경의 예로, 비주얼 스튜디오, 이클립스를 들 수 있다. 이클립스는 자바를 기본적으로 지원하지만, 파이썬, 펄, 루비, 포트란, C, C++, PHP, 코볼, JSP 등과 같은 언어들도 추가적으로 설치할 수 있다. 각 언어의 추가 설치본은 각자 고유의 디버거를 비롯한 다양한 도구들을 가지고 있다.



통합 개발환경	개발사	지원 OS	지원언어	라이선스
Eclipse	IBM 이클립스 재단	MS-Windows Linux, OSX Solaris, AIX	Java, C C++, PHP JSP 외 다수	Eclipse Public
Lazarus	Lazarus Team	MS-Windows Linux, OSX FreeBSD	Pascal	GPL, GNU LGPL, 기타
Anjuta	GNOME 프로젝트	Linux	C C++	GPL
Wide Studio	와이드 스튜디오 프로젝트	Linux	C C++	MIT
Code::Blocks	Code::Blocks Team	MS-Windows Linux OSX	C C++	GPL v3.0
Visual Studio	Microsoft	MS-Windows	Visual Basic .Net Visual C++ Visual C# 등	상용
Delphi	엠바카데로 테크놀로지	MS-Windows	Pascal	상용
C++ Builder	엠바카데로 테크놀로지	MS-Windows	C C++	상용

[표 1-1] 통합 개발환경 종류 및 주요 특징

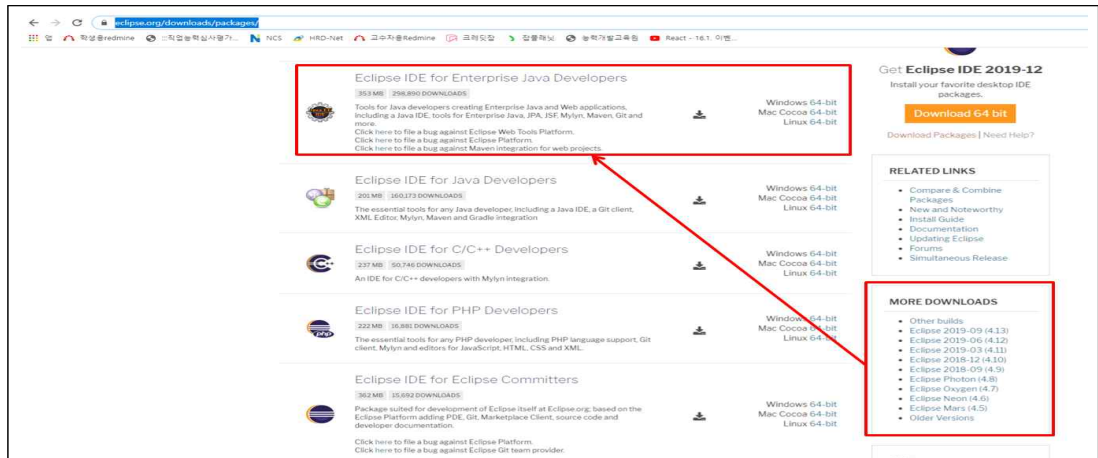
(2) 통합 개발환경의 선정

- Eclipse IDE를 개발도구로 선정: 풍부한 기능과 Plug-In을 보유하고 있으며, 다양한 적용사례가 존재한다.

## 2. 자바 개발도구 Eclipse 설치

(1) eclipse 다운로드

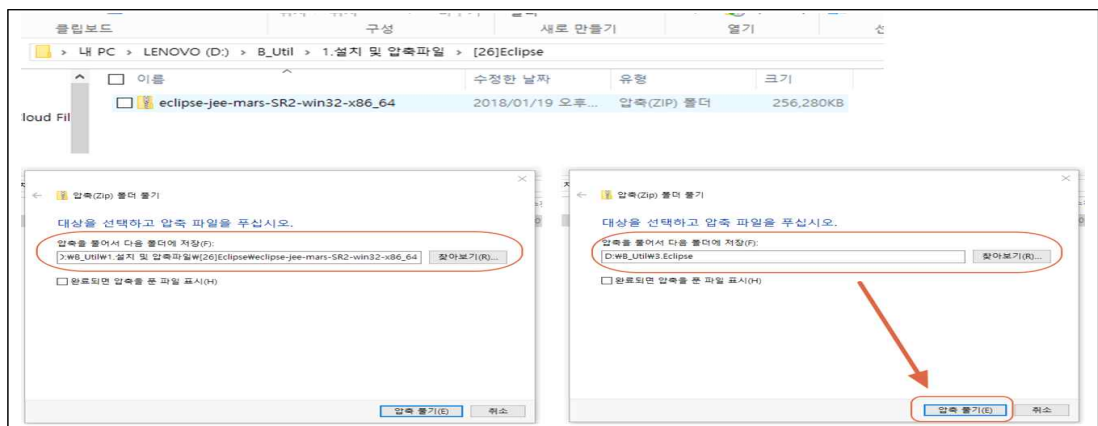
- '<https://www.eclipse.org/downloads/packages/>' 웹사이트로 이동한다.
- 요구사항에 적합한 Eclipse를 다운받아 해당 파일을 확인한다.
- 요구사항에 따라 최신버전일 경우 상단의 Eclipse를 다운로드 한다.
- 이전 버전의 Eclipse가 필요한 경우 우측하단의 Eclipse중 선택적으로 다운로드 한다.
- 다운로드를 진행시 자신의 운영체제에 맞는 Eclipse를 다운로드 한다.



[그림 1-8] Eclipse 다운로드

## (2) Eclipse 설치

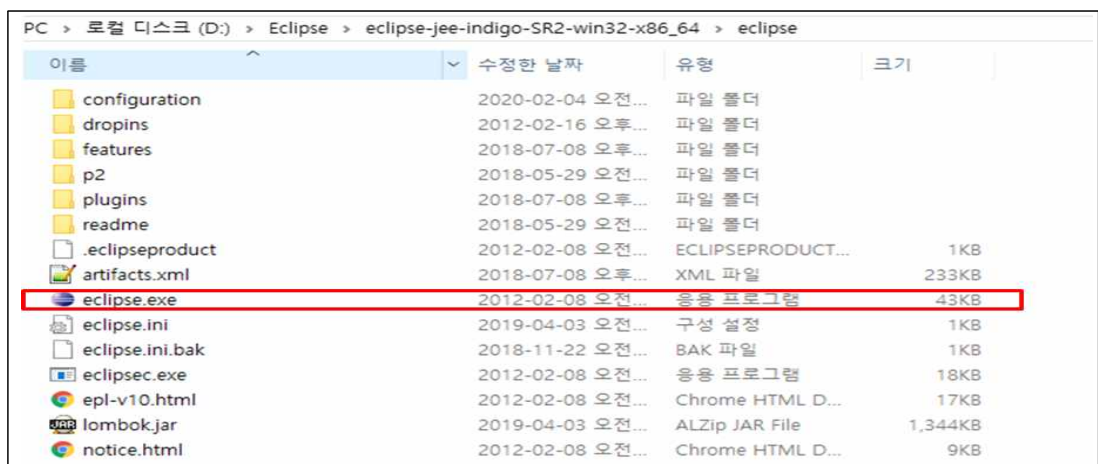
- Eclipse의 경우 설치파일이 아닌 압축을 풀어서 사용한다.
- 다운받은 파일의 압축을 풀어준다.



[그림 1-9] 압축풀기

## (3) Eclipse 실행

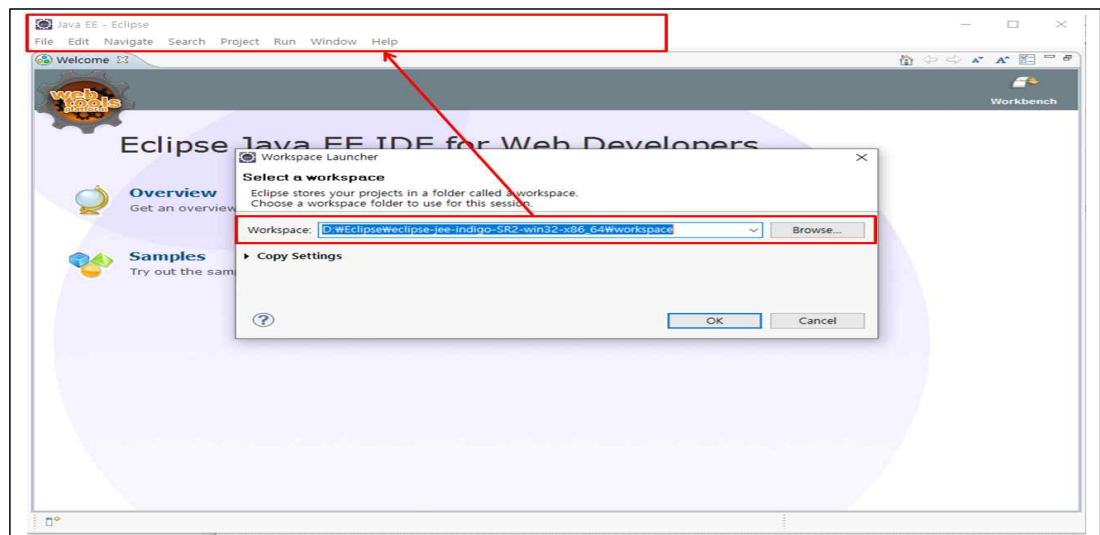
- 압축 해제된 폴더의 eclipse.exe 실행 파일을 선택하여 Eclipse를 실행한다.



[그림 1-10] Eclipse 실행

#### (4) Eclipse 실행 확인

- Eclipse 실행 이후에는 workspace를 생성하고 Eclipse의 실행 화면을 확인한다.



[그림 1-11] Eclipse 실행 확인

## 학습 2 자바 기본 문법

### 2-1. 변수

#### 학습목표

- 응용소프트웨어 개발에 필요한 프로그래밍 언어의 데이터 타입을 적용하여 변수를 사용할 수 있다.

#### 필요 지식 /

#### 1 프로그래밍 언어 활용의 개요

프로그래밍 언어란 컴퓨터 시스템을 동작시키기 위한 프로그램 작성 언어를 말한다. 프로그램은 다소 단순해 보이는 명령어들의 조합으로 구성되는데, 이러한 조합들은 비트(Bit)라고 불리는 0과 1의 값으로 작성되거나 변환되어 컴퓨터가 이해할 수 있도록 한다.

##### 1. 비트(Bit)

비트는 컴퓨터를 이해하기 위한 가장 기본적인 용어로, Binary Digit의 약칭이다. 0과 1로만 구성된 이진법(ex. 101001)을 이용하며, 컴퓨터 이용 시 흔히 접할 수 있는 1바이트(Byte)는 8비트(Bit)를 의미한다.

구분	설명
바이트	1Byte(B) = 8bit, 1bit = 0.125B
킬로바이트	1kilobyte(KB) = 8,192bit, 1bit = 0.000122KB
메가바이트	1megabyte(MB) = 8,388,608bit, 1bit = 1.1921e-7MB
기가바이트	1Gigabyte(GB) = 8.5899e + 9bit, 1bit = 1.1642e-10GB
테라바이트	1Terabyte(TB) = 8.7961e + 12bit, 1bit = 1.1369e-13TB
페타바이트	1Petabyte(PB) = 9.0072e + 15bit, 1bit = 1.1102e-16PB
엑사바이트	1Exabyte(EB) = 9.2234e + 18bit, 1bit = 1.0842e-19EB

[표 2-1] 비트(bit)와 바이트(Byte)의 관계

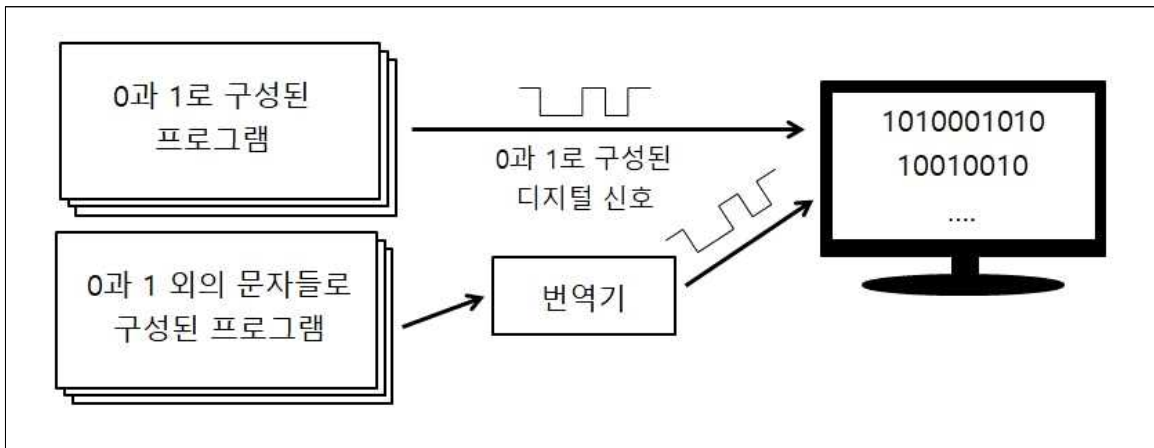
0~9까지의 10진수를 2진법으로 표현하면 [그림 2-1]과 같다.

0 <sub>10</sub> : 0000 <sub>2</sub>	1 <sub>10</sub> : 0001 <sub>2</sub>	2 <sub>10</sub> : 0010 <sub>2</sub>	3 <sub>10</sub> : 0011 <sub>2</sub>	4 <sub>10</sub> : 0100 <sub>2</sub>
5 <sub>10</sub> : 0101 <sub>2</sub>	6 <sub>10</sub> : 0110 <sub>2</sub>	7 <sub>10</sub> : 0111 <sub>2</sub>	8 <sub>10</sub> : 1000 <sub>2</sub>	9 <sub>10</sub> : 1001 <sub>2</sub>

[그림 2-1] 0~9에 대한 10진수의 2진수 표현

## 2. 컴퓨터에 명령 전달 방법

컴퓨터에 명령을 전달할 때에는 비트(Bit)로 전달이 된다. 비트의 0과 1을 컴퓨터가 이해할 수 있는 방식으로 변환하여 전달하게 되는데, 그 방법으로는 전기적 신호를 이용하거나 광 신호를 이용하는 방식 등이 있다. 예를 들어 0은 0V(볼트), 1은 5V(볼트)의 전기적 신호에 매칭시켜 컴퓨터가 이해할 수 있도록 할 수 있다. 그 밖에 불을 깜빡이는 횟수에 차이를 두어 컴퓨터가 신호를 받아들일 수 있도록 하는 방식 등이 있다.



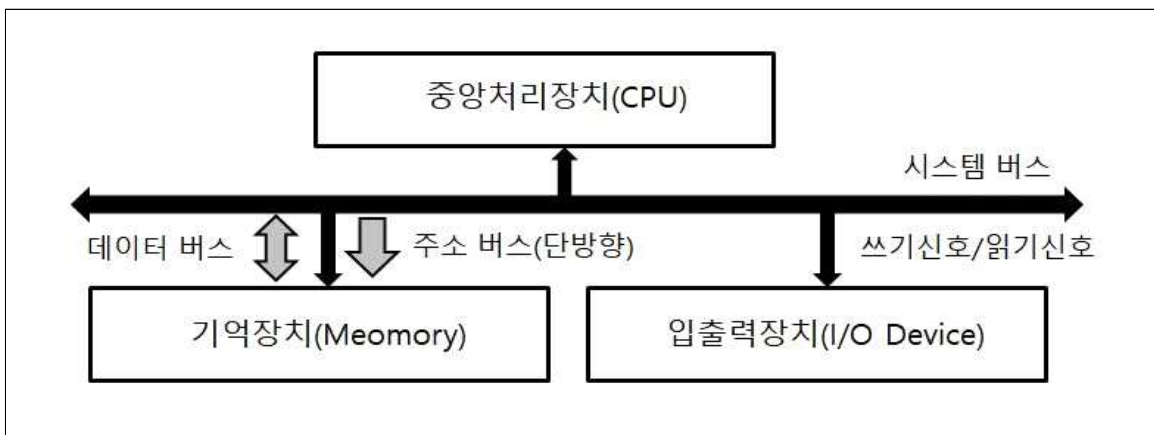
[그림 2-2] 컴퓨터에 명령 전달을 위한 신호 변환 과정

어떠한 프로그래밍 언어를 사용하느냐에 따라 비트(Bit) 변환 없이 프로그램 자체를 0과1로만 작성할 수도 있으나, 이 경우에는 어느 정도의 한계가 있다. 따라서 보다 쉽게 작성하여 컴퓨터에 전달하고, 더욱 복잡한 프로그램을 구현하기 위해 프로그래밍 언어는 점진적으로 발전하게 되었다.

## 3. 컴퓨터 시스템 구조

컴퓨터는 전달받은 0과 1의 값들을 정해진 순서대로 실행하며 그 과정에서 처리(Processing), 저장(Store) 등을 수행한다.

프로그램 실행 및 데이터 처리는 중앙 처리 장치(CPU)에서 수행하고, 저장은 기억 장치(Memory)에서 수행하며, 중앙 처리 장치(CPU)로부터 명령을 받아 데이터를 입력 또는 출력하는 일은 입출력 장치(I/O Device)에서 수행한다.



[그림 2-3] 컴퓨터 시스템 구조

#### 4. 프로그램의 구성 요소

프로그램은 크게 자료 구조와 알고리즘으로 구성된다.

##### (1) 자료 구조

자료 구조는 컴퓨터에 데이터를 삽입, 삭제, 수정하게 해 주는 논리적인 공간 구조를 의미하며, 자료의 형태에 따라 단순 구조, 선형 구조, 비선형 구조, 파일 구조로 분류할 수 있다.

유형	설명
단순 구조(Simple)	프로그래밍 언어에서 제공하는 기본 데이터 타입이다. (ex. int, float, double, char 등)
선형 구조(Liner)	자료들 사이의 선후 관계가 일대일인 구조이다. (ex. 연결 자료 구조, 순차 자료 구조, 스택, 큐 등)
비선형 구조 (Non-Liner)	자료들 사이의 선후 관계가 계층 또는 그물 형태를 가지는 구조이다. (ex. 트리, 그래프 등)
파일구조(File)	보조 기억 장치에 데이터값이 실제로 기억되는 자료 구조이다. (ex. 순차 파일, 색인 파일 등)

[표 2-2] 자료의 형태에 따른 자료 구조 분류

##### (2) 알고리즘

알고리즘은 넓은 의미에서 자료 구조와 함께 프로그램을 구성하는 요소를 의미하며, 좁은 의미에서는 어떤 문제에 대한 답을 찾는 해법을 의미한다. 이 알고리즘은 기본적인 명령어를 통해서도 작성 가능한데, [표 2-3]과 같은 특성을 만족하여야 한다.

유형	설명
입력	외부로부터 입력되는 자료가 0개 이상이어야 한다.
출력	출력되는 결과가 1개 이상이어야 한다.
명확성	각 명령어의 의미가 명확하여야 한다.
유한성	정해진 단계를 지나면 종료되어야 한다.
유효성	모든 명령은 실행이 가능한 연산들이어야 한다.

[표 2-3] 알고리즘의 5가지 특성

알고리즘의 표현은 자연어, 순서도, 의사 코드, 프로그래밍 언어를 이용하는 방법이 있으며, 따라서 프로그래밍 언어가 아니더라도 알고리즘의 표현은 가능하다.

## ② 출력하기

화면에 글자를 출력할 때는 `System.out.println()`을 사용한다. 출력하고자 하는 내용을 괄호() 안에 넣으면 된다.

```
System.out.println("Hello, Java"); //화면에 Hello, Java가 출력된다.
System.out.println("2+5");         //화면에 2+5가 출력된다.
System.out.println(5);             //화면에 5가 출력된다.
System.out.println(2+5);           //화면에 연산의 결과인 7이 출력된다.
```

위의 코드에서 알 수 있듯이 괄호() 안에 숫자를 넣으면 계산된 결과가 출력되지만 큰따옴표 ""안에 넣은 내용은 글자로 간주되어 계산되지 않고 그대로 출력된다.

`System.out.println()` 외에도 `System.out.print()`가 있는데, 이 둘의 차이는 아래와 같다.

```
System.out.println() // 괄호 안의 내용을 출력하고 줄바꿈을 한다.
System.out.print()   // 괄호 안의 내용을 출력하고 줄바꿈을 하지 않는다.
```

줄바꿈을 하지 않으면, 이전에 출력된 내용 바로 뒤에 이어서 출력된다.

### [예제 2-1]

```
class Ex2_1{
    public static void main(String[] args){
        System.out.println("Hello"); // 화면에 Hello를 출력하고 줄바꿈을 한다.
        System.out.println("Java");  // 화면에 Java를 출력하고 줄바꿈을 한다.
        System.out.print("Hi ");     // 화면에 Hi를 출력하고 줄바꿈을 안한다.
        System.out.print("Friends"); // 화면에 Friends를 출력하고 줄바꿈을 안한다.
    }
}
```

결과	Hello Java Hi Friends
----	-----------------------------

[문제 2-1] 결과의 내용과 동일한 결과가 나오도록 코드를 작성해 주세요.

```
class Qu2_1{
    public static void main(String[] args){
        //1. '3+5='를 출력한다.

        //2. 3+5의 연산결과를 출력한다.
    }
}
```

결과	3+5=8
----	-------

### ③ 변수(variable)

중요한 프로그래밍 능력 중의 하나가 바로 '값(data)을 잘 다루는 것'이다. 값을 저장하는 공간인 변수를 잘 이해하고 활용하는 것은 그 능력을 얻기 위한 첫걸음이다.

#### 1. 변수(variable)란?

수학에서 '변수'를 '변하는 수'라고 정의하지만 프로그래밍언어에서는 변수(variable)란, 값을 저장할 수 있는 메모리상의 공간을 의미한다. 이 공간에 저장된 값은 변경될 수 있기 때문에 '변수'라는 수학용어의 정의와 상통하는 면이 있어서 이렇게 이름 붙여졌다.

#### 2. 변수의 선언과 초기화

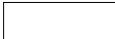
값을 저장할 하려면 공간이 필요하다면 변수를 선언해야 한다. 변수를 선언하는 방법은 다음과 같다.

**변수타입 변수이름 ; // 변수를 선언하는 방법**

변수의 타입은 변수에 저장할 값이 어떤 것이냐에 따라 달라지며, 변수의 이름은 저장공간이 서로 구별될 수 있어야 하기 때문에 필요하다. 예를 들어 정수(integer)를 저장할 공간이 필요하다면 다음과 같이 변수를 선언한다.

**int x ; // 정수(integer)를 저장하기 위한 변수 x를 선언**

위의 문장이 수행되면, x라는 이름의 변수(저장 공간)이 생기며, 그림으로 그리면 다음과 같다.

x 


변수를 선언하였다면 이 변수에 값을 저장할 수 있으며 다음과 같이 한다.

**x = 5 ; // 변수 x에 5를 저장** 

위와 같이 변수를 선언 후 처음으로 값을 저장하는 것을 '변수의 초기화'라고 한다.

수학에서는 '='가 같음을 의미하지만, 자바에서는 오른쪽의 값을 왼쪽에 저장하라는 의미의 '대입 연산자(assignment operation)'이다.

변수는 오직 하나의 값만 저장할 수 있기 때문에, 이미 값이 지정된 변수에 새로운 값을 저장하면 기존의 값은 지워지고 새로 저장된 값만 남는다.

**x = 3 ; // 변수 x에 새로운 값 3을 저장.** 



[문제 2-2] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_2{
    public static void main(String[] args){
        //1. 정수(integer)를 저장할 수 있는 변수 value를 선언하여라.

        //2. 변수 value를 30의 값으로 초기화 하여라.

        //3. 변수 value의 값을 100으로 변경 하여라.

        //4. 정수(integer)를 저장할 수 있는 변수 value2를 선언하고 7의 값으로 초기화 하여라.

    }
}
```

### 3. 변수의 명명규칙

'변수의 이름'처럼 프로그래밍에서 사용하는 모든 이름을 '식별자(identifier)'라고 하며, 식별자는 같은 영역 내에서 서로 구분(식별)될 수 있어야 한다. 그리고 식별자를 만들 때는 다음과 같은 규칙을 지켜야 한다.

(1) 대소문자가 구분되며 길이에 제한이 없다.

`int num ;` // 변수명이 소문자로만 이루어져 있다.

`int nuM ;` // 변수명이 소문자뿐만 아니라 대문자가 들어간다.

위 두 개의 변수는 대소문자를 구분하기 때문에 다른 변수로 식별된다.

(2) 예약어(keyword, reserved word)는 사용할 수 없다.

`int truE ;` // 대문자가 포함된 truE는 예약어가 아니다.

`int true ;` // 소문자로만 이루어진 true의 명칭은 예약어 이다.

※ 팁 : Eclipse에 변수를 선언할 때 명칭의 색이 자주색으로 표현되는 경우 예약어이다.

(3) 숫자로 시작해서는 안 된다.

`int 7top ;` // 변수명의 시작은 숫자가 올수 없다.

`int top10 ;` // 변수명의 뒤쪽에는 숫자가 올수 있다.

(4) 특수문자는 '\_'와 '\$'만을 허용한다.

`int fried_chicken$ ;` // 변수명이 사용가능한 특수문자만 들어가 있으므로 사용할 수 있다.

`int fried&chicken@ ;` // 변수명이 사용불가인 특수문자(&,@)가 있어 사용할 수 없다.

※ 필수는 아니지만 가독성 향상을 위해 자바프로그래머들의 암묵적인 약속들.

(1) 클래스명의 첫 글자는 대문자로 써야 한다.

(2) 여러 단어로 이루어진 경우 첫 번째 이후 단어의 첫 글자는 대문자로 써야 한다.

(3) 상수의 이름은 모두 대문자로 써야 한다. 여러 단어일 경우 '\_'로 구분한다.

#### 4. 변수의 타입

변수를 선언할 때는 저장하려는 값의 특성을 고려하여 가장 알맞은 자료형을 변수의 타입으로 선택하면 된다.

##### (1) 기본형과 참조형

자료형은 크게 '기본형'과 '참조형' 두 가지로 나눌 수 있는데, 기본형 변수는 실제 값(data)을 저장하는 반면, 참조형 변수는 어떤 값이 저장되어 있는 주소(memory address)를 값으로 갖는다.

자바는 C언어와 달리 참조형 변수 간의 연산을 할 수 없으므로 실제 연산에 사용되는 것은 모두 기본형 변수이다.

##### 기본형(primitive type)

- 계산을 위한 실제 값을 저장한다. 모두 8개
- boolean, char, byte, short, int, long, float, double

##### 참조형(reference type)

- 객체의 주소를 저장한다.
- 기본형 8개를 제외한 나머지 타입

참조형 변수(또는 참조변수)를 선언할 때는 변수의 타입으로 클래스의 이름을 사용하므로 클래스의 이름이 참조변수의 타입이 된다. 그래서 새로운 클래스를 작성한다는 것은 새로운 참조형을 추가하는 셈이다.

##### ※ 자료형(data type)과 타입(type)의 차이점

기본형의 종류를 얘기할 때는 '자료형(data type)'이라는 용어를 사용하고 참조형의 경우 객체의 주소를 저장하므로 값(data)이 아닌 객체의 종류에 의해 구분되므로 참조형 변수의 종류를 구분할 때는 '타입(type)'이라는 용어를 사용한다. '타입'이 '자료형'을 포함하는 보다 넓은 의미의 용어이므로 굳이 구분하지 않아도 된다.

##### (2) 기본형(primitive type)의 종류

- 기본형에는 모두 8개의 타입이 있으며, 크게 논리형, 문자형, 정수형, 실수형으로 구분된다.

분류	타입
논리형	boolean
	true와 false중 하나의 값으로 갖으며, 조건식과 논리적 계산에 사용한다.
문자형	char
	문자하나를 저장하는데 사용한다.
정수형	byte, short, int, long
	정수를 저장하는데 사용되며, int가 기본 자료형 이다.
실수형	float, double
	실수를 저장하는데 사용되며, double이 기본 자료형 이다.

[표 2-4] 기본형의 종류

- 기본형의 크기는 1byte, 2byte, 4byte, 8byte의 4가지로 구분된다.

크기	타입
1byte	boolean, byte
2byte	char, short
4byte	int, float
8byte	long, double

[표 2-5] 기본형의 크기

- 기본 자료형의 종류와 크기는 반드시 외워야 하며 [표 2-6]이 도움이 될 것 이다.

종류 \ 크기	1byte	2byte	4byte	8byte
논리형	boolean			
문자형		char		
정수형	byte	short	int	long
실수형			float	double

[표 2-6] 기본형의 종류 및 크기

- 각 타입의 변수가 저장할 수 있는 값의 범위는 [표 2-7]과 같다.

자료형	크기		저장 가능한 값의 범위
	byte	bit	
boolean	1	8	false, true
char	2	16	$0 \sim 2^{16}-1$
byte	1	8	$-2^7 \sim 2^7-1$
short	2	16	$-2^{15} \sim 2^{15}-1$
int	4	32	$-2^{31} \sim 2^{31}-1$
long	8	64	$-2^{63} \sim 2^{63}-1$
float	4	32	$1.4 \times 10^{-45} \sim 3.4 \times 10^{38}$
double	8	64	$4.9 \times 10^{-324} \sim 1.8 \times 10^{308}$

[표 2-7] 기본형의 크기와 범위

### (3) 논리형 - 기본 값 false

- 논리형에는 'boolean' 한가지 밖에 있다.
- boolean은 true또는 false중 하나의 값을 저장할 수 있다.
- boolean은 조건에 의한 분기 처리를 위해 많이 사용된다.
- 데이터를 다루는 최소단위가 1byte임으로 1byte의 크기를 가지게 된다.

ex) boolean switch = false;

(4) 문자형 - 기본 값 ' ' (공백)

- 문자형에는 'char' 한가지 밖에 있다.
- char는 한 글자 단위의 문자를 나타내는 타입이며 'a', '이'와 같이 작은따옴표('')를 이용하여 나타낸다.
- char는 표현방식이 여러 가지로 아래와 같다.

```
ex) char alpha = 'A';           // 문자하나로 표현    ** 주로사용
    char alpha = '\u0041';      // 문자의 Unicode(16진수)로 표현
    char alpha = 65;            // 문자의 10진수 값으로 표현
```

(5) 정수형 - 기본 값 0

- 정수형에는 'byte', 'short', 'int', 'long' 4가지가 있으며 저장하려는 값의 범위에 따라 4개의 정수형중 하나를 선택하면 된다.
- 기본 자료형은 int이다.
- long형 값의 경우 접미사 'L'또는 'l'을 붙여 주어야 한다.

```
ex) int number = 50 ;           // byte, short, int형 크기의 값은 접미사가 붙지 않는다.
    long number = 300000000000L // long형 크기의 값은 접미사가 붙어야 한다.
    byte number = 200;          // 오류 : 변수의 타입이 저장할 수 있는 범위를 넘는 값은
                                // 저장할 수 없다.
```

(6) 실수형 - 기본 값 0.0

- 실수형에는 'float', 'double'두 가지가 있으며 저장하려는 값의 범위뿐만 아니라 정밀도에 따라 두 가지 실수형중 하나를 선택하면 된다.
- 기본 자료형은 double이다.
- float형 값의 경우 접미사 'F'또는 'f'를 붙여 주어야 한다.

```
ex) double pi = 3.1415926548;   // double형의 크기의 값은 접미사가 붙지 않는다.
    float pi = 3.1415926548F;    // float형 크기의 값은 접미사가 붙어야 한다.
```

[문제 2-3] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_3{
    public static void main(String[] args){
        //1. 전원 스위치의 상태를 저장할 수 있는 변수 power를 선언하고 꺼져 있는 상태를
        //   저장 하여라. [hint] 스위치는 ON과 OFF만을 저장하면 된다.

        //2. 로또 당첨금을 저장할 수 있는 변수 lotto를 선언하고 지난주 당첨금으로 초기화
        //   하여라. [hint] 로또 당첨금 최고 금액은 400억 가량이다.

        //3. 하나의 정수를 다른 정수로 나눈 결과를 저장하고자 한다. 변수 result를 선언
        //   하여라. [hint] 결과는 소수점이 있을 수 있다.

    }
}
```

## 5. 상수와 리터럴

### (1) 상수

'상수(constant)'는 변수와 마찬가지로 '값을 저장할 수 있는 공간'이지만, 변수와 달리 한번 값을 저장하면 다른 값으로 변경할 수 없다. 상수를 선언하는 방법은 변수와 동일하며, 단지 변수의 타입 앞에 키워드 'final'을 붙여 주기만 하면 된다.

```
final int MAX_PERSON = 25;
```

일단 상수의 값이 저장된 후에는 상수의 값을 변경하는 것이 허용되지 않는다.

```
final int MAX_PERSON; // 정수형 상수 MAX_PERSON을 선언
MAX_PERSON = 25;      // OK. 상수에 처음으로 값 저장
MAX_PERSON = 20;      // 에러. 상수에 저장된 값을 변경할 수 없음.
```

※ 상수를 선언과 동시에 초기화 하지 않아도 되지만 상수는 선언과 동시에 초기화 하는 습관을 들이는 것이 좋다.

### (2) 리터럴(literal)

원래 12, 123, 3.14, 'A'와 같은 값들이 '상수'인데, 프로그래밍에서는 상수를 '값을 한 번 저장하면 변경할 수 없는 저장 공간'으로 정의하였기 때문에 이와 구분하기 위해 상수를 다른 이름으로 불러야만 했다. 그래서 상수 대신 리터럴이라는 용어를 사용한다. 많은 사람들이 리터럴이라는 용어를 어려워하는데, 리터럴은 우리가 기존에 알고 있던 '상수'의 다른 이름일 뿐이다.

변수(variable) 하나의 값을 저장하기 위한 공간  
상수(constant) 값을 한번만 저장할 수 있는 공간  
리터럴(literal) 그 자체로 값을 의미하는 것

```
int year = 2014;
final int MAX_VALUE = 100;
```

### (3) 상수가 필요한 이유

그냥 리터럴을 직접 쓰면 될 텐데, 굳이 상수가 따로 필요한 이유가 있는가? 라는 의문이 들 것이다. 예를 들어 기본형 변수는 값을 저장할 수 있는 범위가 지정되어 있으며 이를 저장할 해야 한다.

```
int integer_max_value = 2147483647;
```

위와 같이 integer의 최댓값을 지정한다면 누구나 변경하여 사용할 수 있으므로 사용자 별로 다른 값을 가지게 된다. 하지만 상수를 표현한다면 모든 사용자가 정해진 범위 내에서 사용하게 된다.

```
final int INTEGER_MAX_VALUE = 2147483647;
```

## 6. 문자열 리터럴 - String

### (1) 문자 리터럴과 문자열 리터럴

'A'와 같이 작은따옴표로 문자 하나를 감싼 것을 '문자 리터럴'이라고 한다. 두 문자 이상은 큰따옴표로 감싸야 하며 '문자열 리터럴'이라고 한다.

```
char ch = 'A';           // char형 변수는 단 하나의 문자만 저장할 수 있다.
String subject = "Java"; // 변수 subject에 문자열 리터럴 "Java:"를 저장
```

char타입의 변수는 단 하나의 문자만 저장할 수 있으므로, 여러 문자(문자열)를 저장하기 위해서는 String타입을 사용해야 한다. 문자열 리터럴은 “ ” 안에 아무런 문자도 넣지 않는 것을 허용하며, 이를 빈 문자열(empty string)이라고 한다. 그러나 문자 리터럴은 반드시 ' ' 안에 하나의 문자가 있어야 한다

```
String str = "";        // OK. 내용이 없는 빈 문자열
char ch = ";            // 에러. " 안에 반드시 하나의 문자가 필요
char ch = ' ';          // OK. 공백 문자(blank)로 변수 ch를 초기화
```

원래 String은 클래스이므로 아래와 같이 객체를 생성하는 연산자 new를 사용해야 하지만 특별히 이와 같은 표현도 허용한다.

```
String name = new String("Java"); // String객체를 생성
String name = "Java";             // 위의 문장을 간단히. 둘의 차이점은 9장에서 자세히 설명
```

### (2) 문자열 결합

숫자뿐만 아니라 두 문자열을 합칠 때도 덧셈(+)을 사용할 수 있다.

덧셈 연산자(+)는 피연산자가 모두 숫자일 때는 두 수를 더하지만, 피연산자 중 어느 한 쪽이 String이면 나머지 한 쪽을 먼저 String으로 변환한 다음 두 String을 결합한다.

어떤 타입의 변수도 문자열과 덧셈연산을 수행하면 그 결과가 문자열이 되는 것이다.

<b>문자열 + any type -&gt; 문자열 + 문자열 -&gt; 문자열</b> <b>any type + 문자열 -&gt; 문자열 + 문자열 -&gt; 문자열</b>
--

예를 들어 6 + "5"를 계산할 때 6이 String이 아니므로, 먼저 6을 String으로 변환한 다음 "6"+"5"를 수행하여 "65"를 결과로 얻는다. 다음은 문자열 결합의 몇가지 예를 보여준다.

```
7 + " " -> "7" + " " -> "7 "
" " + 7 -> " " + "7" -> " 7"
7 + 7 + " " -> 14 + "A" -> "14" + "A" -> "14A"
```

[문제 2-4] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_4{
    public static void main(String[] args){
        //1. 변수 year에 2020을 저장하여라.

        //2. 변수 month에 2를 저장하여라.

        //3. 변수 day에 14를 저장하여라.

        //4. 변수 year, month, day를 문자열 조합을 이용하여 아래와 같이 출력하여라.
        //   오늘날짜 : 2020년 2월 14일
        //   [hint] “2020년”을 선행하여 출력하여 본다.

    }
}
```

## 7. 오버플로우(overflow)

만일 4bit 2진수의 최대값인 '1111'에 1을 더하면 어떤 결과를 얻을까? 4 bit의 범위를 넘어서는 값이 되기 때문에 에러가 발생할까?

	1	1	1	1
+)	0	0	0	1
	?	?	?	?

원래 2진수 '1111'에 1을 더하면 '10000'이 되지만, 4 bit로는 4자리의 2진수만 저장할 수 있기 때문에 '0000'이 된다. 즉, 5자리의 2진수 '10000'중에서 하위 4 bit만 저장하게 되는 것이다. 이처럼 연산과정에서 해당 타입이 표현할 수 있는 값의 넘어서는 것을 **오버플로우(overflow)**라고 한다. 오버플로우가 발생했다고 해서 발생하는 것은 아니다. 다만 예상했던 결과를 얻지 못할 뿐이다. 애초부터 오버플로우가 발생하지 않게 충분한 크기의 타입을 선택해서 사용해야 한다.

					2진수
					1
					1
					1
					1
+)					1
	1	0	0	0	0

저장할 공간이 없어서 가장 앞의 1은 버려짐

오버플로우는 '자동차 주행표시기(odometer)'나, '계수기(counter)' 등 우리의 일상생활에서도 발견할 수 있는데, 네 자리 계수기라면 '0000'부터 '9999'까지 밖에 표현하지 못하므로 최대값인 '9999' 다음의 숫자는 '0000'이 될 것이다. 원래는 10000이 되어야하는데 다섯 자리는 표현할 수 없어서 맨 앞의 1은 버려지기 때문이다.

※ TV의 채널을 증가시키다가 마지막 채널에서 채널을 더 증가시키면 첫 번째 채널로 이동하고, 첫 번째 채널에서 채널을 감소시키면 마지막 채널로 이동하는 것과 유사하다.

최댓값 +1 --> 최솟값  
최솟값 -1 --> 최댓값

## [예제 2-2]

```
class Ex2_2{
    public static void main(String[] args){
        short sMin = -32768; // short 자료형의 최솟값
        short sMax = 32767; // short 자료형의 최댓값
        char cMin = 0; // char 자료형의 최솟값
        char cMax = 65535; // char 자료형의 최댓값
        System.out.println("sMin-1 : "+(short)(sMin-1)); // 최솟값 -1 = 최댓값
        System.out.println("sMax+1 : "+(short)(sMax+1)); // 최댓값 +1 = 최솟값
        System.out.println("cMin-1 : "+ (int)--cMin); // 최솟값 -1 = 최댓값
        System.out.println("cMax+1 : "+ (int)++cMax); // 최댓값 +1 = 최솟값
    }
}
```

결과

sMin-1 : 32767  
sMax+1 : -32768  
cMin-1 : 65535  
cMax+1 : 0

## 8. 형변환(casting)

모든 변수와 리터럴에는 타입이 있다는 것을 배웠다. 프로그램을 작성하다 보면 같은 타입뿐만 아니라 서로 다른 타입간의 연산을 수행해야 하는 경우도 있다. 이럴 때는 연산을 수행하기 전에 타입을 일치시켜야 하는데, 변수나 리터럴의 타입을 다른 타입으로 변환하는 것을 '형변환(casting)'이라고 한다.

**형변환**이란, 변수 또는 상수의 타입을 다른 타입으로 변환하는 것.

### (1) 형변환 방법

**형변환타입 변수B = (형변환타입) 변수 A;**

여기서 사용되는 괄호()는 '캐스트 연산자'또는'형변환 연산자'라고 하며, 형변환을 '캐스팅(casting)'이라고도 한다.



[문제 2-4] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_4{
    public static void main(String[] args){
        //1. 변수 a를 선언하고 8의 값으로 초기화 하여라.

        //2. 변수 b를 선언하고 3의 값으로 초기화 하여라.

        //3. 변수 result를 선언하고 a의 값을 b의 값으로 나누기한 결과로 초기화 하여라. (정수)

        //4. 변수 result2를 선언하고 a의 값을 b의 값으로 나누기한 결과로 초기화 하여라. (실수)

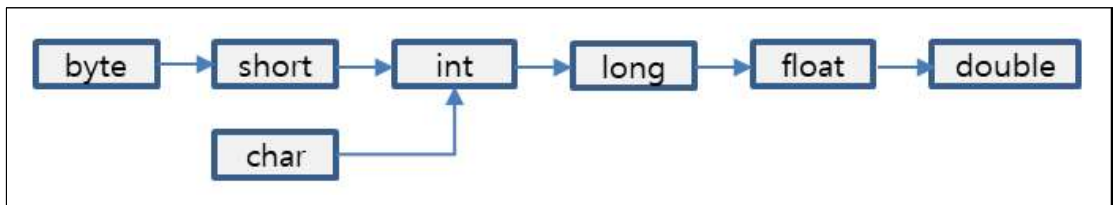
    }
}
```

## (2) 자동 형변환

형변환을 하는 이유는 주로 서로 다른 두 타입을 일치시키기 위해서 인데, 형변환을 생각하면 컴파일러가 알아서 자동적으로 형 변환을 진행하는 경우가 있다.

기존의 값을 최대한 보존할 수 있는 타입으로 자동 형변환한다.

- 기본형 표현범위 포함 관계



[그림 2-4] 기본형의 자동 형 변환 가능한 방향

그림 2-4의 형변환이 가능한 7개의 기본형을 왼쪽부터 오른쪽으로 표현할 수 있는 값의 범위가 작은 것부터 큰 것의 순서로 나열한 것이다.

화살표 방향으로의 변환, 즉 왼쪽에서 오른쪽으로 변환은 형변환 연산자를 사용하지 않아도 자동 형변환이 되며, 그 반대 방향으로의 변환은 반드시 형변환 연산자를 써줘야 한다.

보통 자료형의 크기가 큰 것일수록 값의 표현범위가 크기 마련이지만, 실수형은 정수형과 값을 표현하는 방식이 다르기 때문에 같은 크기일지라도 실수형이 정수형보다 훨씬 더 큰 표현범위를 갖기 때문에 float와 double이 같은 크기인 int와 long보다 오른쪽에 위치해 있다.

1. boolean형을 제외한 나머지 7개의 기본형은 서로 형변환이 가능하다.
2. 기본형과 참조형은 서로 형변환할 수 없다.
3. 서로 다른 타입의 변수간의 연산은 형변환을 하는 것이 원칙이지만, 값의 범위가 작은 타입에서 큰 타입으로의 형변환은 생략할 수 있다.

[문제 2-5] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_5{
    public static void main(String[] args){
        //1. byte형 변수 byte01을 선언하고 33의 값으로 초기화 하여라.

        //2. long형 변수 long01을 선언하고 888의 값으로 초기화 하여라.

        //3. char형 변수 char01을 선언하고 'A'의 값으로 초기화 하여라.

        //4. float형 변수 float01을 선언하고 3.141592의 값으로 초기화 하여라.

        //5. int형 변수 integer01을 선언하고 long01의 변수에 저장된 값으로 초기화 하여라.

        //6. short형 변수 short01을 선언하고 char01의 변수에 저장된 값으로 초기화 하여라.

        //7. int형 변수 integer02을 선언하고 float01의 변수에 저장된 값으로 초기화 하여라.

        //8. int형 변수 integer03를 선언하고 byte01의 변수에 저장된 값으로 초기화 하여라.

        //9. int형 변수 integer03을 선언하고 char01의 변수에 저장된 값으로 초기화 하여라.

    }
}
```

#### 4 연습문제

[2-1] 다음 표의 빈 칸에 8개의 기본형(primitive type)을 알맞은 자리에 넣으시오.

크 기 종 류	1 byte	2 byte	4 byte	8 byte
논리형				
문자형				
정수형				
실수형				

[2-2] 다음의 문장에서 리터럴, 변수, 상수, 키워드를 적으시오.

```
int i = 100;
long l =100L;
final float PI = 3.14f;
```

- 리터럴 :
- 변수 :
- 키워드 :
- 상수 :

[2-3] 다음 중 기본형(primitive type)이 아닌 것은?

1. int
2. Byte
3. double
4. boolean

[2-4] 다음 문장들의 출력결과를 적으세요. 오류가 있는 문장의 경우, 괄호 안에 '오류'라고 적으시오.

```
System.out.println("1" + "2");
System.out.println(true + "");
System.out.println('A' + 'B');
System.out.println('1' + 2);
System.out.println('1' + '2');
System.out.println(4 + 24.3715F);
System.out.println('A' + 3.14);
System.out.println('J' + "ava");
System.out.println(true + null);
```

[2-5] 다음 중 키워드가 아닌 것은?(모두 고르시오)

- |           |          |
|-----------|----------|
| 1. if     | 2. True  |
| 3. NULL   | 4. Class |
| 5. System |          |

[2-6] 다음 중 변수의 이름으로 사용할 수 있는 것은? (모두 고르시오)

- |              |              |
|--------------|--------------|
| 1. \$system  | 2. channel#5 |
| 3. 7eleven   | 4. If        |
| 5. 자바        | 6. new       |
| 7. \$MAX_NUM | 8. hello@com |

[2-7] 참조형 변수(reference type)와 같은 크기의 기본형(primitive type)은? (모두 고르시오)

[Hint] 참조형 변수의 크기는 4byte이다.

- |           |          |
|-----------|----------|
| 1. int    | 2. long  |
| 3. short  | 4. float |
| 5. double |          |

[2-8] 다음 중 형변환을 생략할 수 있는 것은? (모두 고르시오)

```
byte b = 10;  
char ch = 'A';  
int i = 100;  
long l = 1000L;
```

1. b = (byte)i;
2. ch = (char)b;
3. short s = (short)ch;
4. float f = (float)l;
5. i = (int)ch;

[2-9] char타입의 변수에 저장될 수 있는 정수 값의 범위는?

[2-10] 다음중 변수를 잘못 초기화 한 것은? (모두 고르시오)

1. byte b = 256;
2. char c = '';
3. char answer = 'no';
4. float f = 3.14
5. double d = 1.4e3f;

## 2-2. 연산자

### 학습목표

- 프로그래밍 언어의 연산자를 사용하여 애플리케이션에 필요한 기능을 정의하고 사용할 수 있다.

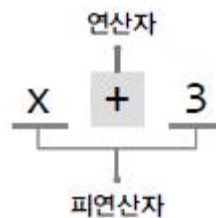
### 필요 지식 /

#### ① 연산자(operator)

연산자는 '연산을 수행하는 기호'를 말한다. 예를 들어 '+' 기호는 덧셈 연산을 수행하며, '덧셈 연산자'라고 한다. 자바에서는 사칙연산(+, -, \*, /)을 비롯해서 다양한 연산자를 제공한다.

##### 1. 연산자와 피연산자

연산자가 연산을 수행하려면 반드시 연산의 대상이 있어야 하는데, 이것을 '피연산자(operand)'라고 한다. 다음과 같이 'x + 3'이라는 식(式)이 있을 때, '+'는 두 피연산자를 더해서 그 결과를 반환하는 덧셈 연산자이고, 변수 x와 상수 3은 이 연산자의 피연산자이다.



이처럼 덧셈 '+'는 두 값을 더한 결과를 반환하므로, 두 개의 피연산자를 필요로 한다. 연산자는 피연산자로 연산을 수행하고 나면 항상 결과 값을 반환한다. 예를 들어 x의 값이 5일 때, 덧셈 연산 'x + 3'의 결과 값은 8이 된다.

연산자와 피연산자를 조합하여 계산하고자 하는 바를 표현한 것을 '식(式, expression)'이라고 한다. 그리고 식을 계산하여 결과를 얻는 것을 '식을 평가(evaluation)한다'고 한다. 하나의 식을 평가(계산)하면, 단 하나의 결과를 얻는다. 만일 x의 값이 5라면, 아래의 식을 평가한 결과는 23이 된다.

```
4 * x + 3
→ 4 * 5 + 3
→ 23
```

식이 평가되어 23이라는 결과를 얻었지만, 이 값이 어디에도 쓰이지 않고 사라지기 때문에 이식은 아무런 의미가 없다. 그래서 아래와 같이 대입 연산자 '='를 사용해서 변수와 같이 값을 저장할 수 있는 결과를 저장해야 한다.

```

y = 4 * x + 3;           // x의 값이 5라면, y의 값은 23이 된다.
System.out.println(y);   // y의 값인 23이 화면에 출력된다.

```

그 다음에 변수 y에 저장된 값을 다른 곳에 사용하거나 화면에 출력함으로써 의미 있는 결과를 얻을 수 있다. 만일 식의 평가결과를 출력하기만 원할 뿐, 이 값을 다른 곳에 사용하지 않을 것이라면 아래처럼 변수에 저장하지 않고 println메서드의 괄호( ) 안에 직접 식을 써도 된다.

