

자 바

1부 자바의 시작

1 자바언의 기본

1.1 자바의 개요

1.1.1 자바 역사

자바는 1990년대 초 미국 Sun Micro 사의 James Gosling이 가전 제품에 이용할 목적으로 파스칼을 모델로 개발하였으나 별다른 반응을 얻지 못했다. 인터넷이 급속도로 확산된 90년대 중반에야 가서 관심의 초점이 되었다. 자바의 역사를 간단히 살펴 보자.

- 1991년 선의 엔지니어들에 의해서 고안된 오크(Oak)라는 언어에서 부터 시작됨.
- 제임스 고슬링과 그와 함께한 엔지니어들은 처음에 C++를 확장하여 가전 제품에 탑재될 소프트웨어를 만드는 것이 목표였으나 C++로 만들기엔 한계가 너무 많다는 것을 알고 C++의 장점을 살려 새로운 언어를 개발하기로 하고하였으나 이미 테스트 되고 있는 Oak언어가 적합하다고 판단하여 그 개발 방향을 바꾸고 Oak이름을 java로 변경하여 추진.
- 1995년 자바로 개발한 웹 브라우저인 핫자바(Hot java)를 발표하고 다음 해에 자바 정식 버전을 발표하게 됨.

1.1.2 자바의 특징

플랫폼의 독립성 - 자바로 작성된 프로그램은 하부의 플랫폼과는 상관없이 자바의 가상 머신 위에서만 운용된다.

객체지향 언어 - 자바는 설계 당시부터 객체 지향성을 고려하여 설계되었기 때문에 C++보다 더욱 막강하고 완벽한 객체 지향성을 보장한다. 따라서 프로그램의 재 사용성과 생산성이 놀랍도록 향상 되었다.

멀티 스레드 지원 - 자바는 멀티 스레드를 언어 차원에서 지원한다. 이 멀티 스레드는 메모리 공유가 가능하면서 프로그램이 보다 효율적으로 실행될 수 있도록 해 준다.

자동 메모리 관리 - 자바 가상 머신이 자동으로 사용되지 않는 메모리를 찾아 해제해 준다.

동시성능 확장 제공 - 프로그램의 성능이 확장되었거나 개선되었을 때 자바는 네트워크를 통해 자동으로 다운로드 되어 설치될 수 있다. 따라서 사용자들은 늘 새로운 기능을 제공받을 수 있고 사용할 수 있게 된다.

네트워크와 분산처리 지원 - 인터넷과 대규모 분산처리 환경을 염두해 두고 개발된 언어임으로 비교적 짧은 시간에 네트워크와 분산처리 지원 프로그램을 개발할 수 있다.

1.2 자바의 구조

자바의 구조를 이루는 부분은 크게 자바 가상 머신과 바이트 코드라는 개념이다.

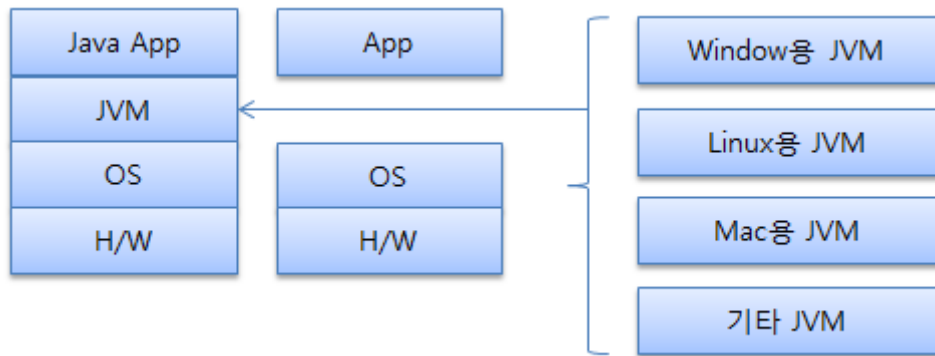
1.2.1 바이트 코드

대부분의 언어는 컴파일러, 어셈블러, 인터프리터라는 번역기에 의해 해석되어 그 해석 결과를 해당 O/S 가 실행할 수 있는 기계어로 바뀌어져 실행된다. 따라서 window에서 번역된 실행 명령어가 Mac에선 실행되지 않는다.