```

System.out.println(4 * x + 3); // x의 값이 5라고 가정하면
→ System.out.println(23);

```

## 2. 연산자 기본

구분	상세	연산자	표현식	설명
단항	증감연산	++ --	A++, ++A A--, --A	1증가 1감소
	비트전환 연산	~	~A	피연산자를 2진수로 표현했을 때 0은 1로 1은 0으로 바꾼다.
	논리부정 연산	!	!A	true이면 false, false이면 true
이항	산술	사칙연산	+	더하기
			-	빼기
			*	곱하기
			/	나누기
	산술	나머지 연산	%	나머지
			<<	비트를 오른쪽으로 이동하여 연산
			>>	비트를 왼쪽으로 이동하여 연산
	대입	비교연산	>>>	비트를 오른쪽으로 이동하여 연산
			=	오른쪽의 값을 왼쪽에 대입
			op=	A의 값에 B를 더하여 A에 대입
	비교	비교연산	A==B	같으면 true, 다르면 false
			A!=B	같지 않으면 true, 같으면 false
			A>B	크면 true, 크기 않으면 false
			A>=B	크거나 같으면 true, 작으면 false
			A<B	작으면 true, 작지 않으면 false
			A<=B	작거나 같으면 true, 크면 false
	논리	논리연산	A&&B	모두 true일 때 true, 아니면 false
			A  B	둘 중 하나라도 true이면 true, 아님 false

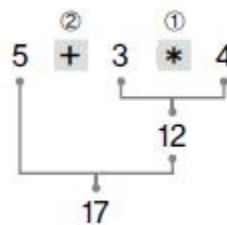
		비트연산	& ! ^	A&B A!B A^B	모두 1일 때 1, 아님 0 반환 모두 0일 때 0, 아님 1반환 다르면 1, 같으면 0
삼항	삼항 연산	? :	(조건식)?식1:식2	조건식이 참이면 식1, 거짓이면 식2를 수행	

[표 2-8] 연산자 기본 정리

피연산자의 개수로 연산자를 분류하기도 하는데, 피연산자의 개수가 하나면 '단항 연산자', 두 개면 '이항 연산자', 세 개면 '삼항 연산자'라고 부른다. 표 2-8에서도 확인할 수 있듯이 대부분의 연산자는 '이항 연산자'이다.

### 3. 연산자 우선순위

식에 사용된 연산자가 둘 이상인 경우, 연산자의 우선순위에 의해서 연산순서가 결정된다. 곱셈과 나눗셈(\*, /)은 덧셈과 뺄셈(+, -)보다 우선순위가 높다는 것은 이미 수학에서 배워서 알고 있을 것이다. 그래서 아래의 식은 '3 \* 4'가 먼저 계산된 다음, 그 결과에 5를 더해서 17을 결과로 얻는다.



이처럼 연산자의 우선순위는 대부분 상식적인 선에서 해결된다. 아래의 표를 통해 이 사실을 한번 확인해 보자.

우선순위	구분	연산자
1	기타	괄호(), 대괄호[]
2	단항 연산자	증감 연산자
3	이항 연산자	산술 연산자
4		비교 연산자
5		논리 연산자
6	삼항 연산자	삼항 연산자
7	대입 연산자	=, op=

[표 2-9] 연산자 우선순위

1. 산술 > 비교 > 논리 > 대입, 대입은 제일 마지막에 수행된다.
2. 단항(1) > 이항(2) > 삼항(3). 단항 연산자의 우선순위가 이항 연산자보다 높다.
3. 단항 연산자와 대입 연산자를 제외한 모든 연산의 진행방향은 왼쪽에서 오른쪽이다.

#### [예제 2-3]

```
class Ex2_3{
    public static void main(String[] args){
        int result = 5 + 15 + 7;
        System.out.println("result : " + result);
        int result2 = 2 + 3 * 4;
        System.out.println("result2 : " + result2);
        int result3 = 3 * ( 4+2 );
        System.out.println("result3 : " + result3);
    }
}
```

결과

```
result : 27
result2 : 14
result3 : 24
```

## 4. 증감 연산자

증감 연산자는 피연산자에 저장된 값을 1 증가 또는 감소시킨다. 증감 연산자의 피연산자로 정수와 실수가 모두 가능하지만, 상수는 값을 변경할 수 없으므로 가능하지 않다.

※ 증감 연산자는 일반 산술 변환에 의한 자동 형 변환이 발생하지 않으며, 연산결과와 타입은 피연산자의 타입과 같다.

**증가 연산자(++)** // 전위형. i의 값을 1증가시킨다.

**감소 연산자(--)** // 후위형. 위의 문장과 차이가 없다.

일반적으로 단항 연산자는 피연산자의 왼쪽에 위치하지만, 증가 연산자 '++'와 감소 연산자 '--'는 양쪽 모두 가능하다. 피연산자의 왼쪽에 위치하면 '전위형(prefix)', 오른쪽에 위치하면 '후위형(postfix)'이라고 한다.

그러나 '++count'와 'count++;'처럼 증감연산자가 수식이나 메서드 호출에 포함되지 않고 독립적인 하나의 문장으로 쓰인 경우에는 전위형과 후위형의 차이가 없다.

**++count;** // 전위형. i의 값을 1증가시킨다.

**count++;** // 후위형. 위의 문장과 차이가 없다.



[예제 2-4]

```
class Ex2_4{
    public static void main(String[] args){
        int number = 4;
        ++number;    // number의 값이 1증가한다. number = 5
        System.out.println(number);
        number++;    // number의 값이 1증가한다. number = 6 독립적인 문장인 경우 위의 결
                     // 과와 같다.
        System.out.println(number);

        int number2 = ++number;
        // 1. number의 값을 1 증가 시킨다. number = 7
        // 2. number의 값을 number2에 저장한다. number2 = 7
        System.out.println("number : "+number);
        System.out.println("number2 : "+number2);

        int number3 = number++;
        // 1. number의 값을 number3에 저장한다. number3 = 7
        // 2. number의 값을 1 증가 시킨다. number = 8
        System.out.println("number : "+number);
        System.out.println("number3 : "+number3);

    }
}
```

결과	5 6 number : 7 number2 : 7 number : 8 number3 : 7
----	--

[문제 2-6] 주석에 적당한 코드를 작성해 주세요.

[hint] char형 'C'의 정수형 값은 67이다.

```
class Qu2_6{
    public static void main(String[] args){
        //1. int형 변수 number를 선언하고 30의 값으로 초기화 하여라.

        //2. char형 변수 ch를 선언하고 'C'의 값으로 초기화 하여라.

        //3. 다음의 출력결과를 작성하여라.
        int result = number++ + 3 + ++ch + ++number;
        System.out.println("number : "+ number); // 결과 :
        System.out.println("ch : "+ ch); // 결과 :
        System.out.println("result : "+ result); // 결과 :

    }
}
```

## 5. 비트전환 연산자 ( ~ )

이 연산자는 정수형과 char형에만 사용이 가능하며 피연산자를 2진수로 표현했을 때, 0은 1로, 1은 0으로 바꾼다. 그래서 연산자 '~'에 의해 비트전환이 되면 피연산자의 부호가 반대로 변경된다. 단. byte, short, char형은 int형으로 변경된 후 전환된다.

원론적으로 컴퓨터는 가산, 즉 덧셈만 가능하다. 그렇기 때문에 감산, 뺄셈을 해야하는 음수의 표현은 불가능한 것이다. 그래서 등장한 것이 비트전환 연산자이다.

비트전환 연산을 하였을 때 int타입으로 변경이 되지만 bit의 수가 많이 지기 때문에 byte타입으로 가정한 예이다.

10진수	2진수	설명
10	0 0 0 0 1 0 1 0	<ul style="list-style-type: none"> <li>10진수 10을 2진수로 표현</li> <li>첫 자리는 부호를 담당</li> <li>부호자리가 0일 경우 0에서 시작 =&gt; 0 + 10</li> </ul>
~10 (-11)	1 1 1 1 0 1 0 1	<ul style="list-style-type: none"> <li>비트 전환 연산자를 통해 0은 1로 1은 0으로 변경한다.</li> <li>부호자리가 1일 경우 최솟값에서 시작 =&gt; -128 + 117 = -11</li> <li>비트 전환된 피연산자의 '1의 보수'를 얻는다.</li> </ul>
~10 + 1 (-10)	1 1 1 1 0 1 1 0	<ul style="list-style-type: none"> <li>1의 보수에 1을 더한 결과 피연산자 반대 부호 값을 얻게 되며 이를 피연산자의 2의 보수라 한다.</li> <li>10의 2의 보수는 -10이 된다.</li> <li>컴퓨터가 인식하는 -10은 왼쪽과 같다.</li> </ul>

[예제 2-5]

```
class Ex2_5{
    public static void main(String[] args){
        byte b = 10;          // byte타입의 b를 선언 후 10으로 초기화
        byte b2 = (byte)~b;    // 변수 b를 비트전환을 수행하였을 때 int타입이 되기 때문에 byte
                               // 타입의 변수 b2에 저장하기 위해 byte타입으로 형 변환
        System.out.println("b2의 값 : "+b2);
    }
}
```

결과 B2의 값 : -11

## 6. 논리부정 연산자( ! )

이 연산자는 boolean형에만 사용 가능하며, 피연산자가 true이면 false를, false면 true를 결과로 반환한다. 간단히 말해서, true와 false를 반대로 바꾸는 것이다.

어떤 값에 논리 부정 연산자'!'를 반복적으로 적용하면, 참과 거짓이 차례대로 반복된다. 이 연산자의 이러한 성질을 이용하면, 한번 누르면 켜지고, 다시 한 번 누르면 TV의 전원버튼과 같은 '토글 버튼(toggle button)'을 논리적으로 구현할 수 있다.

x	!x
true	false
false	true

논리 부정 연산자'!'가 주로 사용되는 곳은 조건문과 반복문의 조건식이며, 이 연산자를 잘 사용하면 조건식이 보다 이해하기 쉬워진다.

**[문제 2-7] 주석에 적당한 코드를 작성해 주세요.**

```
class Qu2_7{
    public static void main(String[] args){
        //1. power변수를 선언하고 false의 값으로 초기화 하여라.

        //2. power변수의 값을 출력하여라.

        //3. power변수의 값을 부정하여 true의 값을 저장하여라.

        //4. power변수의 값을 출력하여라.

    }
}
```

## 7. 산술 연산자

산술 연산자의 종류에는 사칙 연산자, 나머지 연산자, 쉬프트 연산자가 있으면 모두 두 개의 피연산자를 취하는 이항 연산자 이다. 이항 연산자는 두 피연산자의 타입이 일치해야 연산이 가능하므로, 피연산자의 타입이 서로 다르다면 연산 전에 형변환 연산자로 타입을 일치시켜야한다. 예를 들어 int타입과 double타입을 연산하는 경우, 형변환 연산자를 사용해서 피연산자의 타입을 둘 다 int 또는 double로 일치 시켜야 한다.

대부분의 경우, 두 연산자의 타입 중에서 더 큰 타입으로 일치시키는데, 그 이유는 작은 타입으로 형변환하면 원래의 값이 손실될 가능성이 있기 때문이다. 앞서 배운 것과 같이 작은

타입에서 큰 타입으로 형변환하는 경우, 자동적으로 형변환되므로 형변환 연산자를 생략할 수 있다.

```
int number = 19;
double dou = 20.0;
double result = number + dou; // int + double => double + double => double
// 큰 타입으로 형변환시, 형변환 연산자 생략가능
```

이처럼 연산 전에 피연산자 타입의 일치를 위해 자동 형변환되는 것을 '산술 변환' 또는 '일반 산술 변환'이라 하며, 이 변환은 이항 연산자에서만 아니라 단항 연산자에서도 일어난다. '산술 변환'의 규칙은 다음과 같다.

**1. 두 피연산자의 타입을 같게 일치시킨다.(보다 큰 타입으로 일치)**

```
long + int -> long + long -> long
float + long -> float + float -> float
int + char -> int + int -> int
```

**2. 피연산자의 타입이 int보다 작은 타입이면 int로 변환한다.**

```
byte + short -> int + int -> int
char + short -> int + int -> int
```

※ 모든 연산에서 '산술 변환'이 일어나지만, 증감 연산자는 예외이다.

첫 번째 규칙은 앞서 자동 형변환에서 배운 것처럼 피연산자의 값손실을 최소화하기 위한 것이고, 두 번째 규칙은 int보다 작은 타입, 예를 들면 char나 short의 표현범위가 좁아서 연산 중에 오버플로우(overflow)가 발생할 가능성 때문에 있는 것이다.

여기서 한 가지 주목해야할 점은 연산결과와 타입인데, 연산결과와 타입은 피연산자의 타입과 일치한다. 예를 들어 int와 int의 나눗셈 연산결과와 int이다.

```
int / int → int
5 / 2 → 2
```

그래서 아래의 식 '5 나누기 2'의 결과가 2.5가 아닌 2이다. 위의 식에서 2.5라는 실수를 결과로 얻으려면, 피연산자 중 어느 한쪽을 float와 같은 실수형으로 형변환해야 한다. 그러면, 다른 한 쪽은 일반 산술 변환의 첫 번째 규칙에 의해 자동적으로 형변환되어 두 피연산자 모두 실수형이 되고, 연산결과 역시 실수형의 값을 얻을 수 있다.

```
int / (float)int → int / float → float / float → float
5 / (float)2 → 5 / 2.0f → 5.0f / 2.0f → 2.5f
```

[예제 2-6]

```
class Ex2_6{
    public static void main(String[] args){
        System.out.println("5/2 값 : "+ 5/2);
        System.out.println("(float)5/2 값 : "+ (float)5/2);
    }
}
```

결과	5/2 값 : 2 (float)5/2 : 2.5
----	-------------------------------

크기가 작은 자료형의 변수를 큰 자료형의 변수에 저장할 때는 자동으로 형변환(type, conversion, casting)되지만, 반대로 큰 자료형의 값을 작은 자료형의 변수에 저장하려면 명시적으로 형변환 연산자를 사용해서 변환해주어야 한다.

[문제 2-8] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_8{
    public static void main(String[] args){
        //1. byte형 변수 b1을 선언하고 80의 값으로 초기화 하여라

        //2. byte형 변수 b2를 선언하고 100의 값으로 초기화 하여라.

        //3. long형 변수 lo1을 선언하고 642의 값으로 초기화 하여라.

        //4. lo1과 b1의 합을 변수 result1에 저장 하여라.

        //5. result1의 값을 출력 하여라. (결과 첫 번째 줄)

        //6. b1과 b2의 합을 변수 result2에 저장 하여라.

        //7. result2의 값을 출력 하여라. (결과 두 번째 줄)

        //8. byte형 변수 result3을 선언하고 b1과 b2의 합으로 초기화 하여라.

        //9. result3의 값을 출력 하여라. (결과 세 번째 줄)

        //10. 변수 lo2를 선언하고 60000과 80000의 곱을 저장 하여라.

        //11. 변수 lo2에 저장된 값을 출력 하여라. (결과 네 번째 줄)
    }
}
```

결과	result1 : 652 result2 : 180 result3 : -76 lo2 : 4800000000
----	---

## 8. 나머지 연산자 (%)

나머지 연산자는 왼쪽의 피연산자를 오른쪽 피연산자로 나누고 난 나머지 값을 결과로 반환하는 연산자이다. 나눗셈에서처럼 나누는 수(오른쪽 피연산자)로 0을 사용할 수 없고, 피연산자로 정수만 허용한다.

※ 나머지 연산자는 주로 짝수, 홀수 또는 배수 검사 등에 주로 사용된다.

### [예제 2-7]

```
class Ex2_7{
    public static void main(String[] args){
        int x = 18;
        int y = 7;
        int result = x % y;
        System.out.println(result);
    }
}
```

결과	2
----	---

나머지 연산자(%)는 나누는 수로 음수도 허용한다. 그러나 부호는 무시되므로 결과는 음수의 절대값으로 나눈 나머지와 결과가 같다. 그냥 피연산자의 부호를 모두 무시하고, 나머지 연산을 한 결과에 왼쪽 피연산자(나눠지는 수)의 부호를 붙이면 된다.

**System.out.println(10 % 8);** // 10을 8로 나눈 나머지 2가 출력된다.

**System.out.println(10 %-8);** // 위와 같은 결과를 얻는다.

### [문제 2-9] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_9{
    public static void main(String[] args){
        //1. 정수를 저장할 수 있는 변수 numb1을 선언하고 원하는 정수로 초기화 하여라.

        //2. 정수를 저장할 수 있는 변수 numb2를 선언하고 원하는 정수로 초기화 하여라.

        //3. 변수 share를 선언하고 num1을 num2로 나눗셈한 결과로 초기화 하여라.

        //4. 변수 remain를 선언하고 num1을 num2로 나눈 나머지로 초기화 하여라.

        //5. 위의 변수 4개를 활용하여 아래와 같이 출력 하여라.
        //   xxx를 xxx로 나눈 몫은 xxx이고 나머지는 xxx 이다.

    }
}
```

## 9. 쉬프트 연산자 ( <<, >>, >>> )

쉬프트 연산자는 피연산자의 각 자리(2진수로 표현하였을 때)를 '오른쪽(>>, >>>)' 또는 '왼쪽(<<)'으로 이동(shift)한다고 해서 '쉬프트 연산자(shift operator)'라고 이름 붙여졌다.

정수형 변수(byte, short, int, long, char)에서만 가능하며 비트연산을 수행함으로 연산속도가 매우 좋다.

쉬프트 연산을 하였을 때 일반적으로 int타입으로 변경이 되지만 bit의 수가 많이 지기 때문에 byte타입으로 가정한 예이다.

10진수	2진수	설명
10	0 0 0 0 1 0 1 0	<ul style="list-style-type: none"> <li>10진수 10을 2진수로 표현</li> <li>첫 자리는 부호를 담당</li> </ul>
10 << 2	0 0 1 0 1 0 0 0	<ul style="list-style-type: none"> <li>왼쪽으로 두 칸씩 이동한다.</li> <li>비어진 뒤의 두 자리는 부호와 동일한 값으로 채워진다.</li> <li><math>10 * 2^2 = 40</math></li> </ul>
10 >> 2	0 0 0 0 0 0 1 0	<ul style="list-style-type: none"> <li>오른쪽으로 두 칸씩 이동한다.</li> <li>비어진 앞의 두 자리는 0으로 채워진다.</li> <li><math>10 / 2^2 = 2</math></li> </ul>

2진수 n자리를 왼쪽으로 이동하면 피연산자를  $2^n$ 으로 곱한 결과를, 오른쪽으로 이동하면 피연산자를  $2^n$ 으로 나눈 결과를 얻는다는 것을 알 수 있다.

$$x \ll n \text{은 } x * 2^n$$

$$x \gg n \text{은 } x / 2^n$$

'x << n'또는 'x >> n'에서, n의 값이 자료형의 bit수 보다 크면, 자료형의 bit수로 나눈 나머지만큼만 이동한다. 예를 들어 4byte(=32bit)인 경우, 자리수를 32번 바꾸면 결국 제자리로 돌아오기 때문에, '8 << 32'는 아무 일도 일어나지 않는다.

$$8 \ll 32 \rightarrow 8 * 2^{32} \rightarrow 8$$

$$8 \ll 34 \rightarrow 8 * 2^{32} * 2^2 \rightarrow 8 * 2^2 \rightarrow 32$$

당연히 n은 정수만 가능하며 음수인 경우, 부호없는 정수로 자동 변환된다.

## 10. 비교연산자

비교 연산자는 두 피연산자를 비교하는 데 사용되는 연산자다. 주로 조건문과 반복문의 조건식에 사용되며, 연산결과는 오직 true와 false 둘 중의 하나이다.

비교 연산자 역시 이항 연산자이므로 비교하는 피연산자의 타입이 서로 다를 경우에는 자료형의 범위가 큰 쪽으로 자동 형변환하여 피연산자의 타입을 일치시킨 후에 비교한다.

### (1) 대소비교 연산자

두 피연산자의 값의 크기를 비교하는 연산자이다. 참이면 true를, 거짓이면 false를 결과로 반환한다. 기본형 중에서는 boolean형을 제외한 나머지 자료형에 사용할 수 있지만 참조형은 사용할 수 없다.

연산자	연산결과
>	좌변 값이 크면, true 아니면 false
<	좌변 값이 작으면, true 아니면 false
>=	좌변 값이 크거나 같으면, true 아니면 false
<=	좌변 값이 작거나 같으면, true 아니면 false

[표 2-10] 대소비교 연산자의 종류와 연산결과

### (2) 등가비교 연산자

두 피연산자의 값이 같은지 또는 다른지를 비교하는 연산자이다. 대소비교 연산자와 달리, 기본형은 물론 참조형, 즉 모든 자료형에 사용할 수 있다. 기본형의 경우 변수에 저장되어 있는 값이 같은지를 알 수 있고, 참조형의 경우 객체의 주소값을 저장하기 때문에 두 개의 피연산자(참조변수)가 같은 객체를 가리키고 있는지를 알 수 있다.

기본형과 참조형은 서로 형변환이 가능하지 않기 때문에 등가비교 연산자로 기본형과 참조형을 비교할 수 없다.

연산자	연산결과
==	두 값이 같으면, true 아니면 false
!=	두 값이 다르면, true 아니면 false

[표 2-11] 등가비교 연산자의 종류와 연산결과

비교연산자는 수학기호와 유사한 기호와 의미를 가지고 있으므로 이해하는데 별 어려움이 없을 것이다. 한 가지 다른 점은 '두 값이 같다'는 의미로 '='가 아닌 '=='를 사용한다는 것인데, '='는 이미 배운 것과 같이 변수에 값을 저장할 때 사용하는 '대입연산자'이기 때문에 '=='로 두 값이 같은지 비교하는 연산자를 표현한다.

※ '>='와 같이 두 개의 연산자로 이루어진 연산자는 '>='와 같이 기호의 순서를 바꾸거나 '> ='와 같이 중간에 공백이 들어가서는 안 된다.



[문제 2-10] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_10{
    public static void main(String[] args){
        //1. 변수 f01을 선언하고 0.1f의 값으로 초기화 하여라.

        //2. 변수 f02를 선언하고 10.0f의 값으로 초기화 하여라.

        //3. 변수 do01을 선언하고 0.1의 값으로 초기화 하여라.

        //4. 변수 do02를 선언하고 10.0의 값으로 초기화 하여라.

        //5. 변수 str1을 선언하고 “화이팅”의 값으로 초기화 하여라.

        //6. 다음질문의 결과를 출력 하여라. (변수 f01과 do01의 값은 같은가?)

        //7. 다음질문의 결과를 출력 하여라. (변수 f02와 do02의 값은 같은가?)

        //8. 다음질문의 결과를 출력 하여라. (str1의 값이 “화이팅”과 값은 같은가?)

    }
}
```

두 문자열을 비교할 때는, 비교연산자 “==” 대신 equals()라는 메서드를 사용해야 한다. 비교 연산자는 두 문자열이 완전히 같은 것인지 비교할 뿐이므로, 문자열의 내용이 같은지 비교하기 위해서는 equals()를 사용하는 것이다. equals()는 비교하는 두 문자열이 같으면 true, 다르면 false를 반환한다. 문자열에 대한 내용은 5장에서 설명한다.

## 11. 비트연산자

비트 연산자는 피연산자를 비트단위로 논리 연산한다. 피연산자를 이진수로 표현했을 때의 각 자리를 아래의 규칙에 따라 수행하며, 피연산자로 실수는 허용하지 않는다. 정수(문자포함)만 허용한다.

연산자	연산결과
! (OR연산자)	피연산자 중 한 쪽의 값이 1이면, 1을 결과를 얻는다. 그 외는 0
& (AND연산자)	피연산자 양 쪽이 모두 1이어야만 1을 결과를 얻는다. 그 외는 0
^ (XOR연산자)	피연산자의 값이 서로 다를 때만 1을 결과로 얻는다. 같으면 0

[표 2-12] 비트연산자 종류와 연산결과

※ 연산자 '^'는 배타적 XOR(eXclusive OR)라고 하며, 값이 배타적인 경우에만 1의 결과를 얻는다.

연산을 수행하였을 때 일반적으로 int타입으로 변경이 되지만 bit의 수가 많이 지기 때문에 byte타입으로 가정한 예이다.

10진수	2진수	설명
11	0 0 0 0 1 0 1 1	• 10진수 11을 2진수로 표현
20	0 0 0 1 0 1 0 0	• 10진수 20을 2진수로 표현
11   20 ( 31 )	0 0 0 1 1 1 1 1	<ul style="list-style-type: none"> <li>• 11의 오른쪽 마지막 자리의 1, 그리고 20의 오른쪽 마지막 자리의 0의 연산을 통하여 결과를 얻는다.</li> <li>• 모든 자리를 위와 같이 연산한다.</li> </ul>
11 & 20 ( 0 )	0 0 0 0 0 0 0 0	
11 ^ 20 ( 31 )	0 0 0 1 1 1 1 1	

실제로 많이 사용되지는 않지만 OR연산자, AND연산자의 경우 그래픽에서 모두 0인 bit와 모두 1인 비트와의 연산을 통하여 전체 화면의 색상을 변경할 때 사용이 되고 있으며 XOR연산자의 경우 일부분의 색상만을 변경할 때 사용되어 진다.

## 12. 논리연산자

'x가 4보다 작다'라는 조건은 비교연산자를 써서 'x<4'와 같이 표현할 수 있다. 그러면 'x가 4보다 작거나 또는 10보다 크다'와 같이 두 개의 조건이 결합된 경우는 '논리 연산자'를 이용하여 표현이 가능하다. 논리 연산자는 둘 이상의 조건을 '그리고(AND)'나 '또는(OR)'으로 연결하여 하나의 식으로 표현할 수 있도록 해준다.

### (1) 논리연산자 &&, ||

논리연산자 '&&'는 우리말로 '그리고(AND)'의 의미를 가지고 있으며 두 피연산자 모두가 true일 때 true를 결과로 얻는다. '||'는 '또는(OR)'에 해당하며, 두 피연산자 중 어느 한 쪽만 true이어도 true의 결과를 얻는다. 논리 연산자는 피연산자로 boolean형 또는 boolean형 값을 결과로 하는 조건식만을 허용한다.

x	y	x  y	x&& y
true	true	true	true
true	false	true	false
false	true	true	false
false	false	false	false

[표 2-13] 논리 연산자의 연산결과

#### [예제 2-8]

```
class Ex2_8{
    public static void main(String[] args){
        boolean b1 = true;
        boolean b2 = false;
        System.out.println(b1 && b2);
        int in01 = 10;
        boolean re1 = int01 % 2 == 0; // int01이 2의 배수인지 확인. true
        boolean re2 = int01 % 4 == 0; // int01이 4의 배수인지 확인. true
        System.out.println(re1 || re2); // int01이 2 또는 4의 배수인지 확인. true
    }
}
```

x라는 변수에 저장된 값의 범위가 '5이상 25미만이다.'의 문장을 표현을 수학적으로 표현하게 된다면 '5 <= x < 25'로 표현할 수 있지만 자바에서는 해당 형식은 사용할 수 없다. 수학적 표현을 두 부분으로 나누어 보면 '5 <= x'의 부분과 'x < 25'로 나타낼 수 있다. 이를 논리연산자를 이용한다면 '5 <= x && x < 25'로 표현할 수 있다.

#### [문제 2-11] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_11{
    public static void main(String[] args){
        //1. 변수 ch01을 선언하고 원하는 문자로 초기화 하여라.

        //2. 변수 ch01의 값이 영어 대문자일 때 true인 조건식을 작성 하여라.

        //3. 변수 ch01의 값이 영어 소문자일 때 true인 조건식을 작성 하여라.

        //4. 변수 ch01의 값이 영문자일 때 true인 조건식을 작성 하여라.

        //5. 변수 ch01의 값이 ASCII값 숫자 때 true인 조건식을 작성 하여라.

    }
}
```

### 13. 조건 연산자

조건 연산자는 조건식, 식1, 식2 모두 세 개의 피연산자를 필요로 하는 삼항 연산자이며, 삼항 연산자는 조건 연산자 하나뿐이다.



조건 연산자는 첫 번째 피연산자인 조건식의 평가결과에 따라 다른 결과를 반환한다. 조건식의 평가결과가 true이면 식1이, false이면 식2가 연산결과가 된다. 가독성을 높이기 위해 조건식을 괄호( )로 둘러싸는 경우가 많지만 필수는 아니다.

#### [예제 2-9]

```
class Ex2_9{
    public static void main(String[] args){
        // 변수에 저장된 값의 절대값을 구하는 코드이다.
        int num = -10;
        int result = num >= 0 ? num : -num;
        System.out.println(result); // num의 초기화 값이 상관없이 절대값이 저장된다.
    }
}
```

num에 저장된 값이 0보다 크다면 조건식의 결과가 true가 되어 num의 결과, num에 저장된 값이 0보다 작다면 조건식의 결과가 false가 되어 -num의 결과가 result에 저장된다.

**double result = x > 0 ? 0 : 0.5 ;**

위의 식과 같이 식1과 식2의 타입이 다른 경우 두 개의 피연산자의 타입이 동일하게 하는 산술 변환이 발생하게 되기 때문에 result의 타입은 double형이 되어야 한다.

#### [문제 2-12] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_12{
    public static void main(String[] args){
        //1. 변수 x를 선언 및 'B'의 값으로 초기화 하여라.

        //2. 변수 result를 선언 하고 x의 값이 대문자 일 때 “대문자”를 소문자 일 때
        // “소문자”를 그 외에는 “영문자 아님”을 저장해주세요

    }
}
```

## 14. 대입 연산자

대입 연산자는 변수와 같은 저장공간에 값 또는 수식의 연산결과를 저장하는데 사용된다. 이연산자는 오른쪽 피연산자의 값(식이라면 평가값)을 왼쪽 피연산자에 저장한다. 그리고 저장된 값을 연산결과로 반환한다. 예를 들어, 아래의 문장은 변수 x에 3을 저장하고, 연산결과인 3을 화면에 출력한다.

```
System.out.println(x = 3);    // 변수 x에 3이 저장되고
System.out.println(3);        // 연산결과인 3이 출력된다.
```

대입 연산자는 연산자들 중에서 가장 낮은 우선순위를 가지고 있기 때문에 식에서 제일 나중에 수행된다. 그리고 앞서 배운 것처럼 연산 진행 방향이 왼쪽이기 때문에 'x=y=3+4;'에서 '3+4'를 수행 후 'y=7'이 수행되고 마지막으로 'x=y'가 수행된다.

**x = y = 3 + 4;**

대입 연산자는 다른 연산자(op)와 결합하여 'op='와 같은 방식으로 사용될 수 있으며 복합 연산자로 불린다. 예를 들면, 'i = i + 3'은 'i += 3'과 같이 표현될 수 있다. 그리고 결합된 두 연산자는 반드시 공백 없이 붙여 써야 한다.

### [예제 2-10]

```
class Ex2_10{
    public static void main(String[] args){
        int b = 6;
        b = b + 7;           // b자신에 저장된 값이 7을 더한 결과를 b에 저장한다.
        System.out.println(b); // "13"이 출력된다.
        char c = 'A';
        int result = c + 4;   // c + 4의 연산결과가 int형 이므로 result의 타입이 int이어야 한다.
        System.out.println(result); // "69"가 출력된다.
        c += 3;              // 복합연산자의 경우 원래 타입을 유지한다.
        System.out.println(c); // "D"가 출력된다.
    }
}
```

#### ※ 주의사항

- 대입 연산자의 우변이 둘 이상의항으로 이루어져 있는 경우 우측 연산을 실행하고 복합연산을 진행 하여야 한다.
- 복합 연산이 진행되었을 때 연산되는 변수는 원래의 타입을 유지하므로 형변환 하지 않는다.

## 15. 반올림, 임의의 정수 뽑기, 사용자로부터 입력받기

### (1) 반올림

근삿값을 구할 때 4이하의 수는 버리고 5이상의 수는 그 윗자리에 1을 더하여 주는 방식을 의미하며 예를 들어 10.4를 반올림 하면 10, 10.6을 반올림하면 11이 된다. 일반적인 반올림은 소수점 첫 번째 자리에서 반올림하여 정수부분만 취하는 것을 의미 하지만 취하고자 하는 부분이 소수점 아래 부분인 경우들이 있다. 예를 들어 3.141592의 값을 소수점 네 번째 자리에서 반올림하여 세 번째 자리까지 취득한다면 아래의 순서로 진행하면 된다.

`float pi = 3.141592F;`

순서	연산	연산설명	결과
1	<code>pi * 1000</code>	취득하고자 하는 자리까지 정수로 변경하여 준다.	3141.592
2	<code>+ 0.5</code>	0.5를 더하여 반올림 하여 준다.	3142.092
3	<code>(int)</code>	int형으로 형변환을 통해 필요 없는 소수점을 삭제한다.	3142
4	<code>/1000F</code>	정수로 변경 되었던 소수점 부분을 다시 소수점으로 변경한다.	3.142
참고		<ul style="list-style-type: none"> <li>4번 연산시 1000으로 나누면 int/int의 연산이 수행되어 소수점이 없어지게 되어 꼭 1000F로 나누어야 한다.</li> <li>취득하고자하는 소수점 자리에 따라 1번과 4번의 연산 값만 바꾸어 주면 된다.</li> </ul>	

`float result = (int)( pi * 1000 +0.5 ) /1000F;`

### (2) 임의의 정수 뽑기

자신이 원하는 임의의 정수를 뽑기 위해서는 Java에서 지원하는 Math클래스를 사용해야 한다. 임의의 정수를 요구하는 추첨 등에 사용된다.

Math클래스의 random()메서드를 사용하면 0이상 1미만의 double타입의 실수 값을 취득하게 된다.

Math.random()을 활용하여 주사위의 눈의 값을 취득하는 과정은 아래와 같다.

순서	연산	연산설명
1	<code>0 &lt;= Math.random() &lt; 1</code>	Math.random()의 범위
2	<code>0 &lt;= Math.random()*6 &lt; 6</code>	6을 곱해준다.
3	<code>1 &lt;= Math.random()*6+1 &lt; 7</code>	1을 더해준다.
결과	<code>1 &lt;= (int)(Math.random()*6+1) &lt; 7</code>	형변환을 활용하여 정수부분만 취득한다.

```
int random = (int)(Math.random()*6+1); // 0에서 6사이의 임의의 정수 값
```

[문제 2-13] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_13{
    public static void main(String[] args){
        //1. 변수 x를 선언하고 0이상 100미만 사이의 임의의 정수 값으로 초기화 하여라.

        //2. x에 저장된 값을 출력 하여라.

        //3. 변수 x2를 선언하고 43초과 375미만 사이의 임의의 정수 값으로 초기화 하여라.

        //4. x2에 저장된 값을 출력 하여라.

    }
}
```

### (3) 사용자로부터 입력받기

지금까지 System.out.println()을 활용하여 원하는 정보를 Eclipse의 Console창에 출력을 하였다. 필요에 따라 사용자로부터 어떠한 값을 입력받는 경우에는 Scanner클래스를 활용하면 된다. 자세한 내용은 객체지향 프로그래밍이후에 배울 테니 지금은 이해하기보다는 가져다 사용하는 정도로만 활용하면 된다.

Console창으로부터 사용자가 입력한 자료를 받기 위해서 Scanner클래스의 객체를 생성한다.

```
Scanner sc = new Scanner(System.in); // Scanner객체를 생성
```

그리고 nextLine( )이라는 메서드를 호출하면, 입력대기 상태에 있다가 입력을 마치고 '엔터키(Enter)'를 누르면 입력한 내용이 문자열로 반환된다.

```
String input1 = scanner.nextLine(); // 입력받은 내용을 한 줄을 기준으로 저장
String input2 = scanner.next(); // 입력받은 내용을 공백을 기준으로 저장
int input3 = scanner.nextInt(); // 입력받은 정수 저장. 단 숫자이외 입력하였을 때 오류
```

## ② 연습문제

[3-1] 다음중 형변환을 생략할 수 있는 것은? (모두 고르시오)

<pre>byte b = 10; char ch = 'A'; int in = 100; long lo = 1000L;</pre>	<p>(1) b =(byte) in;</p> <p>(2) ch = (char) b;</p> <p>(3) short s = (short) ch;</p> <p>(4) float f = (float) lo;</p> <p>(5) in = (int) ch;</p>
---	--

[3-2] 다음 연산의 결과를 적으시오.

```
class Exercise3_2{
    public static void main(String[] args){
        int x = 2;
        int y = 5;
        char c = 'A'; // 'A'의 문자코드는 65
        System.out.println(1 + x << 33);
        System.out.println(y >= 5 || x < 0 && x > 2);
        System.out.println(y += 10 - x++);
        System.out.println(x+=2);
        System.out.println( !( 'A' <= c && c <='Z' ) );
        System.out.println('C' - c);
        System.out.println('5' - '0');
        System.out.println(c+1);
        System.out.println(++c);
        System.out.println(c++);
        System.out.println(c);
    }
}
```

[3-3] 아래는 변수 num의 값에 따라 “양수”, “음수”, “0”을 출력하는 코드이다. 삼항 연산자를 이용해서 (1)에 알맞은 코드를 넣으시오.

```
class Exercise3_2{
    public static void main(String[] args){
        int num = 10;
        String result = /* (1) */;
        System.out.println(result);
    }
}
```

결과	양수
----	----



[3-4] 아래의 코드는 사과를 담는데 필요한 바구니 (버켓) 의 수를 구하는 코드이다. 만일 사과의 수가 123개이고 하나의 바구니에는 10개의 사과를 담을 수 있다면, 13개의 바구니가 필요할 것이다. (1)에 알맞은 코드를 넣으시오.

```
class Exercise3_4 {
    public static void main(String[] args) {
        int apples = 123;           // 사과의 개수
        int bucket = 10;           // 바구니의 크기(바구니에 담을 수 있는 사과의 개수)
        int numOfBucket = ( /* (1) */ ); // 모든 사과를 담는데 필요한 바구니의 수
        System.out.println("필요한 바구니의 수 : "+numOfBucket);
    }
}
```

결과      필요한 바구니의 수 : 13

[3-5] 아래의 코드는 변수 num의 값 중에서 백의 자리 이하를 버리는 코드이다. 만일 변수 num의 값이 '456'이라면 '400'이 되고, '111'이라면 '100'이 된다. (1)에 알맞은 코드를 넣으시오.

```
class Exercise3_5 {
    public static void main(String[] args) {
        int num = 456;
        int result = /* (1) */;
        System.out.println( result );
    }
}
```

결과      400

[3-6] 아래 코드의 문제점을 수정해서 실행 결과와 같은 결과를 얻도록 하시오.

```
class Exercise3_6 {
    public static void main(String[] args) {
        byte b = 20;
        byte c = a + b;
        char ch = 'A';
        ch = ch + 2;
        float f = 3 / 2;
        long l = 3000 * 3000 * 3000;
        float f2 = 0.1f;
        double d = 0.1;
        boolean result = d==f2;
        System.out.println("c="+c);
        System.out.println("ch="+ch);
        System.out.println("f="+f);
        System.out.println("l="+l);
        System.out.println("result="+result);
    }
}
```

결과      c = 30  
ch = c  
f = 1.5  
l = 27000000000  
result = true

[3-7] 아래는 변수 num의 값보다 크면서도 가장 가까운 10의 배수에서 변수 num의 값을 뺀 나머지를 구하는 코드이다. 예를 들어, 24의 크면서도 가장 가까운 10의 배수는 30이다. 19의 경우 20이고, 81의 경우 90이 된다. 30에서 24를 뺀 나머지는 6이기 때문에 변수 num의 값이 24라면 6을 결과로 얻어야 한다. (1)에 알맞은 코드를 넣으시오.

```
class Exercise3_7 {  
    public static void main(String[] args) {  
        int num = 24;  
        int result = /* (1) */;  
        System.out.println( "result : "+ result );  
    }  
}
```

결과

Result : 6

[3-8] 아래는 화씨(Fahrenheit)를 섭씨(Celcius)로 변환하는 코드이다. 변환공식이 ' $C = 5/9 \times (F - 32)$ '라고 할 때, (1)과 (2)에 알맞은 코드를 넣으시오.

단, 변환 결과값은 소수점 셋째자리에서 반올림해야한다.

```
class Exercise3_8 {  
    public static void main(String[] args) {  
        int fahrenheit = 100;           // 화씨  
        float formula = ( /* (1) */ ); // 변환공식 활용  
        float celcius = ( /* (2) */ ); // formula 소수점 셋째자리에서 반올림  
        System.out.println("Fahrenheit:"+fahrenheit);  
        System.out.println("Celcius:"+celcius);  
    }  
}
```

## 2-3. 명령문

### 학습목표

- 프로그래밍 언어의 명령문을 사용하여 프로그램의 흐름을 바꿀 수 있다.

### 필요 지식 /

#### 1 조건문

조건문은 조건식과 문장을 포함하는 블록{}으로 구성되어 있으며, 조건식의 연산결과에 따라 실행할 문장이 달라져서 프로그램의 실행흐름을 변경할 수 있다.

조건문은 if문과 switch문, 두 가지가 있으며 주로 if문이 많이 사용된다. 처리할 경우의 수가 많을 때는 if문보다 switch문이 효율적이지만, switch문은 if문보다 제약이 많다.

##### 1. if문

###### (1) 기본 if문

if문은 가장 기본적인 조건문이며, 다음과 같이 '조건식'과 '괄호'로 이루어져 있다. 'if'의 뜻이 '만일 ~이라면.'이므로 '만일(if) 조건식이 참(true)이면 괄호안의 문장들을 수행하라.'라는 의미로 이해하면 된다.

```
if(조건식){
    //조건식이 true일 때 수행될 문장들을 적는다.
}
```

만일 다음과 같은 if문이 있을 때, "score >= 60"이 true이면 괄호{} 안의 문장이 수행되어 화면에 "합격입니다."라고 출력되고 false이면, if문 다음의 문장으로 넘어간다.

```
if(score >= 60){
    System.out.println("합격입니다.");
}
```

[문제 2-14] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_14{
    public static void main(String[] args){
        //1. 변수 power를 선언하고 true의 값으로 초기화 하여라.

        //2. power의 값이 true이면 "켜져있습니다."를 출력 하여라.

    }
}
```

## (2) if-else문

if문의 변형인 if-else문의 구조는 다음과 같다. if문에 'else블럭'이 더 추가되었다. 'else'의 뜻이 '그 밖의 다른'이므로 조건식의 결과가 참이 아닐 때, 즉 거짓일 때 else블럭의 문장을 수행하라는 뜻이다.

```
if(조건식){
    //조건식이 true일 때 수행될 문장들을 적는다.
} else {
    // 조건식이 false일 때 수행될 문장들을 적는다.
}
```

조건식의 결과에 따라 이 두 개의 블록 중 어느 한 블록의 내용이 수행되고 전체 if문을 벗어나게 된다. 두 블록의 내용이 모두 수행되거나, 모두 수행되지 않는 경우는 있을 수 없다. 두 가지의 조건에서 하나를 선택해야 할 때 사용된다.

### [문제 2-15] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_15{
    public static void main(String[] args){
        //1. console창으로 사용자로부터 정수를 입력받아 변수 input에 저장 하여라.

        //2. input에 저장된 값이 2의 배수이면 “2의 배수입니다.”를 그 외에는
        // “2의 배수가 아닙니다.”를 출력 하여라.

    }
}
```

## (3) if-else if문

if-else문은 두 가지 경우 중 하나가 수행되는 구조인데, 처리해야 할 경우의 수가 셋 이상인 경우에는 어떻게 해야 할까? 그럴 때는 한 문장에 여러 개의 조건식을 쓸 수 있는 'if-else if'문을 사용하면 된다.

```
if (조건식1) {
    // 조건식1이 true일 때 수행될 문장들을 적는다.
} else if(조건식2) {
    // 조건식1이 false이고 조건식2가 true일 때 수행될 문장들을 적는다.
} else {
    // 조건식1과 조건식2가 모두 false일 때 수행될 문장들을 적는다.
}
```

첫 번째 조건식부터 순서대로 평가해서 결과가 참인 조건식을 만나면, 해당 블록만 수행하고 'if-else if'문 전체를 벗어난다. 만일 결과가 참인 조건식이 하나도 없으면, 마지막에 있는 else블록의 문장들이 수행된다. 그리고 else블록은 생략이 가능하다. else블록이 생략되었을 때는 if- else if문의 어떤 블록도 수행되지 않을 수 있다.

[문제 2-16] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_16{
    public static void main(String[] args){
        //1. 변수 score를 선언하고 0이상 100이하의 정수중에 임의의 값으로 초기화 하여라.

        //2. socre의 값이 90점 이상이면 "A"를 출력, 80점이상 90점 미만이면 "B"를 출력,
        // 70점이상 80점 미만이면 "C"를 출력, 60점이상 70점 미만이면 "D"를 출력
        // 60점 미만이면 "F"를 출력 하여라. (if-else if문을 이용 하여라.)

    }
}
```

#### (4) 중첩 if문

if문의 블록 내에 또 다른 if문을 포함시키는 것이 가능한데 이것을 중첩 if문이라고 부르며 중첩의 횟수에는 거의 제한이 없다.

```
if (조건식1) {
    // 조건식1이 true일 때 수행될 문장들을 적는다.
    if(조건식2){
        // 조건식1과 조건식2 모두 true일 때 수행될 문장들
    }
} else {
    // 조건식1이 false일 때 수행될 문장들을 적는다.
}
```

위와 같이 내부의 if문은 외부의 if문보다 안쪽으로 들여쓰기를 해서 두 if문의 범위가 명확히 구분될 수 있도록 작성해야 한다. 중첩if문에서는 괄호의 생략에 더욱 조심해야 한다. 바깥쪽의 if문과 안쪽의 if문이 서로 엉켜서 if문과 else블록의 관계가 의도한 바와 다르게 형성될 수도 있기 때문이다.

## [예제 2-11]

```
class Ex2_11{
    public static void main(String[] args){
        int score = 96;
        if(score >= 90){
            System.out.print("A");
            if(score >= 95){
                System.out.println("+");
            } else {
                System.out.println("-");
            }
        }
    }
}
```

결과	A+
----	----

위 예제는 모두 3개의 if문으로 이루어져 있으며 if문 안에 또 다른 2개의 if문을 포함하고 있는 모습을 하고 있다. 제일 바깥쪽에 있는 if문에서 점수에 따라 학점(grade)을 결정하고, 내부의 if문에서는 학점을 더 세부적으로 나누어서 평가를 하고 그 결과를 출력한다.

외부 if문의 조건식에 의해 한번 걸러졌기 때문에 내부 if문의 조건식은 더 간단해 질 수 있다.

## 2. switch문

if문은 조건식의 결과가 참과 거짓, 두 가지 밖에 없기 때문에 경우의 수가 많아질수록 else if를 계속 추가해야 하므로 조건식이 많아져서 복잡해지고, 여러 개의 조건식을 계산해야 하므로 처리시간도 많이 걸린다.

이러한 if문과 달리 switch문은 단 하나의 조건식으로 많은 경우의 수를 처리할 수 있고, 표현도 간결하므로 알아보기 쉽다. 그래서 처리할 경우의 수가 많은 경우에는 if문 보다 switch문으로 작성하는 것이 좋다. 다만 switch문은 제약조건이 있기 때문에, 경우의 수가 많아도 어쩔 수 없이 if문으로 작성해야 하는 경우가 있다.

switch문은 조건식을 먼저 계산한 다음, 그 결과와 일치하는 case문으로 이동한다. 이동한 case문 아래에 있는 문장들을 수행하며, break문을 만나면 전체 switch문을 빠져나가게 된다.

```
switch(조건식){
    case 값1 :
        // 조건식의 결과와 값1이 같을 경우 수행될 문장들
        break;
    case 값2 :
        // 조건식의 결과와 값2가 같을 경우 수행될 문장들
        break;
    default :
        // 조건식의 결과와 일치하는 case문이 없을 때 수행될 문장들
}
```

만일 조건식의 결과와 일치하는 case문이 하나도 없는 경우에는 default문으로 이동한다. default문은 if문의 else블럭과 같은 역할을 한다고 보면 이해가 쉬울 것이다. default문의 위치는 어디라도 상관없으나 보통 마지막에 놓기 때문에 break문을 쓰지 않아도 된다.

switch문에서 break문은 각 case문의 영역을 구분하는 역할을 하는데, 만일 break문을 생략하면 case문 사이의 구분이 없어지므로 다른 break문을 만나거나 switch문 블럭의 끝을 만날 때까지 나오는 모든 문장들을 수행한다. 이러한 이유로 각 case문의 마지막에 break문 을 빼먹는 실수를 하지 않도록 주의해야 한다.

#### switch문의 제약조건

1. switch문의 조건식 결과는 정수 또는 문자열이어야 한다.
2. case문의 값은 정수 상수(문자 포함), 문자열만 가능하며, 중복되지 않아야 한다.

#### [예제 2-12] 사용자가 입력한 월에 따른 계절을 출력

```
class Ex2_12{
    public static void main(String[] args){
        System.out.print("현재 월을 입력하세요 : ");
        Scanner sc = new Scanner(System.in);
        int mon = sc.nextInt(); // 화면을 통해 입력받은 숫자를 month에 저장
        switch(mon){
            case 3: case 4: case 5:
                System.out.println("봄 입니다.");
                break;
            case 6: case 7: case 8:
                System.out.println("여름 입니다.");
                break;
            case 9: case 10: case 11:
                System.out.println("가을 입니다.");
                break;
            case 12: case 1: case 2:
                System.out.println("겨울 입니다.");
                break;
            default:
                System.out.println("잘못된 월을 입력하셨습니다.");
        }
    }
}
```

case문은 한 줄에 하나씩 쓰던, 한 줄에 붙여서 쓰던 상관없다.

[문제 2-17] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_17{
    public static void main(String[] args){
        //1. 변수 random을 선언하고 1이상 5이하의 임의의 정수로 초기화 하여라.

        //2. random의 값이 1이면 “32평 아파트 당첨“을, 2이면 “자동차 당첨“,
        // 3이면 “노트북 당첨“, 4이면 “자전거 당첨“, 5이면 “다음 기회에“
        // 를 출력 하여라.

    }
}
```

[문제 2-18] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_18{
    public static void main(String[] args){
        //1. 변수 score을 선언하고 0이상 100이하의 임의의 정수로 초기화 하여라.

        //2. socre의 값이 90점 이상이면 “A“를 출력, 80점이상 90점 미만이면 “B“를 출력,
        // 70점이상 80점 미만이면 “C“를 출력, 60점이상 70점 미만이면 “D“를 출력
        // 60점 미만이면 “F“를 출력 하여라.
        // 단. switch문을 이용 하고 case는 6개만 이용하여라.

    }
}
```



## ② 반복문

반복문은 어떤 작업이 반복적으로 수행되도록 할 때 사용되며, 반복문의 종류로는 for문과 while문, 그리고 while문의 변형인 do-while문이 있다.

for문이나 while문에 속한 문장은 조건에 따라 한 번도 수행되지 않을 수 있지만 do-while문에 속한 문장은 무조건 최소한 한 번은 수행될 것이 보장된다. 반복문은 주어진 조건을 만족하는 동안 주어진 문장들을 반복적으로 수행하므로 조건식을 포함하며 if문과 마찬가지로 조건식의 결과가 true이면 참이고, false면 거짓으로 간주 된다.

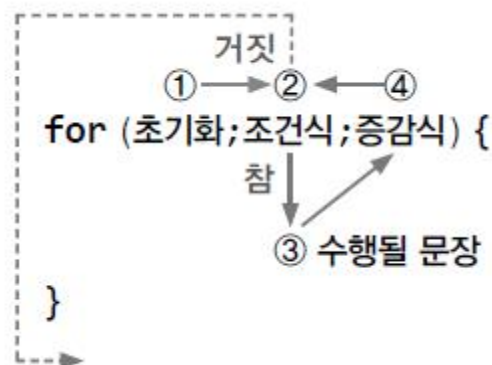
for문과 while문은 구조와 기능이 유사하여 어느 경우에도 서로 변환이 가능하기 때문에 반복문을 작성해야 할 때 for문과 while문 중 어느 쪽을 선택해도 좋으나 for문은 주로 반복횟수를 알고 있을 때 사용한다.

### 1. for문

for문은 반복 횟수를 알고 있을 때 적합하며 아래와 같이 '초기화', '조건식', '증감식', '블록' 모두 4부분으로 이루어져 있으며, 조건식이 true인 동안 블록 내의 문장들을 반복하다 false가 되면 반복문을 벗어난다.

```
for ( 초기화; 조건식 ; 증감식 ) {  
    // 조건식이 true인 동안 수행될 문장들  
}
```

for문이 실행되게 되면 제일 먼저 '초기화'가 수행되고, 그 이후부터는 조건식이 true인 동안 '조건식 -> 수행될 문장 -> 증감식'의 순서대로 계속 반복된다. 그러다가 조건식이 false가 되면, for문 정체를 빠져나가게 된다.



**[예제 2-13] “Hello Java”를 3번 반복 출력**

```
class Ex2_13{
    public static void main(String[] args){
        for(int i = 0; i < 3; i++) {
            System.out.print("Hello Java");
        }
    }
}
```

초기화에서 변수 i에 0을 저장한 다음, 조건에 만족한다면 문장을 수행한 다음, i가 1씩 증가하는 시킨다. 그러다가 i의 값이 3이 되면 조건식의 결과가 false가 되므로 반복문을 마치게 된다. i를 0부터 시작하게 만든다면 조건식에서 횟수를 바꾸면 해당 횟수만큼 반복하게 된다.

**[문제 2-19] 주석에 적당한 코드를 작성해 주세요.**

```
class Qu2_19{
    public static void main(String[] args){
        //1. 0이상 5이하에서 정수를 결과와 같이 출력 하여라.

        //2. 00이상 50이하에서 정수를 결과와 같이 출력 하여라.

        //3. 20이상 100이하에서 정수를 합계를 구하여라.

        //4. 50이상 150미만에서 정수의 곱을 구하여라. ( overflow를 고려하여라.)

        //5. 30이상 4462이하에서 짝수인 정수의 합을 구하여라.
    }
}
```

//6. 7초과 57미만에서 2또는 3의 배수인 정수의 합을 구하여라.

```
}  
}
```

결과	1) 0 1 2 3 4 5 2) 5 4 3 2 1 0 3) 55 4) 3632428800 5) 4979591 6) 1041
----	---

for문 안에 또 다른 for문을 포함시키는 것이 가능하며, 중첩 for문이라고 한다. 중첩횟수는 거의 제한이 없다.

만일 다음과 같이 출력을 하려면 어떻게 해야 할까?

```
*****  
*****  
*****
```

가장 간단한 방법은 다음과 같이 한 줄씩 3번 출력하는 것이다.

```
System.out.println("****") ;  
System.out.println("****");  
System.out.println("****");
```

동일한 문장이 3회 반복 되므로 for문을 이용하여 다음과 같이 작성할 수 있다.

```
for( int count=0 ; count < 3 ; count++) {  
    System.out.println("****") ;  
}
```

' System.out.println("\*\*\*\*");'역시 반복적인 일을 수행한다. 반복할 이 가능한 형태로 출력한다면 다음과 같이 작성할 수 있다.

반복구문을 확인하기 위해서는 가로로 작성하기 보다는 세로로 작성하는 것이 좋다.

```
System.out.print("****");  
System.out.print("****");  
System.out.print("****");
```

```
System.out.println();
```

'System.out.print("\*\*\*);'위의 문장에서 3개의 줄이 반복적으로 일을 수행함을 알 수 있다. for문을 이용하여 다음과 같이 작성할 수 있다.

```
for( int star=0 ; star < 3 ; star++) {  
    System.out.print("**" );  
}  
System.out.println();           // 반복구문에 포함되지 않는 부분
```

선행하여 만들어진 for문에 추가 작성된 for문을 넣으면 다음과 같이 두 개의 for문이 중첩되게 된다.

```
for( int count=0 ; count < 3 ; count++) {  
    for( int star=0 ; star < 3 ; star++) {  
        System.out.print("**" );  
    }  
    System.out.println();  
}
```

#### [예제 2-14] 구구단을 출력

```
class Ex2_14{  
    public static void main(String[] args){  
        for(int dan = 2; dan < 10; dan++) {  
            for(int gob = 1; gob < 10; gob++) {  
                System.out.println(dan + " * " + gob + " = " + dan*gob);  
            }  
        }  
    }  
}
```

#### [문제 2-20] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_20{  
    public static void main(String[] args){  
        //1. 구구단의 짝수 단만 출력 하여라.
```

//2. 구구단의 짝수 단의 홀수 곱을 출력 하여라.

```
}  
}
```

[문제 2-21] 결과와 같이 출력되도록 코드를 작성 하여라.

```
class Qu2_21{  
    public static void main(String[] args){  
  
  
  
  
  
  
  
  
  
    }  
}
```

결과

```
*  
**  
***  
****  
*****
```

## 2. while문

for문에 비해 while문은 구조가 간단하다. if문처럼 조건식과 블록으로 이루어져 있다. 다만 if문과 달리 while문은 조건식이 '참(true)인 동안', 즉 조건식이 거짓이 될 때까지 블록 내의 문장을 반복한다.

```
while (조건식) {  
    // 조건식의 연산결과가 true인 동안 반복될 문장들  
}
```

while문은 먼저 조건식을 평가해서 조건식이 false이면 문장 정체를 벗어나고, true이면 블록 내의 문장을 수행하고 다시 조건식으로 돌아간다. 조건식이 false가 될 때까지 이 과정을 계속 반복한다.

1부터 5까지 정수를 순서대로 출력하는 for문을 while문으로 변경하면 아래와 같다.

```
for( int count=1; count < 6; count ++){  
    System.out.println(count);  
}
```

```
int count = 1; // 초기화  
while(count<6) { // 조건식  
    System.out.println(count);  
    count++; // 증감식  
}
```

위의 두 코드는 완전히 동일하다. for문은 초기화, 조건식, 증감식을 한 곳에 모아 놓은 것일 뿐, while 문과 다르지 않다. 그래서 for문과 while문은 항상 서로 변환이 가능하다.

그래도 이 경우 while문 보다 for문이 더 간결하고 알아보기 쉽다. 만일 초기화이나 증감식이 필요하지 않은 경우라면, while문이 더 적합할 것이다.

**[문제 2-22] 주석에 적당한 코드를 작성해 주세요.**

```
class Qu2_22 { // 1~100까지의 합계를 구하여라.
    public static void main(String[] args){
        //1. while문을 이용하여 1이상 100이하까지의 합계를 구하여라.

    }
}
```

**[문제 2-23] 주석에 적당한 코드를 작성해 주세요.**

```
class Qu2_23 {
    public static void main(String[] args){
        //1. 1부터 1씩 증가하는 정수들을 더하였을 때 합계가 200을 넘게 되는 최초의 값과
        //   합계를 구하여라.

    }
}
```

[Hint] 증감연산자의 위치를 고려해야 한다.

### [예제 2-15] 숫자로 이루어진 문자열의 합계구하기

```
class Ex2_15 {  
    public static void main(String[] args){  
        String str = "184520";  
        int sum = 0;  
        for( int count = 0; count < str.length(); count ++){  
            sum += str.charAt(count)-'0';  
        }  
        System.out.println(sum);  
    }  
}
```

결과 20

str.length()는 str문자열의 길이를 정수형으로 반환하기 때문에 [예제 2-15]에서는 6을 의미한다. 또한 str.charAt(index)는 str문자열의 index(정수)번째 문자 하나를 선택하여 문자형(char)을 반환하여 준다. 예를 들어 str.charAt(2)은 '4'를 반환하여 준다. 문자열 str에서 index번째 값에서 '0'을 빼줌으로써 원하는 정수형 값을 취득할 수 있다.

[문제 2-24] 주식에 적당한 코드를 작성해 주세요.

```
class Qu2_24 {
    public static void main(String[] args) {
        //1. 변수 num을 선언하고 사용자로부터 숫자를 입력 받은 값으로 초기화 하여라.

        //2. 변수 num에 저장된 값의 각 자리의 합계를 구하여라.
        // 예) 사용자가 543을 입력하였다면 합계 결과는 12이다.


    }
}
```

### 3. do-while문

do-while문은 while문의 변형으로 기본적인 구조는 while문과 같으나 조건식과 블록{}의 순서를 바꿔놓은 것이다. 그래서 while문과 반대로 블록{}을 먼저 수행한 후에 조건식을 평가한다. while문은 조건식의 결과에 따라 블록{}이 한 번도 수행되지 않을 수 있지만, do-while문은 최소한 한번은 수행된다.

```
do {  
    // 조건식의 연산결과가 true일때 수행될 문장들  
    // 처음 한번은 조건에 상관없이 실행  
} while (조건식) ;
```

그리 많이 쓰이지는 않지만, 다음의 예제처럼 반복적으로 사용자의 입력을 받아서 처리할 때 유용하다.

#### [예제 2-16] 숫자 맞추기

```
class Ex2_16 {  
    public static void main(String[] args){  
        int input = 0; // 사용자로부터 입력받은 값을 저장할 변수  
        int answer = (int)(Math.random()*100 +1); // 1~100사이의 임의의 정수를 저장할 변수  
        Scanner sc = new Scanner(System.in);  
  
        do{  
            System.out.print("1과 100사이의 정수를 입력하세요 : ");  
            input = sc.nextInt();  
  
            if(input > answer){  
                System.out.println("더 큰 수를 입력해주세요");  
            } else if(input < answer){  
                System.out.println("더 작은 수를 입력해주세요");  
            }  
        } while(input != answer);  
  
        System.out.println("정답입니다.");  
    }  
}
```

Math.random() 을 이용해서 1과 100 사이의 임의의 수를 변수 answer에 저장하고, 이 값을 맞출 때까지 반복하는 예제이다. 사용자 입력인 input이 변수 answer의 값과 다른 동안 반복하다가 두 값이 같으면 반복을 벗어난다.



#### 4. break문

앞서 switch문에서 break문에 대해 배웠던 것을 기억할 것이다. 반복문에서도 break문을 사용할 수 있는데, switch문에서 그랬던 것처럼, break문은 자신이 포함된 가장 가까운 반복문을 벗어난다. 주로 if문과 함께 사용되어 특정 조건을 만족할 때 반복문을 벗어나게 한다.

**[예제 2-17] [문제 2-23]을 break문을 이용하여 풀이**

```
class Ex2_17 {
    public static void main(String[] args){
        int count = 1;
        int sum = 0;
        while(true){
            sum += count;
            if(sum > 200 ){
                break; // 가장 가까운 반복문을 벗어난다.
            }
            count++;
        }
        System.out.println("합계가 200을 넘는 정수는 : " + count + "이다.");
    }
}
```

숫자를 1부터 계속 더해 나가서 몇까지 더하면 합이 200을 넘는지 알아내는 예제이다. count의 값을 1부터 1씩 계속 증가시켜가며 더해서 sum에 저장한다. sum의 값이 200을 넘으면 if문의 조건식이 참이므로 break문이 수행되어 자신이 속한 반복문을 즉시 벗어난다.

이처럼 무한 반복문에는 조건문과 break문이 항상 같이 사용된다. 그렇지 않으면 무한히 반복되기 때문에 프로그램이 종료되지 않을 것이다.

## 5. continue문

continue문은 반복문 내에서만 사용될 수 있으며, 반복이 진행되는 도중에 continue문을 만나면 반복문의 끝으로 이동하여 다음 반복으로 넘어간다. for문의 경우 증감식으로 이동하며, while문과 do-while문의 경우 조건식으로 이동한다.

continue문은 반복문 전체를 벗어나지 않고 다음 반복을 계속 수행한다는 break문과 다르다. 주로 if문과 함께 사용되어 특정 조건을 만족하는 경우에 continue문 이후의 문장들을 수행하지 않고 다음 반복으로 넘어가서 계속 진행하도록 한다.

**[예제 2-18] 3에서부터 20까지의 정수에서 짝수만 출력하는 예제**

```
class Ex2_18 {
    public static void main(String[] args){
        for(int count = 3; count < 21; count++){
            if(count%2 != 0){
                continue;
                // 조건식이 참이 되어 continue문이 수행되면 블록의 끝으로 이동한다. break문과
                // 달리 반복문을 벗어나지 않는다.
            }
            System.out.println(count);
        }
    }
}
```

3과 20사이의 숫자를 출력하되 그 중에서 2의 배수인 것은 제외하도록 하였다. count의 값이 2의 배수인 경우, if문의 조건식 'count%2 != 0'은 참이 되어 continue문에 의해 반복문의 블록 끝으로 이동된다. 즉 continue문과 반복문 블록의 끝 사이의 문장들을 건너뛰고 반복을 이어 가는 것이다.

### ③ 연습문제

[4-1] 다음의 문장들을 조건식으로 표현하라.

- (1) int형 변수 x가 10보다 크고 20보다 작을 때 true인 조건식
- (2) char형 변수 ch가 공백이고 탭이 아닐 때 true인 조건식
- (3) char형 변수 ch가 'x' 또는 'X'일 때 true인 조건식
- (4) char형 변수 ch가 숫자('0'~'9')일 때 true인 조건식
- (5) char형 변수 ch가 영문자(대문자 또는 소문자)일 때 true인 조건식
- (6) boolean형 변수 powerOn가 false일 때 true인 조건식
- (7) 문자열 참조변수 str이 "yes"일 때 true인 조건식

[4-2] 1부터 20까지의 정수 중에서 2 또는 3의 배수가 아닌 수의 총합을 구하시오.

[4-3] 다음의 for문을 while문으로 변경하시오.

```
class Exercise4_3 {  
    public static void main(String[] args) {  
        for(int dan = 2; dan < 10; dan++) {  
            for(int gob = 1; gob < 10; gob++) {  
                System.out.println(dan + " * " + gob + " = " + dan*gob);  
            }  
        }  
    }  
}
```

[4-4] 두 개의 주사위를 던졌을 때, 눈의 합이 6이 되는 모든 경우의 수를 출력하는 프로그램을 작성하시오.

[4-5] 방정식  $2x+4y=10$ 의 모든 해를 구하시오. 단,  $x$ 와  $y$ 는 정수이고 각각의 범위는  $0 \leq x \leq 10$ ,  $0 \leq y \leq 10$  이다.

[4-6] 사용자로부터 두개의 정수(시작, 끝)를 입력받아 시작(포함)에서 끝(포함)까지의 곱을 출력하는 프로그램을 작성하시오.

[4-7]  $1 + (1+2) + (1+2+3) + (1+2+3+4) + \dots + (1+2+3+\dots+10)$  의 결과를 계산하시오.

[4-8]  $1 + (-2) + 3 + (-4) + \dots$  과 같은 식으로 계속 더해나갔을 때, 몇까지 더해야 총합이 100 이상이 되는지 구하시오.

[4-9] 사용자로부터 입력받은 정수의 각 자리의 합을 더한 결과를 출력하는 프로그램을 작성하시오.  
예를 들어 사용자가 53263을 입력하였다면 결과는 19가 되어야 한다.

[4-10] 피보나치(Fibonacci) 수열(數列)은 앞을 두 수를 더해서 다음 수를 만들어 나가는 수열이다. 예를 들어 앞의 두 수가 1과 1이라면 그 다음 수는 2가 되고 그 다음 수는 1과 2를 더해서 3이 되어서 1,1,2,3,5,8,13,21,... 과 같은 식으로 진행된다. 1과 1부터 시작하는 피보나치수열의 10번째 수는 무엇인지 계산하는 프로그램을 작성하시오.

[4-11] 다음은 주어진 문자열(value)이 숫자인지를 판별하는 프로그램이다. (1)에 알맞은 코드를 넣어서 프로그램을 완성하시오.

```
class Exercise4_11 {
    public static void main(String[] args) {
        String value = "12o34";
        char ch = ' ';
        boolean isNumber = true;
        // 반복문과 charAt(int i)를 이용해서 문자열의 문자를
        // 하나씩 읽어서 검사한다.
        for(int i=0; i < value.length() ;i++){
            /*
                (1) 알맞은 코드를 넣어 완성하시오.

            */
        }
        if (isNumber){
            System.out.println(value+"는 숫자입니다.");
        } else {
            System.out.println(value+"는 숫자가 아닙니다.");
        }
    }
}
```

## 2-5. 배열(Array)

### 학습목표

- 배열의 형태를 이해하고 객체생성 및 자료추가의 과정을 이해할 수 있다.
- 배열을 for문을 이용하여 개별 자료에 접근하여 조회할 수 있다.

### 필요 지식 /

#### 1 배열

같은 타입의 여러 변수를 하나의 묶음으로 다루는 것을 '배열(array)'이라고 한다. 많은 양의 데이터를 저장하기 위해서, 그 데이터의 숫자만큼 변수를 선언해야 한다면 매우 혼란스러울 것이다. 10,000개의 데이터를 저장하기 위해 같은 수의 변수를 선언해야한다면 상상하는 것만으로도 상당히 곤혹스러울 것이다.

이런 경우에 배열을 사용하면 많은 양의 데이터를 손쉽게 다룰 수 있다.

**배열은 같은 타입의 여러 변수를 하나의 묶음으로 다루는 것**

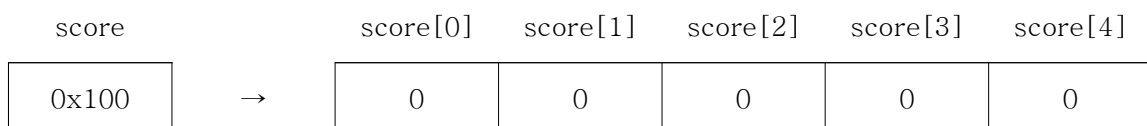
여기서 중요한 것은 '같은 타입'이어야 한다는 것이며, 서로 다른 타입의 변수들로 구성된 배열은 만들 수 없다. 한명의 시험점수를 저장하고자 할 때가 배열을 사용하기 좋은 예이다. 만일 배열을 사용하지 않는다면 5개의 과목 점수를 저장하기 위해서 아래와 같이 5개의 변수를 선언해야 할 것이다.

```
int kor, eng, math, java, oracle;
```

변수 대신 배열을 이용하면 다음과 같이 간단히 처리할 수 있다. 변수의 선언과 달리 다뤄야 할 데이터의 수가 아무리 많아도 단지 배열의 길이만 바꾸면 된다.

```
int[] score = new int[5]; // 5개의 int 값을 저장할 수 있는 배열을 생성한다.
```

아래의 그림은 위의 코드가 실행되어 생성된 배열을 그림으로 나타낸 것이다. 값을 저장할 수 있는 공간은 score[0] 부터 score[4]까지 모두 5개이며, 변수 score는 배열을 다루는데 필요한 참조변수일 뿐 값을 저장하기 위한 공간은 아니다. 잠시 후에 자세히 설명할 것이니까 지금은 참고만 하고 가볍게 넘어가자.



## 1. 배열의 선언과 생성

### (1) 배열의 선언

배열을 선언하는 방법은 간단하다. 원하는 타입의 변수를 선언하고 변수 또는 타입에 배열임을 의미하는 대괄호[]를 붙이면 된다. 대괄호[]는 타입 뒤에 붙여도 되고 변수이름 뒤에 붙여도 되는데, 저자의 경우 대괄호를 타입에 붙이는 쪽을 선호한다. 대괄호가 변수이름의 일부라기보다는 타입의 일부라고 보기 때문이다.

선언 방법	선언 예
타입[] 변수이름;	int[] score; String[] name;
타입 변수이름[];	int score[]; String name[];

### (2) 배열의 생성

배열을 선언한 다음에는 배열을 생성해야 한다. 배열을 선언하는 것은 단지 생성된 배열을 다루기 위한 참조변수를 위한 공간이 만들어질 뿐이고, 배열을 생성해야만 비로소 값을 저장할 수 있는 공간이 만들어지는 것이다. 배열을 생성하기 위해서는 연산자 'new'와 함께 배열의 타입과 길이를 지정해 주어야 한다.

```
타입[] 변수이름;           // 배열을 선언(배열을 다루기 위한 참조변수 선언)
변수이름 = new 타입[길이]; // 배열을 생성(실제 저장공간을 생성)
```

아래의 코드는 '길이가 5인 int배열'을 생성한다.

```
int[] score;           // int타입의 배열을 다루기 위한 참조변수 score선언
score = new int[5];     // int타입의 값 5개를 저장할 수 있는 배열 생성
```

다음과 같이 배열의 선언과 생성을 동시에 하면 간략히 한 줄로 할 수 있는데, 대부분의 경우 이렇게 한다.

```
타입[] 변수이름 = new 타입[길이]; // 배열의 선언 및 생성
int[] score = new int[5];         // 길이가 5인 int형 배열
```

score		score[0]	score[1]	score[2]	score[3]	score[4]
0x100	→	0	0	0	0	0

배열의 주소는 random하게 생성되지만 0x100으로 예를 든 것이다.

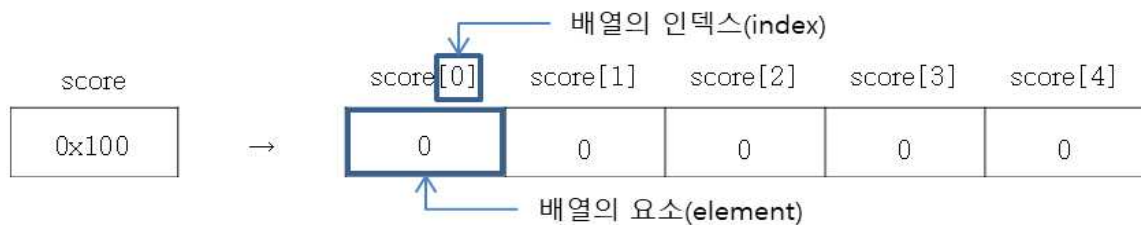
## 2. 배열의 인덱스

생성된 배열의 각 저장공간을 '배열의 요소(element)'라고 하며, '배열이름[인덱스]'의 형식으로 배열의 요소에 접근한다. 인덱스(index)는 배열의 요소마다 붙여진 일련번호로 각 요소를 구별하는데 사용된다. 우리가 변수의 이름을 지을 때 `score1`, `score2`, `score3`과 같이 번호를 붙이는 것과 비슷하다고 할 수 있다. 다만 인덱스는 1이 아닌 0부터 시작한다.

인덱스(index)의 범위는 0부터 배열길이-1까지 이다.

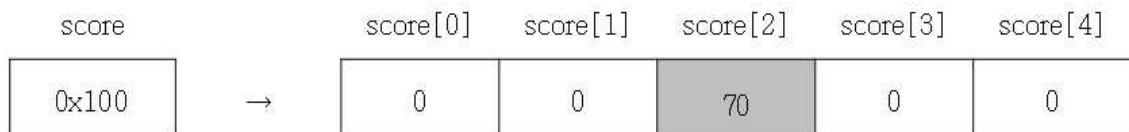
예를 들어 길이가 5인 배열은 모두 5개의 요소(저장공간)를 가지며 인덱스의 범위는 1부터 5까지가 아닌 0부터 4까지, 즉 0,1,2,3,4가 된다.

`int[] score = new int[5];` // 길이가 5인 int형 배열



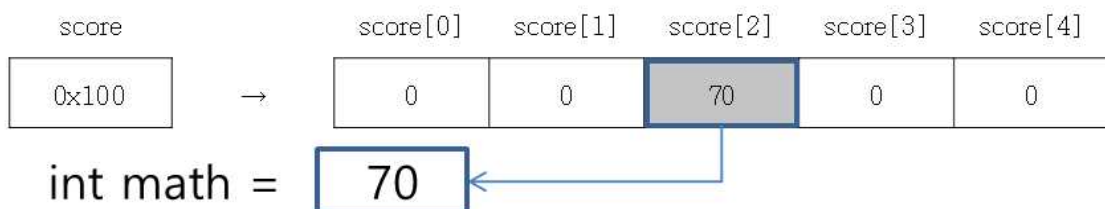
배열에 값을 저장하는 방법은 변수와 같다. 단지 변수이름 대신 '배열이름[인덱스]'를 사용한다는 점만 다르다.

`score[2] = 70;` // 배열 `score`의 3번째 요소에 70을 저장한다.



배열의 요소에서 2번 index의 값을 읽어오는 방법은 아래와 같다.

`int math = score[2];` // 배열 `score`의 3번째 요소의 값을 읽어 새로운 변수 `math`에 저장





### 3. 배열의 초기화

배열은 생성과 동시에 자동적으로 타입에 해당하는 기본값으로 초기화되므로 배열을 사용하기 전에 따로 초기화를 해주지 않아도 되지만, 원하는 값을 저장하려면 아래와 같이 각 요소마다 값을 지정해 줘야한다.

```
int[] score = new int[5]; // 길이가 5인 int형 배열을 생성한다.
score[0] = 0;             // 각 요소에 직접 값을 저장한다.
score[1] = 10;
score[2] = 20;
score[3] = 30;
score[4] = 40;
```

[문제 2-25] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_25 {
    public static void main(String[] args) {
        //1. 위의 내용을 반복문을 이용하여 바꾸어라.

        //2. 1에서 작성된 변수 score에 저장된 요소를 결과와 같이 출력하여라.

    }
}
```

결과	score[0] : 0 score[1] : 10 score[2] : 20 score[3] : 30 score[4] : 40
----	--

위의 예와 같이 배열을 생성하고 값을 저장하는 것을 나누어서 처리 하는 경우도 있지만 아래와 같이 동시에 처리도 가능하다.

```
int[] score = new int[]{ 0,10,20,30,40 }; // 배열의 생성 및 초기화
```

동시에 처리에 주의할 점은 생성과 값의 저장을 따로 이루어 질 때는 'new int[길이];'로 길이를 필수적으로 입력해야 한다. 하지만 동시에 진행할 때는 길이의 부분을 입력을 하면 컴파일 에러가 발생하게 된다. 즉 길이를 기입해서는 안 된다.

## 4. 배열의 활용

### (1) 배열의 길이

자바에서는 JVM이 모든 배열의 길이를 별도로 관리하며, '배열이름.length'를 통해서 배열의 길이에 대한 정보를 얻을 수 있다.

아래의 코드에서 배열 arr의 길이가 5이므로 arr.length의 값 역시 5가 된다.

```
int[] arr = new int[5]; // 길이가 5인 int배열
int tmp = arr.length; // arr.length의 값은 5이고 tmp에 5가 저장된다.
```

배열은 한번 생성하면 길이를 변경할 수 없기 때문에, 이미 생성된 배열의 길이는 변하지 않는다. 따라서 '배열이름.length'는 상수다. 즉, 값을 읽을 수만 있을 뿐 변경할 수 없다.

### [예제 2-19] 배열의 길이 활용

```
class Ex2_19 {
    public static void main(String[] args){
        // 1. 정수 5개를 저장할 수 있는 변수 score를 선언 및 생성
        int[] score = new int[5];

        //2. score의 요소를 출력하는데 실수로 6으로 입력(또는 위에서 배열의 크기 변경)
        // 없는 요소를 요청함으로 ArrayIndexOutOfBoundsException이라는 예외 발생
        for(int i = 0; i < 6; i++){
            System.out.println(score[i]);
        }

        //3. 2와 같은 실수를 하지 않기 위해 배열의 크기만큼만 반복하도록 한다.
        // score.length를 사용하여 score배열의 크기가 변경, 배열크기를 잘못 입력하였을 때
        // 예외를 피할 수 있다.
        for(int i = 0; i < score.length; i++){
            System.out.println(score[i]);
        }
    }
}
```

'배열이름.length'는 배열의 길이가 변경되면 자동적으로 변경된 배열의 길이를 값으로 갖기 때문에, 배열과 함께 사용되는 for문의 조건식을 일일이 변경해주지 않아도 된다.

## (2) 배열의 활용

지금까지 배열의 기본적인 내용은 모두 살펴보았는데, 아직 배열을 어떻게 활용해야 할지 감이 잘 오지 않을 것이다. 이제 다양한 예제를 통해서 배열을 어떻게 활용법을 배워보자.

### [예제 2-20] 배열의 요소들의 합계 및 평균 구하기

```
class Ex2_20 {
    public static void main(String[] args){
        int sum = 0;           // 총점을 저장하기 위한 변수

        int[] score = {40, 80, 73, 84, 92}; // 선언 및 생성을 동시에 할 때 가능
        for(int i = 0; i < score.length; i++){
            sum += score[i];
        }

        float average = 0f; // 평균을 저장하기 위한 변수
        average = (float) sum / score.length; // 계산결과를 float로 얻기 위해서 형변환

        System.out.println("총점 : "+ sum );
        System.out.println("평균 : "+ average );
    }
}
```

결과	총점 : 369
	평균 : 73.8

for문을 이용해서 배열에 저장된 값을 모두 더한 결과를 배열의 개수로 나누어서 평균을 구하는 예제이다. 평균을 구하기 위해 전체 합을 배열의 길이인 score.length로 나누었다.

이때 int와 int 간의 연산은 int를 결과로 얻기 때문에 정확한 평균값을 얻지 못하므로 sum을 float로 변환하여 나눗셈을 하였다.

### [문제 2-26] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_26 {
    public static void main(String[] args) {
        //1. 정수 7개를 저장할 수 있는 변수 socre를 선언 및 생성하여라.

        //2. 변수 score의 각 요소를 0이상 100이하인 임의의 정수를 저장하여라.
    }
}
```

//3. 변수 score에 저장된 요소들의 합계를 구하여라.

//4. 변수 score에 저장된 요소들의 평균을 구하여라.

// 단 소수점 세 번째 자리에서 반올림하여 두 번째 자리까지 표현하여라.

//5. 변수 socre에 저장된 요소에서 최댓값을 구하여라.

//6. 변수 socre에 저장된 요소에서 최솟값을 구하여라.

//7. 1~6번의 결과를 이용하여 결과와 같이 출력하여라.

// 단. 임의의 정수값들의 연산임으로 결과는 다를 수 있음.

}  
}

결과

socre값 : {64, 23, 77, 83, 25, 100, 93 }  
합계 : 465  
평균 : 66.43  
최댓값 : 100  
최솟값 : 23

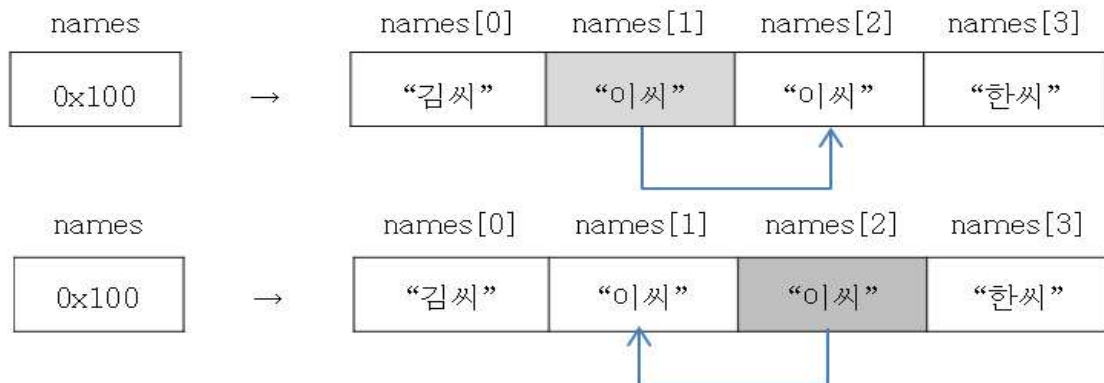
### (3) 배열의 자리 바꾸기

#### [예제 2-21] 배열의 요소의 자리 바꾸기

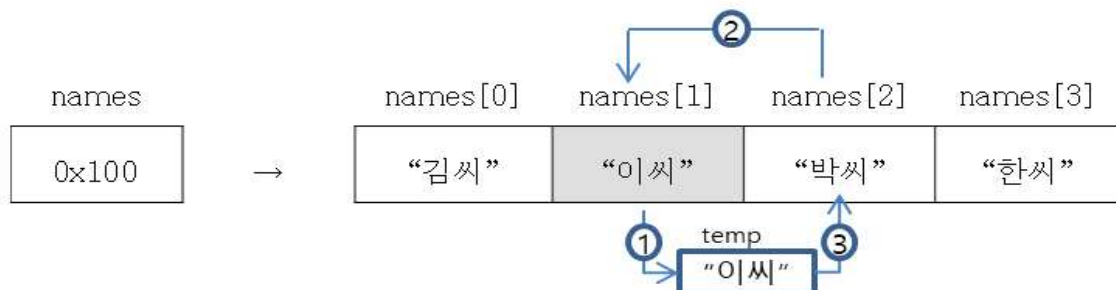
```
class Ex2_21 {  
    public static void main(String[] args){  
        //1. 사람4명의 이름을 저장할 수 있는 변수 names를 선언하고  
        // “김씨”, “이씨”, “박씨”, “한씨”의 이름으로 초기화.  
        String[] names = new String[]{"김씨", "이씨", "박씨", "한씨"};  
        System.out.println("변경전 : "+Arrays.toString(names));  
  
        //2. “이씨”와 “박씨”의 위치를 변경  
        String temp = names[1]; // 1번 인덱스의 값을 임시 저장한다.  
        names[1] = names[2];    // 1번 인덱스의 값을 2번 인덱스의 값으로 변경한다.  
        names[2] = temp;        // 2번 인덱스의 값을 임시저장 되었던 값으로 변경한다.  
  
        System.out.println("변경후 : "+Arrays.toString(names));  
    }  
}
```

**결과**  
변경전 : {"김씨", "이씨", "박씨", "한씨"}  
변경후 : {"김씨", "박씨", "이씨", "한씨"}

1번 인덱스의 값을 2번 인덱스에 저장한 뒤에 2번 인덱스의 값을 1번 인덱스에 저장한다면 2번 인덱스의 값이 변경된 상태에서 1번인덱스에 저장되게 되어 잘못된 결과를 얻게 된다.



그래서 아래와 같이 temp라는 임시 저장소를 이용하여 자리 바꾸기를 진행해야 한다.



#### (4) 정렬

정렬이란 어떤 데이터를 빠르고 쉽게 찾기 위해 일정한 순서대로 가지런히 나열하는 작업을 의미한다. 정렬에는 버블정렬(Bubble sort), 선택정렬(Select sort), 삽입정렬(Insert sort) 등이 있다. 이중 버블 정렬과 선택정렬에 대해서 알아보도록 하자.

##### · 버블정렬(Bubble Sort)

인접한 데이터 간에 교환이 계속해서 일어나면서 정렬이 이루어진다. 오름차순정렬의 경우 한 회전이 완료되었을 때 가장 큰 값이 뒤쪽에 확정이 되게 된다.

아래와 같은 5개의 숫자가 섞여 있을 때 버블정렬을 이용하여 정렬을 진행하여 보자.

5	2	3	1	4
---	---	---	---	---

- 1) 0번 index 요소의 값과 두 번째 요소의 값을 비교하여 1번 index 요소의 값이 더 크다면 두 index의 값을 바꾸어 아래와 같은 결과를 가지게 된다.

2	5	3	1	4
---	---	---	---	---

- 2) 1번 index 요소의 값과 두 번째 요소의 값을 비교하여 2번 index 요소의 값이 더 크다면 두 index의 값을 바꾸어 아래와 같은 결과를 가지게 된다.

2	3	5	1	4
---	---	---	---	---

- 3) 2번 index 요소의 값과 두 번째 요소의 값을 비교하여 3번 index 요소의 값이 더 크다면 두 index의 값을 바꾸어 아래와 같은 결과를 가지게 된다.

2	3	1	5	4
---	---	---	---	---

- 4) 3번 index 요소의 값과 두 번째 요소의 값을 비교하여 4번 index 요소의 값이 더 크다면 두 index의 값을 바꾸어 아래와 같은 결과를 가지게 된다.

2	3	1	4	5
---	---	---	---	---

위의 결과가 1회전이 완료된 결과로써 가장 큰 값인 5가 가장 뒤로 가서 확정되게 된다.  
뒤에서 하나씩 확정되어 때문에 회전수는 항상 '정렬할 숫자의 개수 -1'만큼 회전을 진행하게 된다.

[문제 2-27] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_27{
    public static void main(String[] args) {
        //1. 5개의 숫자 5,2,3,1,4를 오름차순으로 정렬하는 코드를 작성하여라.
        // 단. 버블정렬을 이용하고 결과와 같이 출력되도록 하여라.

    }
}
```

결과

```
1회전
[2, 5, 3, 1, 4]
[2, 3, 5, 1, 4]
[2, 3, 1, 5, 4]
[2, 3, 1, 4, 5]
2회전
[2, 3, 1, 4, 5]
[2, 1, 3, 4, 5]
[2, 1, 3, 4, 5]
[2, 1, 3, 4, 5]
3회전
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
4회전
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

※ 참고 : “뒤에서 부터 확정 된다.”라는 내용을 추가한다면 반복횟수가 16회가 아닌 10회가 된다.

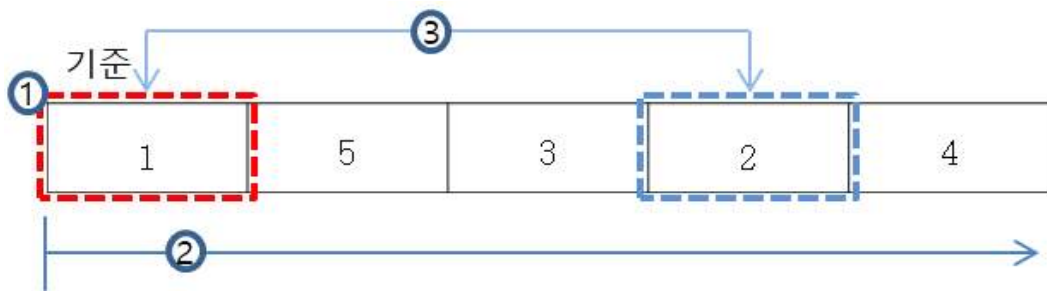
· 선택정렬(Select Sort)

정렬의 대상에서 최솟값을 찾아서 가장 앞의 내용과 교체하는 방식이다. 오름차순정렬의 경우 한 회전이 완료되었을 때 가장 작은 값이 앞쪽에 확정이 되게 된다.

아래와 같은 5개의 숫자가 섞여 있을 때 선택정렬을 이용하여 정렬을 진행하여 보자.

5	2	3	1	4
---	---	---	---	---

- 1) 0번 index를 기준이 된다.
- 2) 기준 이상의 요소들에서 최솟값을 가지고 있는 index를 찾는다.
- 3) 기준 index와 최솟값을 가지고 있는 index의 값을 바꾼다.



위의 순서대로 진행하였을 때 한 회전의 진행이 완료된다. 한 회전에 완료된 후에는 가장 작은 값이 가장 앞에 확정이 된다. 2회전 때에는 기준이 1번 index에서 시작하게 된다. 앞에서부터 하나씩 확정되어 때문에 회전수는 항상 '정렬할 숫자의 개수 -1'만큼 회전을 진행하게 된다.

[문제 2-28] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_28{
    public static void main(String[] args) {
        //1. 5개의 숫자 5,2,3,1,4를 오름차순으로 정렬하는 코드를 작성하여라.
        // 단. 선택정렬을 이용하여라.

    }
}
```



## ② 다차원 배열

지금까지 우리가 배운 배열은 1차원 배열인데 자바에서는 1차원 배열뿐만 아니라 2차원 이상의 다차원 배열도 허용한다. 메모리 용량이 허용하는 한, 차원의 제한은 없지만, 주로 1, 2차원 배열이 사용되므로 2차원 배열만 잘 이해하고 나면 3차원 이상의 배열도 어렵지 않게 다룰 수 있으므로 2차원 배열에 대해서 중점적으로 배울 것이다.

### 1. 2차원 배열의 선언 및 생성

#### (1) 2차원 배열의 선언

이차원 배열을 선언하는 방법은 1차원 배열과 같다. 다만 괄호[]가 하나더 들어갈 뿐이다.

선언 방법	선언 예
타입[][] 변수이름;	int[][] score;
타입[] 변수이름[];	int[] score[];
타입 변수이름[][];	int score[][];

#### (2) 2차원 배열의 생성

2차원 배열은 주로 테이블 형태의 데이터를 담는데 사용되며, 만일 3행 4열의 데이터를 담기 위한 배열을 생성하려면 다음과 같이 한다.

```
int[][] score = new int[3][4];    // 3행 4열의 2차원 배열을 생성한다.
```

위 문장이 수행되면 아래의 그림처럼 4행 3열의 데이터, 모두 12개의 int값을 저장할 수 있는 공간이 마련된다.

	4열			
3행	int	int	int	int
	int	int	int	int
	int	int	int	int

위의 그림에서는 각 요소, 즉 저장공간의 타입을 적어놓은 것이고, 실제로는 배열요소의 타입인 int의 기본값인 0이 저장된다. 배열을 생성하면, 배열의 각 요소에는 배열요소 타입의 기본값이 자동적으로 저장된다.

## 2. 2차원 배열의 index

2차원 배열은 행(row)과 열(column)로 구성되어 있기 때문에 index도 행과 열에 각각 하나씩 존재한다. '행index'의 범위는 '0 ~행의 길이- 1'이고 '열index'의 범위는 '0 ~열의 길이- 1'이다. 그리고 2차원 배열의 각 요소에 접근하는 방법은 '배열이름[행index][열index]'이다.

만일 다음과 같이 배열 score를 생성하면, score[0][0]부터 score[2][3]까지 모두 12개(3×4=12)의 int값을 저장할 수 있는 공간이 마련되고, 각 배열 요소에 접근할 수 있는 방법은 아래의 그림과 같다.

```
int[][] score = new int[3][4]; // 3행 4열의 2차원 배열 score를 생성
```

		4열			
3행	[0]	[0]	[1]	[2]	[3]
	[1]	[0]	[1]	[2]	[3]
	[2]	[0]	[1]	[2]	[3]

배열 score의 1행 1열에 100을 저장하고, 이 값을 출력하려면 다음과 같이 하면 된다.

```
score[0][0] = 100; // 배열 score의 1행 1열에 100을 저장
```

```
System.out.println(score[0][0]); // 배열 score의 1행 1열의 값을 출력
```

## 3. 2차원 배열의 초기화

2차원 배열도 괄호 를 사용해서 생성과 초기화를 동시에 할 수 있다. 다만, 1차원 배열보다 괄호 를 한번 더 써서 행별로 구분해 준다.

```
int[][] arr = new int[][]{ {1, 2, 3}, {4, 5, 6} };
```

```
int[][] arr = { {1, 2, 3}, {4, 5, 6} }; // new int[][]가 생략됨
```

크기가 작은 배열은 위와 같이 간단히 한 줄로 써주는 것도 좋지만, 가능하면 다음과 같이 행별로 줄 바꿈을 해주는 것이 보기도 좋고 이해하기 쉽다.

```
int[][] arr = {  
    {1, 2, 3},  
    {4, 5, 6}  
};
```

※ 앞서 배열의 길이에 대해서 배웠듯이 2차원 배열에서 길이가 있다.

arr.length => 2 : 행의 길이

arr[0].length => 3 : 0번째 행의 배열에 저장된 요소의 수량

[예제 2-22] 2차원 배열 다루기

```
class Ex2_22 {  
    public static void main(String[] args){
```

	국어	영어	수학
1	100	70	30
2	45	76	88
3	37	45	28
4	47	89	100

//1. 학생 4명의 3가지 과목의 점수를 위와 같이 획득 하였다.

// 데이터를 저장할 수 있는 2차원 배열의 생성 및 초기화

```
int[][] score = new int[][]{ // 생성 및 초기화가 동시에 진행시 크기를 기입하지 않는다.  
    {100, 70, 30},  
    {45, 76, 88},  
    {37, 45, 28},  
    {47, 89, 100}, // 마지막 ','는 영향을 주지 않는다.  
};
```

//2. 1번에서 생성된 배열을 이용하여 위의 모양과 같이 출력

```
System.out.println("\t국어\t영어\t수학"); // 상단의 과목을 출력  
for(int hang = 0; hang < score.length; hang++){  
    System.out.print(hang+1); // 번호를 출력 후 이어서 출력하기 위해 print()사용  
    for(int yeol = 0; yeol < score[hang].length; yeol++){  
        System.out.print("\t"+score[hang][yeol]); // 학생의 과목들을 출력  
    }  
    System.out.println(); // 한명의 점수를 출력 후 줄 바꿈을 위해 println()사용  
}  
}
```

[문제 2-29] 주석에 적당한 코드를 작성해 주세요.

```
class Qu2_29{
    public static void main(String[] args) {
        //1. 6명의 학생이름을 저장할 수 있는 변수 names를 선언 및 생성하고 주변 친구들의
        //   이름으로 초기화 하여라.

        //2. 7과목의 점수(정수)를 저장할 수 있는 변수 subjects를 선언 및 생성하고 국어,
        //   영어, 수학, 사회, 과학, Java, Oracle로 초기화 하여라.

        //3. 6명의 7과목의 점수(정수)를 저장할 수 있는 변수 score를 선언 및 생성 하여라.
        //   단. names와 subjects의 길이를 이용하여라.

        //4. score의 모든 요소를 0~100사이의 임의의 값(정수)을 저장 하여라.

        //5. 학생별 합계를 저장할 수 있는 변수 nameSum을 선언 및 생성 하여라.

        //6. nameSum의 요소에 훈련생별 합계를 저장하여라.

        //7. 학생별 평균을 저장할 수 있는 float타입의 변수 nameAvg를 선언 및 생성 하여라.

        //8. nameAvg의 요소에 훈련생별 평균을 저장 하여라.
        //   단. 평균은 소수점 세 번째 자리에서 반올림하여 두 번째 자리까지 표현하여라.

        //9. 과목별 합계를 저장할 수 있는 변수 subSum을 선언 및 생성 하여라.

        //10. subSum의 요소에 과목별 합계를 저장하여라.

        //11. 과목별 평균을 저장할 수 있는 float타입의 변수 subAvg를 선언 및 생성 하여라.
```

```
//12. nameAvg의 요소에 과목별 평균을 저장 하여라.
// 단. 평균은 소수점 세 번째 자리에서 반올림하여 두 번째 자리까지 표현하여라.
```

```
//13. 훈련생별 석차를 저장할 수 있는 변수 rank를 선언 및 생성 하여라.
```

```
//14. rank의 요소에 합계를 기준으로 훈련생별 석차를 저장 하여라.
```

```
//15. 위에서 생성된 변수들을 이용하여 아래와 같이 출력하여라.
```

```
// hidden. 아래의 결과를 합계를 기준으로 내림차순정렬을 하여라.
```

```
}
```

```
}
```

결과										
	국어	영어	수학	사회	과학	Java	Oracle	합계	평균	석차
1번	61	49	42	61	39	50	88	390	55.7	1
2번	91	10	42	73	48	24	81	369	52.7	2
3번	25	11	17	96	96	45	31	321	45.9	3
4번	27	34	3	99	44	37	72	316	45.1	6
5번	2	40	61	100	24	19	71	317	45.3	5
6번	7	18	20	94	93	53	35	320	45.7	4

과목들의 값으로 임의의 값을 이용하기 때문에 결과의 차이가 있을 수 있다. 형태만 똑같이 유지할 수 있도록 하자.

### ③ 연습문제

[5-1] 다음은 배열을 선언하거나 초기화한 것이다. 잘못된 것을 고르고 그 이유를 설명하시오.

- ① `int[] arr[];`
- ② `int[] arr = {1,2,3};`
- ③ `int[] arr = new int[5];`
- ④ `int[] arr = new int[5]{1,2,3,4,5};`
- ⑤ `int arr[5];`
- ⑥ `int[] arr[] = new int[3][];`

[5-2] 다음과 같은 배열이 있을 때,

- (1) `arr[3].length`의 값은 얼마인가?
- (2) `arr.length`의 값은 얼마인가?
- (3) `System.out.println(arr[4][1])`의 출력 결과는 얼마인가?

```
int[][] arr = {  
    { 5, 5, 5, 5, 5},  
    { 10, 10, 10},  
    { 20, 20, 20, 20},  
    { 30, 30}  
};
```

[5-3] 배열 `arr`에 담긴 모든 값을 더하는 프로그램을 완성하시오.

```
class Exercise5_3 {  
    public static void main(String[] args) {  
        int[] arr = {10, 20, 30, 40, 50};  
        int sum = 0;  
        /*  
  
            (1)  
  
        */  
        System.out.println("Sum = "+sum);  
    }  
}
```

결과 Sum = 150

[5-4] 2차원 배열 arr에 담긴 모든 값의 총합과 평균을 구하는 프로그램을 완성하시오.

```
class Exercise5_4 {
    public static void main(String[] args) {
        int[][] arr = {
            { 5, 8, 13, 5, 2 },
            { 22, 13, 28 },
            { 2, 18, 23, 62 }
        };

        int total = 0; // 합계를 저장하기 위한 변수
        float average = 0; // 평균을 저장하기 위한 변수
        /*

            (1)

        */

        System.out.println("total = " + total);
        System.out.println("Average = " + average);
    }
}
```

결과	total = 201
	Average = 16.75

[5-5] 거스름돈을 몇 개의 동전으로 지불할 수 있는지를 계산하는 문제이다. 변수 money의 금액을 동전으로 바꾸었을 때 각각 몇 개의 동전이 필요한지 계산해서 출력하라. 단, 가능한 한 적은 수의 동전으로 거슬러 주어야한다. (1)에 알맞은 코드를 넣어서 프로그램을 완성하시오.

```
class Exercise5_5 {
    public static void main(String[] args) {
        int[] coinUnit = { 500, 100, 50, 10 };
        int money = 2790;
        /*

            (1)

        */

    }
}
```

결과	500원 : 5개
	100원 : 2개
	50원 : 1개
	10원 : 4개

[5-6] 다음은 1과 9사이의 중복되지 않은 숫자로 이루어진 3자리 숫자를 만들어내는 프로그램이다.  
 ※ 임의의 값을 사용했기 때문에 실행결과와 다를 수 있다.

```
class Exercise5_6 {
    public static void main(String[] args) {
        int[] ballArr = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        int[] ball3 = new int[3];

        // ballArr의 index순서대로 index의 요소와 임의의 요소를 골라서 값을 바꾼다.

        // ballArr의 앞에서 3개를 ball3로 복사한다.

        // ball3의 값을 출력한다.

    }
}
```

결과	6
	8
	2

[5-7] 다음은 배열 answer에 담긴 데이터를 읽고 각 숫자의 개수를 세어서 개수만큼 '\*'을 찍어서 그래프를 그리는 프로그램이다.

```
class Exercise5_7 {
    public static void main(String[] args) {
        int[] answer = { 1, 4, 3, 2, 1, 2, 3, 2, 1, 4 };
        int[] counter = new int[4];

        // answer의 요소들 중 1이면 counter의 0번방을 증가, 2이면 counter의 1번방을 증가
        // 3이면 counter의 2번방을 증가, 4이면 counter의 3번방을 증가.

        // counter에 저장된 요소를 결과와 같이 출력 하여라.

        // hide : answer의 값의 범위를 측정하여 최대 5개의 통계 만들기

    }
}
```

결과	1 : 3개 ***
	2 : 2개 **
	3 : 2개 **
	4 : 2개 **



[5-8] 문제 5-5에 동전의 개수를 추가한 프로그램이다. 커맨드라인으로부터 거슬러 줄 금액을 입력받아 계산한다. 보유한 동전의 개수로 거스름돈을 지불할 수 없으면, '거스름돈이 부족합니다.'라고 출력하고 종료한다. 지불할 돈이 충분히 있으면, 거스름돈을 지불한 만큼 가진 돈에서 빼고 남은 동전의 개수를 화면에 출력한다. (1)에 알맞은 코드를 넣어서 프로그램을 완성하시오.

```
class Exercise5_8 {
    public static void main(String[] args) {
        if(args.length!=1){
            System.out.println("한개의 숫자만 입력해 주세요.");
            System.exit(0);
        }
        // 문자열을 숫자로 변환한다. 입력한 값이 숫자가 아닐 경우 예외가 발생한다.
        int money = Integer.parseInt(args[0]);
        System.out.println("money="+money);
        int[] coinUnit = { 500, 100, 50, 10 }; // 동전의 단위
        int[] coin = { 5, 5, 5, 5 }; // 단위별 동전의 개수

        for(int i=0;i<coinUnit.length;i++) {
            int coinNum = 0;
            /* (1) 아래의 로직에 맞게 코드를 작성하시오.
            1. 금액(money)을 동전단위로 나눠서 필요한 동전의 개수(coinNum)를 구한다.
            2. 배열 coin에서 coinNum만큼의 동전을 뺀다.
            (만일 충분한 동전이 없다면 배열 coin에 있는 만큼만 뺀다.)
            3. 금액에서 동전의 개수(coinNum)와 동전단위를 곱한 값을 뺀다.
            */
            System.out.println(coinUnit[i]+"원: "+coinNum);
        }

        if(money > 0) {
            System.out.println("거스름돈이 부족합니다.");
            System.exit(0); // 프로그램을 종료한다.
        }

        System.out.println("=남은 동전의 개수 =");
        for(int i=0;i<coinUnit.length;i++) {
            System.out.println(coinUnit[i]+"원:"+coin[i]);
        }
    }
}
```

[5-9] 주어진 배열을 시계방향으로 90도 회전시켜서 출력하는 프로그램을 완성 하시오.

```
class Exercise5_9 {
    public static void main(String[] args) {
        char[][] star = {
            {'*', '*', ' ', ' ', ' ', ' '},
            {'*', '*', ' ', ' ', ' ', ' '},
            {'*', '*', '*', '*', '*', '*'},
            {'*', '*', '*', '*', '*', '*'}
        };

        char[][] result = new char[star[0].length][star.length];

        for(int i=0; i < star.length;i++) {
            for(int j=0; j < star[i].length;j++) {
                System.out.print(star[i][j]);
            }
            System.out.println();
        }
        System.out.println();

        for(int i=0; i < star.length;i++) {
            for(int j=0; j < star[i].length;j++) {
                /*
                 (1) 알맞은 코드를 넣어 완성하시오.
                */
            }
        }

        for(int i=0; i < result.length;i++) {
            for(int j=0; j < result[i].length;j++) {
                System.out.print(result[i][j]);
            }
            System.out.println();
        }
    }
}
```

결과

```

**
**
*****
*****
****
****
**
**
**
```

## 학습 3 객체지향 프로그래밍

### 3-1. 클래스와 객체

#### 학습목표

- 프로그래밍 언어 Java의 특성을 파악하고 설명할 수 있다.
- Java프로그래밍 언어의 특성을 적용하여 애플리케이션을 구현할 수 있다.