그러나 자바는 컴파일을 실행하면 기계어가 아닌 **"바이트 코드"**라는 중간 단계의 컴파일 결과를 생성한다. 이 코드는 하드웨어나 O/S와는 무관하게 작동되며 이는 자바가 설치되어 있으면 재 컴파일 없이 바로 될 수 있도록 되어 있다.

1.2.2 자바 가상 머신

자바 가상 머신이란 소프트웨어적으로 만들어진 일종의 CPU라 생각하면 쉽겠다. 이 자바 가상 머신은 단독으로 존재 할 수도 있지만 다양한 웹 브라우저 내에 존재 할수 도 있다. 자바 가상 머신이 바로 위에서 설명한 바이트 코드를 실행시켜 주는 소프트웨어 CPU이다.



OS와 사용자 App 사이에 다양한 플랫폼을 지원하는 JVM이 존재하기 때문에 개발자는 특정 플랫폼에 종속적이지 않고 프로그램을 개발 할 수 있는 것이다. 반면 C++과 같은 언어는 특정 플랫폼에 종속적 으로 컴파일 되기 때문에 다른 플랫폼에서 실행되기 힘들다.

1.2.3 자바 프로그램의 유형

1) Applet

애플릿은 네트워크의 원격 컴퓨터에서 전송받아 웹 브라우저나 애플릿 뷰어에서 실행되는 프로그램의 형태이다. 자바의 대표적인 실행 형태이다. 이 형태는 대부분이 웹 서버에 위치하고 있다가 각 클라이언트에게 전송되어 실행되어 진다. 그러나 보안 이슈등의 이유로 많이 사용되지 않는다.

2) Application

독립적인 자바 실행 프로그램의 형태를 갖는다. 이것은 자바 가상 머신이 위치하고 있는 어떤 플랫폼에서도 실행이 가능하지만 네트워크에서 다운로드되어 동적으로 실행할 수는 없다. 그러나 Applet에 비해 시스템의 자원에 쉽게 접근할 수 있다는 장점을 갖는다. 하지만 UI를 갖고 있는 프로그램들 또한 독립적인 실행 파일 생성이 복잡하고 다른 언어들로 만들어진 프로그램들 보다 상대적으로 느린 속도를 보이기 때문에 많이 사용되지 않는다. 그럼에도 불구하고 자바 언어의 사용이 항상 상위에 링크되는 이유는 하드웨어에 밀접한 관계를 갖고 있는 네트워크와 같이 백 그라운드에서 실행되는 프로그램을 작성할 때 특정 하드웨어에 종속적이지 않는 자바가 많이 사용된다.

3) Servlet

웹 서버를 통하여 서비스를 제공하는 형태로 기존의 CGI를 대체하는 기능을 제공한다. CGI 보다 성능이 월등히 높다.

4) Beans

자바로 만들어진 일종의 컴포넌트이다. 컴포넌트란 독립적인 기능과 화면을 갖고 있는 실행 모듈로서

프로그램의 재생산성을 향상시킨다. 따라서 빈즈는 독립적으로 사용되기 보다는 Applet, Application, Servlet 등에서 부품으로 사용된다.