#### 필요 지식 /

#### ① 객체지향 언어

객체지향언어는 기존의 프로그래밍언어와 다른 전혀 새로운 것이 아니라, 기존의 프로그래밍 언어에 몇 가지 새로운 규칙을 추가한 보다 발전된 형태의 것이다. 이러한 규칙들을 이용해서 코드 간에 서로 관계를 맺어 줌으로써 보다 유기적으로 프로그램을 구성하는 것이 가능해졌다. 기존의 프로그래밍 언어에 익숙한 사람이라면 자바의 객체지향적인 부분만 새로 배우면 된다. 다만 절차적 언어에 익숙한 프로그래밍 습관을 객체지향적으로 바꾸도록 노력해야 할 것이다. 객체지향언어의 주요 특징은 다음과 같다.

##### 1. 코드의 재사용성이 높다.

새로운 코드를 작성할 때 기존의 코드를 이용하여 쉽게 작성할 수 있다.

##### 2. 코드의 관리가 용이하다.

코드간의 관계를 이용해서 적은 노력으로 쉽게 코드를 변경할 수 있다.

##### 3. 신뢰성이 높은 프로그래밍을 가능하게 한다.

제어자와 메서드를 이용해서 데이터를 보호하고 올바른 값을 유지하도록 하며, 코드의 중복을 제거하여 코드의 불일치로 인한 오동작을 방지할 수 있다.

객체지향언어의 가장 큰 장점은 '코드의 재사용성이 높고 유지보수가 용이하다.'는 것이다. 이러한 객체지향언어의 장점은 프로그램의 개발과 유지보수에 드는 시간과 비용을 획기적으로 개선하였다.

앞으로 상속, 다형성과 같은 객체지향개념을 학습할 때 재사용성과 유지보수 그리고 중복된 코드의 제거, 이 세 가지 관점에서 보면 보다 쉽게 이해할 수 있을 것이다.

객체지향 프로그래밍은 프로그래머에게 거시적 관점에서 설계할 수 있는 능력을 요구하기 때문에 객체지향개념을 이해했다 하더라도 자바의 객체지향적 장점들을 충분히 활용한 프로

그램을 작성하기란 쉽지 않을 것이다.

너무 객체지향개념에 얽매어서 고민하기 보다는 일단 프로그램을 기능적으로 완성한 다음 어떻게 하면 보다 객체지향적으로 코드를 개선할 수 있을지를 고민하여 점차 개선해 나가는 것이 좋다.

이러한 경험들이 축적되어야 프로그램을 객체지향적으로 설계할 수 있는 능력이 길러지는 것이지 처음부터 이론을 많이 안다고 해서 좋은 설계를 할 수 있는 것은 아니다.

## ② 클래스와 객체

### 1. 클래스와 객체의 의미

객체의 사전적인 정의는, '실제로 존재하는 것'이다. 우리가 주변에서 볼 수 있는 책상, 의자, 자동차와 같은 사물들이 곧 객체이다. 객체지향이론에서는 사물과 같은 유형적인 것뿐만 아니라, 개념이나 논리와 같은 무형적인 것들도 객체로 간주한다.

<b>객체의 정의</b>	실제로 존재하는 것, 사물 또는 개념
<b>객체의 용도</b>	객체가 가지고 있는 기능과 속성에 따라 다름
<b>유형의 객체</b>	책상, 의자, 자동차, TV와 같은 사물
<b>무형의 객체</b>	수학공식, 프로그램 에러와 같은 논리나 개념

클래스란 '객체를 정의해놓은 것.' 또는 클래스는 '객체의 설계도 또는 틀'이라고 정의할 수 있다. 클래스는 객체를 생성하는데 사용되며, 객체는 클래스에 정의된 대로 생성된다.

<b>클래스의 정의</b>	클래스란 객체를 정의해 놓은 것
<b>클래스의 용도</b>	클래스는 객체를 생성하는데 사용

클래스와 객체의 관계를 우리가 살고 있는 실생활에서 예를 들면, 제품 설계도와 제품과의 관계라고 할 수 있다. 예를 들면, TV설계도(클래스)는 TV라는 제품(객체)을 정의한 것이며, TV(객체)를 만드는데 사용된다.

또한 클래스는 단지 객체를 생성하는데 사용될 뿐, 객체 그 자체는 아니다. 우리가 원하는 기능의 객체를 사용하기 위해서는 먼저 클래스로부터 객체를 생성하는 과정이 선행되어야 한다.

우리가 TV를 보기 위해서는, TV(객체)가 필요한 것이지 TV설계도(클래스)가 필요한 것은 아니며, TV설계도(클래스)는 단지 TV라는 제품(객체)을 만드는 데만 사용될 뿐이다.

그리고 TV설계도를 통해 TV가 만들어진 후에야 사용할 수 있다. 프로그래밍에서는 먼저 클래스를 작성한 다음, 클래스로부터 객체를 생성하여 사용한다.

## 2. 객체의 구성요소

객체는 속성과 기능, 두 종류의 구성요소로 이루어져 있으며, 일반적으로 객체는 다수의 속성과 다수의 기능을 갖는다. 즉, 객체는 속성과 기능의 집합이라고 할 수 있다. 그리고 객체가 가지고 있는 속성과 기능을 그 객체의 멤버(구성원, member)라 한다.

클래스란 객체를 정의한 것이므로 클래스에는 객체의 모든 속성과 기능이 정의되어 있다. 클래스로부터 객체를 생성하면, 클래스에 정의된 속성과 기능을 가진 객체가 만들어지는 것이다.

보다 쉽게 이해할 수 있도록 TV를 예로 들어보자. TV의 속성으로는 전원상태, 크기, 길이, 높이, 색상, 볼륨, 채널과 같은 것들이 있으며, 기능으로는 켜기, 끄기, 볼륨 높이기, 볼륨 낮추기, 채널 변경하기 등

속성	크기, 길이, 높이, 색상, 볼륨, 채널 등
기능	켜기, 끄기, 볼륨 높이기, 볼륨 낮추기, 채널 변경하기 등



위에서 분석한 내용을 토대로 Tv클래스를 만들면 다음과 같다.

### [예제 3-1] Tv클래스 정의하기

```
class Tv {
    String color;    // 색상
    boolean power;   // 전원상태
    int channel;     // 채널

    void power(){    // 전원을 켜고 끄는 메서드
        power = !power;
    }
    void channelUp(){ // 채널을 하나 증가시키는 메서드
        channel++;
    }
    void channelDown(){ // 채널을 하나 감소시키는 메서드
        channel--;
    }
}
```

실제 TV가 갖는 기능과 속성은 이 외에도 더 있지만, 프로그래밍에 필요한 속성과 기능만을

선택하여 클래스를 작성하면 된다.

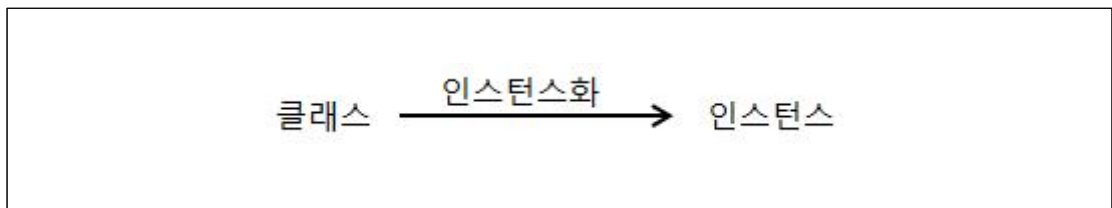
### 3. 객체와 인스턴스

클래스로부터 객체를 만드는 과정을 클래스의 인스턴스화(instantiate)라고 하며, 어떤 클래스로부터 만들어진 객체를 그 클래스의 인스턴스(instance)라고 한다.

예를 들면, Tv클래스로부터 만들어진 객체를 Tv클래스의 인스턴스라고 한다. 결국 인스턴스는 객체와 같은 의미이지만, 객체는 모든 인스턴스를 대표하는 포괄적인 의미를 갖고 있으며, 인스턴스는 어떤 클래스로부터 만들어진 것인지를 강조하는 보다 구체적인 의미를 갖고 있다.

예를 들면, '책상은 인스턴스다.'라고 하기 보다는 '책상은 객체다.'라는 쪽이, '책상은 책상 클래스의 객체이다.'라고 하기 보다는 '책상은 책상 클래스의 인스턴스다.'라고 하는 것이 더 자연스럽다.

인스턴스와 객체는 같은 의미이므로 두 용어의 사용을 엄격히 구분할 필요는 없지만, 위의 예에서 본 것과 같이 문맥에 따라 구별하여 사용하는 것이 좋다.



### 4. 인스턴스의 생성과 사용

Tv클래스를 선언한 것은 Tv설계도를 작성한 것에 불과하므로, Tv인스턴스를 생성해야 제품(Tv)을 사용할 수 있다. 클래스로부터 인스턴스를 생성하는 방법은 여러 가지가 있지만 일반적으로는 다음과 같이 한다.

```
클래스명 변수명;           // 클래스의 객체를 참조하기 위한 참조변수를 선언
변수명 = new 클래스명(); // 클래스의 객체를 생성 후, 객체의 주소를 참조변수에 저장
클래스명 변수명 = new 클래스명();

Tv t;                     // Tv클래스 타입의 참조변수 t를 선언
t = new Tv();             // Tv인스턴스를 생성한 후, 생성된 Tv인스턴스의 주소를 t에 저장
Tv t = new Tv();
```

[예제 3-2] [예제 3-1]에서 만든 클래스의 인스턴스 생성 및 사용

```

class Ex3_2 {
    public static void main(String[] args) {
        Tv t;           // Tv인스턴스를 참조하기 위한 변수 t를 선언
        t = new Tv();    // Tv인스턴스를 생성 후 참조변수 t에 저장
        t.channel = 7;   // Tv인스턴스의 멤버변수 channel의 값을 7로 변경
        t.channelDown(); // Tv인스턴스의 메서드 channelDown()를 호출
        System.out.println("현재 채널은 " + t.channel + "입니다.");
    }
}

```

이 예제는 Tv클래스로부터 인스턴스를 생성하고 인스턴스의 속성(channel)과 메서드(channelDown( ))를 사용하는 방법을 보여 주는 것이다.

#### (1) Tv t;

Tv클래스 타입의 참조변수 t를 선언한다. 메모리에 참조변수 t를 위한 공간이 마련된다. 아직 인스턴스가 생성되지 않았으므로 이 참조변수로 할 수 있는 것은 아무것도 없다.

#### (2) t = new Tv();

연산자 new에 의해 Tv클래스의 인스턴스가 메모리의 빈 공간에 생성된다. 주소가 0x100인 곳에 생성되었다고 가정하자. 이 때, 멤버변수는 각 자료형에 해당하는 기본값으로 초기화 된다.

color는 참조형이므로 null로, power는 boolean이므로 false로, 그리고 channel은 int이므로 0으로 초기화 된다.

#### (3) t.channel = 7 ;

참조변수 t에 저장된 주소에 있는 인스턴스의 멤버변수 channel에 7을 저장한다. 여기서 알 수 있는 것처럼, 인스턴스의 멤버변수 (속성) 를 사용하려면 '참조변수.멤버변수'와 같이 하면 된다.

#### (4) t.channelDown();

참조변수 t가 참조하고 있는 Tv인스턴스의 channelDown메서드를 호출한다. channel Down메서드는 멤버변수 channel에 저장되어 있는 값을 1 감소시킨다.

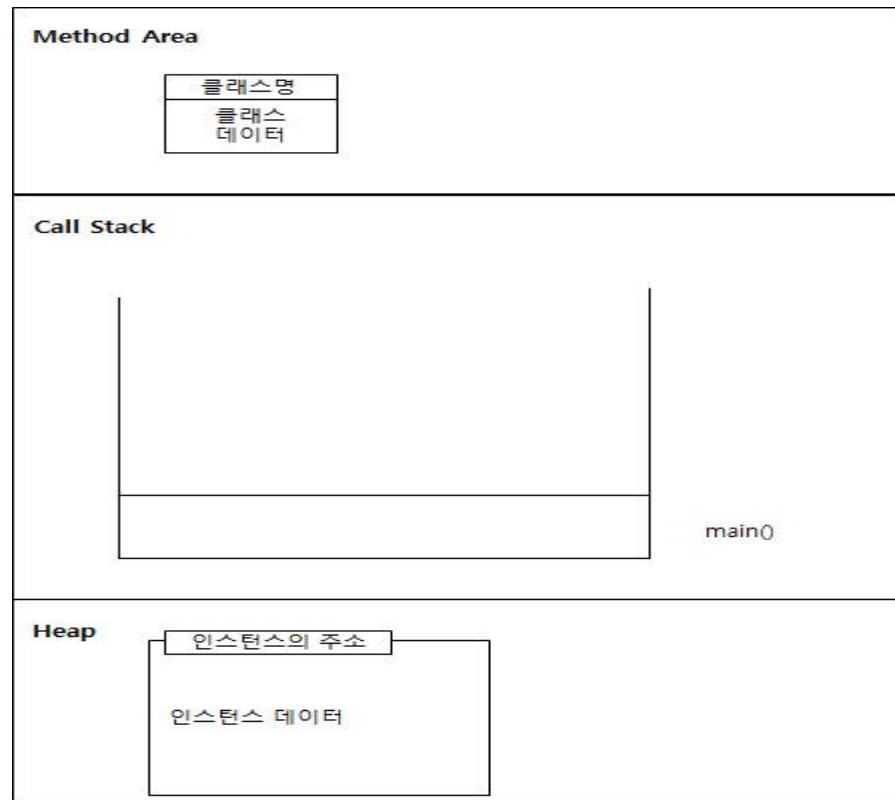
channelDown( )에 의해서 channel의 값은 7에서 6이 된다.

인스턴스는 참조변수를 통해서만 다룰 수 있으며, 참조변수의 타입은 인스턴스의 타입과 일치해야 한다.

#### 4. JVM의 메모리 구조

응용프로그램이 실행되면, JVM은 시스템으로부터 프로그램을 수행하는데 필요한 메모리를 할당받고 JVM은 이 메모리를 용도에 따라 여러 영역으로 나누어 관리 한다.

그 중 3가지 주요 영역(method area, call stack, heap)에 대해서 알아보자.



##### (1) 메서드 영역(Method Area)

프로그램 실행 중 어떤 클래스가 사용되면, JVM은 해당 클래스의 클래스파일(\*.class)을 읽어서 분석하여 클래스에 대한 정보(클래스 데이터)를 이곳에 저장한다. 이 때, 그 클래스의 클래스변수(class variable)도 이 영역에 함께 생성된다.

##### (2) 호출 스택(Call Stack)

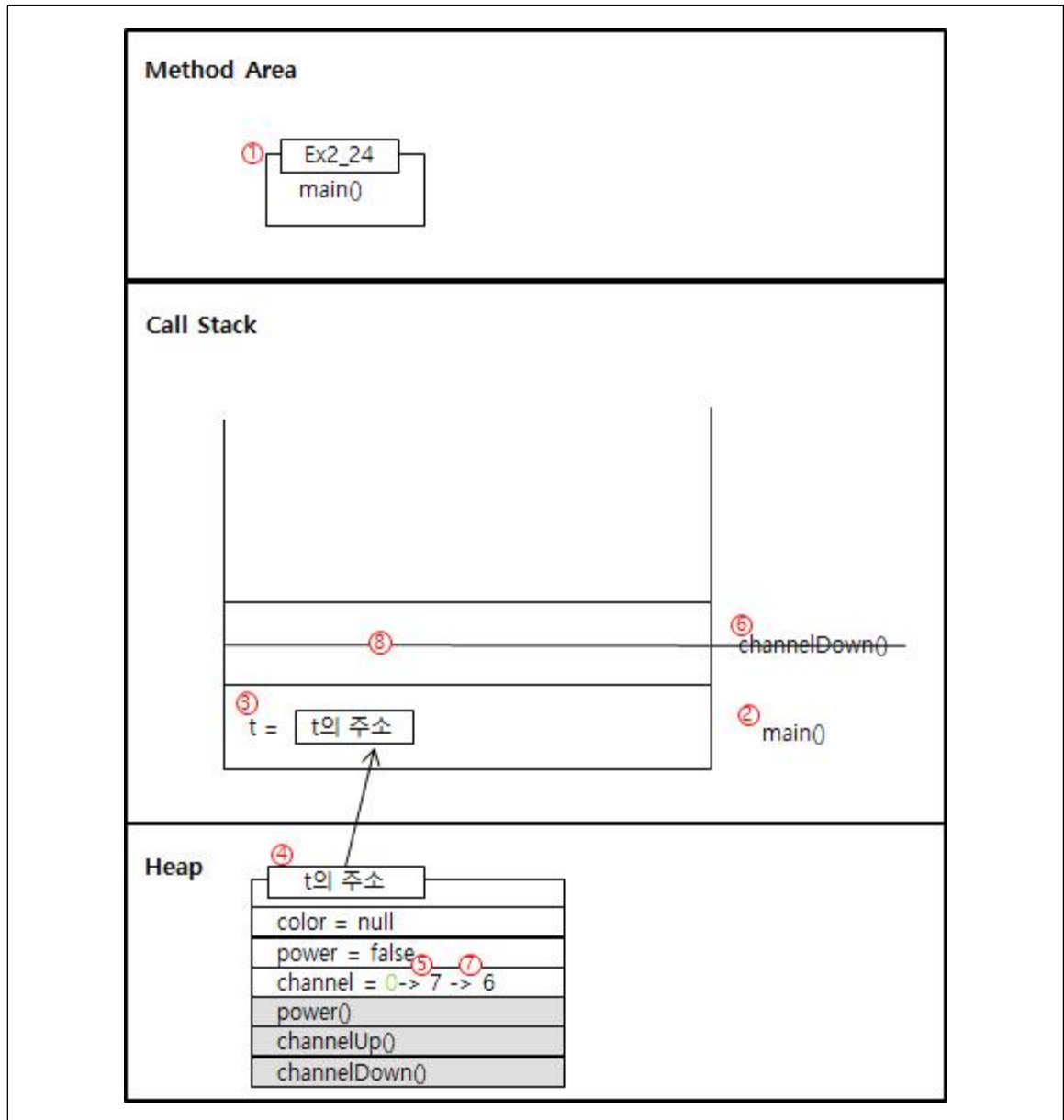
호출스택은 메서드의 작업에 필요한 메모리 공간을 제공한다. 메서드가 호출되면, 호출스택에 호출된 메서드를 위한 메모리가 할당되며, 이 메모리는 메서드가 작업을 수행하는 동안 지역 변수(매개변수 포함)들과 연산의 중간결과 등을 저장하는데 사용된다. 그리고 메서드가 작업을 마치면 할당되었던 메모리공간은 반환되어 비워진다.

##### (3) 힙(Heap)

인스턴스가 생성되는 공간으로 프로그램 실행 중 생성되는 인스턴스는 모두 이곳에 생성된다. 즉, 인스턴스변수(instance variable)들이 생성되는 공간이다.



[예제 3-3] [예제 3-2] 내용을 JVM메모리 구조로 그리기



- ① JVM에 의해서 Ex2\_24클래스가 Method Area에 메모리공간이 할당되고 클래스멤버인 main메서드가 해당 공간에 로드된다.
- ② main메서드가 호출됨으로써 프로그램이 시작된다.
- ③ main메서드의 지역변수 t가 선언이 된다.
- ④ Tv클래스의 인스턴스가 생성되어 Heap영역에 메모리 공간이 할당된다. 인스턴스 멤버중에 인스턴스 변수인 color, power, channel이 로드되고 기본값으로 초기화 되고 인스턴스 메서드인 power(), channelUp(), channelDown()가 같이 로드 된다.
- ⑤ t의 주소에 있는 channel의 값을 7로 변경된다.
- ⑥ t의 주소에 있는 channelDown()가 호출되어 진행되던 main()가 끝난 것은 아니므로 호출

스택에 대기상태로 남아 있고 `channelDown()`가 실행이 된다.

⑦ `channelDown()`가 실행되어 `channel`의 값이 1감소하여 6이 된다.

⑧ `channelDown()`가 수행이 완료되어 호출스택에서 사라지고 `main()`로 되돌아간다.

각 메서드를 위한 메모리상의 작업 공간은 서로 구별되며, 첫 번째로 호출된 메서드를 위한 작업공간이 호출스택의 맨 밑에 마련되고, 첫 번째 메서드 수행 중에 다른 메서드를 호출하면, 첫 번째 메서드의 바로 위에 두 번째로 호출된 메서드를 위한 공간이 마련된다.

이 때 첫 번째 메서드는 수행을 멈추고, 두 번째 메서드가 수행되기 시작한다. 두 번째로 호출된 메서드가 수행을 마치게 되면, 두 번째 메서드를 위해 제공되었던 호출스택의 메모리공간이 반환되며, 첫 번째 메서드는 다시 수행을 계속하게 된다. 첫 번째 메서드가 수행을 마치면, 역시 제공되었던 메모리 공간이 호출스택에서 제거되며 호출스택은 완전히 비워지게 된다. 호출 스택의 제일 상위에 위치하는 메서드가 현재 실행 중인 메서드이며, 나머지는 대기상태에 있게 된다.

따라서, 호출스택을 조사해 보면 메서드 간의 호출관계와 현재 수행중인 메서드가 어느 것인지 알 수 있다. 호출스택의 특징을 정리해보면 다음과 같다.

메서드가 호출되면 수행에 필요한 만큼의 메모리를 스택에 할당받는다.  
메서드가 수행을 마치고나면 사용했던 메모리를 반환하고 스택에서 제거된다.  
호출스택의 제일 위에 있는 메서드가 현재 실행 중인 메서드이다.  
아래에 있는 메서드가 바로 위의 메서드를 호출한 메서드이다.

반환타입(`return type`)이 있는 메서드는 종료되면서 결과값을 자신을 호출한 메서드(`caller`)에게 반환한다. 대기상태에 있던 호출한 메서드(`caller`)는 넘겨받은 반환값으로 수행을 계속 진행하게 된다.

## 3-2. 변수와 메서드

### 학습목표

- 클래스의 구성 요소인 변수인 메서드의 종류를 설명할 수 있다.
- 클래스의 구성 요소인 변수와 메서드를 활용하여 클래스를 구성할 수 있다.

### 필요 지식 / 변수

#### ① 선언위치에 따른 변수의 종류

변수는 클래스변수, 인스턴스변수, 지역변수 모두 세 종류가 있다. 변수의 종류를 결정짓는 중요한 요소는 '변수의 선언 위치'이므로 변수의 종류를 파악하기 위해서는 변수가 어느 영역에 선언되었는지를 확인하는 것이 중요하다. 멤버변수를 제외한 나머지 변수들은 모두 지역변수이며, 멤버변수 중 static이 붙은 것은 클래스변수, 붙지 않은 것은 인스턴스변수이다.

#### [예제 3-4] 변수의 종류

```

class Ex2_26 {
    int iv;           // 인스턴스 변수
    static int cv;    // 클래스 변수

    void method(){
        int lv = 0;  // 지역변수
    }
}

```

클래스영역

메서드영역

변수의 종류	선언위치	생성시기
클래스변수 ( class variable )	클래스 영역	클래스가 메모리에 로드될 때
인스턴스변수 ( instance variable)		인스턴스가 생성되었을 때
지역 변수 ( local variable )	클래스 영역 이외의 영역 ( 메서드, 생성자, 초기화 블록 내부)	변수 선언문이 수행되었을 때

• **인스턴스변수 (instance variable)** - 클래스 영역에 선언되며, 인스턴스를 생성할 때 만들어진다. 그래서 인스턴스 변수의 값을 읽어 오거나 저장하려면 먼저 인스턴스를 생성해야한다. 인스턴스마다 별도의 저장공간을 가지므로 서로 다른 값을 가질 수 있다. 인스턴스마다 고유한 상태를 유지해야하는 속성의 경우, 인스턴스변수로 선언한다.

• **클래스변수 (class variable)** - 클래스 변수를 선언하는 방법은 인스턴스변수 앞에 `static`을 붙이기만 하면 된다. 인스턴스마다 독립적인 저장공간을 갖는 인스턴스변수와는 달리, 클래스변수는 모든 인스턴스가 공통된 저장공간(변수)을 공유하게 된다. 한 클래스의 모든 인스턴스들이 공통적인 값을 유지해야하는 속성의 경우, 클래스변수로 선언해야 한다.

클래스 변수는 인스턴스변수와 달리 인스턴스를 생성하지 않고 언제라도 바로 사용할 수 있다는 특징이 있으며, '클래스이름.클래스변수'와 같은 형식으로 사용한다.

• **지역변수 (local variable)** - 메서드 내에 선언되어 메서드 내에서만 사용 가능하며, 메서드가 종료되면 소멸되어 사용할 수 없게 된다. `for`문 또는 `while`문의 블록 내에 선언된 지역변수는, 지역변수가 선언된 블록 내에서만 사용 가능하며, 블록을 벗어나면 소멸되어 사용할 수 없게 된다. 우리가 6장 이전에 선언한 변수들은 모두 지역변수이다.

## ② 클래스변수와 인스턴스변수

클래스 변수와 인스턴스변수의 차이를 이해하기 위한 기업들에서 많이 사용하는 출입카드를 클래스로 정의해 보자.



[그림 3-1] 출입카드

출입카드 클래스를 작성하기 위해서는 먼저 카드를 분석해서 속성과 기능을 알아 내야한다. 속성으로는 카드의 폭, 높이, 사진, 이름, 직원번호 정도를 생각할 수 있을 것이다.

이 중에서 어떤 속성을 클래스 변수로 선언할 것이며, 또 어떤 속성들은 인스턴스변수로 선언한 것인지 한번 생각해 보자.

속성	사진 이름 직원번호 폭 높이	<pre>class Card {     String photo; // 사진     String name; // 이름     int number; // 직원번호     static int width = 100; // 폭     static int height = 250; // 높이 }</pre>
기능	• • • •	

출입카드를 여러 장 만드는 경우 즉 Car클래스의 인스턴스를 생성하였을 때 폭(width)과 높이(height)는 모든 인스턴스가 공통적으로 같은 값을 유지해야하므로 클래스변수로 선언하였고, 사진(photo), 이름(name), 직원번호(number)는 인스턴스마다 다른 값을 유지해야하므로 인스턴스변수로 선언 하였다.

[문제 3-1] 주석에 적당한 코드를 작성해 주세요.

```
class CardTest {
    public static void main(String[] args) {
        //1. Card클래스의 폭을 출력하여라.

        //2. Card클래스의 높이를 출력하여라.

        //3. Card클래스의 객체를 생성해 주세요. 변수명 : cd1

        //4. 변수 cd1의 이름을 "Daniel"로 변경하여라.

        //5. 변수 cd1의 직원 번호를 "19961210"로 변경하여라.

        //6. Card클래스의 객체를 생성해 주세요. 변수명 : cd2

        //7. 변수 cd2의 이름을 "nayeon"로 변경하여라.

        //8. 변수 cd2의 직원 번호를 "19950922"로 변경하여라.

        //9. 결과의 출력1과 같이 출력하여라.

        //10. Card클래스의 폭을 70으로 변경하여라.

        //11. Card클래스의 높이를 100으로 변경하여라.

        //12. 결과의 출력2와 같이 출력하여라.

    }
}

class Card {
    String name; // 이름
    int number; // 직원번호
    static int width = 100; // 폭
    static int height = 250; // 높이
}
```

결과

출력1  
1번 카드의 이름은 Daniel, 직원번호 19961210,  
폭은 100, 높이는 250이다.  
2번 카드의 이름은 nayeon, 직원번호 19950922,  
폭은 100, 높이는 250이다.

출력2  
1번 카드의 이름은 Daniel, 직원번호 19961210,  
폭은 70, 높이는 100이다.  
2번 카드의 이름은 nayeon, 직원번호 19950922,  
폭은 70, 높이는 100이다.

[문제 3-2] [문제 3-1]을 JVM메모리 구조로 도식화 하여라.

Method Area

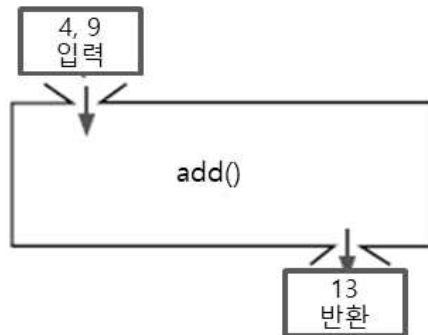
Call Stack

Heap

인스턴스변수는 인스턴스가 생성될 때 마다 생성되므로 인스턴스마다 각기 다른 값을 유지 할 수 있지만, 클래스 변수는 모든 인스턴스가 하나의 저장공간을 공유하므로, 항상 공통된 값을 갖는다.

### ③ 메서드

'메서드(method)'는 특정 작업을 수행하는 일련의 문장들을 하나로 묶은 것이다. 기본적으로 수학의 함수와 유사하며, 어떤 값을 입력하면 이 값으로 작업을 수행해서 결과를 반환한다. 예를 들어 두개의 정수를 입력받아 그 결과를 반환해주는 'add()'를 만들었다면 아래와 같이 도식화 할 수 있다.



[그림 3-2] add메서드 도식화

그저 메서드가 작업을 수행하는데 필요한 값만 넣고 원하는 결과를 얻으면 될 뿐, 이 메서드가 내부적으로 어떤 과정을 거쳐 결과를 만들어내는지 전혀 몰라도 된다. 즉, 메서드에 넣을 값(입력)과 반환하는 결과(출력)만 알면 되는 그래서 메서드를 내부가 보이지 않는 '블랙박스(black box)'라고도 한다.

#### 1. 메서드를 사용하는 이유

##### (1) 높은 재사용성(reuseability)

이미 Java API에서 제공하는 메서드들을 사용하면서 경험한 것처럼 한번 만들어 놓은 메서드는 몇 번이고 호출할 수 있으며, 다른 프로그램에서도 사용이 가능하다.

##### (2) 중복된 코드의 제거

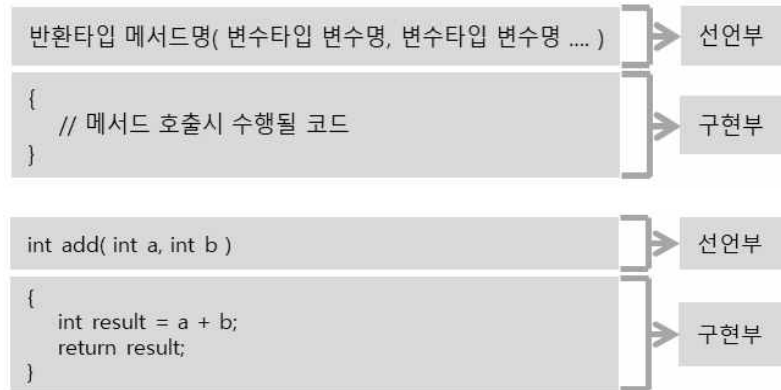
프로그램을 작성하다보면, 같은 내용의 문장들이 여러 곳에 반복해서 나타나곤 한다. 이렇게 반복되는 문장들을 묶어서 하나의 메서드로 작성해 놓으면, 반복되는 문장들 대신 메서드를 호출하는 한 문장으로 대체할 수 있다. 그러면, 전체 소스 코드의 길이도 짧아지고, 변경사항이 발생했을 때 수정해야할 코드의 양이 줄어들어 오류가 발생할 가능성도 함께 줄어든다.

##### (3) 프로그램의 구조화

지금까지는 main메서드 안에 모든 문장을 넣는 식으로 프로그램을 작성해 왔다. 코드의 양이 얼마 되지 않는 작은 프로그램을 작성할 때는 이렇게 해도 별 문제가 없지만, 많은 양의 코드를 작성할 때는 이런 식으로 작성할 수 없다. 큰 규모의 프로그램에서는 문장들을 작업단위로 나눠서 여러 개의 메서드를 담아 프로그램의 구조를 단순화 시키는 것이 필수적이다.

## 2. 메서드의 선언과 구현

메서드는 크게 두 부분, '선언부(header, 머리)'와 '구현부(body, 몸통)'로 이루어져 있다. 메서드를 정의한다는 것은 선언부와 구현부를 작성하는 것을 뜻하며 다음과 같은 형식으로 메서드를 정의한다.



[그림 3-3] 메서드 정의 방식과 add메서드 정의

### (1) 메서드의 선언부

메서드 선언부는 '메서드의 이름'과 '매개변수 선언', 그리고 '반환타입'으로 구성되어 있으며, 메서드가 작업을 수행하기 위해 어떤 값들을 필요로 하고 작업의 결과로 어떤 타입의 값을 반환하는지에 대한 정보를 제공한다. 예를 들어 [그림 3-3]에 정의된 메서드 add는 두 개의 정수를 입력받아서, 두 값을 더한 결과(int타입의 값)를 반환한다.

메서드의 선언부는 후에 변경사항이 발생하지 않도록 신중히 작성해야한다. 메서드의 선언부를 변경하게 되면, 그 메서드가 호출되는 모든 곳이 함께 변경되어야 하기 때문이다.

#### • 매개변수 선언( parameter declaration )

매개변수는 메서드가 작업을 수행하는데 필요한 값들(입력)을 제공받기 위한 것이며, 필요한 값의 개수만큼 변수를 선언하며 각 변수 간의 구분은 쉼표','를 사용한다. 한 가지 주의할 점은 일반적인 변수선언과 달리 두 변수의 타입이 같아도 변수의 타입을 생략할 수 없다는 것이다.

```
int add(int x, int y) { ... } // OK.
```

```
int add(int x, y) { ... } // 에러. 매개변수 y의 타입이 없다.
```

선언할 수 있는 매개변수의 개수는 거의 제한이 없지만, 만일 입력해야할 값의 개수가 많은 경우에는 배열이나 참조변수를 사용하면 된다. 만일 값을 전혀 입력받을 필요가 없다면 괄호( ) 안에 아무 것도 적지 않는다.



#### · 반환타입( return type )

메서드의 작업수행 결과(출력)인 '반환값(return value)'의 타입을 적는다. 단, 반환값이 없는 경우 반환타입으로 'void'를 적어야한다.

### (2) 메서드의 구현부

메서드의 선언부 다음에 오는 괄호 를 '메서드의 구현부'라고 하는데, 여기에 메서드를 호출했을 때 수행될 문장들을 넣는다. 우리가 그동안 작성해온 문장들은 모두 main메서드의 구현부에 속한 것이었으므로 지금까지 하던 대로만 하면 된다.

#### · return 문

return문은 현재 실행중인 메서드를 종료하고 호출한 메서드로 되돌아간다. 지금까지 반환값이 있을 때만 return문을 썼지만, 원래는 반환값의 유무에 관계없이 모든 메서드에는 적어도 하나의 return문이 있어야 한다. 그런데도 반환타입이 void인 경우, return문 없이도 아무런 문제가 없었던 이유는 컴파일러가 메서드의 마지막에 'return;'을 자동적으로 추가해주었기 때문이다.

```
void printHi() {  
    System.out.println("Hi");  
    return; // 반환타입이 void이므로 생략 가능.  
}
```

그러나 반환타입이 void가 아닌 경우, 즉 반환값이 있는 경우, 반드시 return문이 있어야 한다. return문이 없으면 컴파일 에러(error: missing return statement)가 발생한다.

```
int add(int a, int b) {  
    int result = a + b;  
    return result; // 반환타입이 void가 아니므로 생략 불가.  
}
```

### 3. 메서드의 호출

메서드를 정의했어도 호출되지 않으면 아무 일도 일어나지 않는다. 메서드를 호출해야만 구현부{}의 문장들이 수행된다.

#### · 인자(argument)와 매개변수(parameter)

메서드를 호출할 때 괄호( ) 안에 지정해준 값들을 '인자(argument)' 또는 '인수'라고 하는데, 인자의 개수와 순서는 호출된 메서드에 선언된 매개변수와 일치해야 한다.

그리고 인자는 메서드가 호출되면서 매개변수에 대입되므로, 인자의 타입은 매개변수의 타입과 일치하거나 자동 형변환이 가능한 것이어야 한다.

```
int add( int a, int b ) {  
    int result = a + b;  
    return result;  
}
```

1. `add(3, 8);` // add메서드에 선언된 매개변수의 개수와 타입이 같음. (O)
2. `add(3, 4, 8);` // add메서드에 선언된 매개변수의 타입은 같지만 개수가 다름 (X)
3. `add(7, "9");` // add메서드에 선언된 매개변수의 개수는 같지만 타입이 다름 (X)

### [예제 3-5] 메서드 호출 방법

```
public class Ex3_5 {  
    public static void main(String[] args) {  
        // 클래스메서드 임으로 '클래스명.메서드명()'으로 호출  
        // 반환값이 없기 때문에 변수에 저장하거나 출력 불가  
        Ex.classMethod(5);  
  
        // 인스턴스메서드를 호출하기 위해서는 인스턴스를 먼저 생성해야 한다.  
        Ex e = new Ex();  
  
        // 인스턴스메서드는 '참조변수명.메서드명()'으로 호출  
        // 반환값이 있기 때문에 반환타입과 일치하는 변수에 저장  
        int re = e.instanceMethod(5, 7);  
        System.out.println(re); // 5와 7의 곱인 35를 출력  
    }  
}  
  
class Ex {  
    static void classMethod(int a) {  
        System.out.println("입력된 값은 "+ a +"이다.");  
    }  
  
    int instanceMethod(int a, int b) {  
        int result = a * b;  
        return result;  
    }  
}
```

[문제 3-3] 주석에 적당한 코드를 작성해 주세요.

```
class Qu3_3 {
    public static void main(String[] args) {
        //8. MethodTest클래스의 add01메서드를 호출 하여라.

        //9. MethodTest클래스의 add02메서드를 호출 하여라.

        //10. MethodTest클래스의 add03메서드를 호출 하여라.

        //11. MethodTest클래스의 add04메서드를 호출 하여라.

    }
}

class MethodTest {
    //1. 클래스변수 a를 선언하고 10의 값으로 초기화 하여라.

    //2. 클래스변수 b를 선언하고 20의 값으로 초기화 하여라.

    //3. 인스턴스변수 c를 선언하고 50의 값으로 초기화 하여라.

    //4. 클래스변수 a와 b의 합을 출력하는 클래스메서드 add01를 구현 하여라.

    //5. 클래스변수 a, b, int 타입의 매개변수 c의 합을 반환하는 클래스메서드 add02를
    // 구현 하여라.

    //6. 인스턴스변수 c와 int 타입의 매개변수 c의 합을 반환하는 인스턴스메서드 add03를
    // 구현 하여라.

    //7. 인스턴스변수 c와 0~100사이의 임의의 정수의 합을 출력하는 인스턴스메서드 add04
    // 를 구현 하여라.

}
```

[문제 3-4] [문제 3-3]를 JVM메모리 구조로 도식화 하여라.

Method Area

Call Stack

Heap

[문제 3-5] 계산기를 만드는 프로그램이다. 주석에 적당한 코드를 작성해 주세요.

```
class Qu3_5 {
    public static void main(String[] args) {
        //5. 사용자로부터 정수를 입력 받아 변수 firstNum에 저장 하여라.

        //6. 사용자로부터 부호를 입력 받아 변수 buho에 저장 하여라.

        //7. 사용자로부터 정수를 입력 받아 변수 secondNum에 저장 하여라.

        //8. buho에 저장된 값이 '+'이면 add메서드, '-'이면 subtract메서드
        //   '*'이면 multiply메서드, '/'이면 divide메서드를 호출하고 결과를 출력하여라.
        //   단. buho의 저장된 값이 위의 4가지 경우가 아니라면 '연산종료'를 출력하여라.


    }
}

class Calc {
    //1. 두 개의 int타입 입력받아 두 인자의 합의 결과를 반환하는 인스턴스메서드
    //   add를 구현 하여라.


    //2. 두 개의 int타입 입력받아 앞의 인자에서 뒤의 인자를 뺀 결과를 반환하는
    //   인스턴스메서드 subtract를 구현 하여라.


    //3. 두 개의 int타입 입력받아 두 인자의 곱의 결과를 반환하는 인스턴스메서드
    //   multiply를 구현 하여라.
    //   단. overflow를 고려하여라.


    //4. 두 개의 int타입 입력받아 앞의 인자를 뒤의 인자로 나누기한 결과를 반환하는
    //   인스턴스메서드 divide를 구현 하여라.
    //   단. 결과는 소수점 두 번째 자리에서 반올림하여 첫 번째 자리까지 표현 하여라.


}
```

#### 4. 기본형 매개변수와 참조형 매개변수

자바에서는 메서드를 호출할 때 매개변수로 지정한 값을 메서드의 매개변수에 복사해서 넘겨 준다. 매개변수의 타입이 기본형(primitive type)일 때는 기본형 값이 복사되겠지만, 참조형(reference type)이면 인스턴스의 주소가 복사된다. 메서드의 매개변수를 기본형으로 선언하면 단순히 저장된 값만 얻지만, 참조형으로 선언하면 값이 저장된 곳의 주소를 알 수 있기 때문에 값을 읽어 오는 것은 물론 값을 변경하는 것도 가능하다.

**기본형 매개변수** 변수의 값을 읽기만 할 수 있다. (read only)

**참조형 매개변수** 변수의 값을 읽고 변경할 수 있다. (read & write)

[문제 3-6] 주석에 적당한 코드를 작성해 주세요.

```
class Data {
    int x;
}

public class Qu3_6 {
    public static void main(String[] args) {
        //1. Data클래스의 객체를 생성하여라. 변수명 d1

        //2. 참조변수 d1의 x값을 10으로 변경하여라.

        //3. 참조변수 d1의 x값을 출력하여라.

        //4. 인자값으로 참조변수 d1의 x값을 가지는 change메서드를 호출하여라.

        //5. 참조변수 d1의 x값을 출력하여라.

        //6. Data클래스의 객체를 생성하여라. 변수명 d2

        //7. 참조변수 d2의 x값을 10으로 변경하여라.

        //8. 참조변수 d2의 x값을 출력하여라.

        //9. 인자값으로 참조변수 d2를 가지는 change메서드를 호출하여라.

        //10. 참조변수 d2의 x값을 출력하여라.

    }
}
```

```

static void change(int x) { // 기본형 매개변수
    x = 1000;
    System.out.println("change() 기본형 : "+x);
}

static void change(Data d1) { // 기본형 매개변수
    d1.x = 1000;
    System.out.println("change() 참조형 : "+d1.x);
}
}

```

int타입의 매개변수를 가지는 change메서드는 매개변수 x의 값을 1000으로 변경된 것이다. 즉, 원본이 아닌 복사본이 변경된 것이라 원본에는 아무런 영향을 미치지 못한다. 이처럼 기본형 매개변수는 변수에 저장된 값만 읽을 수만 있을 뿐 변경할 수는 없다.

Data타입의 매개변수를 가지는 change메서드는 매개변수 x에 참조변수 d2가 저장하고 있는 주소를 매개변수 d1에 복사된다. 그래서 main메서드의 d2와 change메서드의 참조변수 d1은 같은 객체를 가리키게 된다. 그래서 매개변수 d1으로 x의 값을 읽고 변경하는 것이 모두 가능하다.

기본형 매개변수는 단순히 값을 복사되게 되고, 참조형 매개변수는 참조변수가 저장하고 있는 주소를 복사하게 된다.

[문제 3-6]의 프로그램을 실행하였을 때 아래와 같은 결과를 가지게 된다.

```

10 // 3번 출력결과
change() 기본형 : 1000 // 4번에서 호출된 change메서드 출력결과
10 // 5번 출력결과
10 // 8번 출력결과
change() 참조형 : 1000 // 9번에서 호출된 change메서드 출력결과
1000 // 10번 출력결과

```

[문제 3-7] [문제 3-6]를 JVM메모리 구조로 도식화 하여라.

Method Area

Call Stack

Heap



## 5. 메서드 간의 호출과 참조

같은 클래스에 속한 멤버들 간에는 별도의 인스턴스를 생성하지 않고도 서로 참조 또는 호출이 가능하다. 단, 클래스멤버가 인스턴스 멤버를 참조 또는 호출하고자 하는 경우에는 인스턴스를 생성해야 한다.

그 이유는 인스턴스 멤버가 존재하는 시점에 클래스 멤버는 항상 존재하지만, 클래스멤버가 존재하는 시점에 인스턴스 멤버가 존재하지 않을 수도 있기 때문이다.

### [예제 3-6] 인스턴스멤버와 클래스멤버간 호출

```
public class Ex3_6 {
    int iv;          // 인스턴스변수
    static int cv;   // 클래스변수

    void instanceMethod() { // 인스턴스메서드
        System.out.println(iv); // 인스턴스변수를 사용할 수 있다.
        System.out.println(cv); // 클래스변수를 사용할 수 있다.
        classMethod();          // 클래스메서드 호출할 수 있다.
    }

    static void classMethod() { // 클래스메서드
        // System.out.println(iv); // 에러 클래스메서드에서 인스턴스변수를 사용할 수 없다.
        System.out.println(cv); // 클래스변수를 사용할 수 있다.
        // instanceMethod();     // 인스턴스메서드를 호출할 수 없다.
    }
}
```

위의 코드는 같은 클래스 내의 인스턴스 메서드와 static메서드 간의 호출에 대해서 설명하고 있다. 같은 클래스 내의 메서드는 서로 객체의 생성이나 참조변수 없이 직접 호출이 가능하지만 클래스메서드는 인스턴스 메서드를 호출할 수 없다.

메서드간의 호출과 마찬가지로 인스턴스 메서드는 인스턴스변수를 사용할 수 있지만, 클래스 메서드는 인스턴스변수를 사용할 수 없다.

## 6. 재귀호출 (recursive call)

메서드의 내부에서 메서드 자신을 다시 호출하는 것을 '재귀호출(recursive call)'이라 하고, 재귀호출하는 메서드를 '재귀 메서드'라 한다.

어떻게 메서드가 자기자신을 호출할 수 있는지 의아하겠지만, 메서드 입장에서는 자기 자신을 호출하는 것과 다른 메서드를 호출하는 것은 차이가 없다. '메서드 호출'이라는 것이 그저 특정 위치에 저장되어 있는 명령들을 수행하는 것일 뿐이기 때문이다.

호출된 메서드는 '값에 의한 호출(call by value)'을 통해, 원래의 값이 아닌 복사된 값으로 작업하기 때문에 호출한 메서드와 관계없이 독립적인 작업수행이 가능하다.

하지만 무한히 자기 자신을 호출하기 때문에 무한 반복에 빠지게 된다. 무한반복문이 조건문과 함께 사용되어야하는 것처럼, 재귀호출도 조건문이 필수적으로 따라다닌다.

반복문은 그저 같은 문장을 반복해서 수행하는 것이지만, 메서드를 호출하는 것은 반복문보다 몇 가지 과정, 예를 들면 매개변수 복사와 종료 후 복귀할 주소저장 등이 추가로 필요하기 때문에 반복문보다 재귀호출의 수행시간이 더 오래 걸린다. 하지만 재귀호출을 사용하는 이유는 재귀호출이 주는 논리적 간결함 때문이다. 몇 겹의 반복문과 조건문으로 복잡하게 작성된 코드가 재귀호출로 작성하면 보다 단순한 구조로 바뀌 수도 있다. 아무리 효율적이라도 알아보기 힘들게 작성하는 것보다 다소 비효율적이더라도 알아보기 쉽게 작성하는 것이 논리적 오류가 발생할 확률도 줄어 들고 나중에 수정하기 좋다.

어떤 작업을 반복적으로 처리해야한다면, 먼저 반복문으로 작성해보고 너무 복잡하면 재귀호출로 간단히 할 수 없는지 고민해볼 필요가 있다. 재귀호출은 비효율적이므로 재귀호출에 드는 비용보다 재귀호출의 간결함이 주는 이득이 충분히 큰 경우에만 사용해야 한다.

재귀호출의 대표적인 예는 팩토리얼(factorial)을 구하는 것이다. 팩토리얼은 한 숫자가 1이 될 때까지 1씩 감소시켜가면서 계속해서 곱해 나가는 것인데,  $n!$ ( $n$ 은 양의 정수)과 같이 표현한다. 예를 들어 '5! :  $5*4*3*2*1$ '로 120의 결과가 도출된다.

#### [예제 3-7] 재귀호출을 이용한 팩토리얼 구하기

```
public class Ex3_7 {
    public static void main(String[] args){
        int result = factorial(4);
        System.out.println(result);
    }

    static int factorial(int n){
        int result = 0;
        if(n==1){
            result = 1;
        } else {
            result = n * factorial(n-1);
        }
        return result;
    }
}
```

[문제 3-8] [예제 3-7]를 JVM메모리 구조로 도식화 하여라.

Method Area

Call Stack

Heap

[문제 3-9] 주사위 두개를 이용한 이동거리 구하기

```
public class Qu3_9 {  
    public static void main(String[] args) {  
        //6. doubleDice메서드를 호출 하여라.  
  
        //7. 총 이동거리를 출력하여라.  
        // 예를 들어 주사위 값이 (3,3)이 나오게 되면 한 번 더 던질 수 있는 기회가 주어  
        // 지며 두 번째 던진 주사위 값이 (5,2)가 되었다면 총 이동거리는 13이 된다.  
  
    }  
  
    static int doubleDice(){  
        //1. 첫 번째 주사위 : 1~6사이의 임의의 정수를 변수 firstDice에 저장하여라.  
  
        //2. 두 번째 주사위 : 1~6사이의 임의의 정수를 변수 secondDice에 저장하여라.  
  
        //3. 두 주사위의 합을 변수 result에 저장하여라.  
  
        //4. 만일 두 주사위의 값이 같다면 주사위 두개를 한 번 더 던지도록 하여라.  
  
        //5. 전체 주사위 값의 합을 반환 하여라.  
  
    }  
}
```

## 3-3. 오버로딩

### 학습목표

- 오버로딩의 의미를 이해하고 설명할 수 있다.

### 필요 지식 / 메서드

#### ① 오버로딩(overloading)

메서드도 변수와 마찬가지로 같은 클래스 내에서 서로 구별될 수 있어야 하기 때문에 각기 다른 이름을 가져야 한다. 그러나 자바에서는 한 클래스 내에 이미 사용하려는 이름과 같은 이름을 가진 메서드가 있더라도 매개변수의 개수 또는 타입이 다르면, 같은 이름을 사용해서 메서드를 정의할 수 있다.

이처럼, 한 클래스 내에 같은 이름의 메서드를 여러 개 정의하는 것을 '메서드 오버로딩 (method overloading)' 또는 간단히 '오버로딩(overloading)'이라 한다.

같은 이름의 메서드를 정의한다고 해서 무조건 오버로딩인 것은 아니다. 오버로딩이 성립하기 위해서는 다음과 같은 조건을 만족해야한다.

1. 메서드 이름이 같아야 한다.
2. 매개변수의 개수 또는 타입이 달라야 한다.
3. 반환타입은 영향을 주지 않는다.

비록 메서드의 이름이 같아 하더라도 매개변수가 다르면 서로 구별될 수 있기 때문에 오버로딩이 가능한 것이다. 위의 조건을 만족시키지 못하는 메서드는 중복 정의로 간주되어 컴파일 시에 에러가 발생한다. 그리고 오버로딩된 메서드들은 매개변수에 의해서만 구별될 수 있으므로 반환 타입은 오버로딩을 구현하는데 아무런 영향을 주지 못한다는 것에 주의하자.

오버로딩의 예로 가장 대표적인 것은 `println` 메서드이다. 지금까지 여러분은 `println` 메서드의 괄호 안에 값만 지정해주면 화면에 출력하는데 아무런 어려움이 없었다.

하지만, 실제로는 `println` 메서드를 호출할 때 매개변수로 지정하는 값의 타입에 따라서 호출되는 `println` 메서드가 달라진다. `PrintStream` 클래스에는 어떤 종류의 매개변수를 지정해도 출력할 수 있도록 정의해놓고 있다.

```
void println();  
void println(boolean x);  
void println(char x);
```

[예제 3-7] 메서드 오버로딩

```
public class Ex3_7 {
    int count;           // 인스턴스변수
    static int staticCount; // 클래스변수
    void count() { // 1
        System.out.println(count);
    }
    int count() { // 2
        return count;
    }
    int count(int a) { // 3
        int result = count + a;
        return result;
    }
    float count(float f) { // 4
        float result = count + a;
        return result;
    }
}
```

구분	1	2	3
2	메서드명 : 일치 매개변수 개수 : 일치 매개변수 타입 : 일치 <b>사용 불가</b>		
3	메서드명 : 일치 매개변수 개수 : 불일치 매개변수 타입 : 불일치 <b>사용 가능</b>	2번사용 불가로 비교하지 않는다.	
4	메서드명 : 일치 매개변수 개수 : 불일치 매개변수 타입 : 불일치 <b>사용가능</b>	2번사용 불가로 비교하지 않는다.	메서드명 : 일치 매개변수 개수 : 일치 매개변수 타입 : 불일치 <b>사용가능</b>

2번메서드 생성시 1번 메서드와 오버로딩 조건이 일치하지 않으므로 사용할 수 없다. 하지만 3,4번 메서드의 경우 매개변수 개수의 불일치 또는 매개변수 타입의 불일치로 메서드 오버로딩 조건에 만족하여 사용이 가능하다.

※ 동일한 메서드를 생성시 '메서드명 is already defined (이미 정의되어 있는 메서드명)'이라는 컴파일 에러가 발생하게 된다.

[문제 3-8] 주석에 적당한 코드를 작성해 주세요.

```
public class Qu3_8 {
    public static void main(String[] args) {
        //7. 3번의 add메서드를 호출하여라.

        //8. 4번의 add메서드를 호출하여라.

        //9. 5번의 add메서드를 호출하여라.

        //10. 6번의 add메서드를 호출하여라.

    }
}

class MyAdd {
    //1. 클래스변수 a를 선언하고 20의 값으로 초기화 하여라.

    //2. 인스턴스변수 b를 선언하고 15의 값으로 초기화 하여라.

    //3. int타입의 매개변수가 하나이며 변수 a와 합을 반환하는 클래스메서드를 add를 작성
    // 하여라.

    //4. int타입의 매개변수가 두개이고 매개변수의 합을 반환하는 인스턴스메서드를 add를
    // 작성 하여라.

    //5. int타입, long타입 각 한 개의 매개변수, 매개변수의 합을 반환하는 인스턴스
    // 메서드 add를 작성 하여라.

    //6 char타입, float타입 각 한 개의 매개변수, 매개변수의 합을 반환하는 인스턴스
    // 메서드 add를 작성 하여라.

}
```

※ hint : 변수의 타입에서 배웠듯이 long타입과 float타입의 값은 점미사가 붙어야 한다.

## 3-4. 생성자

### 학습목표

- 생성자의 의미를 이해하고 활용할 수 있다.

### 필요 지식 / 메서드 오버로딩

#### ① 생성자

생성자는 인스턴스가 생성될 때 호출되는 '인스턴스 초기화 메서드'이다. 따라서 인스턴스 변수의 초기화 작업에 주로 사용되며, 인스턴스 생성 시에 실행되어야 하는 작업을 위해서도 사용된다.

생성자 역시 메서드처럼 클래스 내에 선언되며, 구조도 메서드와 유사하지만 리턴값이 없다는 점이 다르다. 그렇다고 해서 생성자 앞에 리턴값이 없음을 뜻하는 키워드 `void`를 사용하지는 않고, 단지 아무 것도 적지 않는다. 생성자의 조건은 다음과 같다.

1. 생성자의 이름은 클래스의 이름과 같아야 한다.
2. 생성자는 리턴 값이 없다.

※ 생성자도 메서드이기 때문에 리턴값이 없다는 의미의 `void`를 붙여야 하지만, 모든 생성자가 리턴값이 없으므로 `void`를 생략할 수 있게 한 것이다.

생성자는 다음과 같이 정의한다. 생성자도 오버로딩이 가능하므로 하나의 클래스에 여러 개의 생성자가 존재할 수 있다.

```
클래스명(타입 변수명, 타입 변수명, ...){  
    // 인스턴스 생성시 수행될 코드  
    // 주로 인스턴스 변수의 초기화 코드를 적는다.  
}
```

연산자 `new`가 인스턴스를 생성하는 것이지 생성자가 인스턴스를 생성하는 것이 아니다. 생성자라는 용어 때문에 오해하기 쉬운데, 생성자는 단순히 인스턴스 변수들의 초기화에 사용되는 조금 특별한 메서드일 뿐이다. 생성자가 갖는 몇 가지 특징만 제외하면 메서드와 다르지 않다.



## 1. 기본 생성자 (default constructor)

지금까지는 생성자를 모르고도 프로그래밍을 해 왔지만, 사실 모든 클래스에는 반드시 하나 이상의 생성자가 정의되어 있어야 한다.

그러나 지금까지 클래스에 생성자를 정의하지 않고도 인스턴스를 생성할 수 있었던 이유는 컴파일러가 제공하는 '기본 생성자(default constructor)' 덕분이었다.


컴파일 할 때, 해당 클래스의 생성자가 하나도 정의되지 않은 경우 컴파일러는 자동적으로 아래와 같은 내용의 기본 생성자를 추가하여 컴파일 한다.

<pre>클래스이름() { // 기본 생성자 }</pre>	
컴파일 전	컴파일 후
<pre>class Card {  }</pre>	<pre>class Card {     Card() { } }</pre>

컴파일러가 자동적으로 추가해주는 기본 생성자는 이와 같이 매개변수도 없고 아무런 내용도 없는 아주 간단한 것이다. 그동안 우리는 인스턴스를 생성할 때 컴파일러가 제공한 기본 생성자를 사용해왔던 것이다.

### [예제 3-8] 기본 생성자

```
class TestData_1{
    int value;
}
class TestData_2{
    int value;
    TestData_2(int x){
        value = x;
    }
}
public class Ex3_8{
    public static void main(String[] args) {
        TestData_1 td1 = new TestData_1();
        TestData_2 td2 = new TestData_2(); // 컴파일 에러
    }
}
```



이 예제를 컴파일하면 위와 같은 에러를 확인할 수 있다. 이것은 TestData\_2클래스에서 TestData\_2( )라는 생성자를 찾을 수 없다는 내용의 에러 메시지인데, TestData\_2클래스에 생성자 TestData\_2()가 정의되어 있지 않기 때문에 에러가 발생한 것이다.

TestData\_1의 인스턴스를 생성하는 코드에는 에러가 없는데, TestData\_2의 인스턴스를 생성하는 코드에서 에러가 발생하는 이유는 무엇일까?

그 이유는 TestData\_1에는 정의되어 있는 생성자가 하나도 없으므로 컴파일러가 기본 생성자를 추가해주었지만, TestData\_2에는 이미 생성자 TestData\_2(int x)가 정의되어 있으므로 기본생성자가 추가되지 않았기 때문이다.

컴파일러가 자동적으로 기본 생성자를 추가해주는 경우는 '클래스 내에 생성자가 하나도 없을 때'뿐이라는 것을 명심해야한다.

```
TestData_1 td1 = new TestData_1();  
TestData_2 td2 = new TestData_2(); //에러
```

→

```
TestData_1 td1 = new TestData_1();  
TestData_2 td2 = new TestData_2(10); //OK
```

이 예제에서 컴파일 에러가 발생하지 않도록 하기 위해서는 위의 오른쪽 코드와 같이 TestData\_2 클래스의 인스턴스를 생성할 때 생성자 TestData\_2(int x)를 사용하던가, 아니면 TestData\_2클래스에 생성자 TestData\_2( )를 추가로 정의해주면 된다.

**기본 생성자가 컴파일러에 의해서 추가되는 경우는**

**클래스에 정의된 생성자가 하나도 없을 때 뿐이다.**

## 2. 매개변수가 있는 생성자

생성자도 메서드처럼 매개변수를 선언하여 호출 시 값을 넘겨받아서 인스턴스의 초기화 작업에 사용할 수 있다. 인스턴스마다 각기 다른 값으로 초기화되어야 하는 경우가 많기 때문에 매개변수를 사용한 초기화는 매우 유용하다.

아래의 코드는 자동차를 클래스로 정의한 것인데, 단순히 color, gearType, door 세 개의 인스턴스변수와 두 개의 생성자만을 가지고 있다.

### [예제 3-9] 생성자

```
class Car {  
    String color;      // 색상  
    String gearType;   // 변속종류  
    int door;          // 문의 개수  
  
    Car() {            // 기본 생성자  
  
    }  
  
    Car(String c, String g, int d) { // 매개변수가 3개인 생성자  
        color = c;  
        gearType = g;  
        door = d;  
    }  
}
```

Car인스턴스를 생성할 때, 생성자 Car( )를 사용한다면, 인스턴스를 생성한 다음에 인스턴스 변수들을 따로 초기화해주어야 하지만, 매개변수가 있는 생성자 Car(String color, String gearType, int door)를 사용한다면 인스턴스를 생성하는 동시에 원하는 값으로 초기화를 할 수 있게 된다.

인스턴스를 생성한 다음에 인스턴스변수의 값을 변경하는 것보다 매개변수를 갖는 생성자를 사용하는 것이 코드를 보다 간결하고 직관적으로 만든다.

```
Car c = new Car();  
c.color = "white";  
c.gearType = "auto";  
c.door = 4;
```

→

```
Car c = new Car("white", "auto", 4);
```

위의 양쪽 코드 모두 같은 내용이지만, 오른쪽의 코드가 더 간결하고 직관적이다.

### 3. 객체 자신을 가리키는 참조변수 - this

[예제 3-8]에서 매개변수가 있는 생성자의 매개변수로 선언된 지역변수 `c`의 값을 인스턴스변수 `color`에 저장한다. 이 때 변수 `color`와 `c`는 이름만으로도 서로 구별되므로 아무런 문제가 없다. 하지만 생성자의 매개변수로 선언된 변수의 이름이 `c`이기 때문에 이 생성자를 사용하는 쪽에서는 어떠한 값을 입력해야 할지 알 수 없기 때문에 가독성을 위하여 `color`로 변경한다면 인스턴스변수 `color`와 매개변수인 `color`의 변수명이 같아진다. 이때 두 변수를 구별하기 위하여 인스턴스변수 앞에 'this'를 사용하여 구분할 수 있게 된다.

<pre>Car(String c, String g, int d) {     color = c;     gearType = g;     door =d; }</pre>	→	<pre>Car(String color, String gearType, int door) {     this.color = color;     this.gearType = gearType;     this.door =door; }</pre>
---	---	--

이렇게 하면 `this.color`는 인스턴스변수이고, `color`는 생성자의 매개변수로 정의된 지역변수로 서로 구별이 가능하다. 만일 오른쪽코드에서 '`this.color= color`'대신 '`color = color`'와 같이하면 둘 다 지역변수로 간주된다.

이처럼 생성자의 매개변수로 인스턴스변수들의 초기값을 제공받는 경우가 많기 때문에 매개변수와 인스턴스변수의 이름이 일치하는 경우가 자주 있다. 이때는 왼쪽코드와 같이 매개변수이름을 다르게 하는 것 보다 'this'를 사용해서 구별되도록 하는 것이 의미가 더 명확하고 이해하기 쉽다.

'this'는 참조변수로 인스턴스 자신을 가리킨다. 참조변수를 통해 인스턴스의 멤버에 접근할수 있는 것처럼, 'this'로 인스턴스변수에 접근할 수 있는 것이다.

하지만, 'this'를 사용할 수 있는 것은 인스턴스멤버뿐이다. 클래스 메서드에서는 인스턴스 멤버들을 사용할 수 없는 것처럼, 'this' 역시 사용할 수 없다. 왜냐하면, 클래스메서드는 인스턴스를 생성하지 않고도 호출될 수 있으므로 클래스메서드가 호출된 시점에 인스턴스가 존재하지 않을 수도 있기 때문이다.

사실 생성자를 포함한 모든 인스턴스메서드에는 자신이 관련된 인스턴스를 가리키는 참조변수 'this'가 지역변수로 숨겨진 채로 존재한다.

#### 4. 생성자에서 다른 생성자 호출하기 - this()

생성자의 호출은 인스턴스가 생성될 때만 호출이 가능하다. 하지만 생성자에서 다른 생성자를 호출해야 하는 경우가 있다. 생성자 간에도 서로 호출이 가능하기 위해서는 다음의 두 조건을 만족시켜야 한다.

- 생성자의 이름으로 클래스이름 대신 this를 사용한다.
- 한 생성자에서 다른 생성자를 호출할 때는 반드시 첫 줄에서만 호출이 가능하다.

```
Car (String color) {  
    door = 5;  
    Car(color, "auto", 4); // 예러1. 생성자의 두 번째 줄에서 다른 생성자 호출  
                           // 예러2. 클래스명 대신 this를 사용해야 한다.  
}
```

생성자 내에서 다른 생성자를 호출할 때는 클래스이름인 'Car'대신 'this'를 사용해야하는데 그렇지 않아서 예러이고, 또 다른 예러는 생성자 호출이 첫 번째 줄이 아닌 두 번째 줄이기 때문에 예러이다.

생성자에서 다른 생성자를 첫 줄에서만 호출이 가능하도록 한 이유는 생성자 내에서 초기화 작업도중에 다른 생성자를 호출하게 되면, 호출된 다른 생성자 내에서도 멤버변수들의 값을 초기화를 할 것이므로 다른 생성자를 호출하기 이전의 초기화 작업이 무의미해질 수 있기 때문이다.

#### [예제 3-10] this()가 사용되어야 하는 경우

```
class Car {  
    String color;      // 색상  
    String gearType;   // 변속종류  
    int door;          // 문의 개수  
  
    Car() {  
//        color = "black";  
//        gearType = "stick";  
//        door =4;  
        this("black", "stick",4); // this()이 이용하여 위의 3줄을 한줄로 변경 가능  
    }  
}
```

```

Car(String color){
    this();
    this.color = color;
}

Car(String color, String gearType, int door) {
    this.color = color;
    this.gearType = gearType;
    this.door =door;
}

}

public class Ex3_10 {
    public static void main(String[] args){
        Car c1 = new Car();           // 기본차량
        Car c2 = new Car("red");      // 색상을 red로 변경한 차량
        Car c3 = new Car("blue","auto",2); // 색상을 blue, 기어를 auto, 문은 2개인 차량

        System.out.println("기본차량");
        System.out.println("\t차량색상 : " + c1.color + ", 기어타입 : " + c1.gearType
            + ", 문의 개수 : " + c1.door);

        System.out.println("색상변경 차량");
        System.out.println("\t차량색상 : " + c2.color + ", 기어타입 : " + c2.gearType
            + ", 문의 개수 : " + c2.door);

        System.out.println("풀옵션 차량");
        System.out.println("\t차량색상 : " + c3.color + ", 기어타입 : " + c3.gearType
            + ", 문의 개수 : " + c3.door);
    }
}

```

결과

```

기본차량
차량색상 : black, 기어타입 : stick, 문의 개수 : 4

색상변경 차량
차량색상 : red, 기어타입 : stick, 문의 개수 : 4

풀옵션 차량
차량색상 : blue, 기어타입 : auto, 문의 개수 : 2

```

같은 클래스 내의 생성자들은 일반적으로 서로 관계가 깊은 경우가 많아서 이처럼 서로 호출하도록 하여 유기적으로 연결해주면 더 좋은 코드를 얻을 수 있다. 그리고 수정이 필요한 경우에 보다 적은 코드만을 변경하면 되므로 유지보수가 쉬워진다.

[문제 3-9] [예제 3-10]을 JVM메모리 구조로 도식화 하여라.

Method Area

Call Stack

Heap

## 5. 변수의 초기화

변수를 선언하고 처음으로 값을 저장하는 것을 '변수의 초기화'라고 한다. 변수의 초기화는 경우에 따라서 필수적이기도 하고 선택적이기도 하지만, 가능하면 선언과 동시에 적절한 값으로 초기화 하는 것이 바람직하다.

멤버변수는 초기화를 하지 않아도 자동적으로 변수의 자료형에 맞는 기본값으로 초기화가 이루어지므로 초기화하지 않고 사용해도 되지만, 지역변수는 사용하기 전에 반드시 초기화해야 한다.

### (1) 명시적 초기화(explicit initialization)

변수를 선언과 동시에 초기화하는 것을 명시적 초기화라고 한다. 가장 기본적인면서도 간단한 초기화 방법이므로 여러 초기화 방법 중에서 가장 우선적으로 고려되어야 한다.

```
class Car {  
    int door = 4;           // 기본형 변수의 초기화  
    Test t = new Test();    // 참조형 변수의 초기화  
}
```

### (2) 초기화 블록 초기화(initialization block)

초기화 블록에는 '클래스 초기화 블록'과 '인스턴스 초기화 블록' 두 가지 종류가 있다. 클래스 초기화 블록은 클래스변수의 초기화에 사용되고, 인스턴스 초기화 블록은 인스턴스변수의 초기화에 사용된다.

**클래스 초기화 블록** 클래스변수의 복잡한 초기화에 사용된다.  
**인스턴스 초기화 블록** 인스턴스변수의 복잡한 초기화에 사용된다.

초기화 블록을 작성하려면, 인스턴스 초기화 블록은 단순히 클래스 내에 블록{} 만들고 그 안에 코드를 작성하기만 하면 된다. 그리고 클래스 초기화 블록은 인스턴스 초기화 블록 앞에 단순히 static을 덧붙이기만 하면 된다.

지역변수와 달리 멤버변수는 각 타입의 기본값으로 자동 초기화 된다. 그 다음에 명시적 초기화, 초기화 블록, 생성자의 순서로 초기화 된다. 그리고 클래스 변수(cv)가 인스턴스 변수(iv) 보다 먼저 초기화 된다. 멤버변수의 초기화에 대해서는 이 두 가지만 기억하면 된다.

1. 클래스변수 초기화 -> 인스턴스 변수 초기화
2. 자동초기화 -> 명시적초기화(간단) -> 초기화 블록, 생성자(복잡)



### [예제 3-11] 변수의 초기화

```
class Init {
    static int action;
    int action2 = 5;

    static { // 클래스 초기화 블록
        action = 4;
    }

    { // 인스턴스 초기화 블록
        action2 = 4;
    }

    Init(){
        action2 = 7;
    }

    Init(int action2){
        this();
        this.action2 = action2;
    }
}

public class Ex3_11 {
    public static void main(String[] args){
        Init init = new Init(9);
    }
}
```

#### 1. Init init;

- Init클래스가 메모리에 로드되고 클래스변수인 action이 생성되며 기본값인 0으로 초기화한다.
- 클래스 초기화 블록을 통해 action의 값이 4로 변경한다.

#### 2. new Init(7);

- 연산자 'new'에 의해 인스턴스가 생성된다. 이때 인스턴스변수인 action2가 메모리에 같이 생성되고 명시적초기화를 통해 5의 초기화 된다.
- 인스턴스 초기화 블록이 실행되어 action2의 값을 4로 변경 한다.
- Init(int action2) 생성자가 실행 -> 기본생성자 호출 -> action2의 값이 7로 변경 -> 기본생성자 종료 -> Init(int action2) 생성자에서 action2의 값이 9로 변경 : 결과적으로 action2는 9의 값을 가지게 된다.

## 3-5. 상속

### 학습목표

- 상속의 정의와 장점을 설명할 수 있다.
- 상속을 이용한 클래스 구성을 할 수 있다.

### 필요 지식 / 클래스

#### ① 상속

상속이란, 기존의 클래스를 재사용하여 새로운 클래스를 작성하는 것이다. 상속을 통해서 클래스를 작성하면 보다 적은 양의 코드로 새로운 클래스를 작성할 수 있고 코드를 공통적으로 관리할 수 있기 때문에 코드의 추가 및 변경이 매우 용이하다.

이러한 특징은 코드의 재사용성을 높이고 코드의 중복을 제거하여 프로그램의 생산성과 유지보수에 크게 기여한다.

자바에서 상속을 구현하는 방법은 아주 간단하다. 새로 작성하고자 하는 클래스의 이름 뒤에 상속받고자 하는 클래스의 이름을 키워드 'extends'와 함께 써 주기만 하면 된다.

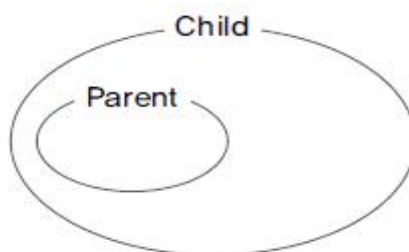
예를 들어 새로 작성하려는 클래스의 이름이 Child이고 상속받고자 하는 기존 클래스의 이름이 Parent라면 다음과 같이 하면 된다.

```
class Parent{  
}  
  
class Child extends Parent{  
}
```

Parent  
↑  
Child

이 두 클래스는 서로 상속 관계에 있다고 하며, 상속해주는 클래스를 '조상 클래스'라 하고 상속 받는 클래스를 '자손 클래스'라 한다.

프로그램이 커질수록 클래스간의 관계가 복잡해지는데, 이 때 아래와 같이 그림으로 표현하면 클래스간의 관계를 보다 쉽게 이해할 수 있다.



자손 클래스는 조상 클래스의 모든 멤버를 상속 받으므로 항상 조상 클래스보다 같거나 많은 멤버를 갖는다. 즉, 상속에 상속을 거듭할수록 상속받는 클래스의 멤버 개수는 점점 늘어나게 된다.

그래서 상속을 받는다는 것은 조상 클래스를 확장(extend)한다는 의미로 해석할 수도 있으며 이것이 상속에 사용되는 키워드가 'extends'인 이유이기도 하다.

- 자손 클래스는 조상 클래스의 모든 멤버를 상속받는다.  
( 단, 생성자와 초기화 블록은 상속되지 않는다.)
- 자손 클래스의 멤버 개수는 조상 클래스보다 항상 같거나 많다.

#### [예제 3-12] 상속

```
public class Ex3_12 {
    public static void main(String[] args) {
        Child ci = new Child();
        //자식
        ci.volume = 15;
        ci.volumeUp();
        System.out.println(ci.volume);
        //부모
        ci.channel = 3;
        ci.channelUp();
        System.out.println(ci.channel);
    }
}

class Child extends Parent{
    int volume;
    void volumeUp(){
        volume++;
    }
}

class Parent {
    int channel;
    void channelUp(){
        channel++;
    }
}
```

[문제 3-10] [예제 3-12]을 JVM메모리 구조로 도식화 하여라.

Method Area

Call Stack

Heap

## ② 포함

상속이외에도 클래스를 재사용하는 또 다른 방법이 있는데, 그것은 클래스 간에 '포함(Composite)' 관계를 맺어 주는 것이다. 클래스 간의 포함관계를 맺어 주는 것은 한 클래스의 멤버변수로 다른 클래스 타입의 참조변수를 선언하는 것을 뜻한다.

원(Circle)을 표현하기 위한 Circle클래스와 좌표상의 한 점을 다루기 위한 Point클래스를 다음과 같이 작성되어 있다고 가정하자.

```
class Circle{
    int x; // 원점의 x좌표
    int y; // 원점의 y좌표
    int r; // 원의 반지름
}
```

```
class Point{
    int x; // x좌표
    int y; // y좌표
}
```

Point클래스를 재사용하여 Circle클래스를 작성한다면 다음과 같이 작성할 수 있다.

```
class Circle{
    Point c = new Point(); // 원점
    int r;                  // 반지름
}
```

이와 같이 한 클래스를 작성하는 데 다른 클래스를 멤버변수로 선언하여 포함시키는 것은 좋은 생각이다. 하나의 거대한 클래스를 작성하는 것보다 단위별로 여러 개의 클래스를 작성한 다음, 이 단위 클래스들을 포함관계로 재사용하면 보다 간결하고 손쉽게 클래스를 작성할 수 있다. 또한 작성된 단위 클래스들은 다른 클래스를 작성하는데 재사용될 수 있을 것이다.

**[문제 3-11] 삼각형과 원 만들기.**

```
class Point{
    int x;
    int y;

    Point(int x, int y){
        this.x = x;
        this.y = y;
    }
}
```

```

class Circle{
    //1. 중심점을 저장할 수 있는 변수 c를 선언하여라.

    //2. 반지름(정수)을 저장할 수 있는 변수 r을 선언하여라.

    //3. 매개변수가 두 개인 생성자를 작성하여라.
    //    단. 두 개의 매개변수를 이용하여 c와 r을 초기화 하여라.

    //4. 기본 생성자를 작성하여라.
    //    단. 매개변수가 두 개인 생성자를 호출하여 중심점은(100,100)으로 반지름은 200이
    //    저장되도록 하여라.

}

class Triangle{
    //5. 점 여러 개를 저장할 수 있는 변수 p를 선언하여라.

    //6. 매개변수가 하나인 생성자를 이용하여 p를 초기화 하여라.

    //7. 매개변수가 세 개인 생성자를 이용하여 p를 초기화 하여라.

}

public class Qu3_11{
    public static void main(String[] args){
        //8. Circle의 객체를 생성하여라.
        //    단. 매개변수가 두 개인 생성자를 이용하여라. (중심점은 200,200 반지름 50)

        //9. Triangle의 객체를 생성하여라.
        //    단. 매개변수가 하나인 생성자를 이용하여라.

    }
}

```

[문제 3-12] 카드 한 벌 만들기

```

public class Qu3_12{
    public static void main(String[] args){
        //9. 카드 한 벌의 객체를 생성

        //10. 0번 index의 카드 한 장을 뽑기

        //11. 임의의 index번째 카드 한 장을 뽑기

        //12. 카드를 자동 섞기

        //13. 0번 index의 카드 한 장을 뽑기

        //14. 카드 1000번 섞기

        //15. 0번 index의 카드 한 장을 뽑기

    }
}

class Card{
    static final int KIND_MAX = 4; //카드 무늬수
    static final int NUM_MAX = 13; //무늬별 카드수

    static final int SPADE = 4;
    static final int DIAMOND = 3;
    static final int HEART = 2;
    static final int CLOVER = 1;

    int kind;    // 카드 무늬
    int num;     // 카드 숫자

    Card(){
        this(SPADE,1);
    }

    Card(int kind, int number){
        this.kind = kind;
        this.number = number;
    }
}

```

```

@Override
public String toString(){
    String kind = "";
    String number="";

    switch(this.kind){
        case 4 :
            kind = "SPADE";
            break;
        case 3 :
            kind = "DIAMOND";
            break;
        case 2 :
            kind = "HEART";
            break;
        case 1 :
            kind = "CLOVER";
            break;
        default :
    }

    //1. 인스턴스변수 number의 값이 11이면 지역변수 number의 값을 "J"로, 12이면
    // "Q"로, 13이면 "K"로, 1이면 "A"로 그 외의 경우는 번호 그대로를
    // 저장하여라.

    return "kind : " + kind + ", number : " + number;

}
}

class Deck{ // 카드 한 벌(52장)을 Deck이라고 한다.
    //2. 카드의 수량을 저장할 수 있는 변수 CARD_NUM을 선언하고 Card클래스의 변수를
    // 사용하여 초기화하여라.

```



//3. 카드 52장을 저장할 수 있는 변수 c를 선언하고 생성하여라.

//4. 기본생성자를 작성 하여라.

// 단. 변수 c의 모든방에 카드 1,1 ~ 4,13까지 생성하여 저장하여라.

//5. c에서 임의의 index번째 카드 한 장을 반환하는 메서드(pick)를 작성하여라.

//6. 사용자로부터 입력받은 index번째 카드 한 장을 반환하는 메서드(pick)를 작성하여라.

// 단. 입력받은 값이 0~51사이의 정수라면 입력받은 index번째 카드 한 장을 반환하고

// 그렇지 않은 경우에는 “랜덤번호“를 출력하고 임의의 한 장을 반환하도록 하여라.

//7. c의 카드를 섞는 메서드(shuffle)을 작성하여라. - 자동 섞기

// 단. 카드 섞는 법 : 연습문제 5-6의 방법을 활용

//8. 사용자로부터 입력받은 횟수만큼 c의 카드를 섞는 메서드(shuffle)을 작성하여라.

// 단. 임의의 방 두개를 뽑아 두개의 index번째 요소의 위치를 바꾼다. 이를 사용자로

// 부터 입력받은 횟수만큼 반복한다.

}

### ③ 클래스 간의 관계 결정하기

클래스를 작성하는데 있어서 상속관계를 맺어 줄 것인지 포함관계를 맺어 줄 것인지 결정하는 것은 때때로 혼동스러울 수 있다. 아래와 같이 문장을 만들어 보면 클래스 간의 관계가 보다 명확해 진다.

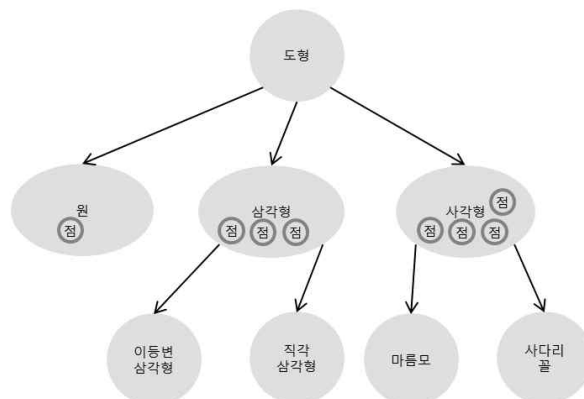
원(Circle)은 점(Point)이다. - Circle **is a** Point  
 원(Circle)은 점(Point)을 가지고 있다.. - Circle **has a** Point

원은 원점(Point)과 반지름으로 구성되므로 위의 두 문장을 비교해 보면 첫 번째 문장보다 두 번째 문장이 더 옳다는 것을 알 수 있을 것이다.

이처럼 클래스를 가지고 문장을 만들었을 때 '~은 ~이다.'라는 문장이 성립한다면, 서로 상속관계를 맺어 주고, '~은 ~을 가지고 있다.'는 문장이 성립한다면 포함관계를 맺어 주면 된다. 그래서 Circle클래스와 Point클래스 간의 관계는 상속관계 보다 포함관계를 맺어 주는 것이 더 옳다.

상속관계 ' ~은 ~이다. (is -a)'  
 포함관계 ' ~은 ~을 가지고 있다. (has -a)'

클래스 : 원, 이등변 삼각형, 직각삼각형, 마름모, 사다리꼴



원, 삼각형, 사각형은 각각 점을 한개, 세 개, 네 개인 점을 포함하고 있기 때문에 위와 같이 표현을 할 수 있다. 또한 마름모와 사다리꼴의 상위로 사각형을 만들 듯이 원, 삼각형, 사각형의 공통되는 부분을 모은다면 도형이라는 상위 클래스를 만들 수 있다.

#### ④ 단일상속(single inheritance)

또 다른 객체지향언어인 C++에서는 여러 조상 클래스로부터 상속받는 것이 가능한 '다중 상속(multiple inheritance)'을 허용하지만 자바에서는 단일 상속만을 허용한다. 그래서 둘 이상의 클래스로부터 상속을 받을 수 없다. 예를 들어, Tv클래스와 DVD클래스가 있을 때, 이 두 클래스로부터 상속을 받는 TvDVD클래스를 작성할 수 없다. 그래서 TvDVD클래스는 조상 클래스로 Tv클래스와 DVD클래스 중 하나만 선택해야한다.

```
class TvDVD extends Tv, DVD{ // 에러. 조상은 하나만 허용 된다.
    // ....
}
```

다중상속을 허용하면 여러 클래스로부터 상속받을 수 있기 때문에 복합적인 기능을 가진 클래스를 쉽게 작성할 수 있다는 장점이 있지만, 클래스간의 관계가 매우 복잡해진다는 것과 서로 다른 클래스로부터 상속받은 멤버간의 이름이 같은 경우 구별할 수 있는 방법이 없다는 단점을 가지고 있다.

<pre>class Tv{     int b;     void method(){         System.out.println("Tv의 메서드");     } }</pre>	<pre>class Dvd{     int c;     void method(){         System.out.println("Dvd의 메서드");     } }</pre>
---	---

위의 두개의 클래스를 다중 상속을 받을 수 있다고 가정해 보자. 상속이라 함은 해당 클래스의 모든 멤버를 상속받게 된다. Tv클래스의 변수 b와 Dvd클래스의 c는 명칭이 다르기 때문에 어떤 것을 상속받아도 문제가 발생하지 않는다. 하지만 method()의 경우에는 어떤 쪽의 것을 상속 받게 되는지 알 수 없게 된다.

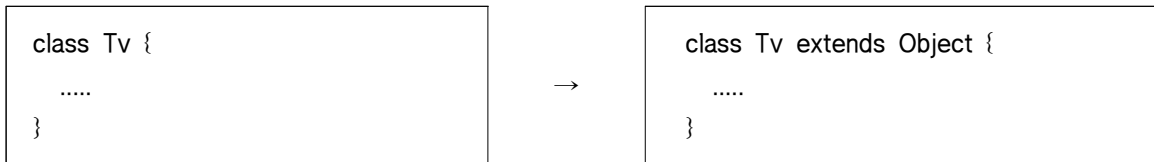
이것을 해결하는 방법은 조상 클래스의 메서드의 이름이나 매개변수를 바꾸는 방법 밖에 없다. 이렇게 하면 그 조상 클래스의 method( )메서드를 사용하는 모든 클래스들도 변경을 해야하므로 그리 간단한 문제가 아니다.

자바에서는 다중상속의 이러한 문제점을 해결하기 위해 다중상속의 장점을 포기하고 단일상속만을 허용한다. 단일 상속이 하나의 조상 클래스만을 가질 수 있기 때문에 다중상속에 비해 불편한 점도 있지만, 클래스 간의 관계가 보다 명확해지고 코드를 더욱 신뢰할 수 있게 만들어 준다는 점에서 다중상속보다 유리하다.

## ⑤ Object클래스 - 모든 클래스의 조상

Object클래스는 모든 클래스 상속계층도의 최상위에 있는 조상클래스이다. 다른 클래스로부터 상속 받지 않는 모든 클래스들은 자동적으로 Object클래스로부터 상속받게 함으로써 이것을 가능하게 한다.

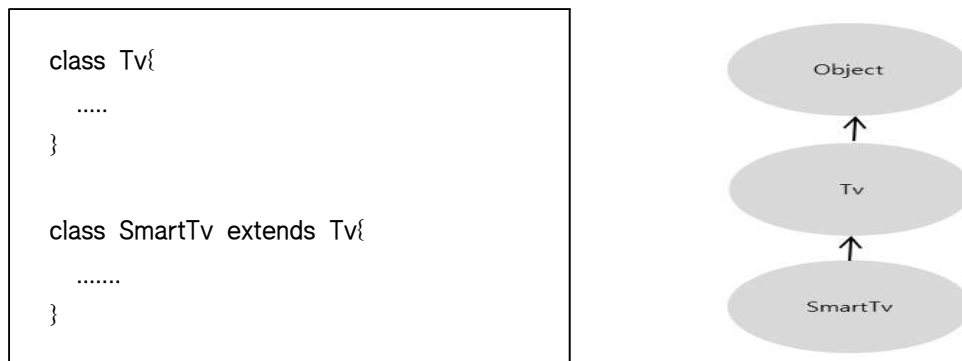
만일 다음과 같이 다른 클래스로부터 상속을 받지 않는 Tv클래스를 정의하였다고 하자.



위의 코드를 컴파일 하면 컴파일러는 위의 코드를 다음과 같이 자동적으로 'extends Object'를 추가하여 Tv클래스가 Object클래스로부터 상속받도록 한다.

이렇게 함으로써 Object클래스가 모든 클래스의 조상이 되도록 한다. 만일 다른 클래스로부터 상속을 받는다고 하더라도 상속계층도를 따라 조상클래스, 조상클래스의 조상클래스를 찾아 올라가다 보면 결국 마지막 최상위 조상은 Object클래스일 것이다.

※ 이미 어떤 클래스로부터 상속받도록 작성된 클래스에 대해서는 컴파일러가 'extends Object'를 추가하지 않는다.



위와 같이 Tv클래스가 있고, Tv클래스를 상속받는 SmartTV가 있을 때 상속계층도는 다음과 같다.

※ 상속계층도를 단순화하기 위해서 Object클래스를 생략하는 경우가 많다.

이처럼 모든 상속계층도의 최상위에는 Object클래스가 위치한다. 그래서 자바의 모든 클래스들은 Object클래스의 멤버들을 상속 받기 때문에 Object클래스에 정의된 멤버들을 사용할 수 있다. 그동안 toString( )이나 equals(Object o)와 같은 메서드를 따로 정의하지 않고도 사용할 수 있었던 이유는 이 메서드들이 Object클래스에 정의된 것들이기 때문이다.

## ⑥ super, super()

## 1. 참조변수 super

super는 자손 클래스에서 조상 클래스로부터 상속받은 멤버를 참조하는데 사용되는 참조변수이다. 멤버변수와 지역변수의 이름이 같을 때 this를 붙여서 구별했듯이 상속받은 멤버와 자신의 멤버와 이름이 같을 때는 super를 붙여서 구별할 수 있다.

### [예제 3-13] 참조변수 super

```
public class Ex3_13 {  
    public static void main(String[] args) {  
        Child c = new Child();  
        c.method(50);  
    }  
}  
class Parent{  
    int x = 10;  
}  
class Child extends Parent{  
    int x=30;  
  
    void method(int x){  
        System.out.println("x : "+ x);  
        System.out.println("this.x : "+ this.x);  
        System.out.println("super.x : "+ super.x);  
    }  
}
```

결과	x : 50 this.x : 30 super.x : 10
----	---------------------------------------

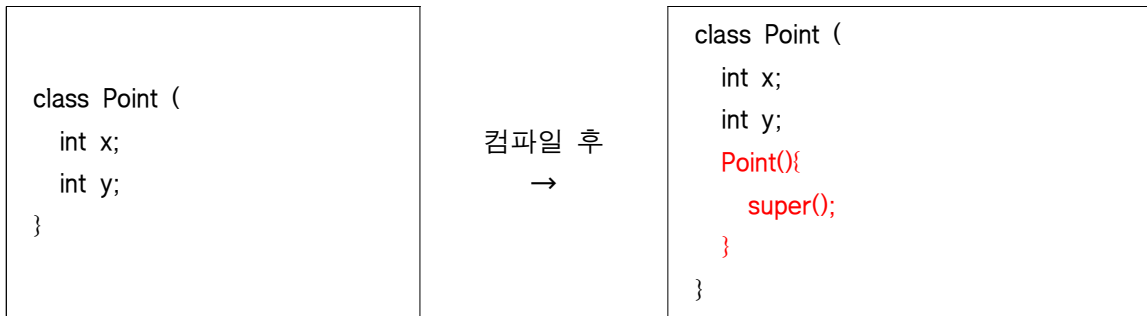
method()의 첫 번째 출력문 x는 지역변수를 의미한다.

Child클래스의 인스턴스 변수와 지역변수 x의 명칭이 같기 때문에 그 두 가지를 구분하기 위해 참조변수 this를 사용하였다.

Child클래스는 조상인 Parent클래스로부터 x를 상속받는데, 공교롭게도 자신의 멤버인 x와 이름이 같아서 이 둘을 구분할 방법이 필요하다. 바로 이럴 때 사용되는 것이 super다.

## 2. super() - 조상의 생성자

this()처럼 super()도 생성자이다. this()는 같은 클래스의 다른 생성자를 호출하는데 사용되지만, super()는 조상의 생성자를 호출하는데 사용된다.



위의 코드에서는 Point클래스는 생성자를 없기 때문에 컴파일러가 기본생성자인 Point()를 생성하고 기본생성자의 첫 줄에 super()까지 만들어 주게 된다.

**Object클래스를 제외한 모든 클래스의 생성자 첫 줄에 생성자, this()또는 super()를 호출해야 한다. 그렇지 않으면 컴파일러가 자동적으로 'super();'를 생성자의 첫줄에 삽입한다.**

인스턴스를 생성할때 클래스를 선택하는 것만큼 생성자를 선택하는 것도 중요하다.

#### [예제 3-14] super()

```
class Point{
    int x;
    int y;

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

class Point3D extends Point{
    int z;

    Point3D(int x, int y, int z){ // 컴파일 에러
        // 생성자의 첫 줄에서 다른 생성자를 호출하지 않기 때문이 컴파일러가 super()호출
        this.x = x;
        super.y = y;
        this.z = z;
    }
}
```

```
public class Ex3_14 {
    public static void main(String[] args) {
        Point3D p3 = new Point3D(1,2,3);
    }
}
```

이 예제는 컴파일하면 컴파일에러가 발생한다. Point3D클래스의 생성자에서 조상 클래스의 생성자인 Point()를 찾을 수 없다는 내용이다.

Point3D클래스의 생성자의 첫 줄이 생성자(조상의 것이든 자신의 것이든)를 호출하는 문장이 아니기 때문에 컴파일러는 다음과 같이 자동적으로 'super();'를 Point3D클래스의 생성자의 첫 줄에 넣어 준다.

```
Point3D(int x, int y, int z){
    super();
    this.x = x;
    super.y = y;
    this.z = z;
}
```

그래서 Point3D클래스의 인스턴스를 생성하면, 생성자 Point3D(int x, int y, int z)가 호출되면서 첫 문장인 'super();'를 수행하게 된다. super()는 Point3D클래스의 조상인 Point클래스의 기본 생성자인 Point()를 뜻하므로 Point()가 호출된다.

그러나 Point클래스에 생성자 Point()가 정의되어 있지 않기 때문에 위와 같은 컴파일 에러가 발생한 것이다. 이 에러를 수정하려면 Point클래스에 기본 생성자 Point()를 추가해주는 방법과 생성자 Point3D(int x, int y, int z)의 첫 줄에서 Point(x, y)를 호출하도록 변경하면 된다.

완성된 Point3D클래스의 생성자는 아래와 같다.

```
Point3D(int x, int y, int z){
    super(x, y);
    this.z = z;
}
```

## 3-6. 오버라이딩

### 학습목표

- 오버라이딩을 이용하여 조상클래스의 메서드를 재정의 할 수 있다.

### 필요 지식 / 상속

#### ① 오버라이딩

조상 클래스로부터 상속받은 메서드의 내용을 변경하는 것을 오버라이딩이라고 한다. 상속받은 메서드를 그대로 사용하기도 하지만, 자손 클래스 자신에 맞게 변경해야하는 경우가 많다. 이럴 때 조상의 메서드를 오버라이딩한다.

※ override의 사전적 의미는 '위에 덮어쓰다(over wite)'이다.

2차원 좌표계의 한 점을 표현하기 위한 Point클래스가 있을 때, 이를 조상으로 하는 Point3D 클래스, 3차원 좌표계의 한 점을 표현하기 위한 클래스를 다음과 작성하였다고 하자.

```
class Point {
    int x;
    int y;
    String getLocation(){
        return "x : "+ x +", y : "+ y;
    }
}

class Point3D extends Point {
    int z;
    String getLocation(){
        return "x : "+ x +", y : "+ y +", z : "+ z;
    }
}
```

Point클래스의 getLocation( )은 한 점의 x, y 좌표를 문자열로 반환하도록 작성되었다.

이 두 클래스는 서로 상속관계에 있으므로 Point3D클래스는 Point클래스로부터 getLocation( )을 상속받지만, Point3D클래스는 3차원 좌표계의 한 점을 표현하기 위한 것이므로 조상인 Point클래스로부터 상속받은 getLocation( )은 Point3D클래스에 맞지 않는다. 그래서 이 메서드를 Point3D클래스 자신에 맞게 z축의 좌표값도 포함하여 반환하도록 오버라이딩 하였다.



## ② 오버라이딩의 조건

오버라이딩은 메서드의 내용만을 새로 작성하는 것이므로 메서드의 선언부는 조상의 것과 완전히 일치해야 한다. 그래서 오버라이딩이 성립하기 위해서는 다음과 같은 조건을 만족해야 한다.

자손 클래스에서 오버라이딩하는 메서드는 조상 클래스의 메서드와

- 이름이 같아야 한다.
- 매개변수의 개수와 타입이 같아야 한다.
- 반환타입이 일치해야 한다.

접근 제어자(access modifier)와 예외(exception)는 제한된 조건 하에서만 다르게 변경할 수 있다. 예외(Exception)에 대해서 아직 배우지 않았지만, 일단 알아두자.

### 1. 접근 제어자는 조상 클래스의 메서드보다 좁은 범위로 변경 할 수 없다.

만일 조상 클래스에 정의된 메서드의 접근 제어자가 protected라면, 이를 오버라이딩하는 자손클래스의 메서드는 접근 제어자가 protected나 public이어야 한다. 같은 범위의 접근 제어자를 사용한다. 접근 제어자의 접근범위를 넓은 것에서 좁은 나열하면 public,protected, (default) , private이다.

### 2. 조상 클래스의 메서드보다 많은 수의 예외를 선언할 수 없다.

만일 조상 클래스에 정의된 메서드의 2개의 예외를 선언하였다면 자식 클래스에서는 2개 이하의 예외만 선언이 가능하다.

**오버로딩(overloading)** 기존에 없는 새로운 메서드를 정의하는 것 (new)  
**오버라이딩(overriding)** 상속받은 메서드의 내용을 재정의 하는 것(modify)

### [예제 3-15] 오버로딩과 오버라이딩

```
public class Ex3_15 {  
    public static void main(String[] args) {  
        OverGo og = new OVerGo();  
        System.out.println(og.upMethod());  
        System.out.println(og.downMethod());  
        System.out.println(og.downMethod(9));  
    }  
}
```

```

class OverUp{
    int x = 1;
    int y = 2;

    String upMethod(){
        return "overUp"+x+y;
    }
}
class OverGo extends OverUp{
    int z=3;

    //오버라이딩
    @Override
    String upMethod(){
        return x+y+z+ " : 오버라이딩";
    }

    //오버로딩
    String downMethod(){
        return "Basic";
    }
    String downMethod(int k){
        return x+y+z+k+ " : 오버로딩";
    }
}

```

결과  
6 : 오버라이딩  
Basic  
15 : 오버로딩

[예제 3-13]에서 OverGo클래스의 upMethod()는 부모의 upMethod()의 내용인 반환값을 재정의해 사용하였다. downMethod()의 경우에는 OverUp클래스에 선언부가 일치하는 메서드가 없지만 동일한 클래스인 OverGo클래스에 메서드명이 동일하고 매개변수가 개수가 다른 오버로딩의 특징을 가지게 된다.

오버라이딩된 upMethod()의 선언부 위쪽에 @override 어노테이션이 붙어 있는데 이는 부모의 메서드를 오버라이드 하였다는 정의이며 생략이 가능하다. 하지만 가독성을 위하여 붙여주는 것이 좋다.

## 3-7. 제어자

**학습목표**      클래스를 구성함에 있어 제어자를 적절하게 활용할 수 있다.

### 필요 지식 / 클래스

#### ① 제어자란?

제어자(modifier)는 클래스, 변수 또는 메서드의 선언부에 함께 사용되어 부가적인 의미를 부여한다. 제어자의 종류는 크게 접근 제어자와 그 외의 제어자로 나눌 수 있다.

접근 제어자	public, protected, default, private
그 외	static, final, abstract, native, transient, synchronized, volatile, strictfp

제어자는 클래스나 멤버변수와 메서드에 주로 사용되며, 하나의 대상에 대해서 여러 제어자를 조합하여 사용하는 것이 가능하다.

단, 접근 제어자는 한 번에 네 가지 중 하나만 선택해서 사용할 수 있다. 즉, 하나의 대상에 대해서 public과 private을 함께 사용할 수 없다는 것이다.

※ 제어자들 간의 순서에 관계없지만 주로 접근 제어자를 제일 왼쪽에 놓는 경향이 있다.

예를 들어 그 외 제어자 static과 접근제어자인 public을 같이 사용할 때 'static public'도 가능하지만 일반적으로는 'public static'의 형태로 많이 사용된다.

#### ② 그 외 제어자

그 외 제어자 중 static, final, abstract를 제외하고는 자주 사용되지 않는다. 일단 자주 사용되는 제어자에 대해서 알아보도록 하자.

##### 1. static - 클래스의, 공통적인

static은 '클래스의' 또는 '공통적인'의 의미를 가지고 있다. 인스턴스변수는 하나의 클래스로부터 생성되었더라도 각기 다른 값을 유지 하지만, 클래스변수는 인스턴스에 관계없이 같은 값을 갖는다.

static이 붙은 멤버변수와 메서드, 그리고 초기화 블록은 인스턴스가 아닌 클래스에 관계된 것이기 때문에 인스턴스를 생성하지 않고도 사용할 수 있다.

인스턴스 멤버를 사용하지 않는 메서드는 `static`을 붙여서 클래스메서드로 선언하는 것을 고려해 보도록 하자. 가능하다면 `static`메서드로 하는 것이 인스턴스를 생성하지 않고도 호출이 가능해서 더 편리하고 속도도 더 빠르다.

제어자	대상	의미
static	멤버변수	<ul style="list-style-type: none"> <li>- 모든 인스턴스에 공통적으로 사용되는 클래스변수가 된다.</li> <li>- 클래스변수는 인스턴스를 생성하지 않고도 사용 가능하다.</li> <li>- 클래스가 메모리에 로드될 때 생성된다.</li> </ul>
	메서드	<ul style="list-style-type: none"> <li>- 인스턴스를 생성하지 않고도 호출이 가능하다.</li> <li>- 클래스메서드 내에서는 인스턴스를 직접 사용할 수 없다.</li> </ul>

※ `static`이 사용될 수 있는 곳 - 멤버변수, 메서드, 초기화 블록

## 2. final - 마지막의, 변경될 수 없는

`final`은 '마지막의'또는 '변경될 수 없는'의 의미를 가지고 있으며 거의 모든 대상에 사용될 수 있다.

변수에 사용되면 값을 변경할 수 없는 상수가 되며, 메서드에 사용되면 오버라이딩을 할 수 없게 되고 클래스에 사용되면 자신을 확장하는 자손클래스를 정의하지 못하게 된다.

제어자	대상	의미
final	클래스	변경될 수 없는 클래스, 확장될 수 없는 클래스가 된다. <code>final</code> 로 지정된 클래스는 다른 클래스의 조상이 될 수 없다.
	메서드	변경될 수 없는 메서드가 된다. <code>final</code> 로 지정된 메서드는 오버라이딩을 통해 재정의 될 수 없다.
	멤버변수	변수 앞에 <code>final</code> 이 붙으면, 값을 변경할 수 없는 상수가 된다.
	지역변수	

※ `final`이 사용될 수 있는 곳 - 클래스, 메서드, 멤버변수, 지역변수

## 3. abstract - 추상의, 미완성의

`abstract`는 '미완성'의 의미를 가지고 있다. 메서드의 선언부만 작성하고 실제 수행내용은 구현하지 않은 추상 메서드를 선언하는데 사용된다.

그리고 클래스에 사용되어 클래스 내에 추상메서드가 존재한다는 것을 쉽게 알 수 있게

한다. 보다 자세한 내용은 '추상 클래스'에서 다룬다.

제어자	대상	의미
abstract	클래스	클래스 내에 추상 메서드가 선언되어 있음을 의미
	메서드	선언부만 작성하고 구현부는 작성되지 않은 완성되지 않은 추상 메서드임을 의미

※ abstract가 사용될 수 있는 곳 - 클래스, 메서드

추상 클래스는 아직 완성되지 않은 메서드가 존재하는 '미완성된 설계도'이므로 인스턴스를 생성할 수 없다.

### ③ 접근제어자

접근 제어자는 멤버 또는 클래스에 사용되어, 해당하는 멤버 또는 클래스를 외부에서 접근하지 못하도록 제한하는 역할을 한다.

접근 제어자가 default임을 알리기 위해 실제로 default를 붙이지는 않는다. 클래스나 멤버변수, 메서드, 생성자에 접근 제어자가 지정되어 있지 않다면, 접근 제어자가 default임을 뜻한다.

제어자	같은 클래스	같은 패키지	자손클래스	전체
public				
protected				
default				
private				

[표 3-1] 접근제어자

**private** 같은 클래스 내에서만 접근이 가능하다.  
**default** 같은 패키지 내에서만 접근이 가능하다.  
**protected** 같은 패키지 내에서, 그리고 다른 패키지의 자손클래스에서 접근이 가능하다.  
**public** 접근 제한이 전혀 없다.

접근 범위가 넓은 쪽에서 좁은 쪽의 순으로 왼쪽부터 나열

**public > protected > default > private**

public은 접근 제한이 전혀 없는것이고, private은 같은 클래스 내에서만 사용하도록 제한하는 가장 높은 제한이다. 그리고 default는 같은 패키지내의 클래스에서만 접근이 가능하도록 하는 것이다.

마지막으로 protected는 패키지에 관계없이 상속관계에 있는 자손클래스에서 접근할 수 있도록 하는 것이 제한목적이지만, 같은 패키지 내에서도 접근이 가능하다. 그래서 protected가 default보다 접근범위가 더 넓다.

#### [예제 3-16] 접근제어자

```
/* package OOP_01 */
public class AccessModifier_01{
    private int a = 10;
    int b = 20;           // 접근제어자가 없다면 default가 생략되어있음
    protected int c = 30;
    public int d = 40;

    int getA(){ // 인스턴스 변수 a에 접근하기 위한 메서드
        return a;
    }
}

/* package OOP_02 */
public class AccessModifier_02{
    private int a = 10;
    int b = 20;
    protected int c = 30;
    public int d = 40;

    int getA(){ // 인스턴스 변수 a에 접근하기 위한 메서드
        return a;
    }
}

/* package OOP_01 */
public class AccessModifier_03 extends AccessModifier_02{
    private int a3 = 50;
    int b3 = 60;
    protected int c3 = c + 30;
    public int d3 = 80;
}
```

```

/* package OOP_01 */
public class Ex3_15 {
    public static void main(String[] args) {
        /* Ex3_15와 AccessModifier_01 같은 패키지 */
        AccessModifier_01 am01 = new AccessModifier_01();
        // System.out.println(am01.a); // 접근불가
        System.out.println(am01.b);
        System.out.println(am01.c);
        System.out.println(am01.d);

        /* Ex3_15와 AccessModifier_02 다른 패키지 */
        AccessModifier_02 am02 = new AccessModifier_02();
        // System.out.println(am02.a); // 접근불가
        // System.out.println(am02.b); // 접근불가
        // System.out.println(am02.c); // 접근불가
        System.out.println(am02.d); // d의 접근제어자가 public임으로 접근가능

        /* Ex3_15와 AccessModifier_03 다른 패키지 */
        AccessModifier_03 am03 = new AccessModifier_03();
        // System.out.println(am03.a); // 접근불가
        // System.out.println(am03.b); // 접근불가
        System.out.println(am03.c3); // 자손클래스를 통해 접근
        System.out.println(am03.d); // 접근가능

    }
}

```

구분	같은 패키지 (AccessModifier_01)	다른 패키지 + 자손 클래스 (AccessModifier_03)	다른 패키지 (AccessModifier_02)
Ex3_15	b, c, d	c, d	d

접근제어자가 private의 경우에는 같은 클래스가 아니면 접근이 불가능하다. 그래서 getA()를 이용하여 변수의 값을 얻을 수 있으며 변수의 값을 변경해야하는 경우에는 해당 변수의 값을 변경해 주는 메서드를 작성해야 한다.

#### ④ 캡슐화

클래스나 멤버, 주로 멤버에 접근 제어자를 사용하는 이유는 클래스의 내부에 선언된 데이터를 보호하기 위해서이다. 데이터가 유효한 값을 유지하도록, 또는 비밀번호와 같은 데이터를 외부에서 함부로 변경하지 못하도록 하기 위해서는 외부로부터의 접근을 제한하는 것이 필요하다.

이것을 데이터 감추기(data hiding)라고 하며, 객체지향개념의 캡슐화(encapsulation)에 해당하는 내용이다.

또 다른 이유는 클래스 내에서만 사용되는, 내부 작업을 위해 임시로 사용되는 멤버변수나 부분작업을 처리하기 위한 메서드 등의 멤버들을 클래스 내부에 감추기 위해서이다.

외부에서 접근할 필요가 없는 멤버들을 private으로 지정하여 외부에 노출시키지 않음으로써 복잡성을 줄일 수 있다. 이 것 역시 캡슐화에 해당한다.

#### 접근 제어자를 사용하는 이유

- 외부로부터 데이터를 보호하기 위해
- 외부에는 불필요한, 내부적으로만 사용되는, 부분을 감추기 위해

#### [문제 3-13] TimeVO (Time Variable Objcet) 만들기

```
public class Qu3_13{
    public static void main(String[] args){
        TimeVO tv = new TimeVO();
        tv.setTime(46);
        tv.setMinute(112);
        tv.setSecond(4853);
        System.out.println("현재시간 : "+ tv.getTime() + "시 " + tv.getMinute()
                           + "분 " + tv.getSecond() + "초");
    }
}

class TimeVO{
    private int hour;
    private int minute;
    private int second;

    public int getHour() {
        return hour;
    }

    public void setHour(int hour) {
```



```

//1. 시간은 0~23시까지 이다. 매개변수에 저장된 값이 0~23로만 저장되도록
//   setHour메서드를 구현 하여라.

}

public int getMinute() {
    return minute;
}

public void setMinute(int minute) {
    //2. 분은 0~59시까지 이다. 매개변수에 저장된 값이 0~59로만 저장되도록
    //   setMinute메서드를 구현 하여라.
    //   단. 분이 60을 넘는다면 1시간을 의미하므로 시간이 추가되도록 하여라.
    //   예) 143분 => 2시간 23분

}

public int getSecond() {
    return second;
}

public void setSecond(int second) {
    //3. 분은 0~59시까지 이다. 매개변수에 저장된 값이 0~59로만 저장되도록
    //   setMinute메서드를 구현 하여라.
    //   단. 분이 60을 넘는다면 1분을 의미하므로 분이 추가되도록 하여라.
    //   예) 3930초 => 1시간 5분 30초

}

}
}

```

결과 현재시간 : 1시 12분 53초

## 3-8. 다형성

## 학습목표

- 다형성의 특징을 활용하여 객체지향적 프로그래밍을 할 수 있다.

## 필요 지식 / 상속

### ① 다형성(polymorphism)

다형성이란 '여러 가지 형태를 가질 수 있는 능력'을 의미하며, 자바에서는 한 타입의 참조변수로 여러 타입의 객체를 참조할 수 있도록 함으로써 다형성을 프로그램적으로 구현하였다.

이를 좀 더 구체적으로 말하자면, 조상클래스 타입의 참조변수로 자손클래스의 인스턴스를 참조할 수 있도록 하였다는 것이다.

```
class Car {
    String color;
    int door;
    void drive(){
        System.out.println("출발~");
    }
}

class FireCar extends Car{
    void water(){
        System.out.println("물뿌려~");
    }
}
```

지금까지 우리는 생성된 인스턴스를 다루기 위해서, 인스턴스의 타입과 일치하는 타입의 참조변수만을 사용했다. 즉, Car인스턴스를 다루기 위해서는 Car타입의 참조변수를 사용하고, FireCar인스턴스를 다루기 위해서는 FireCar타입의 참조변수를 사용했다.

```
Car c = new Car();
FireCar fc = new FireCar();
```

이처럼 인스턴스의 타입과 참조변수의 타입이 일치하는 것이 보통이지만, Tv와 CaptionTv클래스가 서로 상속관계에 있을 경우, 다음과 같이 조상 클래스 타입의 참조변수로 자손 클래스의 인스턴스를 참조하도록 하는 것도 가능하다.

```
Car c = new FireCar(); // 조상 타입의 참조변수로 자손의 인스턴스를 참조
```

이제 인스턴스를 같은 타입의 참조변수로 참조하는 것과 조상타입의 참조변수로 참조하는 것은 어떤 차이가 있는지에 대해서 알아보도록 하자.

```
FireCar fc = new FireCar();  
Car c = new FireCar();
```

위의 코드에서 FireCar인스턴스 2개를 생성하고, 참조변수 c와 fc가 생성된 인스턴스를 하나씩 참조하도록 하였다. 이 경우 실제 인스턴스가 FireCar타입이라 할지라도, 참조변수 c로는 FireCar인스턴스의 모든 멤버를 사용할 수 없다.

Car타입의 참조변수로는 FireCar인스턴스 중에서 Car클래스의 멤버들(상속받은 멤버포함)만 사용할 수 있다. 따라서, 생성된 FireCar인스턴스의 멤버 중에서 Car클래스에 정의되지 않은 멤버, water()는 참조변수 c로 사용이 불가능하다. 즉, c.water()와 같이 할 수 없다는 것이다. 둘 다 같은 타입의 인스턴스지만 참조변수의 타입에 따라 사용할 수 있는 멤버의 개수가 달라진다.

반대로 아래와 같이 자손타입의 참조변수로 조상타입의 인스턴스를 참조하는 것은 가능할까?

1. FireCar fc1 = new Car(); // 컴파일 에러
2. FireCar fc2 = (FireCar) new Car(); // 런타임 에러

그렇지 않다. 위의 코드중 1번에서 컴파일 에러가 발생한다. 1번에서 발생한 컴파일 에러를 처리하기 위해서는 변수에서의 형변환과 같이 참조변수도 형변환이 가능하다. 그래서 Car클래스가 FireCar클래스의 상위클래스 이기 때문에 형변환을 해주어야 한다. 1번의 코드를 형변환하여 컴파일 에러를 처리한다면 2번의 예제가 된다. 하지만 2번의 경우에는 프로그램이 실행이 되었을때 에러가 발생하게 된다. 그 이유는 실제 인스턴스인 Car의 멤버 개수보다 참조변수 fc2가 사용할 수 있는 멤버 개수가 더 많기 때문이다. 그래서 이를 허용하지 않는다.

조상타입의 참조변수로 자손타입의 인스턴스를 참조할 수 있다.  
반대로 자손타입의 참조변수로 조상타입의 인스턴스를 참조할 수는 없다.

## ② 참조변수의 형변환

기본형 변수처럼 참조변수도 형변환이 가능하다. 단, 서로 상속관계에 있는 클래스사이에서만

가능하기 때문에 자손타입의 참조변수를 조상타입의 참조변수로, 조상타입의 참조변수를 자손타입의 참조변수로의 형변환만 가능하다.

※ 바로 윗 조상이나 자손이 아닌, 조상의 조상으로도 형변환이 가능하다. 따라서 모든 참조변수는 모든 클래스의 조상인 Object클래스 타입으로 형변환이 가능하다.



만일 위와 같이 Car클래스가 있고 이를 상속받는 FireEngine, Ambulance클래스가 있을때, FireEngine타입의 참조변수 f는 조상타입인 Car로 형변환 가능하다. 반대로 Car타입의 참조변수를 자손타입인 FireEngine으로 형변환하는 것도 가능하다. 그러나 FireEngine과 Ambulance는 상속관계가 아니므로 형변환이 불가능하다.

만일 위와 같이 Car클래스가 있고 이를 상속받는 FireEngine, Ambulance클래스가 있을때, FireEngine타입의 참조변수 f는 조상타입인 Car로 형변환 가능하다. 반대로 Car타입의 참조변수를 자손타입인 FireEngine으로 형변환하는 가능하다. 그러나 FireEngine과 Ambulance는 상속 관계가 아니므로 형변환이 불가능하다.

```
FireEngine f = new FireEngine();
Car c = (Car)f;           // OK. 조상인 Car타입으로 형변환(생략가능)
FireEngine f2 = (FireEngine)c; //OK. 자손인 FireEngine타입으로 형변환(생략불가)
Ambulance a = (Ambulance)f; // 에러. 상속관계가 아닌 클래스 간의 형변환 불가
```

기본형의 형변환과 달리 참조형의 형변환은 변수에 저장된 값(주소)이 변환되는 것이 아니다.

```
Car c = (Car)f;           // f의 값(객체의 주소)을 c에 저장.
                          // 타입을 일치시키기 위해 형변환 필요(생략가능)
f = (FireEngine) c;       // 조상타입을 자손타입으로 형변환하는 경우 생략불가
```

참조변수의 형변환은 그저 리모컨(참조변수)을 다른 종류의 것으로 바꾸는 것 뿐이다. 리모컨의 종류를 바꾸는 이유는 사용할 수 있는 멤버의 개수를 조절하기 위한 것이고, 그 이유는 곧 설명할 것이다. 위와 같이 조상 타입으로의 형변환을 생략할 수 있는 이유는 조상타입으

로 형변환하면 다룰 수 있는 멤버의 개수가 줄어들기 때문에 항상 안전하기 때문이다. 반대로 실제 객체가 가진 기능보다 리모컨의 버튼이 더 많으면 안된다.

서로 상속관계에 있는 타입간의 형변환은 양방향으로 자유롭게 수행될 수 있으나, 참조변수가 가리키는 인스턴스의 자손타입으로 형변환은 허용되지 않는다. 그래서 참조변수가 가리키는 인스턴스의 타입이 무엇인지 먼저 확인하는 것이 중요하다.

### [예제 3-17] 형변환

```
public class Ex3_17{
    public static void main(String[] args) {
        Car car = new Car();
        FireCar fe = new FireCar();
        car.drive();
        fe.water();
        car = fe; // up-casting : 형변환 생략
        //car.water(); // Car타입의 참조변수는 water()를 호출할수 없다.
        fe = (FireCar) car; // down-casting : 형변환 생략불가
        fe.water();
        Car car2 = new Ambulance();
        Ambulance fe2 = (Ambulance) car2; // 문법상 문제는 없지만 런타임시 에러
        fe2.stop();
    }
}

class Car{
    String color;
    int door;

    void drive(){
        System.out.println("드라이브 gogo~!");
    }

    void stop(){
        System.out.println("멈추시요~! stop");
    }
}

class FireCar extends Car{
    void water(){
        System.out.println("불났어..물뿌려..");
    }
}
```

```

    }
}

class Ambulance extends Car{
    void aed(){
        System.out.println("심폐소생");
    }
}

```

자손타입 --> 조상타입(Up-casting) : 형변환 생략가능  
 조상타입 --> 자손타입(Down-casting) : 형변환 생략불가

참조변수간의 형변환 역시 캐스트연산자를 사용하며, 괄호()안에 변환하고자 하는 타입의 이름(클래스명)을 적어주면 된다.

### ③ instanceof연산자

참조변수가 참조하고 있는 인스턴스의 실제 타입을 알아보기 위해 instanceof연산자를 사용한다. 주로 조건문에 사용되며, instanceof의 왼쪽에는 참조변수를 오른쪽에는 타입(클래스명)이 피연산자로 위치한다. 그리고 연산의 결과로 boolean값인 true와 false중의 하나를 반환한다.

instanceof를 이용한 연산결과로 true를 얻었다는 것은 참조변수가 검사한 타입으로 형변환이 가능하다는 것을 뜻한다.

```

void doWork(Car c){
    if(c instanceof FireCar){    //1. 인스턴스의 타입을 확인
        FireCar fc = (FireCar)c; //2. 형변환
        ....
    }
}

```

위의 코드는 Car타입의 참조변수 c를 매개변수로 하는 메서드이다. 이 메서드가 호출될 때, 매개변수로 Car클래스 또는 그 자손 클래스의 인스턴스를 넘겨받겠지만 메서드 내에서는 정확히 어떤 인스턴스인지 알 길이 없다. 그래서 instanceof연산자로 참조변수 c가 가리키고 있는 인스턴스의 타입을 체크하고, 적절히 형변환한 다음에 작업을 해야 한다.

조상타입의 참조변수로 자손타입의 인스턴스를 참조할 수 있기 때문에, 참조변수의 타입과 인스턴스의 타입이 항상 일치하지는 않는다는 것을 배웠다. 조상타입의 참조변수로 실제

인스턴스의 멤버들을 모두 사용할 수 없기 때문에, 실제 인스턴스와 같은 타입의 참조변수로 형변환을 해야만 인스턴스의 모든 멤버들을 사용할 수 있다.

#### [예제 3-18] instanceof연산자

```
public class Ex3_18 {
    public static void main(String[] args) {
        FireEngine2 fe = new FireEngine2();
        Car2 c2 = new Car2();
        if(fe instanceof FireEngine2){
            System.out.println("이것은 파이어엔진의 인스턴스이다.");
            FireEngine2 fe2 = fe;
        }
        if(fe instanceof Car2){
            System.out.println("이것은 카의 인스턴스 이다.");
            Car2 fe2 = fe;
        }
        if(fe instanceof Object){
            System.out.println("이것은 오브젝트의 인스턴스 이다.");
            Object fe2 = fe;
        }
        if(c2 instanceof FireEngine2){ // false임으로 실행 불가
            System.out.println("fe 는 c2의 인스턴스이다.");
            FireEngine2 fe2 = c2;
        }
        if(c2 instanceof Object){
            System.out.println("c2 는 Object의 인스턴스이다.");
            Object fe2 = fe;
        }
    }
}

class Car2{}

class FireEngine2 extends Car2{}
```

결과  
이것은 파이어엔진의 인스턴스이다.  
이것은 카의 인스턴스 이다.  
이것은 오브젝트의 인스턴스 이다.  
c2 는 Object의 인스턴스이다.

#### ④ 매개변수의 다형성과 여러객체 배열

##### 1. 매개변수의 다형성

참조변수의 다형적인 특징은 메서드의 매개변수에도 적용된다. 아래와 같이 Car, FireCar, Ambulance, Buyer클래스가 정의되어 있다고 가정하자.

<pre> class Car{ }  class FireCar extends Car{ }  class Ambulance extends Car{ } </pre>	<pre> class Buyer {     void buyFireCar(FireCar fc){ }     void buyAmbulance(Ambulance ab){ } } </pre>
---	--

Buyer클래스에서 Car를 구매하기 위한 메서드, FireCar를 구매하기 위한 메서드, Ambulance를 구매하기 위한 메서드를 만들어 놓았다. 하지만 차의 종류가 추가 된다면 Buyer클래스의 메서드도 추가되어야 하는 단점이 있다. 이를 매개변수의 다형성을 이용하여 해결할 수 있다.

<pre> class Buyer {     void buyCar(Car c){ } } </pre>
--

위와 같이 메서드를 작성한다면 매개변수의 다형성을 이용하여 buyCar메서드의 매개변수로 Car클래스를 상속받는 모든 자손클래스를 매개변수로 받을수 있게 된다.

## 2. 여러 종류의 객체 배열 - Vector

조상타입의 참조변수로 자손타입의 객체를 참조하는 것이 가능하므로, Product클래스가 Tv, Computer클래스의 조상일 때 다음과 같이 할 수 있는 것을 이미 배웠다.

```

Product p1 = new Tv();
Product p2 = new Computer();

```

같은 타입의 여러 변수를 하나의 변수로 다루기 위해 배열을 활용한다면 다음과 같다.

```

Product[] p = new Product[2];
p[0] = new Tv();
p[1] = new Computer();

```

이처럼 조상타입의 참조변수 배열을 사용하면, 공통의 조상을 가진 서로 다른 종류의 객체를 배열로 묶어서 다룰 수 있다. 또는 묶어서 다루고 싶은 객체들의 상속관계를 따져서 가장 가까운 공통조상 클래스 타입의 참조변수 배열을 생성해서 객체들을 저장하면 된다.

위의 내용을 이용하여 [문제 3-14]에 구매자가 구매한 내역을 만들 수 있다. 하지만 배열을



생성할 때 처음의 크기는 10으로 만든다면 10개 이상의 구매내역은 저장할 수 없게 된다. 이를 처리하기 위해 크기를 크기 만든다면 그만큼 메모리의 낭비를 가져올 수 있다.

이런 경우, Vector클래스를 사용하면 되나. Vector클래스는 내부적으로 Object타입의 배열을 가지고 있어서, 이 배열에 객체를 추가하거나 제거할 수 있게 작성되어 있다. 또한 배열의 크기를 알아서 관리해 주기 때문에 저장할 인스턴스의 개수에 신경 쓰지 않아도 된다.

메서드/생성자	설명
Vector()	10개의 객체를 저장할 수 있는 Vector인스턴스를 생성한다. 10개 이상의 인스턴스가 저장되면, 자동적으로 크기가 증가된다.
boolean add(Object o)	Vector에 객체를 추가한다. 추가에 성공하면 결과값으로 true, 실패하면 false를 반환한다.
boolean remove(Object o)	Vector에 저장되어 있는 객체를 제거한다. 제거에 성공하면 true, 실패하면 false를 반환한다.
boolean isEmpty()	Vector가 비어있는지 검사한다. 비어있으면 true, 비어있지 않으면 false를 반환한다.
Object get(int index)	지정된 위치(index)의 객체를 반환한다. 반환타입이 Object 타입이므로 저장한 타입으로의 형변환이 필요하다.
int size()	Vector에 저장된 객체의 개수를 반환한다.

[표 3-2] Vector의 메서드와 생성자

[문제 3-14]의 Buyer클래스에 회원의 구매내역을 저장할 인스턴스변수 item을 선언 및 생성하며 아래와 같다.

**Vector<클래스명> item = new Vector<클래스명>();**

'<클래스명>'은 생략이 가능하며 생략한다면 item에 Object타입(최상위 타입)으로 저장이 되게 된다. 하지만 클래스명에 'Car'를 입력 시 item에는 Car타입과 Car를 상속받는 하위 클래스 타입만 저장이 가능해 진다. 배열에 사용되는 '변수타입'이라고 생각하면 편하다.

물건을 구매 시 Vector클래스의 add메서드를 이용하여 구매내역을 추가해 주면 된다.

#### [문제 3-14] 물건구매

```
public class Qu3_14{
    public static void main(String[] args){
        //12. Tv의 객체를 생성하여라. - 가격 : 200
    }
}
```

```

//13. Computer의 객체를 생성하여라. - 가격 : 300

//14. Buyer의 객체를 생성하여라. - 돈 : 1000

//15. Computer를 구매하여라.

//16. Tv를 2대 구매하여라.

//17. 구매한 물품의 계산서를 출력하여라.

//18. Tv한대를 반품하여라.

//19. 구매한 물품의 계산서를 출력하여라.

}
}

class Product{
    int price;
    int bonusPoint;
    //1. 매개변수로 price하나를 가지는 생성자를 작성하여라.
    // 매개변수의 값으로 인스턴스변수 price를 초기화 하고 가격의 10%에 해당되는
    // 금액을 bonusPoint에 저장하여라.

}

class Tv extends Product{
    //2. 매개변수가 하나인 생성자를 작성하여라. (매개변수 : int타입의 price)
    // Product의 매개변수가 하나인 생성자를 호출하여라. 이때 인자로 매개변수로 받은
    // price에 저장된 값을 넘겨 준다.

    //3. Object클래스의 toString()를 오버라이드 하여 “Tv”를 반환하도록 하여라.

```

```

}

class Computer extends Product{
    //4. 매개변수가 하나인 생성자를 작성하여라. (매개변수 : int타입의 price)
    //   Product의 매개변수가 하나인 생성자를 호출하여라. 이때 인자로 매개변수로 받은
    //   price에 저장된 값을 넘겨 준다.

    //5. Object클래스의 toString()를 오버라이드 하여 “Computer“를 반환하도록 하여라.

}

class Buyer{
    int money;
    int bonusPoint;
    Vector item = new Vector(); // 구매내역을 저장

    //6. 매개변수가 하나인 생성자를 작성하여라. (매개변수 : int타입의 money)
    //   매개변수의 저장된 값으로 인스턴스변수 money의 값을 초기화 하여라.

    //7. Tv를 구매하는 buy메서드를 작성하여라.

    //8. Computer를 구매하는 buy메서드를 작성하여라.

    //9. 매개변수의 다형성을 이용하여 7,8번 메서드를 하나로 작성하여라.
    //   물건의 종류가 늘어나도 메서드를 여러 개 만들지 않아도 된다.

```

//10. 구매한 물품을 반품하는 refund메서드를 작성하여라.

//11. 지금까지 구매한 내역을 결과와 같이 출력하는 summary메서드를 작성하여라.

```
}
    결과
    영 수 증
    구매내역
        Tv : 100만원
        Computer : 200만원
        Tv : 100만원

    구매하신 물품의 총 금액은 400만원이며
    고객님의 보너스 포인트는 40만 포인트 입니다.
    이용해 주셔서 감사합니다.
```

### 3-8. 추상클래스

## 학습목표

- 응용소프트웨어 개발에 필요한 하드웨어 및 소프트웨어의 필요 사항을 검토하고 이에 따라, 개발환경에 필요한 준비를 수행할 수 있다.

## 필요 지식 /

### ① 추상 클래스( abstract class )

#### 1. 추상 메서드

메서드는 선언부와 구현부(몸통)로 구성되어 있다고 했다. 선언부만 작성하고 구현부는 작성하지 않은 채로 남겨 둔 것이 추상메서드이다. 즉, 설계만 해 놓고 실제 수행될 내용은 작성하지 않았기 때문에 미완성 메서드인 것이다.

메서드를 이와 같이 미완성 상태로 남겨 놓는 이유는 메서드의 내용이 상속받는 클래스에 따라 달라질 수 있기 때문에 조상 클래스에서는 선언부만을 작성하고, 주석을 덧붙여 어떤 기능을 수행할 목적으로 작성되었는지 알려 주고, 실제 내용은 상속받는 클래스에서 구현하도록 비워 두는 것이다. 그래서 추상클래스를 상속받는 자손 클래스는 조상의 추상메서드를 상황에 맞게 적절히 구현해주어야 한다.

추상메서드 역시 키워드 'abstract'를 앞에 붙여 주고, 추상메서드는 구현부가 없으므로 괄호 {} 대신 문장의 끝을 알리는 ';'을 적어준다.

**abstract 리턴타입 메서드이름();**

추상클래스로부터 상속받는 자손클래스는 오버라이딩을 통해 조상인 추상클래스의 추상메서드를 모두 구현해주어야 한다. 만일 조상으로부터 상속받은 추상메서드 중 하나라도 구현하지 않는다면, 자손클래스 역시 추상클래스로 지정해 주어야 한다.

실제 작업내용인 구현부가 없는 메서드가 무슨 의미가 있을까 싶기도 하겠지만, 메서드를 작성할 때 실제 작업내용인 구현부보다 더 중요한 부분이 선언부이다.

메서드를 사용하는 쪽에서는 메서드가 실제로 어떻게 구현되어있는지 몰라도 메서드의 이름과 매개변수, 리턴타입, 즉 선언부만 알고 있으면 되므로 내용이 없을 지라도 추상메서드를 사용하는 코드를 작성하는 것이 가능하며, 실제로는 자손클래스에 구현된 완성된 메서드가 호출되도록 할 수 있다.

#### 2. 추상 클래스

클래스를 설계도에 비유한다면, 추상 클래스는 미완성 설계도에 비유할 수 있다. 미완성 설계도란, 단어의 뜻 그대로 완성되지 못한 채로 남겨진 설계도를 말한다.

클래스가 미완성이라는 것은 멤버의 개수에 관계된 것이 아니라, 단지 미완성 메서드(추상메서드)를 포함하고 있다는 의미이다.

미완성 설계도로 완성된 제품을 만들 수 없듯이 추상 클래스로 인스턴스는 생성할 수 없

다. 추상 클래스는 상속을 통해서 자손클래스에 의해서만 완성될 수 있다.

**추상클래스**   미완성 설계도, 인스턴스 생성불가.  
미완성 메서드(추상 메서드)를 포함하고 있는 클래스

추상 클래스 자체로는 클래스로서의 역할을 다 못하지만, 새로운 클래스를 작성하는데 있어서 바탕이 되는 조상 클래스로서 중요한 의미를 갖는다. 새로운 클래스를 작성할 때 아무것도 없는 상태에서 시작하는 것보다는 완전하지는 못하더라도 어느 정도 틀을 갖춘 상태에서 시작하는 것이 나을 것이다.

실생활에서 예를 들자면, 같은 크기의 TV라도 기능의 차이에 따라 여러 종류의 모델이 있지만, 사실 이 들의 설계도는 아마 90%정도는 동일할 것이다. 서로 다른 세 개의 설계도를 따로 그리는 것보다는 이들의 공통부분만을 그린 미완성 설계도를 만들어 놓고, 이 미완성 설계도를 이용해서 각각의 설계도를 완성하는 것이 훨씬 효율적일 것이다.

추상 클래스는 키워드 'abstract'를 붙이기만 하면 된다. 이렇게 함으로써 이 클래스를 사용할 때, 클래스 선언부의 abstract를 보고 이 클래스에는 추상메서드가 있으니 상속을 통해서 구현해주어야 한다는 것을 쉽게 알 수 있을 것이다.

```
abstract class 클래스이름 {
    .....
}
```

추상 클래스는 추상 메서드를 포함하고 있다는 것을 제외하고는 일반 클래스와 전혀 다르지 않다. 추상 클래스에도 생성자가 있으며, 멤버변수와 메서드도 가질 수 있다.

#### [예제 3-19] 이차원 게임을 이용한 상위 클래스 만들기

상속 적용 전	상속 적용 후
<pre>class Marine {     int x;     int y;     void move(int x, int y){ }     void stop(){ }     void run(){ } }  class Tank {     int x;     int y;</pre>	<pre>class Unit {     int x;     int y;     void move(int x, int y){ }     void stop(){ } }  class Marine extends Unit{     void run(){ } }</pre>

<pre> void move(int x, int y){ } void stop(){ } void changeMode(){ } }  class Dropship {     int x;     int y;     void move(int x, int y){ }     void stop(){ }     void load(){ } } </pre>	<pre> class Tank extends Unit{     void changeMode(){ } }  class Dropship extends Unit{     void load(){ } } </pre>
--	---

상속에서 배운 상속을 이용하여 Marine, Tank, Dropship의 공통 멤버인 x,y좌표와 move(), stop()를 포함한 Unit클래스를 만들어진 예이다. 하지만 Marine의 이동방식은 걸어서 이동, Tank의 이동방식은 바퀴를 굴려서 이동, Dropship의 이동방식은 날아서 이동이고 이동속도 또한 다르다. 그렇기 때문에 Unit클래스는 '이동한다'라는 공통의 행위만 가지게 되고 Unit클래스를 상속받은 각 각이 이동방식을 구현하는 방식을 취하면 예제3-20의 형태로 클래스를 구성하게 된다.

#### [예제 3-20] 이차원 게임을 이용한 추상 메서드와 추상 클래스 만들기

<pre> /* 추상 메서드를 가지고 있음으로 추상    클래스로 변경 */  abstract class Unit {     int x;     int y;     /* 추상메서드로 변경 */     abstract void move(int x, int y);     void stop(){ } } </pre>	<pre> /* 추상 클래스를 상속받는 클래스의 구현    방법 */  //1. 추상클래스가 되는 방법 abstract class Marine extends Unit{     void run(){ } }  //2. 추상메서드 구현하는 방법 class Tank extends Unit{     void changeMode(){ }      @override     void move(int x, int y){         System.out.println("엔진가동");     } }  class Dropship extends Unit{     void load(){ } } </pre>
---	---

```
@override  
void move(int x, int y){  
    System.out.println("날아서 이동");  
}  
}
```



## 3-9. 인터페이스

### 학습목표

- 응용소프트웨어 개발에 필요한 하드웨어 및 소프트웨어의 필요 사항을 검토하고 이에 따라, 개발환경에 필요한 준비를 수행할 수 있다.

### 필요 지식 / 추상클래스

#### 1 인터페이스 (interface)

##### 1. 인터페이스

인터페이스는 일종의 추상클래스이다. 인터페이스는 추상클래스처럼 추상메서드를 갖지만 추상클래스보다 추상화 정도가 높아서 추상클래스와 달리 몸통을 갖춘 일반 메서드 또는 멤버 변수를 구성원으로 가질 수 없다. 오직 추상메서드와 상수만을 멤버로 가질 수 있으며, 그 외의 다른 어떠한 요소도 허용하지 않는다.

추상클래스를 부분적으로만 완성된 '미완성 설계도'라고 한다면, 인터페이스는 구현된 것은 아무 것도 없고 밑그림만 그려져 있는 '기본 설계도'라 할 수 있다.

인터페이스도 추상클래스처럼 완성되지 않은 불완전한 것이기 때문에 그 자체만으로 사용되기 보다는 다른 클래스를 작성하는데 도움 줄 목적으로 작성된다.

인터페이스를 작성하는 것은 클래스를 작성하는 것과 같다. 다만 키워드로 class 대신 interface를 사용한다는 것만 다르다. 그리고 interface에도 클래스처럼 접근제어자로 public 또는 default만 사용할 수 있다.

```
interface 인터페이스이름{
    public static final 타입 상수이름 = 값;
    public abstract 메서드이름(매개변수 목록);
}
```

일반적인 클래스와 달리 인터페이스의 멤버들은 다음과 같은 제약사항이 있다.

- 모든 멤버변수는 public static final 이어야 하며, 이를 생략할 수 있다.
- 모든 메서드는 public abstract 이어야 하며, 이를 생략할 수 있다.  
단, static메서드와 디폴트 메서드는 예외 (JDK1.8부터)

인터페이스에 정의된 모든 멤버에 예외없이 적용되는 사항이기 때문에 제어자를 생략할 수 있는 것이며, 편의상 생략하는 경우가 많다. 생략된 제어자는 컴파일 시에 컴파일러가 자동적으로 추가해준다.

## 2. 인터페이스 상속

인터페이스는 인터페이스로부터만 상속받을 수 있으며, 클래스와는 달리 다중상속, 즉 여러 개의 인터페이스로부터 상속을 받는 것이 가능하다.

※ 인터페이스는 클래스와 달리 Object클래스와 같은 최고 조상이 없다.

```
interface Movable{
    void move(int x, int y);
}
interface Attackable{
    void attack(Unit u);
}
interface Fightable extends Movable, Attackable{ }
```

자손 인터페이스(Fightable)는 조상 인터페이스(Movable, Attackable)에 정의된 멤버를 모두 상속받는다.

그래서 Fightable 자체에는 정의된 멤버가 하나도 없지만 조상 인터페이스로부터 상속받은 두 개의 추상메서드, move(int x, int y)와 attack(Unit u)을 멤버로 갖게 된다.

인터페이스가 멤버로 가질 수 있는 멤버는 상수와 추상 메서드뿐이기 때문에 인터페이스가 클래스를 상속받게 된다면 구현된 메서드나 일반 변수를 멤버로 가지게 되기 때문에 인터페이스의 특징을 잃어버리게 된다.

## 3. 인터페이스 구현

인터페이스도 추상클래스처럼 그 자체로는 인스턴스를 생성할 수 없으며, 추상클래스가 상속을 통해 추상메서드를 완성하는 것처럼, 인터페이스도 자신에 정의된 추상메서드의 몸통을 만들어주는 클래스를 작성해야 하는데, 그 방법은 추상클래스가 자신을 상속받는 클래스를 정의하는 것과 다르지 않다. 다만 클래스는 확장한다는 의미의 키워드 'extends'를 사용하지만 인터페이스는 구현한다는 의미의 키워드 'implements'를 사용할 뿐이다.

```
class 클래스이름 implements 인터페이스 이름{
    // 인터페이스에 정의된 추상메서드를 모두 구현해야 한다.
}
```

만일 구현하는 인터페이스의 메서드 중 일부만 구현한다면, abstract를 붙여서 추상클래스로 선언해야 한다.

[예제 3-20] 인터페이스의 활용

```
public class Ex3_20 {
    public static void main(String[] args) {
        Fighter f = new Fighter();
        if(f instanceof Unit3){
            System.out.println("f는 Unit클래스의 자손 입니다.");
            Unit3 u3 = f;
        }

        if(f instanceof Fightable){
            System.out.println("f는 Fightable 인터페이스를 구현하였습니다.");
            Fightable f3 = f;
        }

        if(f instanceof Movable){
            System.out.println("f는 Movable 인터페이스를 구현하였습니다.");
            Movable m3 = f;
        }

        if(f instanceof Attackable){
            System.out.println("f는 Attackable 인터페이스를 구현하였습니다.");
            Attackable a3 = f;
        }

        if(f instanceof Object){
            System.out.println("f는 Object클래스의 자손 입니다.");
            Object o3 = f;
        }
    }
}

class Fighter extends Unit3 implements Fightable{
    //1. public접근 제어자 지정
    @Override
    public void move(int x, int y){ }
    @Override
    public void attack(Unit u){ }
}
```

```

class Unit3{
    int currentHp; //유닛의 체력
    int x; //유닛의 위치 x좌표
    int y; //유닛의 위치 y좌표
}

interface Fightable extends Movable, Attackable{ }

interface Movable{
    //2. public abstract 가 생략 되어져 있다.
    void move(int x, int y);
}

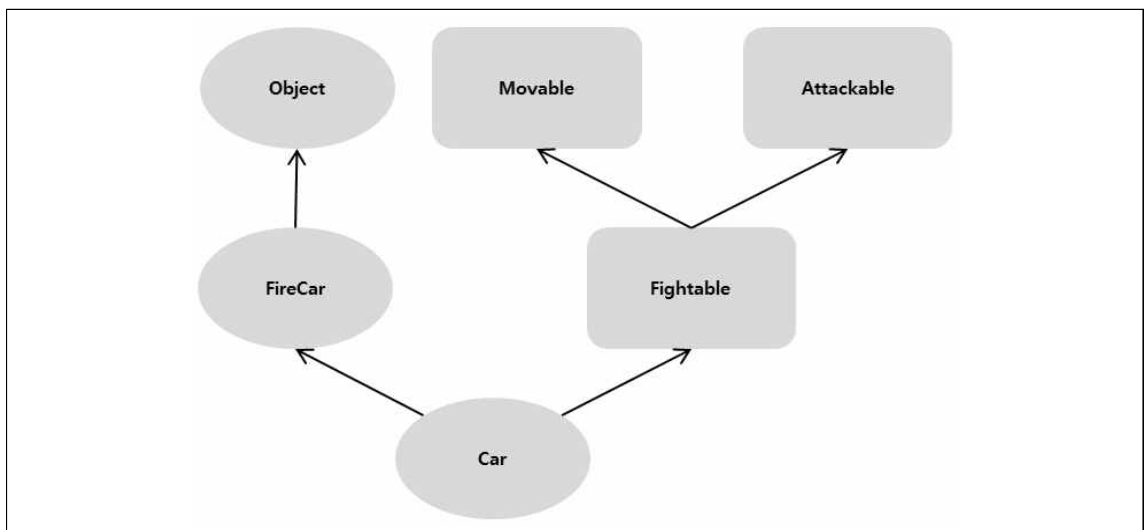
interface Attackable{
    void attack(Unit u);
}

```

결과

f는 Unit클래스의 자손입니다.  
f는 Fightable 인터페이스를 구현하였습니다.  
f는 Movable 인터페이스를 구현하였습니다.  
f는 Attackable 인터페이스를 구현하였습니다.  
f는 Object클래스의 자손입니다.

예제 3-20에 사용된 클래스와 인터페이스간의 관계를 그려보면 다음과 같다.



인터페이스는 상속 대신 구현이라는 용어를 사용하지만, 인터페이스로부터 상속받은 추상메서드를 구현하는 것이기 때문에 인터페이스도 조금은 다른 의미의 조상이라고 할 수 있다. 여기서 주의 깊게 봐두어야 할 것은 Movable인터페이스에 정의된 'void move(int x, int y)'를 Fightable클래스에서 구현할 때 접근 제어자를 public으로 했다.

오버라이딩 할 때는 조상의 메서드보다 넓은 범위의 접근 제어자를 지정해야 한다는 것을 기억할 것이다. Movable인터페이스에 void move(int x, int y)'와 같이 정의되어 있지만 사실 'public abstract'가 생략된 것이다. 그래서, 이를 구현하는 Fighter클래스에서는 move메서드의 접근제어자는 반드시 public으로 해야 한다.

#### 4. 인터페이스를 이용한 다형성

다형성을 학습할 때 자손클래스의 인스턴스를 조상타입의 참조변수로 참조하는 것이 가능하다는 것을 배웠다. 인터페이스 역시 이를 구현한 클래스의 조상이라 할 수 있으므로 해당 인터페이스 타입의 참조변수로 이를 구현한 클래스의 인스턴스를 참조할 수 있으며, 인터페이스 타입으로의 형변환도 가능하다.

인터페이스 `Fightable`을 클래스 `Fighter`가 구현했을 때, 다음과 같이 `Fighter`인스턴스를 `Fightable`타입의 참조변수로 참조하는 것이 가능하다.

```
Fightable f = (Fightable) new Fighter(); // 캐스트 연산자 생략 가능
```

인터페이스 타입의 참조변수로 이 인터페이스를 구현한 클래스의 인스턴스를 참조할 수 있다.

## 5. 인터페이스의 장점

인터페이스를 사용하는 이유와 그 장점을 정리해 보면 다음과 같다.

### (1) 개발시간을 단축시킬 수 있다.

일단 인터페이스가 작성되면, 이를 사용해서 프로그램을 작성하는 것이 가능하다. 메서드를 호출하는 쪽에서는 메서드의 내용에 관계없이 선언부만 알면 되기 때문이다.

그리고 동시에 다른 한 쪽에서는 인터페이스를 구현하는 클래스를 작성하게 하면, 인터페이스를 구현하는 클래스가 작성될 때까지 기다리지 않고도 양쪽에서 동시에 개발을 진행할 수 있다.

### (2) 표준화가 가능하다.

프로젝트에 사용되는 기본 틀을 인터페이스로 작성한 다음, 개발자들에게 인터페이스를 구현하여 프로그램을 작성하도록 함으로써 보다 일관되고 정형화된 프로그램의 개발이 가능하다.

### (3) 서로 관계없는 클래스들에게 관계를 맺어 줄 수 있다.

서로 상속관계에 있지도 않고, 같은 조상클래스를 가지고 있지 않은 서로 아무런 관계도 없는 클래스들에게 하나의 인터페이스를 공통적으로 구현하도록 함으로써 관계를 맺어 줄 수 있다.

### (4) 독립적인 프로그래밍이 가능하다.

인터페이스를 이용하면 클래스의 선언과 구현을 분리시킬 수 있기 때문에 실제구현에 독립적인 프로그램을 작성하는 것이 가능하다. 클래스와 클래스간의 직접적인 관계를 인터페이스를 이용해서 간접적인 관계로 변경하면, 한 클래스의 변경이 관련된 다른 클래스에 영향을 미치지 않는 독립적인 프로그래밍이 가능하다.

## 학습 4 예외처리

### 4-1. 예외처리

#### 학습목표

- 프로그램 오류에 대한 예외처리를 활용할 수 있다.

#### 필요 지식 /

#### ① 프로그램 오류

##### 1. 프로그램 오류 개요

프로그램이 실행 중 어떤 원인에 의해서 오작동을 하거나 비정상적으로 종료되는 경우가 있다. 이러한 결과를 초래하는 원인은 프로그램 에러 또는 오류라고 한다.

이를 발생시점에 따라 '컴파일 에러'와 '런타임 에러'로 나눌 수 있는데, 글자 그대로 '컴파일 에러'는 컴파일 할 때 발생하는 에러이고 프로그램의 실행도중에 발생하는 에러를 '런타임 에러'라고 한다.

<b>컴파일 에러</b>	컴파일 시에 발생하는 에러
<b>런타임 에러</b>	실행 시에 발생하는 에러

소스코드를 컴파일 하면 컴파일러가 소스코드에 대해 오타나 잘못된 구문, 자료형 체크 등의 기본적인 검사를 수행하여 오류가 있는지를 알려 준다. 이때 발생하는 에러가 컴파일 에러이다. 이러한 컴파일 에러들을 모두 수정해서 컴파일을 성공적으로 마치고 나면, 클래스 파일이 생성되고, 생성된 클래스 파일을 실행할 수 있게 되는 것이다.

하지만 컴파일을 에러없이 성공적으로 마쳤다고 해서 프로그래밍의 실행 시에도 에러가 발생하지 않는 것이 아니다. 컴파일러가 소스코드의 기본적인 사항은 컴파일 시에 모두 걸러줄 수는 있지만, 실행도중에 발생할 수 있는 잠재적인 오류까지 검사할 수 없기 때문에 컴파일은 잘되었어도 실행 중에 에러에 의해서 잘못된 결과를 얻거나 프로그램이 비정상적으로 종료될 수 있다.

런타임 에러를 방지하기 위해서는 프로그램의 실행도중 발생할 수 있는 모든 경우의 수를 고려하여 이에 대한 대비를 하는 것이 필요하다.

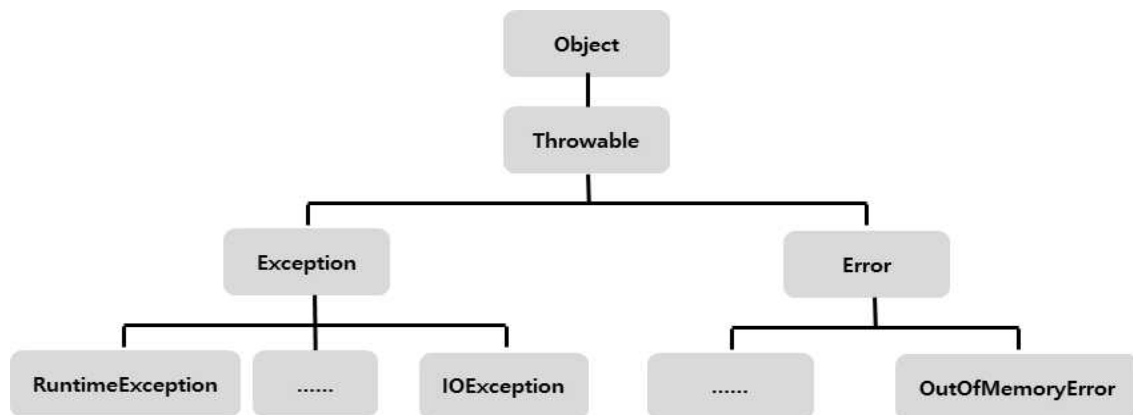
자바에서는 프로그램 실행 시 발생할 수 있는 프로그램 오류를 '에러(error)'와 '예외(Exception)', 두 가지로 구분하였다.

에러가 발생하면, 프로그램의 비정상적인 종료를 막을 길이 없지만, 예외는 발생하더라도 프로그래머가 이에 대한 적절한 코드를 미리 작성해 놓음으로써 프로그램의 비정상적인 종료를 막을 수 있다.

<b>에러(error)</b>	프로그램 코드에 의해서 수습될 수 없는 심각한 오류
<b>예외(exception)</b>	프로그램 코드에 의해서 수습될 수 있는 다소 미약한 오류

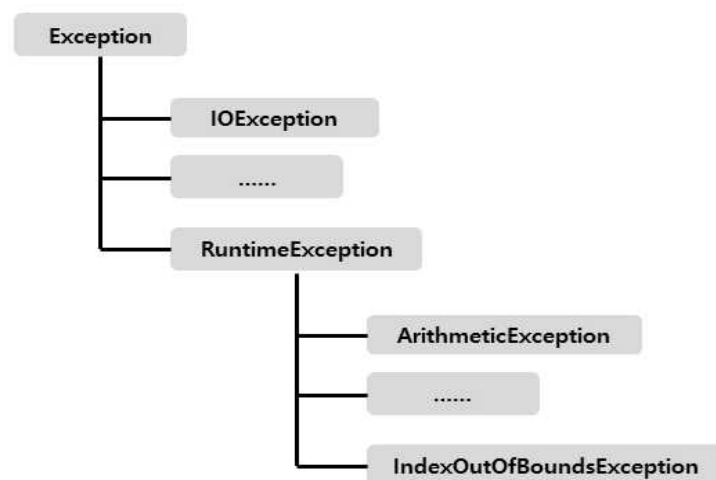
## 2. 예외 클래스의 계층구조

자바에서 실행 시 발생할 수 있는 오류(Exception과 Error)를 클래스로 정의하였다. 앞서 배운 것처럼 모든 클래스의 조상은 Object클래스이므로 Exception과 Error클래스 역시 Object클래스의 자손들이다.



[그림 4-1] 예외클래스 계층도

모든 예외의 최고 조상은 Exception클래스이며, 상속계층도를 Exception클래스부터 도식화하면 다음과 같다.



[그림 4-2] Exception클래스의 상속 계층도

그림 4-2에서 볼 수 있듯이 예외 클래스들은 다음과 같이 두 그룹으로 나뉘질 수 있다.

- ① RuntimeException클래스와 그 자손들
- ② Exception클래스와 그 자손들 (RuntimeException 제외)

앞으로 RuntimeException클래스와 그 자손 클래스들을 'RuntimeException클래스들'이라 하고, 그 외 클래스들을 'Exception클래스들'이라 하겠다.

RuntimeException클래스들은 주로 프로그래머의 실수에 의해서 발생할 수 있는 예외들로 자바의 프로그래밍 요소들과 관계가 깊다. 예를 들면, 배열을 배우면서 많이 보았던 배열의 범위를 벗어나던가(IndexOutOfBoundsException), 객체지향 프로그래밍을 배우면서 보았던 객체를 생성하지 않고 값이 null인 참조변수의 멤버를 호출 했다면가(NullPointerException) 하는 경우에 발생하게 된다.

Exception클래스들은 주로 외부의 영향으로 발생할 수 있는 것들로서, 프로그램의 사용자들의 동작에 의해서 발생하는 경우가 많다. 예를 들면, 존재하지 않는 파일의 이름을 입력 했다면가 (FileNotFoundException), 입력한 데이터 형식이 잘못된(DataFormatException) 경우에 발생한다.

Exception클래스들	외적인 요인에 의해 발생하는 예외
RuntimeException클래스들	프로그래머의 실수로 발생하는 예외

### 3. 예외 처리하기 (try-catch)

프로그램의 실행도중에 발생하는 에러는 어쩔 수 없지만, 예외는 프로그래머가 이에 대한 처리를 미리 해주어야 한다.

예외처리란? 프로그램 실행 시 발생할 수 있는 예기치 못한 예외의 발생에 대비한 코드를 작성하는 것이며, 예외처리의 목적은 예외의 발생으로 인한 실행 중인 프로그램의 갑작스런 비정상 종료를 막고, 정상적인 실행상태를 유지할 수 있도록 하는 것이다.

예외를 처리하기 위해서는 try-catch문을 사용하며, 그 구조는 다음과 같다.

```
try{
    // 예외가 발생할 가능성이 있는 문장을 넣는다.
} catch (Exception e1) {
    // Exception1이 발생하였을 경우, 이를 처리하기 위한 문장을 적는다.
}
```



하나의 try블럭 다음에는 여러 종류의 예외를 처리할 수 있도록 하나 이상의 catch블럭이 올 수 있으며, 이 중 발생한 예외의 종류와 일치하는 단 한 개의 catch블럭만 수행된다. 발생한 예외의 종류와 일치하는 catch블럭이 없으면 예외는 처리되지 않는다.

#### [예제 4-1] 오류처리

```
public class Ex4_1 {
    public static void main(String[] args) {
        int number = 100;
        int result = 0;
        // Math.random()을 사용하였기 때문에 결과는 다를 수 있다.
        for(int i=0; i < 10; i++){
            result = number/(int)(Math.random()*5);
            System.out.println(result);
        }

        // 반복횟수는 10회이지만 3회만 진행하고 프로그램이 종료 되었다.
        // 예외가 발생하더라도 프로그램을 진행 되어야 한다.
        for (int i = 0; i < 10; i++) {
            try {
                result =number/(int)(Math.random()*5);
                System.out.println(result);
            } catch (ArithmeticException e) {
                System.out.println("0으로 나누지 마시요.");
            }
        }
    }
}
```

**결과**

```
12
20
11
Exception in thread "main"
java.lang.ArithmeticException: / by zero
at G_exception.ExceptionEx.main(ExceptionEx.java:9)
```

**결과**

```
0으로 나누지 마시요.
100
100
25
0으로 나누지 마시요.
50
0으로 나누지 마시요.
25
0으로 나누지 마시요.
```

위의 예제는 변수 number에 저장되어 있는 값 100을 0~5사이의 임의의 정수로 나눈 결과를 출력하는 일을 10번 반복한다.

random()을 사용했기 때문에 매번 실행할 때마다 결과가 다르겠지만, 대부분의 경우 10번이 출력되기 이전에 예외가 발생하여 프로그램이 비정상적으로 종료될 것이다.

결과에 나타난 메시지를 보면 예외의 발생원인과 위치를 알 수 있다. 이 메시지를 보면, 0으로 나누려(/ by zero) 했기 때문에 'ArithmeticException'이 발생했고, 예외가 발생한 위치는 ExceptionEx클래스의 main메서드의 9번째 라인에서 발생하였다는 것을 알 수 있다.

예외가 발생할 가능성이 있는 문장((int)(Math.random()\*5))을 try-catch를 이용하여 예외처리를 진행한 결과 1,5,7,9번째 ArithmeticException이 발생하였지만 10회 반복되었고 예외가 발생하였을 때 '0으로 나누지 마시요'로 처리가 되었다.

#### 4. 예외처리 흐름

try-catch문에서, 예외가 발생한 경우와 발생하지 않았을 때의 흐름(문장의 실행순서)이 달라 지는데, 아래에 이 두 가지 경우에 따른 문장 실행순서를 정리 하였다.

##### 1. try블럭 내에서 예외가 발생한 경우

- 발생한 예외와 일치하는 catch블럭이 있는지 확인한다.
- 일치하는 catch블럭을 찾게 되면, 그 catch블럭 내의 문장을 수행하고 try-catch문을 빠져나가서 그 다음 문장을 계속해서 수행한다. 만일 일치하는 catch블럭을 찾지 못 하면, 예외는 처리되지 못한다.

##### 2. try블럭 내에서 예외가 발생하지 않을 경우

- catch블럭을 거치지 않고 전체 try-catch문을 빠져나가서 수행을 계속한다.

#### [예제 4-2] try-catch흐름

```
public class Ex4_2 {  
    public static void main(String[] args) {  
        int a = 3;  
        int b = 0;  
        try {  
            System.out.println(a);  
            System.out.println(a/b);  
            System.out.println(4);  
        } catch (ArithmeticException e) {  
            e.printStackTrace();  
            System.out.println(5);  
        } catch (Exception e){  
            System.out.println(e.getMessage());  
            System.out.println(6);  
        }  
        System.out.println(7);  
    }  
}
```

결과

```
3  
java.lang.ArithmeticException: / by zero  
at G_exception.ExceptionEx2.main(ExceptionEx2.java:13)  
5  
7
```

위의 예제의 결과를 보면, 3을 출력한 다음 try블럭에서 예외가 발생했기 때문에 try블럭을 바로 벗어나서 'System.out.println(4);'는 실행되지 않는다. 그리고는 발생한 예외에 해당하는 catch블럭으로 이동하여 문장들을 수행하여 5를 출력하게 된다. 다음엔 전체 try-catch문을 벗어나서 그 다음 문장을 실행하여 7을 출력하게 된다.

try블럭에서 예외가 발생하면, 예외가 발생한 위치 이후에 있는 try블럭의 문장들을 수행되지 않으므로 try블럭에 포함시킬 코드의 범위를 잘 선택해야 한다.

예제 4-2의 결과에서 2,3번 줄을 보게 되면 예외가 발생했지만 프로그램은 진행된 것처럼 보이겠지만 사실은 'e.printStackTrace()'가 출력한 메시지이다. 또한 try블럭 내에서 Arithmetic Exception이 아닌 다른 예외가 발생하였다면 'e.getMessage()'와 6을 출력하게 된다.

예외가 발생했을 때 생성되는 예외 클래스의 인스턴스에는 발생한 예외에 대한 정보가 담겨져 있으며, getMessage()와 printStackTrace()를 통해서 이 정보들을 얻을 수 있다.

catch블럭의 괄호()dp 선언된 참조변수를 통해 이 인스턴스에 접근할 수 있다. 이 참조변수는 선언된 catch블럭 내에서만 사용 가능하다. 자주 사용되는 메서드는 다음과 같다.

<b>printStackTrace()</b>	예외발생 당시에 호출스택(Call Stack)에 있었던 메서드의 정보와 예외 메시지를 화면에 출력한다.
<b>getMessage()</b>	발생한 예외클래스의 인스턴스에 저장된 메시지를 얻을 수 있다.

## 5. 예외발생 시키기

키워드 throw를 사용해서 프로그래머가 고의로 예외를 발생시킬 수 있으며, 방법은 아래의 순서를 따르면 된다.

1. 먼저, 연산자 new를 이용해서 발생시키려는 예외 클래스의 객체를 만든 다음  
Exception e = new Exception("발생 사유");
2. 키워드 throws를 이용해서 예외를 발생시킨다.  
throw e;

### [예제 4-3] 예외발생 시키기

```
public class Ex4_3 {
    public static void main(String[] args) {
        Exception e1 = new Exception("고의로 발생!");
        try{
            throw e1; // 예외처리를 강제한다.
        } catch(Exception e) {
            e.printStackTrace();
        }

        throw new RuntimeException();
        // 예외처리를 강제 하지 않으나 다음 줄이 컴파일 에러
        System.out.println("프로그램이 정상 종료 되었습니다.");
    }
}
```

위의 예제에서 특이한 사항을 확인할 수 있다. Exception으로 만든 객체는 예외처리를 강제 하지만 RuntimeException의 경우에는 예외처리를 강제 하지 않는다. 이 차이점 때문에 예외를 RuntimeException클래스들과 Exception클래스들로 분리한 것이다.

RuntimeException클래스들에 해당하는 예외는 프로그래머가 실수로 발생하는 것들이기 때문에 예외처리를 강제하지 않는 것이다. 하지만 예외처리를 해주지 않았기 때문에 프로그램이 강제 종료되게 된다.

예외처리를 확인하지 않는 RuntimeException클래스들은 'unchecked예외'라고 부르며, 예외처리를 확인하는 Exception클래스들은 'checked예외'라고 부른다.

## 6. 메서드에 예외 선언하기

예외를 처리하는 방법에는 지금까지 배워 온 try-catch문을 사용하는 것 외에, 예외를 메서드에 선언하는 방법이 있다.

메서드에 예외를 선언하려면, 메서드의 선언부에 키워드 throws를 사용해서 메서드 내에서 발생할 수 있는 예외를 적어주기만 하면 된다. 그리고 예외가 여러 개일 경우에는 쉼표(,)로 구분한다.

```
void method() throws Exception1, Exception2,...{  
    //메서드의 내용  
}
```

만일 예외의 최고조상인 Exception클래스를 메서드에 선언하면, 이 메서드는 모든 종류의 예외가 발생할 가능성이 있다는 뜻이다.

```
void method() throws Exception{  
    // 메서드의 내용  
}
```

이렇게 예외를 선언하면, 이 예외뿐만 아니라 그 자손타입의 예외까지도 발생할 수 있다는 점에 주의하자. 앞서 오버라이딩에서 살펴본 것과 같이, 오버라이딩할 때는 단순히 선언된 예외의 개수가 아니라 상속관계까지 고려해야 한다.

메서드의 선언부에 예외를 선언함으로써 메서드를 사용하려는 사람이 메서드의 선언부를 보았을 때, 이 메서드를 사용하기 위해서는 어떠한 예외들이 처리되어야 하는지 쉽게 알 수 있다.

기존의 많은 언어들에서는 메서드에 예외선언을 하지 않기 때문에, 경험 많은 프로그래머가

아니고서는 어떤 상황에 어떤 종류의 예외가 발생할 가능성이 있는지 충분히 예측하기 힘들기 때문에 그에 대한 대비를 하는 것이 어려웠다.

그러나 자바에서는 메서드를 작성할 때 메서드 내에서 발생할 가능성이 있는 예외를 메서드의 선언부에 명시하여 이 메서드를 사용하는 쪽에서는 이에 대한 처리를 하도록 강요하기 때문에, 프로그래머들의 짐을 덜어 주는 것은 물론이고 보다 견고한 프로그램 코드를 작성할 수 있도록 도와준다.

#### [예제 4-4] 메서드의 예외 선언

```
public class Ex4_4 {
    public static void main(String[] args) {
        Service.serviceMethod();
    }
}

class Service{
    static void serviceMethod(){
        try{
            Dao.daoMethod();
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}

class Dao{
    static void daoMethod() throws Exception{
        throw new Exception("DaoMethod에서 발생");
    }
}
```

결과

```
java.lang.Exception: DaoMethod에서 발생
at Dao.daoMethod(ExceptionThrows.java:22)
at Service.serviceMethod(ExceptionThrows.java:13)
at ExceptionThrows.main(ExceptionThrows.java:5)
```

위의 실행결과를 보면, 프로그램의 실행도중 java.lang.Exception이 발생하여 비정상적으로 종료했다는 것과 예외가 발생했을 때 호출스택(call stack)의 내용을 알 수 있다.

daoMethod()에서 'throw new Exception();'문장에 의해 예외가 강제적으로 발생했으나 try-catch문으로 예외처리를 해주지 않았으므로, daoMethod()는 종료되면서 예외를 자신을 호출한 serviceMethod()에게 넘겨준다. 예외를 넘겨받은 serviceMethod()에서 예외처리를 해주었기 때문에 main()에는 영향을 주기 않는다.

## 7. finally블럭

finally블럭은 try-catch문과 함께 예외의 발생여부에 상관없이 실행되어야할 코드를 포함시킬 목적으로 사용된다. try-catch문의 끝에 선택적으로 덧붙여 사용할 수 있으며, try-catch-finally의 순서로 구성된다.

```
try{
    // 예외가 발생할 가능성이 있는 문장들을 넣는다.
} catch(Exception1 e1){
    // 예외처리를 위한 문장을 적는다.
} finally {
    // 예외의 발생여부에 관계없이 항상 수행되어야하는 문장들을 넣는다.
    // finally블럭은 try-catch문의 맨 마지막에 위치해야한다.
}
```

예외가 발생한 경우에는 'try -> catch -> finally'의 순으로 실행되고, 예외가 발생하지 않은 경우에는 'try -> finally'의 순으로 실행된다.

### [예제 4-5] finally활용

```
public class Ex4_5 {
    public static void main(String[] args) {
        try{
            startInstall();
            copyFiles();
            deleteTempFiles();
        } catch (Exception1 e1) {
            e1.printStackTrace();
            deleteTempFiles();
        } finally {
            deleteTempFiles();
        }
    }

    static void startInstall(){ } // 프로그램설치
    static void copyFiles(){ } // 파일들을 복사
    static void deleteTempFiles(){ } // 임시파일 삭제
}
```

이 예제가 하는 일은 프로그램설치를 위한 준비를 하고 파일들을 복사하고 설치가 완료되면, 프로그램을 설치하는데 사용된 임시파일들을 삭제하는 순서로 진행된다.

프로그램의 설치과정 중에 예외가 발생하더라도, 설치에 사용된 임시파일들이 삭제되도록 catch블럭에 `deleteTempFiles()`메서드를 넣었다.

결국 try블럭의 문장을 수행하는 동안에(프로그램을 설치하는 과정에), 예외의 발생여부에 관계없이 `deleteTempFiles()`메서드는 실행되어야 하는 것이다.

이럴 때 finally블럭을 사용하여 예외발생 여부에 상관없이 진행될 수 있도록 만들 수 있다.

## 학습 5 자바 주요 클래스

### 5-1. java.lang 패키지

#### 학습목표

- 자바의 주요 클래스들의 종류와 특징에 대해 설명할 수 있다.

#### 필요 지식 / 클래스

#### ① java.lang 패키지

java.lang패키지는 자바프로그래밍에 가장 기본이 되는 클래스들을 포함하고 있다. 그렇기 때문에 java.lang패키지의 클래스들은 import문 없이도 사용할 수 있게 되어 있다. 그 동안 String클래스나 System클래스를 import문 없이 사용할 수 있었던 이유가 바로 java.lang패키지에 속한 클래스들이기 때문이었던 것이다. 우선 java.lang패키지의 여러 클래스들 중에서 자주 사용되는 클래스 몇 가지만 골라서 학습해 보자.

#### ② Object클래스

Object클래스에 대해서는 클래스의 상속을 학습할 때 배웠지만, 여기서는 보다 자세히 알아 보자. Object클래스는 모든 클래스의 최고 조상이기 때문에 Object클래스의 멤버들은 모든 클래스에서 바로 사용 가능하다.

Object클래스의 메서드	설명
protected Object clone()	객체 자신의 복사본을 반환한다.
public boolean equals(Object obj)	객체 자신과 객체 obj가 같은 객체인지 알려준다.
protected void finalize()	객체가 소멸될 때 가비지 컬렉터에 의해 자동적으로 호출된다. 이 때 수행되어야하는 코드가 있을 때 오버라이딩한다.
public Class getClass()	객체 자신의 클래스 정보를 담고 있는 Class인스턴스를 반환
public int hashCode()	객체 자신의 해시코드를 반환한다.
public String toString()	객체 자신의 정보를 문자열로 반환한다.
public void notify()	객체 자신을 사용하려고 기다리는 스레드를 하나만 깨운다.
public void notifyAll()	객체 자신을 사용하려고 기다리는 모든 스레드를 깨운다.



<pre> public void wait() public void wait(long time) public void wait(long timeout, int nanos) </pre>	<p>다른 스레드가 notify(), notifyAll()을 호출할 때까지 현재 스레드를 무한히 또는 지정된 시간동안 기다리게 한다. (timeout : 1/1000초, nanos : 1/10초)</p>
---	---

[표 5-1] Vector의 메서드와 생성자

Object클래스는 멤버변수는 없고 오직 11개의 메서드만 가지고 있다. 이 메서드들은 모든 인스턴스가 가져야 할 기본적인 것들이며, 우선 이 중에서 중요한 몇 가지만 살펴보자.

### 1. equals(Object obj)

매개변수로 객체의 참조변수를 받아서 비교하여 그 결과를 boolean값으로 알려 주는 역할을 한다.

아래의 코드는 Object클래스에 정의되어 있는 equals메서드의 실제 내용이다.

```

public boolean equals(Object obj){
    return this == obj ;
}

```

위의 코드에서 알 수 있듯이 두 객체의 같고 다름을 참조변수의 값으로 판단한다. 그렇기 때문에 서로 다른 두 객체를 equals메서드로 비교하면 항상 false를 결과로 얻게 된다.

#### [예제 5-1] 기본 equals메서드

```

public class Ex5_1 {
    public static void main(String[] args) {
        Value v1 = new Value(10);
        Value v2 = new Value(10);
        System.out.println("v1 == v2 : "+(v1 == v2)); // ①
        System.out.println("v1.equals(v2) : "+v1.equals(v2)); // ②
    }
}

class Value{
    int value;
    Value(int value){
        this.value = value;
    }
}

```

결과  
v1 == v2 : false  
v1.equals(v2) : false

위 예제에서 v1에는 v1객체의 주소가 v2에는 v2객체의 주소를 저장하고 있다. ①의 결과는 v1과 v2가 저장하고 있는 값, 즉 주소를 비교하기 때문에 false의 결과를 가지게 된다. ②의 equals메서드는 Object 클래스로부터 상속받은 equals메서드를 사용하기 때문에 참조변수가 같은 객체를 참조하고 있는지 즉 ①과 동일하게 주소를 비교하게 된다. equals메서드로 Value인스턴스가 가지고 있는 value값을 비교 하도록 할 수는 없을까? Value클래스에서 equals메서드를 오버라이딩하여 주소가 아닌 객체에 저장된 내용을 비교하도록 변경하면 된다.

#### [문제 5-1] equals메서드 override

```
public class Qu5_1{
    public static void main(String[] args){
        Person p1 = new Person(8808082347856);
        Person p2 = new Person(8808082347856);
        System.out.println("p1 == p2 : "+(v1 == v2);
        System.out.println("p1.equals(p2) : "+p1.equals(p2));

    }
}

class Person{
    long regNo;
    Perons(long regNo){
        this.regNo = regNo;
    }
    @override
    public boolean equals(Obejct obj){
        //1. 매개변수에 저장된 인자 값의 객체가 Person의 객체라면 현재 객체의 regNo와
        // 매개변수의 regNo의 값이 같으면 true반환, 그렇지 않으면 false를 반환하는
        // 오버라이드된 equals메서드를 완성 하여라.

    }
}
```

※ equals메서드가 오버라이드 되어 있는 클래스들 : String, Date, File, wrapper클래스들...

## 2. toString()

이 메서드는 인스턴스에 대한 정보를 문자열(String)로 제공할 목적으로 정의한 것이다. 인스턴스의 정보를 제공한다는 것은 대부분의 경우 인스턴스 변수에 저장된 값들을 문자열로 표현한다는 뜻이다.

Object클래스에 정의된 toString()은 아래와 같다.

```
public String toString(){
    return getClass().getName()+"@"+Integer.toHexString(hashCode());
}
```

클래스를 작성할 때 toString()을 오버라이딩하지 않는다면, 위와 같은 내용이 그대로 사용될 것이다. 즉, toString()을 호출하면 클래스이름에 16진수의 해시코드를 얻게 될 것이다.

### [예제 5-2] toString()오버라이드

```
public class Ex5_2 {
    public static void main(String[] args) {
        Card c = new Card();
        System.out.println(c.toString());
        Card2 c2 = new Card2();
        System.out.println(c2.toString());
    }
}

class Card{
    String kind;
    int number;
    Card(){
        this("SPADE",1);
    }
}

class Card2{
    String kind;
    int number;
    Card2(){
        this("SPADE",1);
    }
    public String toString(){
        return "kind : "+kind+", number : "+number;
    }
}
```

결과  
Card@6c89db9a  
kind : SPADE, number : 1

위 예제에서 Card클래스는 toString메서드가 없지만 Object클래스를 상속 받기 때문에 Object클래스의 toString메서드를 사용하게 된다. 그래서 Card객체의 toString메서드를 호출한 결과는 주소를 출력하게 된다. 하지만 Card2클래스의 toString메서드를 호출 시 Object클래스의 toString메서드를 호출하는 것이 아닌 오버라이드된 Card2클래스의 toString메서드를 호출하여 반환되는 'kind : SPADE, number : 1'의 값을 출력하게 된다.

### 3. getClass()

클래스의 정보가 필요할 때, 먼저 Class객체에 대한 참조를 얻어 와야 하는데, 해당 Class객체에 대한 참조를 얻는 방법은 여러 가지가 있다.

생성될 객체로 부터 얻는 방법	Class obj = new Card().getClass();
클래스 리터럴로부터 얻는 방법	Class obj = Card.class;
클래스의 이름으로 부터 얻는 방법	Class obj = Class.forName("Card");

특히 forName()은 특정 클래스 파일, 예를 들어 데이터베이스 드라이버를 메모리에 올릴때 주로 사용한다.

Class객체를 이용하면 클래스에 정의된 멤버의 이름이나 개수 등, 클래스에 대한 모든 정보를 얻을 수 있기 때문에 Class객체를 통해서 객체를 생성하고 메서드를 호출하는 등 보다 동적인 코드를 작성할 수 있다.

#### [예제 5-3] getClass()사용법

```
public class Ex5_3 {
    public static void main(String[] args) {
        Card c = new Card();
        Class obj1 = c.getClass();
        Class obj2 = Card.class;
        Class obj3 = Class.forName("Card"); /* 주의 : '패키지명.클래스명'으로 작성 */
        System.out.println(obj1.getName());
    }
}

class Card{
    String kind;
    int number;
    Card(){
        this("SPADE",1);
    }
}
```

## ② String클래스

기존의 다른 언어에서는 문자열을 char형의 배열로 다루었으나 자바에서는 문자열을 위한 클래스를 제공한다. 그 것이 바로 String클래스인데, String클래스는 문자열을 저장하고 이를 다루는데 필요한 메서드를 제공한다.

한번 생성된 String인스턴스가 갖고 있는 문자열은 읽어 올 수만 있고, 변경할 수는 없다.

예를 들어 문자열 'a'와 문자열 'b'를 '+'연산자를 이용해서 문자열을 결합하는 경우 인스턴스내의 문자열이 바뀌는 것이 아니라 새로운 문자열("ab")이 담긴 String인스턴서가 생성되는 것이다.

문자열간의 결합이나 추출 등 문자열을 다루는 작업이 많이 필요한 경우에는 String클래스 대신 StringBuffer클래스를 사용하는 것이 좋다. StringBuffer클래스는 뒤에서 알아 보도록 하자.

### 1. 문자열의 비교

문자열을 만들 때는 두 가지 방법, 지금까지 문자열을 다루어온 방식인 문자열 리터럴을 지정하는 방법과 String클래스의 생성자를 사용해서 만드는 방법이 있다.

#### [예제 5-4] 문자열의 비교

```
public class Ex5_4 {
    public static void main(String[] args) {
        String str1 = "abc";
        String str2 = "abc";

        System.out.println("str1 == str2 ? "+(str1==str2));
        System.out.println("str1.equals(str2) ? "+(str1.equals(str2)));

        String str3 = new String("abc");
        String str4 = new String("abc");
        System.out.println("str3 == str4 ? "+(str3==str4));
        System.out.println("str3.equals(str4) ? "+(str3.equals(str4)));
    }
}
```

결과

```
str1 == str2 ? true
str1.equals(str2) ? true
str3 == str4 ? false
str3.equals(str4) ? true
```

String클래스의 생성자를 이용한 경우에는 new연산자에 의해서 메모리에 할당이 이루어지기 때문에 항상 새로운 String인스턴스가 생성된다. 그러나 문자열 리터럴은 이미 존재하는 것을 재사용 하는 것이기 때문에 str1과 str2의 주소를 비교하였을 때 true의 결과를 가지게 된다.

※ 문자열 리터럴은 클래스가 메모리에 로드될 때 자동적으로 미리 생성된다.

## 2. String클래스의 생성자와 메서드

아래의 표는 String클래스 내에 정의된 생성자와 메서드의 목록이다. 전체 목록은 아니고, 자주 사용될만한 것들만 뽑아 보았다.

메서드/설명	예제	결과
String(String s)	String s = new String("Hello");	s="Hello"
주어진 문자열(s)을 갖는 String인스턴스를 생성한다.		
char charAt(int index)	String s = "Hello"; char c = s.charAt(1);	c='e'
지정된 위치(index)에 있는 문자열을 알려준다.		
int compareTo(String str)	int i = "aaa".compareTo("aaa"); int i2 = "aaa".compareTo("aab");	i=0 i2=-1
문자열(str)과 사전순서를 비교한다. 같으면 0을 사전순으로 이전이면 음수를, 이후면 양수를 반환한다.		
String concat(String str)	String s = "Hello"; String s2 = s.concat(" Oracle");	s2="Hello Oracle"
문자열(str)을 뒤에 덧붙인다.		
boolean contains(CharSequence s)	String s = "JavaSoEz"; boolean b = s.contains("S");	b=true
지정된 문자열(s)이 포함되었는지 검사한다.		
boolean endsWith(String suffix)	String s = "JavaSoEz,java"; boolean b = s.endsWith("java");	b=true
지정된 문자열(suffix)로 끝나는지 검사한다.		
boolean equals(Object obj)	String s = "Java"; boolean b = s.equals("java");	b=false
매개변수로 받은 문자열(obj)과 String인스턴스의 문자열을 비교한다. obj가 String이 아니거나 문자열이 다르면 false를 반환한다.		
int indexOf(int ch)	String s = "Hello"; int b = s.indexOf('l');	b=2
주어진 문자(ch)가 문자열에 존재하는지 확인하여 위치(index)를 알려준다. 못 찾으면 -1을 반환한다.		
int lastIndexOf(int ch)	String s = "Hello"; int b = s.lastIndexOf('l');	b=3;
지정된 문자열을 인스턴스의 문자열 끝에서부터 찾아서 위치(index)를 알려준다. 못 찾으면 -1을 반환		
int length()	String s = "My Length"; int b = s.length();	b=9
문자열의 길이를 알려준다.		
String replace(CharSequence old, CharSequence nw)	String s = "Hello Oracle"; String b = s.replace("Oracle", "Java");	b="Hello Java"
문자열 중의 문자열(old)을 새로운 문자열(nw)로 모두 바꾼 문자열을 반환한다.		

String[] split(String regex)	문자열을 지정된 분리자(regex)로 나누어 문자열 배열에 담아 반환한다.	String s = "dog-cat-snake"; String[] sArr = s.split("-");	sArr[0]="dog" sArr[1]="cat" sArr[2]="snake"
String substring(int begin, int end)			
주어진 시작위치(begin)부터 끝 위치(end)범위에 포함된 문자열을 더한다. 이 때, 시작위치의 문자는 범위에 포함되지만, 끝 위치의 문자는 포함되지 않는다.		String s = "I Love Java"; String b = s.substring(2,6);	b="Love"
String toLowerCase()	String인스턴스에 저장되어 있는 모든 문자열을 소문자로 변환하여 반환한다.	String s = "I Love Java"; String b = s.toLowerCase();	b="i love java"
String toUpperCase()			
String trim()	문자열의 왼쪽 끝과 오른쪽 끝에 있는 공백을 없앤 결과를 반환한다. 이 때 문자열 중간에 있는 공백은 제거되지 않는다.	String s = " My Trim "; String b = s.trim();	b="My Trim"
static String valueOf(boolean b)			
지정된 값을 문자열로 변환하여 반환한다. 참조변수의 경우, toString()을 호출한 결과를 반환한다.(매개변수로 모든 타입이 올 수 있다.)		String b = String.valueOf(true); String c = String.valueOf(100); String d = String.valueOf('a');	b="true" b="100" b="a"

[표 5-3] String Methods

### ③ StringBuffer클래스와 StringBuilder클래스

String클래스는 인스턴스를 생성할 때 지정된 문자를 변경할 수 없다. 문자열의 변경을 위해서는 StringBuffer클래스를 사용해야 한다.

StringBuffer클래스의 인스턴스를 생성할 때, 적절한 길이의 char형 배열이 생성되고, 이 배열은 문자열을 저장하고 편집하기 위한 공간(buffer)으로 사용된다.

StringBuffer인스턴스를 생성할 때는 생성자 StringBuffer(int length)를 사용해서 StringBuffer인스턴스에 저장될 문자열의 길이를 고려하여 충분히 여유있는 크기로 지정하는 것이 좋다. StringBuffer인스턴스를 생성할 때, 버퍼의 크기를 지정해주지 않으면 16개의 문자를 저장할 수 있는 크기의 버퍼를 생성한다.

StringBuffer클래스와 StringBuilder클래스의 StringBuffer의 경우에는 멀티쓰레드에 안전하도록 동기화되어 있다. 멀티쓰레드나 동기화에 대해서 배우지 않았지만, 동기화가 StringBuffer의 성능을 떨어뜨린다는 것만 이해하면 된다.

아래의 코드는 문자열 str에 “a”를 여러 번 더하는 과정이다.

```
String str = “”;
str += “a”;
str += “a”;
⋮
```

위의 코드가 간단해 보이지만 실제로는 다음과 코드를 진행하게 된다.

```
String str = “”;
str = new StringBuffer(str).append(“a”).toString();
str = new StringBuffer(str).append(“a”).toString();
⋮
```

순서는 StringBuffer객체를 만들 때 기존에 저장된 문자열로 초기화를 하고 append메서드를 이용하여 문자열 “a”를 더해준다. StringBuffer의 append메서드의 반환 타입은 StringBuffer타입이기 때문에 toString메서드를 이용하여 문자열로 변환하여 주는 과정을 반복하게 되는 것이다. 이를 StringBuffer클래스나 StringBuilder클래스를 이용하면 어느 정도 빨라지는지 다음 예제를 통해 알아보자.

#### [예제 5-5] 문자열 결합

```
public class Ex5_5 {
    public static void main(String[] args) {
        /* String클래스를 이용한 문자열 결합 */
        long start = System.currentTimeMillis();
        String str = “a”;
        for (int i = 0; i < 300000; i++) {
            str += “a”;
        }
        long stop = System.currentTimeMillis();
        System.out.println(“String결합 : “+(stop - start));

        /* StringBuffer클래스를 이용한 문자열 결합 */
        long startB = System.currentTimeMillis();
        StringBuffer strB = new StringBuffer(“a”);
        for (int i = 0; i < 30000000; i++) {
            strB.append(“a”);
        }
        long stopB = System.currentTimeMillis();
    }
}
```



```

System.out.println("StringBuffer결합 : "+(stopB - startB));

/* StringBuilder클래스를 이용한 문자열 결합 */
long startBi = System.currentTimeMillis();
StringBuilder strBi = new StringBuilder("a");
for (int i = 0; i < 30000000; i++) {
    strBi.append("a");
}
long stopBi = System.currentTimeMillis();
System.out.println("StringBuilder결합 : "+(stopBi - startBi));
}
}

```

**결과**  
String결합 : 23204  
StringBuffer결합 : 386  
StringBuilder결합 : 258

※ 자신의 컴퓨터 성능에 따라 결과는 다를 수 있다.

예제 5-5의 결과 String클래스를 이용해 문자열을 30만회 합치는 반복문은 23.204초가 소요되었다. 하지만 StringBuffer클래스를 이용해 문자열을 3000만회 합치는 반복문은 0.386초가 소요되었다. 횟수는 100배 증가 하였지만 약 1/60정도의 시간만 소요되었다.

문자열의 변경, 결합 등이 자주 일어날 경우에는 String클래스보다는 StringBuffer클래스를 활용하는 것이 현명한 선택이 된다.

StringBuilder클래스는 StringBuffer클래스보다 조금의 차이뿐 이므로 StringBuffer클래스로 작성된 코드를 StringBuilder클래스로 굳이 바꿀 필요는 없다.

#### ④ wrapper클래스

객체지향 개념에서 모든 것은 객체로 다루어져야 한다. 그러나 자바에서는 9개의 기본형을 객체로 다루지 않는데 이것이 바로 자바가 완전한 객체지향 언어가 아니라는 얘기를 듣는 이유이다. 그 대신 보다 높은 성능을 얻을 수 있었다.

때로는 기본형(primitive type)변수도 어쩔 수 없이 객체로 다뤄야 하는 경우가 있다. 예를 들면, 매개변수로 객체를 요구할 때, 기본형 값이 아닌 객체로 저장해야할 때, 객체간의 비교가 필요할 때 등등의 경우에는 기본형 값들을 객체로 변환하여 작업을 수행해야 한다.

이 때 사용되는 것이 래퍼(wrapper)클래스이다. 8개의 기본형을 대표하는 8개의 래퍼클래스가 있는데, 이 클래스들을 이용하면 기본형 값을 객체로 다룰 수 있다.

래퍼클래스의 생성자는 매개변수로 문자열이나 각 자료형의 값들을 인자로 받는다. 이 때 주의해야할 것은 생성자의 매개변수로 문자열을 제공할 때, 각 자료형에 알맞은 문자열을 사용해야한다는 것이다. 예를 들어 'new Integer( "1.0" );'과 같이 하면 NumberFormatException이 발생한다.

기본형	래퍼클래스	생성자	활용 예제
boolean	Boolean	Boolean(boolean value) Boolean(String s)	Boolean b = new Boolean(true); Boolean b = new Boolean("true");
char	Character	Character(char value)	Character c = new Character('a');
byte	Byte	Byte(byte value) Byte(String s)	Byte b = new Byte(10); Byte b = new Byte("10");
short	Short	Short(short value) Short(String s)	Short b = new Short(10); Short b = new Short("10");
int	Integer	Integer(int value) Integer(String s)	Integer b = new Integer(10); Integer b = new Integer("10");
long	Long	Long(long value) Long(String s)	Long b = new Long(10); Long b = new Long("10");
float	Float	Float(double value) Float(float value) Float(String s)	Float b = new Float(1.0); Float b = new Float(1.0f); Float b = new Float("1.0f");
double	Double	Double(double value) Double(String s)	Double b = new Double(1.0); Double b = new Double("1.0");

[표 5-4] wrapper클래스와 생성자

#### [예제 5-6] wrapper클래스

```

public class Ex5_6 {
    public static void main(String[] args) {
        Integer i = new Integer(100);
        Integer i2 = new Integer(100);
        System.out.println(i == i2);
        System.out.println(i.equals(i2)); // ①
        System.out.println(i); // ②

        // 1. 최대값, 최소값
        System.out.println(Integer.MAX_VALUE);
        System.out.println(Integer.MIN_VALUE);

        // 2. size = > bit
        System.out.println(Integer.SIZE);

        // 3.BYTES => byte
        //JDK1.8부터 지원한다.
        //System.out.println(Integer.BYTES);
    }
}

```

```

//4. 진법적용
Integer i7 = new Integer(100,2);
System.out.println(i7); // ③

//5. autoboxing, unboxing
int i3 = new Integer(50); // ④
Integer i4 = 50; // ⑤
Integer[] iArr = new Integer[3];
iArr[0] = new Integer(20);
iArr[1] = 25;
iArr[2] = 50;
int i5 = iArr[0];
}
}

```

**결과**

```

false
true
100
2147483647
-2147483648
32

```

예제 5-6의 ①의 경우는 Object클래스의 equals메서드를 사용하게 되었다면 다른 두 개의 객체이기 때문에 false의 결과가 나와야 하지만 결과는 true가 나왔다. 즉, wrapper클래스들은 equals메서드가 오버라이드 되어있다는 결과를 도출할 수 있다.

②의 경우 객체의 주소가 출력되어야 하지만 실력 출력결과는 해당 객체가 가지고 있는 값을 출력한다. 즉, wrapper클래스들은 toString메서드가 오버라이드 되어있다는 결과를 도출할 수 있다.

i7의 객체가 생성될 때 매개변수가 두개인 생성자를 활용 하였다. ③에서 4의 출력결과를 얻는데 이것은 i7의 객체가 생성될 때 100이라는 숫자가 2진수로 표기되어 출력되었을 때 다시 10진수의 값으로 출력된 예제이다.

④번의 코드는 Integer객체를 저장된 값만을 int타입의 i3에 저장되는 예제이면 wrapper클래스의 객체 타입을 기본형 타입으로 변경하여 저장하는 것을 언박싱(unboxing)이라고 하며 자동적으로 이루어진다.

⑤번의 코드는 기본형 값을 wrapper클래스타입의 참조변수에 저장하는 예제로 이를 오토박싱(autoboxing)이라고 한다. 이 또한 자동으로 이루어진다.

<b>오토박싱(autoboxing)</b>	기본형 값을 wrapper클래스의 객체로 변환해 주는 기능.
<b>언박싱(unboxing)</b>	wrapper클래스의 객체를 기본형 값으로 변환해 주는 기능

## 5-2. 정규식

### 학습목표

- 사용자의 요청에 따른 정규식을 작성할 수 있다.

### 필요 지식 /

#### ① 정규식 (Regular Expression)

정규식이란 텍스트 데이터 중에서 원하는 조건(패턴-pattern)과 일치하는 문자열을 찾아내기 위해 사용하는 것으로 미리 정의된 기호와 문자를 이용해서 작성한 문자열을 말한다.

정규식을 이용하면 많은 양의 텍스트 파일 중에서 원하는 데이터를 손쉽게 뽑아낼 수도 있고 입력된 데이터가 형식에 맞는지 체크할 수도 있다. 예를 들면 html문서에서 전화번호나 이메일 주소만을 따로 추출한다던가, 입력한 비밀번호가 숫자와 영문자의 조합으로 되어 있는지 확인할 수도 있다.

정규식을 정의하고 데이터를 비교하는 과정을 단계별로 설명하면 다음과 같다.

1. 정규식을 매개변수로 `Pattern`클래스의 `static`메서드인 `compile(String regex)`을 호출하여 `Pattern` 인스턴스를 얻는다. (패턴등록)  
`Pattern p = Pattern.compile("[a-z]*");`
2. 정규식으로 비교할 대상을 매개변수로 `Pattern`클래스의 `Matcher matcher(CharSequence input)`를 호출해서 `Matcher`인스턴스를 얻는다. (비교문장 등록)  
`Matcher m = p.matcher("aasdf");`
3. `Matcher`인스턴스에 `boolean matches()`를 호출해서 정규식에 부합하는지 확인한다.  
`System.out.println(m.matches());`

정규식 패턴	설명
<code>^</code>	문자열의 시작
<code>\$</code>	문자열의 종료
<code>.</code>	임의의 한 문자(문자의 종류를 가리지 않음) 단, '\ '는 넣을 수 없음
<code>*</code>	앞 문자가 없을 수도 무한정 많을 수도 있음
<code>+</code>	앞 문자가 하나 이상

?	앞 문자가 없거나 하나 있음
[]	문자의 집합이나 범위를 나타내며 두 문자 사이는 - 기호로 범위를 나타낸다. []내에서 '^'가 성행하여 존재하면 not을 의미한다.
{}	횟수 또는 범위를 나타낸다.
()	소괄호 안의 문자를 하나의 문자로 인식
	패턴 안에서 or연산을 수행할 때 사용
\s	공백문자
\S	공백문자가 아닌 나머지 문자
\w	알파벳이나 숫자 [A-Za-z0-9]와 같은 의미
\W	알파벳이나 숫자를 제외한 나머지 문자
\d	숫자를 의미 [0-9]와 동일
\D	숫자를 제외한 모든 문자

[표 5-5] 정규식 패턴

#### [문제 5-2] 정규식 정의하기

```

public class Qu5_2{
    public static void main(String[] args){
        String regEx1 = "[a-z]{2,3}"; // 문자열이 영어소문자 2개 또는 3개로 구성
        System.out.println(Pattern.matches(regEx1, "ssss"));

        //1. 텍스트가 영문자가 3회 반복되고 이후에 숫자가 하나이상으로 구성

        //2. 텍스트가 핸드폰 번호 형태인 '숫자3개-숫자4개-숫자4개'로 구성

        //3. 텍스트가 핸드폰 번호로 구성
        // 01 다음 0,1,7,8,9 - 0을 제외한 숫자, 숫자3개 - 숫자4개

        //4. 텍스트가 주민등록번호로 구성
        // 년도 월 일 - 1~4 숫자6개
    }
}

```

```
//5. 텍스트가 이메일로 구성
// 시작은 영문자 이어야 하고 특수기호는 - , _ , \ , . 4가지가 포함될 수 있다.
// @ 이후 영문자가 1개~7개가 포함될 수 있다.
// . 이후 영문자가 2~3개가 포함되어야 한다.
// .kr이 없을 수도 하나 존재 할 수도 있다.
```

```
}
}
```

## 6-1. 컬렉션 프레임워크

### 학습목표

- 컬렉션 프레임워크의 인터페이스와 이를 구현한 클래스를 설명할 수 있다.
- 컬렉션의 특징에 따라 개발에 활용할 수 있다.

### 필요 지식 / 객체지향 프로그래밍

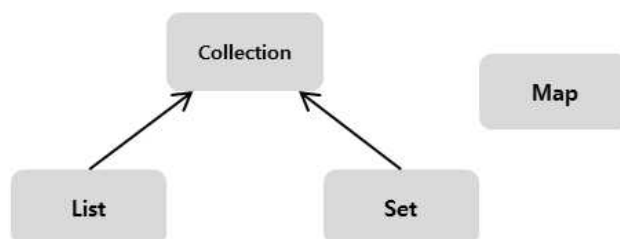
#### ① 컬렉션 프레임워크

컬렉션 프레임워크이란, '데이터 군을 저장하는 클래스들을 표준화한 설계'를 뜻한다. 컬렉션(Collection)은 다수의 데이터, 즉 데이터 그룹을, 프레임워크는 표준화된 프로그래밍 방식을 의미한다.

컬렉션 프레임워크는 컬렉션, 다수의 데이터를 다루는 데 필요한 다양하고 풍부한 클래스들을 제공하기 때문에 프로그래머의 짐을 상당히 덜어 주고 있으며, 또한 인터페이스와 아형성을 이용한 객체지향적 설계를 통해 표준화되어 있기 때문에 사용법을 익히기에도 편리하고 재사용성이 높은 코드를 작성할 수 있다는 장점이 있다.

#### 1. 컬렉션 프레임워크의 핵심 인터페이스

컬렉션 프레임워크에서는 컬렉션을 크게 3가지 타입이 존재한다고 인식하고 각 컬렉션을 다루는데 필요한 기능을 가진 3개의 인터페이스를 정의하였다. 그리고 인터페이스 List와 Set의 공통된 부분을 다시 뽑아서 새로운 인터페이스인 Collection을 추가로 정의 하였다.



[그림 6-1] 컬렉션 프레임워크의 핵심 인터페이스간의 상속계층도

인터페이스는 List와 Set을 구현한 컬렉션 클래스들은 서로 많은 공통부분이 있어서, 공통된 부분을 다시 뽑아 Collection인터페이스를 정의할 수 있었지만 Map인터페이스는 이들과는 전

혀 다른 상태로 컬렉션을 다루기 때문에 같은 상속계층도에 포함되지 못했다.

이러한 설계는 객체지향언어의 장점을 극명히 보여주는 것으로 객체지향개념을 학습하는 사람들에게 많은 것을 느끼게 한다. 후에 프로그래밍 실력을 어느 정도 갖추게 되었을 때 컬렉션 프레임워크의 실제 소스를 분석해보면 객체지향적인 설계능력을 향상시키는데 많은 도움이 될 것이다.

인터페이스	특 징
List	순서가 있는 데이터의 집합, 데이터의 중복을 허용한다. 예) 대기자 명단
	구현 클래스 : ArrayList, LinkedList, Stack, Vector 등
Set	순서를 유지하지 않는 데이터의 집합, 데이터의 중복을 허용하지 않는다. 예) 양의 정수집합, 소수의 집합
	구현 클래스 : HashSet, TreeSet 등
Map	키(key)와 값(value)의 쌍(pair)으로 이루어진 데이터의 집합 순서는 유지되지 않으며, 키는 중복을 허용하지 않고, 값은 중복을 허용한다. 예) 우편번호, 지역번호(전화번호)
	구현클래스 : HashMap, TreeMap, Hashtable, Properties 등

[표 6-1] 컬렉션 프레임워크의 핵심 인터페이스와 특징

실제 개발 시에는 다루고자 하는 컬렉션의 특징을 파악하고 어떤 인터페이스를 구현한 컬렉션 클래스를 사용해야하는지 결정해야하므로 위의 표5-1에 적힌 각 인터페이스의 특징과 차이를 잘 이해하고 있어야 한다.

컬렉션 프레임워크의 모든 컬렉션 클래스들은 List, Set, Map 중의 하나를 구현하고 있으며, 구현한 인터페이스의 이름이 클래스의 이름에 포함되어있어서 이름만으로도 클래스의 특징을 쉽게 알 수 있도록 되어 있다.

그러나 Vector, Stack, Hashtable, Properties와 같은 클래스들은 컬렉션 프레임워크가 만들어지기 이전부터 존재하던 것이기 때문에 컬렉션 프레임워크의 명명법을 따르지 않는다.

Vector나 Hashtable과 같은 기존의 컬렉션 클래스들은 호환을 위해, 설계를 변경해서 남겨두었지만 가능하면 사용하지 않는 것이 좋다. 그 대신 새로 추가된 ArrayList와 HashMap을 사용하자.



## 2. Collection인터페이스

List와 Set의 조상인 Collection인터페이스에는 다음과 같은 메서드들이 정의되어 있다.

메서드	설 명
boolean add(Object o) boolean addAll(Collection c)	지정된 객체(o) 또는 Collection(c)의 객체들을 Collection에 추가한다.
void clear()	Collection의 모든 객체를 삭제한다.
boolean contains(Object o) boolean containsAll(Collection c)	지정된 객체(o)또는 Collection의 객체들이 Collection에 포함되어 있는지 확인한다.
boolean equals(Object o)	동일한 Collection인지 비교한다.
int hashCode()	Collection의 hash code를 반환한다.
boolean isEmpty()	Collection이 비어있는지 확인한다.
Iterator iterator()	Collection의 iterator를 얻어서 반환한다.
boolean remove(Object o)	지정된 객체를 삭제한다.
boolean removeAll(Collection c)	지정된 Collection에 포함된 객체들을 삭제한다.
boolean retainAll(Collection c)	지정된 Collection에 포함된 객체만을 남기고 다른 객체들은 Collection에서 삭제한다. Collection에 변화가 있으면 true를 그렇지 않으면 false를 반환한다.
int size()	Collection에 저장된 객체의 개수를 반환한다.
Object[] toArray()	Collection에 저장된 객체를 객체배열(Object[])로 반환한다.
Object[] toArray(Object[] a)	지정된 배열에 Collection의 객체를 저장해서 반환한다.

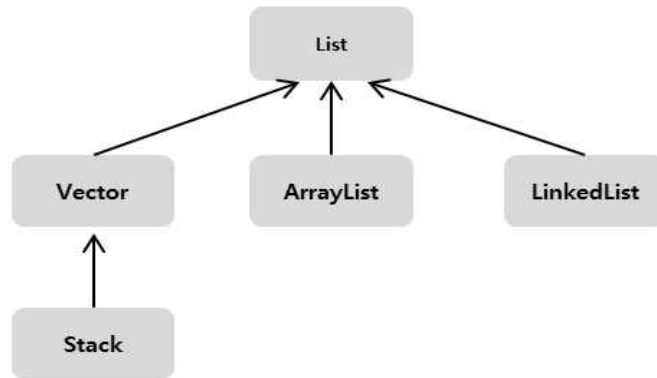
[표 6-2] Collection인터페이스에 정의된 메서드

Collection인터페이스는 컬렉션 클래스에 저장된 데이터를 읽고, 추가하고, 삭제하는 등 컬렉션을 다루는데 가장 기본적인 메서드들을 정의하고 있다.

## ② List인터페이스

### 1. List 기본

List인터페이스는 중복을 허용하면서 자장순서가 유지되는 컬렉션을 구현하는데 사용된다.



[그림 6-2] List의 상속계층도

List인터페이스에 정의된 메서드는 다음과 같다. Collection인터페이스로부터 상속받은 것들은 제외하였다.

메서드	설 명
void add(int index, Object element) boolean addAll(int index, Collection c)	지정된 위치(index)에 객체(element)또는 컬렉션에 포함된 객체를 추가한다.
Object get(int index)	지정된 위치(index)에 있는 객체를 반환한다.
int indexOf(Object o)	지정된 객체의 위치(index)를 반환한다.(정방향)
int lastIndexOf(Object o)	지정된 객체의 위치(index)를 반환한다.(역방향)
ListIterator listIterator()	List의 객체에 접근할 수 있는 ListIterator를 반환한다.
Object remove(int index)	지정된 위치(index)에 있는 객체를 삭제하고 삭제된 객체를 반환한다.
Object set(int index, Object element)	지정된 위치(index)에 객체(element)를 저장한다.
void sort(Comparator c)	지정된 비교자(comparator)로 List를 정렬한다.
List subList(int from, int to)	지정된 범위(from부터 to)에 있는 객체를 반환한다.

[표 6-3] List인터페이스에 정의된 메서드

## 2. ArrayList

ArrayList는 컬렉션 프레임워크에서 가장 많이 사용되는 컬렉션 클래스일 것이다. 이 ArrayList는 List인터페이스를 구현하기 때문에 데이터의 저장순서가 유지되고 중복을 허용한다는 특징을 갖는다.

ArrayList는 기존의 Vector를 개선한 것으로 Vector와 구현원리와 기능적인 측면에서 동일하다고 할 수 있다. 앞에서 얘기했던 것과 같이 Vector는 기존에 작성된 소스와의 호환성을 위해서 계속 남겨 두고 있을 뿐이기 때문에 가능하면 Vector보다는 ArrayList를 사용하자.

ArrayList는 Object배열을 이용해서 데이터를 순차적으로 저장한다. 예를 들면, 첫 번째로 저장한 객체는 Object배열의 0번째 위치에 저장되고 그 다음에 저장하는 객체는 1번째 위치에 저장된다. 이런 식으로 계속 배열에 순서대로 저장되며, 배열에 더 이상 저장할 공간이 없으면 보다 큰 새로운 배열을 생성해서 기존의 배열에 저장된 내용을 새로운 배열로 복사한 다음에 저장된다.

메서드	설 명
ArrayList()	크기가 0인 ArrayList를 생성
ArrayList(Collection c)	주어진 컬렉션이 저장된 ArrayList를 생성
ArrayList(int initialCapacity)	지정된 초기용량을 갖는 ArrayList를 생성
boolean add(Object o)	ArrayList의 마지막에 객체를 추가. 성공하면 true
void add(int index, Object element)	지정된 위치(index)에 객체를 저장
boolean addAll(Collection c)	주어진 컬렉션의 모든 객체를 저장한다.
void clear()	ArrayList를 완전히 비운다.
Object clone()	ArrayList를 복제한다.
boolean contains(Object o)	지정된 객체(o)가 ArrayList에 포함되어 있는지 확인
Object get(int index)	지정된 위치(index)에 저장된 객체를 반환한다.
int indexOf(Object o)	지정된 객체가 저장된 위치를 찾아 반환한다.
boolean isEmpty()	ArrayList가 비어있는지 확인한다.
Iterator iterator()	ArrayList의 iterator객체를 반환
Object remove(int index)	지정된 위치(index)에 있는 객체를 제거한다.

<code>boolean remove(Object o)</code>	지정된 객체를 삭제한다.
<code>boolean removeAll(Collection c)</code>	지정된 Collection에 저장된 것과 동일한 객체들을 ArrayList에서 제거한다.
<code>boolean retainAll(Collection c)</code>	ArrayList에 저장된 객체 중에서 주어진 컬렉션과 공통된 것들만을 남기고 나머지는 삭제한다.
<code>Object set(int index, Object element)</code>	주어진 객체(element)를 지정된 위치(index)에 저장한다.
<code>int size()</code>	ArrayList에 저장된 객체의 개수를 반환한다.
<code>void sort(Comparator c)</code>	지정된 정렬기준(c)으로 ArrayList를 정렬
<code>List subList(int fromIndex, int toIndex)</code>	fromIndex부터 toIndex사이에 저장된 객체를 반환한다.
<code>Object[] toArray()</code>	Collection에 저장된 객체를 객체배열(Object[])로 반환한다.
<code>Object[] toArray(Object[] a)</code>	지정된 배열에 Collection의 객체를 저장해서 반환한다.

[표 6-4] ArrayList의 생성자와 메서드

#### [예제 6-1] ArrayList생성자와 메서드

```

public class Ex6_1 {
    public static void main(String[] args) {
        List<Integer> list1 = new ArrayList<Integer>();
        list1.add(new Integer(5));
        list1.add(new Integer(4));
        list1.add(new Integer(3));
        list1.add(new Integer(2));
        list1.add(new Integer(1));

        ArrayList<Integer> list2 = new ArrayList<Integer>(list1.subList(2, 4));
        System.out.println("1 : "+list1);
        System.out.println("1 : "+list2);

        Collections.sort(list1);
        Collections.sort(list2);
        System.out.println("2 : "+list1);
        System.out.println("2 : "+list2);
    }
}

```

```

System.out.println(list1.containsAll(list2));
System.out.println(list2.containsAll(list1));
list2.add(4);
list2.add(7);
System.out.println("3 : "+list1);
System.out.println("3 : "+list2);

list2.add(2, 11);
System.out.println("4 : "+list1);
System.out.println("4 : "+list2);

list2.set(2, 33);
System.out.println("5 : "+list1);
System.out.println("5 : "+list2);
}
}

```

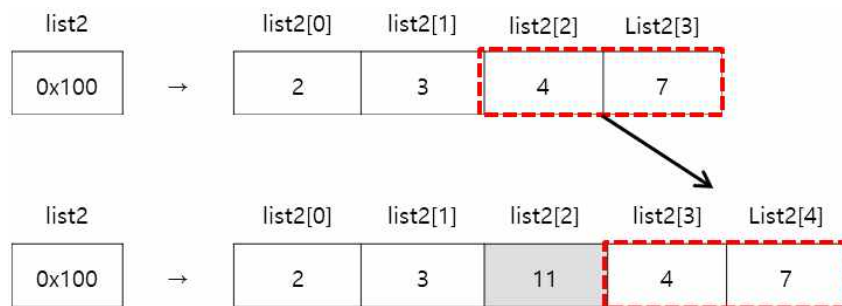
결과

```

1 : [5, 4, 3, 2, 1]
1 : [3, 2]
2 : [1, 2, 3, 4, 5]
2 : [2, 3]
true
false
3 : [1, 2, 3, 4, 5]
3 : [2, 3, 4, 7]
4 : [1, 2, 3, 4, 5]
4 : [2, 3, 11, 4, 7]
5 : [1, 2, 3, 4, 5]
5 : [2, 3, 33, 4, 7]

```

위의 예제는 ArrayList의 기본적인 메서드를 이용해서 객체를 다루는 방법을 보여 준다. ArrayList는 List인터페이스를 구현했기 때문에 저장된 순서를 유지한다는 것을 알 수 있다. 그리고 Collection클래스의 sort메서드를 이용해서 ArrayList에 저장된 객체들을 정렬하였는데 Collection클래스에 대한 내용과 정렬(sort)하는 방법에 대해서는 후에 자세히 다룰 것이므로 간단한 정렬방법이 있다는 정도만 이해하고 넘어가자.



[그림 6-3] ArrayList에 요소 추가

위의 그림과 같이 list2에 2번 index에 11의 값이 추가되면서 해당 index부터 뒤에 있는 요소들은 뒤로 밀려나게 된다. ArrayList에 요소가 추가 되거나 삭제 될 때 많은 요소들을 이동시켜야 한다.

### 3. LinkedList

배열은 가장 기본적인 형태의 자료구조로 구조가 간단하며 사용하기 쉽고 데이터를 읽어오는데 걸리는 시간이 가장 빠르다는 장점을 가지고 있지만 다음과 같은 단점도 가지고 있다.

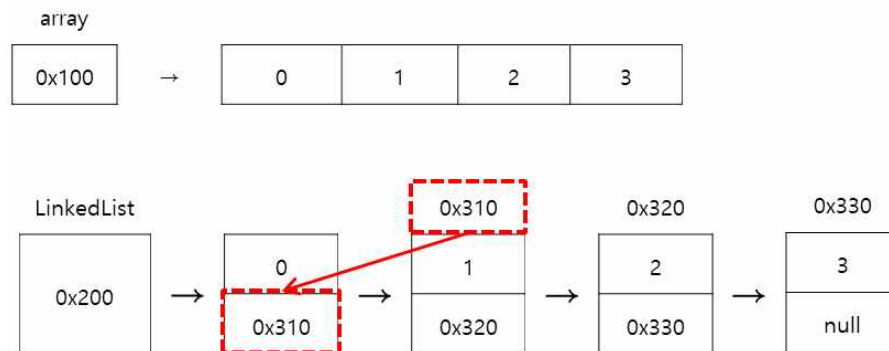
#### 1. 크기를 변경할 수 없다.

- 크기를 변경할 수 없으므로 새로운 배열을 생성해서 데이터를 복사하는 작업이 필요하다.
- 실행속도를 향상시키기 위해서는 충분히 큰 크기의 배열을 생성해야 하므로 메모리가 낭비된다.

#### 2. 비순차적인 데이터의 추가 또는 삭제에 시간이 많이 걸린다.

- 차례대로 데이터를 추가하고 마지막에서부터 데이터를 삭제하는 것이 빠르지만, 배열의 중간에 데이터를 추가하려면, 빈자리를 만들기 위해 다른 데이터들을 복사해서 이동해야 한다.

이러한 배열의 단점을 보완하기 위해서 LinkedList라는 자료구조가 고안 되었다. 배열은 모든 데이터가 연속적으로 존재하지만 LinkedList는 불연속적으로 존재하는 데이터를 서로 연결한 형태로 구성되어 있다.

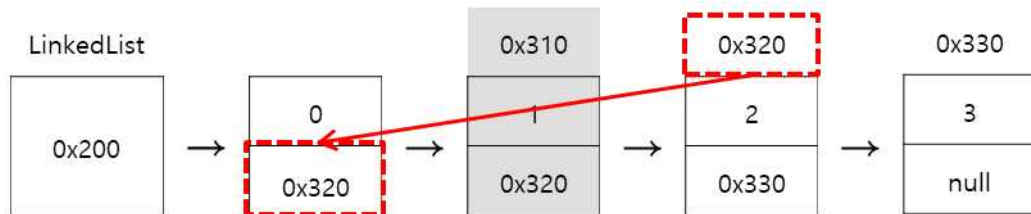


[그림 6-4] 배열과 LinkedList

그림 6-4에서 알 수 있듯이 LinkedList의 각 요소(node)들은 자신과 연결된 다음 요소에 대한 참조(주소값)와 데이터로 구성되어 있다.

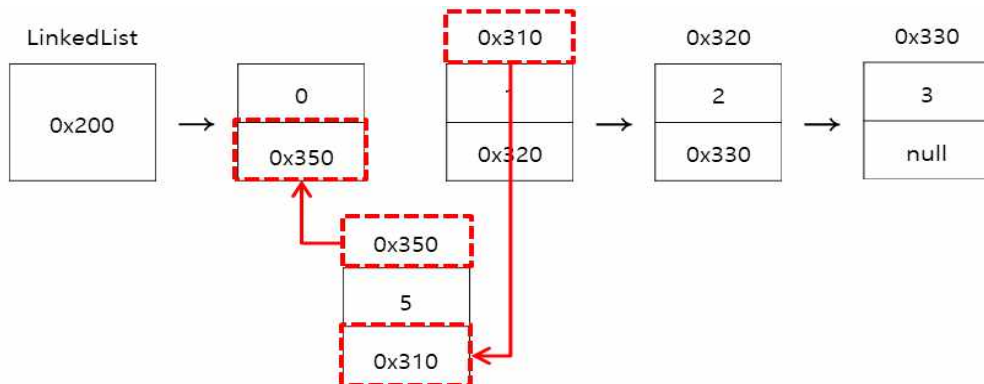
```
class Node{
    Object obj;    // 데이터를 저장
    Node next;    // 다음 요소의 주소를 저장
}
```

LinkedList에서의 데이터 삭제는 간단하다. 삭제하고자 하는 요소의 이전요소가 삭제하고자 하는 요소의 다음 요소를 참조하도록 변경하기만 하면 된다. 단 하나의 참조만 변경하면 삭제가 이루어지는 것이다. 배열처럼 데이터를 이동하기 위해 복사하는 과정이 없기 때문에 처리 속도가 매우 빠르다.



[그림 6-5] LinkedList에서의 데이터 삭제

새로운 데이터를 추가할 때는 새로운 요소를 생성한 다음 추가하고자 하는 위치의 이전 요소의 참조를 새로운 요소에 대한 참조로 변경해주고, 새로운 요소가 그 다음 요소를 참조하도록 변경하기만 하면 되므로 처리속도가 매우 빠르다.



[그림 6-6] LinkedList에서의 데이터 추가

링크드 리스트는 이동방향이 단방향이기 때문에 다음 요소에 대한 접근은 쉽지만 이전 요소에 대한 접근은 어렵다. 이점을 보완한 것이 DoubleLinkedList이다.

DoubleLinkedList는 단순히 링크드 리스트에 참조변수를 하나 더 추가하여 다음 요소에 대한 참조뿐 아니라 이전 요소에 대한 참조가 가능하도록 했을 뿐, 그 외에는 LinkedList와 같다.

```
class Node{
    Node previous; // 이전 요소의 주소를 저장
    Object obj;    // 데이터를 저장
    Node next;    // 다음 요소의 주소를 저장
}
```

#### 4. Stack과 Queue

스택은 마지막에 저장한 데이터를 가장 먼저 꺼내게 되는 LIFO(Last In First Out)구조로 되어 있고, 큐는 처음에 저장한 데이터를 가장 먼저 꺼내게 되는 FIFO(First In First Out)의 구조로 되어 있다.

쉽게 얘기하자면 스택은 동전통과 같은 구조로 양 옆과 바닥이 막혀 있어서 한 방향으로만 뺄 수 있는 구조이고, 코는 양 옆만 막혀 있고 위아래로 뚫려 있어서 한 방향으로만 넣고 한 방향으로만 빼는 파이프와 같은 구조로 되어 있다.

메서드	설 명
boolean empty()	Stack이 비어있는지 알려준다.
Object peek()	Stack의 맨 위에 저장된 객체를 반환. pop()과 달리 Stack에서 객체를 꺼내지는 않음.
Object pop()	Stack의 맨 위에 저장된 객체를 꺼낸다.
Object push(Object item)	Stack에 객체(item)를 저장한다.
int search(Object c)	Stack에서 주어진 객체(o)를 찾아서 그 위치를 반환. 못찾으면 -1을 반환.

[표 6-5] Stack의 메서드

메서드	설 명
boolean add(Object o)	지정된 객체를 Queue에 추가한다. 성공하면 true를 반환
Object remove()	Queue에서 객체를 꺼내 반환
Object element()	삭제없이 요소를 읽어온다.
boolean offer(Object o)	Queue에 객체를 저장, 성공하면 true, 실패하면 false를 반환
Object poll()	Queue에서 객체를 꺼내서 반환. 비어있으면 null반환
Object peek()	삭제없이 요소를 읽어 온다. Queue가 비어있으면 null을 반환

[표 6-6] Queue의 메서드

**스택의 활용 예** - 수식계산, 수식괄호검사, 웹브라우저의 뒤로/앞으로  
**큐의 활용 예** - 최근사용문서, 인쇄작업 대기목록



#### [예제 6-2] Stack과 Queue의 활용

```
public class Ex6_2 {
    public static void main(String[] args) {
        Stack st = new Stack();
        st.push("0");
        st.push("1");
        st.push("2");

        System.out.println("===== Stack =====");
        while(!st.empty()){
            System.out.println(st.pop());
        }

        Queue q = new LinkedList();
        q.offer("0");
        q.offer("1");
        q.offer("2");
        System.out.println("===== Queue =====");
        while(!q.isEmpty()){
            System.out.println(q.poll());
        }
    }
}
```

결과

```
===== Stack =====
2
1
0
===== Queue =====
0
1
2
```

스택과 큐에 각각 “0”, “1”, “2”를 같은 순서로 넣고 꺼내었을 때의 결과가 다른 것을 알 수 있다. 코는 먼저 넣은 것이 먼저 꺼내지는 구조(FIFO)이기 때문에 넣을 EO와 같은 순서이고, 스택은 먼저 넣은 것이 나중에 꺼내지는 구조(LIFO)이기 때문에 넣을 때의 순서와 반대로 꺼내진 것을 알 수 있다.

자바에서는 스택을 Stack클래스로 구현하여 제공하고 있지만 큐는 Queue인터페이스로만 정의해 놓았을 뿐 별도의 클래스를 제공하지 있지 않다. 대신 Queue인터페이스를 구현한 클래스들이 있어서 이들 중의 하나를 선택해서 사용하면 된다.

#### 스택응용 문제

순서가 A, B, C, D로 정해진 입력 자료를 스택에 입력하였다가 출력한 결과로 가능한 것이 아닌 것은?

- ① D, C, B A      ② B, C, D, A      ③ C, B, A, D      ④ D, B, C, A

## 5. Iterator

컬렉션 프레임워크에서 컬렉션에 저장된 요소들을 읽어오는 방법을 표준화하였다. 컬렉션에 저장된 각 요소에 접근하는 기능을 가진 `Iterator` 인터페이스를 정의하고, `Collection` 인터페이스에는 “`Iterator`(`Iterator`를 구현한 클래스의 인스턴스)“를 반환하는 `iterator()`를 정의하고 있다.

`iterator()`는 `Collection` 인터페이스에 정의된 메서드이므로 `Collection` 인터페이스의 자손인 `List`와 `Set`에도 포함되어 있다. 그래서 `List`나 `Set` 인터페이스를 구현하는 컬렉션은 `iterator()`가 각 컬렉션의 특징에 알맞게 작성되어 있다. 컬렉션 클래스에 대해 `iterator()`를 호출하여 `Iterator`를 얻은 다음 반복문, 주로 `while`문을 사용해서 컬렉션 클래스의 요소들을 읽어 올 수 있다.

메서드	설 명
<code>boolean hasNext()</code>	읽어 올 요소가 남아있는지 확인한다.
<code>Object next()</code>	다음 요소를 읽어 온다. <code>next()</code> 를 호출하기 전에 <code>hasNext()</code> 를 호출해서 읽어 올 요소가 있는지 확인하는 것이 안전하다.
<code>void remove()</code>	<code>next()</code> 로 읽어 온 요소를 삭제한다. <code>next()</code> 를 호출한 다음에 <code>remove()</code> 를 호출해야 한다.

[표 6-7] `Iterator` 인터페이스의 메서드

### [예제 6-3] `Iterator`의 활용

```
public class Ex6_3 {
    public static void main(String[] args) {

        List<String> list = new ArrayList<>();
        list.add("123");
        list.add("456");
        list.add("789");

        List<String> list2 = new ArrayList<>(list);

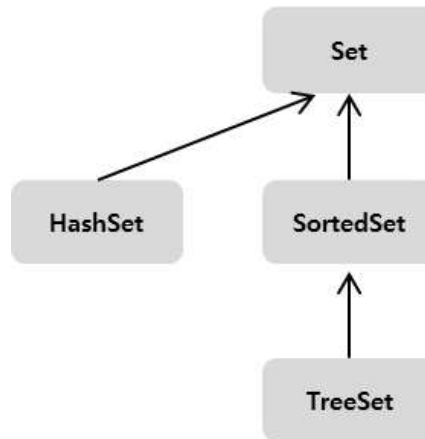
        //Iterator : 단방향 이동만 된다.
        Iterator<String> it = list.iterator();
        while(it.hasNext()){
            System.out.println("list : "+it.next());
            it.remove();
        }
    }
}
```

```
//ListIterator
//Iterator의 단방향성을 보완하기 위해 나왔다.
//단 List인터페이스를 구현한 컬렉션에서만 사용한다.
ListIterator<String> li = list2.listIterator();
System.out.println("list2 : "+li.hasNext());
System.out.println("list2 : "+li.next());
System.out.println("list2 : "+li.hasPrevious());
System.out.println("list2 : "+li.previous());
}
}
```

ListIterator는 Iterator를 상속받아서 기능을 추가한 것으로, 컬렉션의 요소에 접근할 때 Iterator는 단방향으로만 이동할 수 있는 데 반해 ListIterator는 양방향으로 이동이 가능하다. 다만 ArrayList나 LinkedList와 같이 List인터페이스를 구현한 컬렉션에서만 사용할 수 있다.

### ③ Set인터페이스

Set인터페이스는 중복을 허용하지 않고 저장순서가 유지되지 않는 컬렉션 클래스를 구현하는데 사용된다. Set인터페이스를 구현한 클래스로는 HashSet, TreeSet등이 있다.



[그림 6-7] Set의 상속계층도

#### 1. HashSet

HashSet은 Set인터페이스를 구현한 가장 대표적인 컬렉션이며, Set인터페이스의 특징대로 HashSet은 중복된 요소를 저장하지 않는다.

HashSet에 새로운 요소를 추가할 때는 add메서드나 addAll메서드를 사용하는데, 만일 HashSet에 이미 저장되어 있는 요소와 중복된 요소를 추가하고자 한다면 이 메서드들은 false를 반환함으로써 중복된 요소이기 때문에 추가에 실패했다는 것을 알린다.

이러한 HashSet의 특징을 이용하면, 컬렉션 내의 중복된 요소들을 쉽게 제거 할 수 있다.

ArrayList와 같이 List인터페이스를 구현한 컬렉션과 달리 HashSet은 저장순서를 유지하지 않으므로 저장순서를 유지하고자 한다는 LinkedHashSet을 사용해야 한다.

#### [문제 6-1] HashSet의 활용

```
public class Qu6_1 {  
    public static void main(String[] args) {  
        //1. Set인터페이스 타입의 참조변수 set에 HashSet객체의 주소를 저장하여라.  
  
        //2. 참조변수 set에 중복되지 않는 1~45사이의 임의의 정수를 6개 저장하여라.    }  
}
```

```

//3. 참조변수 set에 저장된 값을 출력하여라.

//4. 참조변수 set에 저장된 정수들을 오름차순으로 정렬하여라.


//5. 정렬된 정수를 출력하여라.

    }
}

```

## 2. TreeSet

TreeSet은 이진 검색 트리(binary search tree)라는 자료구조의 형태로 데이터를 저장하는 컬렉션 클래스이다. 이진 검색 트리는 정렬, 검색, 범위검색에 높은 성능을 보이는 자료구조이며 TreeSet은 이진 검색 트리의 성능을 향상시킨 '레드-블랙 트리'로 구현되어 있다.

그리고 Set인터페이스를 구현했으므로 중복된 데이터의 저장을 허용하지 않으며 정렬된 위치에 저장하므로 저장순서를 유지하지도 않는다.

### [예제 6-4] TreeSet의 활용

```

public class Ex6_4 {
    public static void main(String[] args) {
        TreeSet ts = new TreeSet();
        int[] score = {80, 95, 50, 35, 45, 65, 10, 100};

        for(int i = 0; i < score.length; i++){
            ts.add(new Integer(score[i]));
        }

        System.out.println("50보다 작은 값 : "+ ts.headSet(new Integer(50)));
        System.out.println("50보다 큰 값 : "+ ts.tailSet(new Integer(50)));
    }
}

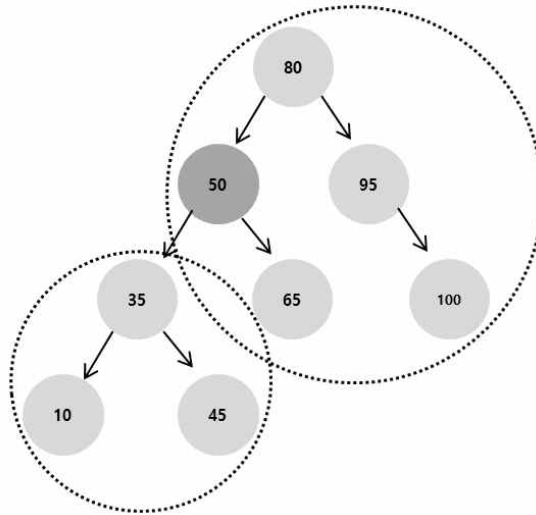
```

결과  
50보다 작은 값 : [10, 35, 45]  
50보다 큰 값 : [50, 65, 80, 95, 100]

headSet메서드와 tailSet메서드를 이용하면, TreeSet에 저장된 객체 중 지정된 기준 값보다 큰

값의 객체들과 작은 값의 객체들을 얻을 수 있다.

예제에 사용된 값들로 이진 검색 트리를 구성해 보면 다음 그림과 같다.



[그림 6-8] 예제 11-29에 사용된 데이터로 구성된 이진 검색 트리

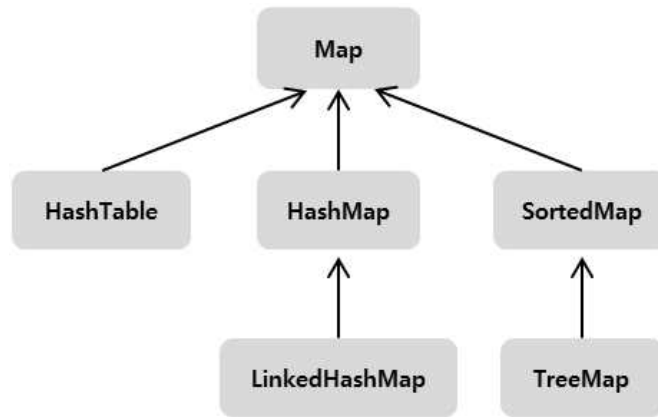
위의 그림을 보면 50이 저장된 노드의 왼쪽노드와 그 아래 연결된 모든 노드의 값을 50보다 작고, 나머지 다른 노드의 값들은 50보다 같거나 크다는 것을 알 수 있다.

#### 이진 검색 트리(binary search tree)는

- 모든 노드는 최대 두 개의 자식노드를 가질 수 있다.
- 왼쪽 자식노드의 값은 부모노드의 값보다 작고 오른쪽자식노드의 값은 부모노드의 값보다 커야한다.
- 노드의 추가 삭제에 시간이 걸린다. (순차적으로 저장하지 않으므로)
- 검색(범위검색)과 정렬에 유리하다.
- 중복된 값을 저장하지 못한다.

## ④ Map인터페이스

Map인터페이스는 키(key)와 값(value)을 하나의 쌍으로 묶어서 저장하는 컬렉션 클래스를 구현하는 데 사용된다. 키는 중복될 수 없지만 값은 중복을 허용한다. 기존에 저장된 데이터와 중복된 키와 값을 저장하면 기존의 값은 없어지고 마지막에 저장된 값이 남게 된다. Map인터페이스를 구현한 클래스로는 HashTable, HashMap, TreeMap 등이 있다.



[그림 6-9] Map의 상속계층도

메서드	설 명
HashMap()	HashMap객체를 생성
HashMap(int initialCapacity)	지정된 값을 초기용량으로 하는 HashMap객체를 생성
HashMap(Map m)	지정된 m의 모든 요소를 포함하는 HashMap을 생성
void clear()	Map의 모든 객체를 삭제한다.
boolean containsKey(Object key)	지정된 key객체와 일치하는 Map의 key객체가 있는지 확인 한다.
boolean containsValue(Object value)	지정된 value객체와 일치하는 Map의 value객체가 있는지 확인한다.
Set entrySet()	Map에 저장되어 있는 key-value쌍을 Map.Entry타입의 객체로 저장한 Set으로 반환한다.
boolean equals(Object o)	동일한 Map인지 비교한다.
Object get(Object key)	지정한 key객체에 대응하는 value객체를 찾아서 반환한다.
int hashCode()	해시코드를 반환한다.
boolean isEmpty()	Map이 비어있는지 확인한다.

Set keySet()	Map에 저장된 모든 key객체를 반환한다.
Object put(Object key, Object value)	Map에 value객체를 key객체에 연결하여 저장한다.
void putAll(Map t)	지정된 Map의 모든 key-value쌍을 추가한다.
Object remove(Object key)	지정한 key객체와 일치하는 key-value객체를 삭제한다.
int size()	Map에 저장된 key-value쌍의 개수를 반환한다.
Collection values()	Map에 저장된 모든 value객체를 반환한다.

[표 6-8] Map인터페이스에 정의된 메서드

values()에서는 반환타입이 Collection이고, keySet()에서는 반환타입이 Set인 것에 주목하자. Map인터페이스에서 값(value)은 중복을 허용하기 때문에 Collection타입으로 반환하고, 키(key)는 중복을 허용하지 않기 때문에 Set타입으로 반환한다.

#### [예제 6-5] HashMap의 활용

```
public class Ex6_5 {
    public static void main(String[] args) {
        Map<String, Integer> map = new HashMap<>();
        map.put("김자바", new Integer(90));
        map.put("한자바", new Integer(100));
        map.put("이자바", new Integer(80));
        map.put("강자바", new Integer(90));
        map.put("안자바", new Integer(100));
        System.out.println(map);

        System.out.println(map.get("김자바")); // 키에 해당하는 value값 가져오기
        // "김자바"라는 키를 포함한 요소 삭제
        map.remove("김자바");
        System.out.println(map);
        // 동일한 키가 있다면 새로운 값으로 변경
        map.put("김자바", new Integer(50));
        System.out.println(map);
    }
}
```

결과

```
{한자바=100, 강자바=90, 이자바=80, 김자바=90, 안자바=100}
90
{한자바=100, 강자바=90, 이자바=80, 안자바=100}
{한자바=100, 강자바=90, 김자바=50, 이자바=80, 안자바=100}
```



## 7-1. JDBC프로그래밍

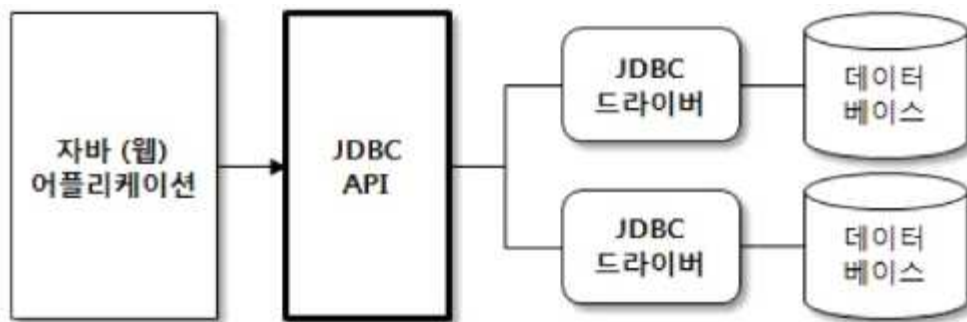
### 학습목표

- JDBC를 이용하여 데이터베이스 연동할 수 있다.

### 필요 지식 / DataBase(Oracle), Java

#### ① JDBC프로그래밍 이란?

Java DataBase Connectivity의 약자로서 자바 프로그램 내에서 DB와 관련된 작업을 처리할 수 있도록 도와주는 일을 한다. Java에서 데이터베이스를 사용할 때에는 JDBC API를 이용하여 프로그래밍 한다. 자바는 DBMS 종류에 상관없이 하나의 JDBC API를 사용하여 데이터베이스 작업을 처리할 수 있기 때문에 알아두면 어떤 DBMS든 작업을 처리할 수 있게 된다.



[그림 7-1] JDBC API사용 애플리케이션의 기본 구성

JDBC API에 없었던 옛날에는 각각의 데이터베이스마다(ms-sql, my-sql, oracle ..) 각각의 SQL 문을 사용했다. 그래서 DB의 종류에 따라 SQL문의 작성 방법이 너무나 차이가 나서 개발자들의 항의가 많았다. 그러다가 더 이상 무한 경쟁을 하지 말고 메서드나 일부 용어를 통일하여 만들자고 회사 간의 합의가 이루어졌다.

그 결과로 function이나 structure , 전역변수 등을 통합하여 같은 문법으로 통일시키고 JDBC API에 명세하였다. 즉, JDBC API를 사용할 경우 DBMS에 알맞은 JDBC드라이버만 있다면 어떤 DB라도 사용 가능하게 되었다.

본 장에서는 오라클(Oracle) DBMS를 활용하는 방법을 알아보도록 하자.

#### ② 오라클 JDBC드라이버 다운로드 및 등록

오라클을 이용한 프로그래밍을 위해서 오라클의 JDBC드라이버를 설치해야 한다. eclipse를 이

용한 프로그래밍을 할 때 ojdbc6.jar파일을 프로젝트에 추가해 주면 된다.

## 1. 오라클 드라이버 다운 ( oracle 11g 기준)

‘<https://www.oracle.com/database/technologies/jdbcdriver-ucp-downloads.html>’ 을 홈페이지에서 ojdbc6.jar파일을 다운로드 한다.



Download	Release Notes
 ojdbc6.jar	(2,739,670 bytes) - (SHA1 Checksum: a483a046eee2f404d864a6ff5b09dc0e1be3fe6c) Certified with JDK8, JDK7, and JDK6;
 ojdbc5.jar	(2,091,137 bytes) - (SHA1 Checksum: 5543067223760fc2277fe3f603d8c4630927679c) For use with JDK1.5;
 ucp.jar	(506,301 bytes) - (SHA1 Checksum: 5520b4e492939b477cc9ced90c03bc72710dcaf3)
 ojdbc.policy	(10,591 bytes) - Sample security policy file for Oracle Database JDBC drivers

[그림 7-2] ojdbc6.jar 다운로드

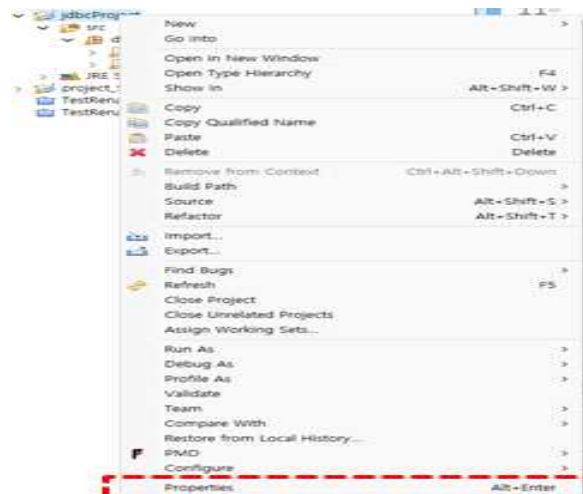
## 2. eclipse에 jar파일 등록하기

1에서 다운받은 ojdbc6.jar파일을 프로그래밍에 사용하기 위해서는 만들어진 프로젝트에 등록을 해주어야 한다. jar파일을 등록하는 방식중에 한 가지 방법을 알아보자.

### (1) 프로젝트 설정으로 이동

프로젝트명에 오른쪽 마우스를 클릭하여 'properties'를 선택한다.

(단축키 프로젝트 클릭 후 Alt+Enter)

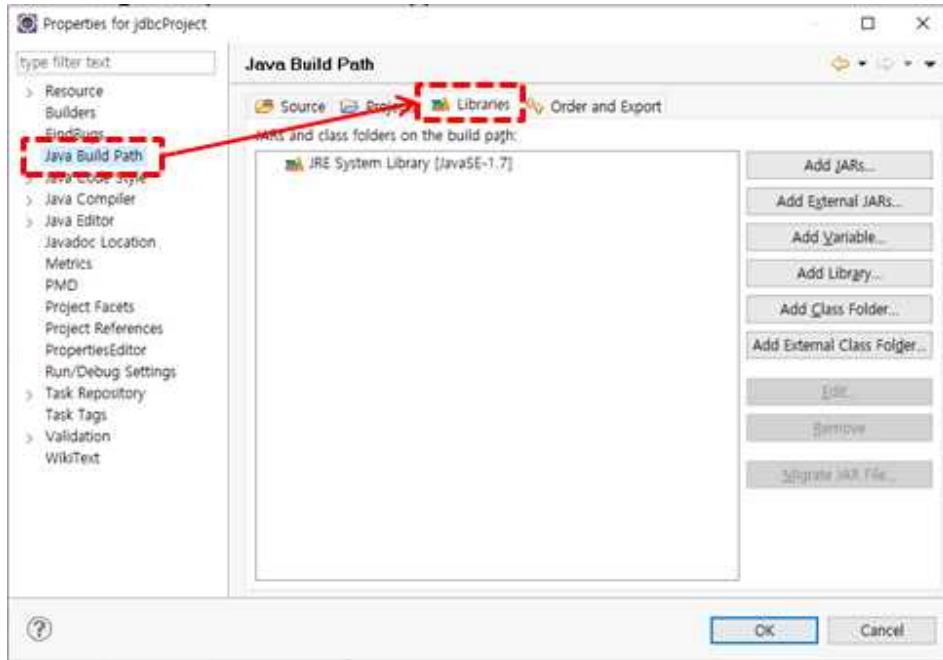


[그림 7-3] 프로젝트설정

### (2) 라이브러리 설정으로 이동

왼쪽의 여러 가지 설정 중에서 'Java Build Path'를 선택을 하고 오른쪽 탭에서 'Libraries'를

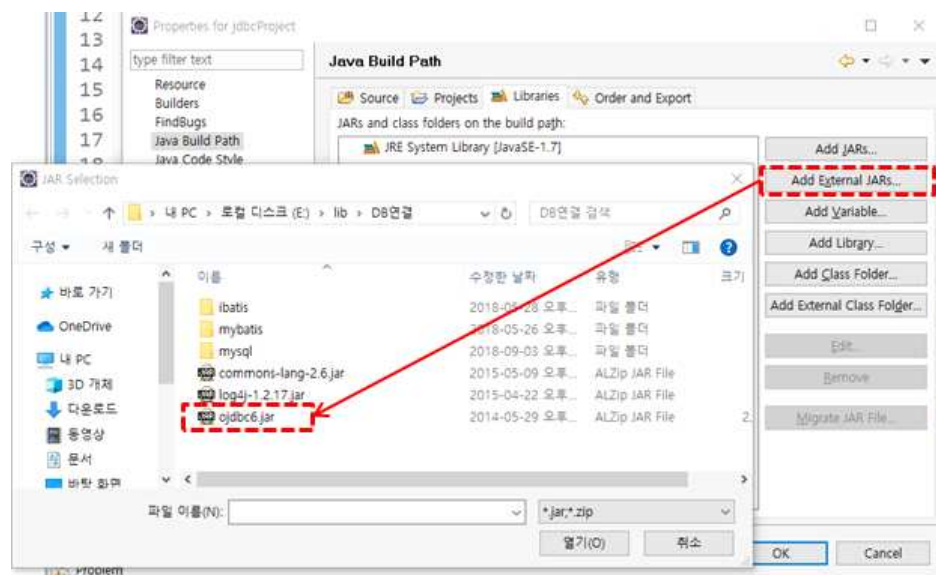
선택한다.



[그림 7-4] 라이브러리 설정

### (3) jar파일 등록

우측의 항목 중 'Add External Jars...'를 선택하면 파일 선택창이 나오게 되는데 다운로드한 jar파일을 선택하여 준다.



[그림 7-5] jar파일 등록

## ③ JDBC의 활용

## 1. JDBC활용 순서

JDBC 드라이버를 등록하였다면 활용하는 방법에 대하여 알아보도록 하자.

JDBC를 이용한 데이터베이스의 처리 순서는 아래와 같다.

순서	설명	사용법
1	JDBC드라이버 로딩	Class.forName("oracle.jdbc.driver.OracleDriver");
2	DataBase접속	DriverManager.getConnection()
3	질의(SQL명령 수행)	Statement객체 또는 PreparedStatement객체를 이용
4	질의결과 수집	질의문장이 select의 경우 ResultSet객체를 이용 그 외의 경우 정수 값(성공한 record수)을 반환.
5	종료(자원반납)	생성된 자원 반납

[표 7-1] 데이터베이스 처리순서

### [예제 7-1] JDBC를 이용한 데이터베이스 처리

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class Ex7_1 {
    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // 1. 드라이버 로딩(옵션)
            Class.forName("oracle.jdbc.driver.OracleDriver");
            // 2. DB에 접속 (Connection객체 생성)
            String url = "jdbc:oracle:thin:@localhost:1521/xes";
            String userId = "sem";
            String password = "java";
            // 실제로 OracleDriver가 사용되는 부분
            conn = DriverManager.getConnection(url, userId, password);
            // 3. Statement객체 생성 ==> Connection객체를 이용한다.
```

```

stmt = conn.createStatement();
// 4. SQL문을 Statement객체를 이용하여 실행하고
// 실행결과를 ResultSet객체에 저장한다.
String sql = "select * from lprod"; // 실행할 SQL문
rs = stmt.executeQuery(sql); // SQL문이 select일 경우에는
// 5. ResultSet객체에 저장되어 있는 자료를 반복문과 next()메서드를
// 이용하여 차례로 읽어와 처리한다.
System.out.println("실행한 쿼리문 : " + sql);
System.out.println("=== 쿼리문 실행 결과 ===");
while(rs.next()){
    System.out.println("lprod_id : " + rs.getInt("lprod_id"));
    System.out.println("lprod_gu : " + rs.getString("lprod_gu"));
    System.out.println("lprod_nm : " + rs.getString("lprod_nm"));
    System.out.println("-----");
}
System.out.println("출력 끝...");
} catch (SQLException e) {
    e.printStackTrace();
} catch (ClassNotFoundException e) {
    e.printStackTrace();
} finally{
    // 6. 종료( 사용했던 자원을 모두 반납한다. )
    if(rs!=null)try {
        rs.close();
    } catch (SQLException e2) {}
    if(stmt!=null)try {
        stmt.close();
    } catch (SQLException e2) {}
    if(conn!=null)try {
        conn.close();
    } catch (SQLException e2) {}
}
}
}

```

결과

```

실행한 쿼리문 : select * from lprod
=== 쿼리문 실행 결과 ===
lprod_id : 1
lprod_gu : P101
lprod_nm : 컴퓨터제품
-----
lprod_id : 2
lprod_gu : P102
lprod_nm : 전자제품
-----
lprod_id : 3
lprod_gu : P201
lprod_nm : 여성캐주얼
-----
lprod_id : 4
lprod_gu : P202
lprod_nm : 남성캐주얼
-----
출력 끝...

```

위의 예제에서 데이터베이스에 저장된 데이터를 읽어(Read)오는 select쿼리는 executeQuery()를 사용하지만 생성(Create), 갱신(Update), 삭제>Delete)의 경우에는 executeUpdate()를 사용해야 한다. 또한 컬럼의 자료를 가져오는 방법으로 'rs.get자료형이름("컬럼명")'사용하였지만 'rs.get자료형이름(컬럼번호)'를 사용할 수 있다. 단. 컬럼번호는 1부터 시작이다.

## 2. DTO와 VO

DTO는 Data Transfer Object의 약자로 레이어를 이동할 때 데이터를 들고 있는 객체를 말한

다. 풀 명칭에서 유추할 수 있듯이, 데이터를 오브젝트로 변환하는 객체이다.

VO는 Value Object의 약자로 값 오브젝트이다. 값 오브젝트는 말 그대로 값을 위해 쓰는 것이다. DTO와 동일한 개념이나 차이점은 Read-Only 속성 객체이다.

DTO와 VO모두 데이터를 저장하고 있다는 점이 있지만 프로젝트마다 정의를 다르게 하고 있기 때문에 프로젝트의 정의에 따라 사용하는 것이 좋다.

#### [예제 7-2] MemberVO

```
public class MemberVO {  
    private String mem_id;           //회원ID  
    private String mem_pass;        //패스워드  
    private String mem_name;        //회원명  
    private String mem_regno;       //생년월일  
    private String mem_add;         //집주소 mem_add1+mem_add2  
    private String mem_hp;          //전화번호  
    private int mem_mileage;         //마일리지  
    private String mem_job;         //직업  
  
    public String getMem_id() {  
        return mem_id;  
    }  
    public void setMem_id(String mem_id) {  
        this.mem_id = mem_id;  
    }  
    public String getMem_pass() {  
        return mem_pass;  
    }  
    public void setMem_pass(String mem_pass) {  
        this.mem_pass = mem_pass;  
    }  
    public String getMem_name() {  
        return mem_name;  
    }  
    public void setMem_name(String mem_name) {  
        this.mem_name = mem_name;  
    }  
    public String getMem_regno() {  
        return mem_regno;  
    }  
    public void setMem_regno(String mem_regno) {
```

```

        this.mem_regno = mem_regno;
    }
    public String getMem_add() {
        return mem_add;
    }
    public void setMem_add(String mem_add) {
        this.mem_add = mem_add;
    }
    public String getMem_hp() {
        return mem_hp;
    }
    public void setMem_hp(String mem_hp) {
        this.mem_hp = mem_hp;
    }
    public int getMem_mileage() {
        return mem_mileage;
    }
    public void setMem_mileage(int mem_mileage) {
        this.mem_mileage = mem_mileage;
    }
    public String getMem_job() {
        return mem_job;
    }
    public void setMem_job(String mem_job) {
        this.mem_job = mem_job;
    }
}

```

#### [문제 7-1] 회원들의 목록 가져오기

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class Qu7_1 {
    public static void main(String[] args) {

        //1. 교사용 PC의 오라클 DataBase에 접속하여 회원들의 정보를 memList에
        // 저장하여야.
    }
}

```

```
// 교사용 PC의 ip : 192.168.호실.1, port : 1521, SID : xe, id : sem, pw : java
List<MemberVO> memLlst = new ArrayList<>();
```

```
}
```

```
}
```

### 3. Statement와 PreParedStatement

Statement와 PreParedStatement의 가장 큰 차이점은 캐시(cache) 사용여부이다. 쿼리의 수행은 쿼리문장 분석, 컴파일, 실행의 순서로 수행되게 된다. Statement를 사용하면 매번 쿼리를 수



행할 때마다 쿼리수행을 위한 3단계를 거치게 되고, PreparedStatement는 처음 한번만 세 단계를 거친 후 캐시에 담아 재사용을 한다는 것이다. 만약 동일한 쿼리를 반복적으로 수행한다면 PreparedStatement가 DB에 훨씬 적은 부하를 주며, 성능도 좋다.

#### (1) Statement

```
String sqlstr = "SELECT name, memo FROM TABLE WHERE num = " + num
Statement stmt = conn.createStatement();
ResultSet rst = stmt.executeQuery(sqlstr);
```

Statement는 executeQuery()나 executeUpdate()를 실행하는 시점에 파라미터로 SQL문을 전달하는데, 이 때 전달되는 SQL 문은 완성된 형태로 한눈에 무슨 SQL 문인지 파악하기 쉽다. 하지만, 이 녀석은 SQL문을 수행하는 과정에서 매번 컴파일을 하기 때문에 성능상의 이슈가 있다.

#### (2) PreparedStatement

```
String sqlstr = "SELECT name, memo FROM TABLE WHERE num = ? "
PreparedStatement stmt = conn.prepareStatement(sqlstr);
pstmt.setInt(1, num);
ResultSet rst = pstmt.executeQuery();
```

PreparedStatement은 이름에서부터 알 수 있듯이 준비된 Statement 이다. 이 준비는 컴파일 (Parsing) 을 이야기하며, 컴파일이 미리 되어있는 녀석이기에 Statement 에 비해 성능상 이점이 있다. 요 녀석은 보통 조건절과 함께 사용되며 재사용이 되는데, ? 부분에만 변화를 주어 지속적으로 SQL을 수행하기 때문에 한눈에 무슨 SQL 문인지 파악하기는 어렵다.

## ④ Singleton Pattern

싱글톤 패턴은 디자인 패턴중 하나로 애플리케이션이 시작될 때 어떤 클래스가 최초 한번만 메모리를 할당하고 그 메모리에 인스턴스를 만들어 사용하는 디자인 패턴이다.

생성자가 여러 차례 호출되더라도 실제로 생성되는 객체는 하나고 최초 생성 이후에 호출된 생성자는 최초에 생성한 객체를 반환한다. 생성자에 접근제어자를 `private`로 선언해서 객체의 생성을 제한하고 메서드를 통해서만 객체를 취득할 수 있도록 한다.

싱글톤 패턴을 사용하는 이유는 아래와 같다.

1. 고정된 메모리 영역을 얻으면서 한번의 `new`로 인스턴스를 사용하기 때문에 메모리 낭비를 방지할 수 있다.
2. 싱글톤으로 만들어진 클래스의 인스턴스는 전역 인스턴스이기 때문에 다른 클래스의 인스턴스들이 데이터를 공유하기 쉽다.

#### [예제 7-3] Singleton Pattern 활용

```
public class Ex7_3 {
    public static void main(String[] args) {
        Service sv1 = Service.getInstance();
        Service sv2 = Service.getInstance();
        System.out.println(sv1 == sv2);    // true
    }
}

public class Service{
    static Service service;

    private Service(){

    }

    static Service getInstance(){
        if(service == null){
            service = new Service();
        }
        return service;
    }
}
```



- 최범균(2013). “JSP2.2 웹프로그래밍“. 가메출판사.
- 최희탁(2014). “서블릿 컨테이너의 이해“. 한빛미디어
- 전준식(2014). “거침없이 배우는 JBOSS(오픈소스 미들웨어 JBOSS EAP 6 & AS 7 이해하기)“. 지앤선
- 피터 클라리엔 저/유윤선 역(2010).  
“스프링 시큐리티3 스프링 프레임워크 기반 표준 보안 솔루션“위키북스
- 이동국(2013) “마이바티스 프로그래밍“ 에이콘
- 이일민(2014) “토비의 스프링 3.1“ 에이콘
- 고모리유키 저/김정환 역(2012) “프로가 되기위한 웹 기술 입문“위키북스
- 전자정부프레임워크 위키. (<http://www.egovframe.go.kr/wiki/doku.php>)
- JAVA EE 7 Technologies. (<http://www.oracle.com/technetwork/java/javaee/tech/index.html>)
- JAVA EE 7 Tutorial. (<https://docs.oracle.com/javaee/7/tutorial/>)
- Tomcat Document. (<http://tomcat.apache.org>)
- WebLogic Document.  
([http://docs.oracle.com/cd/E21764\\_01/web.1111/e13712/configurewebapp.htm](http://docs.oracle.com/cd/E21764_01/web.1111/e13712/configurewebapp.htm))
- Commons library. (<http://commons.apache.org>)
- Jackson library. (<https://github.com/FasterXML/jackson>)
- MyBatis Product (<http://blog.mybatis.org/p/products.html>)
- MyBatis Document (<http://mybatis.github.io/mybatis-3/>)
- MAVEN. (<http://maven.apache.org>)
- Spring Reference :  
(<http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle>)
- Spring Security Reference :  
(<http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle>)
- JAXB Tutorial. (<https://docs.oracle.com/javase/tutorial/jaxb/intro/index.html>)
- 국제 표준 규격(ISO/IEC 29119,ISO/IEC 25000,ISO/IEC 9126,IEEE 829) 기준
- 공공 부문 공개 SW 적용지원 센터의 [공개 SW 테스트 가이드]
- 미래창조과학부 SW공학센터의 S[W 개발 품질 관리 매뉴얼]

- STEN 테스트 관련 학습 서비스
- STA테스팅컨설팅 테스트 관련 학습 서비스
- 미래창조과학부(소프트웨어자산뱅크)의 개발 도우미 서비스
- 정보통신산업진흥원(공개SW역량프라자)의 정보마당 서비스