1.3 자바 설치

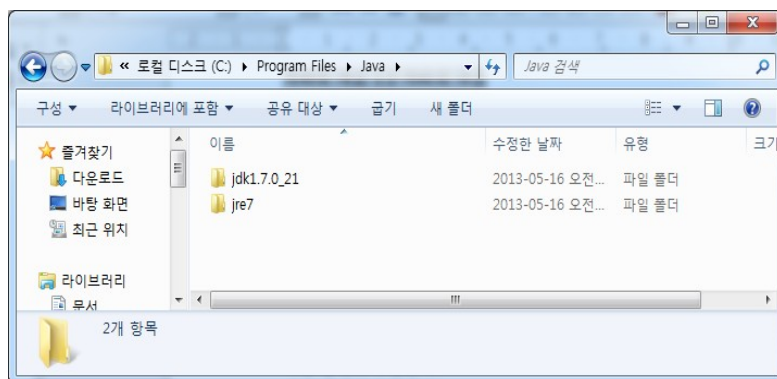
1.3.1 다운로드 및 설치

<http://www.oracle.com> 사이트를 방문하여 사용하고 있는 OS에 맞는 자바 JDK를 다운로드 받아 기본값으로 설치하면 끝이다.

이때 주의 해야 할 점은 자바 설치 파일이 jdk로 시작하는 것이 있고 jre로 시작하는 설치파일이 있다. 단지, 자바로 만들어진 파일을 실행하기 위해서라면 jre로 시작하는 설치파일을 설치해도 상관없지만 개발하려 할 때는 반드시 jdk용 설치 파일을 사용하여 설치 해야 한다.

윈도우에 기본값으로 설치했다면 "C:\Program Files\Java" 폴더안에 설치된 자바 버전명으로 하위 폴더가 만들어져 설치된다.

아래의 그림은 설치된 후의 예이다.



위에서 설명한것 처럼 설치된 디렉토리를 살펴 보면 2개의 하위 디렉토리가 생성되는데 jdk로 시작하는 폴더는 실행및 개발과 관련된 프로그램들이 설치되어 있는 디렉토리이며, jre로 시작하는 디렉토리는 자바 프로그램을

실행만할 수 있는 여러가지 내용들만 설치된 디렉토리가 된다.

자바로 만들어진 프로그램들을 실행할 때는 jre로 시작하는 디렉토리만 있어도 되지만 개발할 때는 반드시 jdk로 시작하는 디렉토리가 필요하다.

1.3.2 환경설정파일 등록

설치된 자바 JDK를 자바가 필요한 프로그램들이 원활히 사용하려 하면 세 종류의 환경 변수를 등록해야 한다.

- CLASSPATH
- PATH
- JAVA_HOME

1.3.3 CLASSPATH

JVM에 의해 실행 가능한 파일을 만들려면 javac.exe 를 사용하여 컴파일 해야 한다. 이때 정상적으로 컴파일되면 확장자가 "class"인 파일이 만들어지는데 이 파일들의 위치(경로)를 지정하는 환경변수가 "CLASSPATH"이다. 그러나 전문 개발 툴을 사용하여 개발하는 경우에는 대부분의 개발 툴이 자체적으로 소스들을 관리하기 위한 체계를 갖고 있기 때문에 CLASSPATH에는 현재 경로에서 "class"파일을 찾으라는 의미로 ";" 만을 지정하는 경우가 많다.

1.3.4 PATH

해당 OS에서 실행할 수 있는 실행 가능한 파일의 경로를 지정하는 환경 변수이다. 윈도우를 예를 들면 확장자의 종류가 "com","exe","bat"중에 하나이다. 따라서 자바와 관련된 컴파일러, 실행파일, 기타 실행파일들의 경로를 추가해 주어야 하는데 기본값으로 자바를 설치했다면

"C:\Program Files\Java\JDK1.7.0_21\bin"

과 같은 디렉토리를 path에 등록해 주면 자바와 관련된 각종 프로그램을 시키기 위해서 위의 경로로 이동하지 않아도 된다. 그러나 만약 path에 위의 문장이 등록되어 있지 않거나 오타자가 발생하면 자바를 정상적으로 실행시킬 수 없을 것이다.

JDK1.7.0_21에서 "1.7.0_21"은 설치되는 버전과 릴리즈버전에 따라 달라질 수 있다.

1.3.5 JAVA_HOME

환경변수명에서도 의미하듯이 자바가 설치되어 있는 메인 경로를 의미한다.

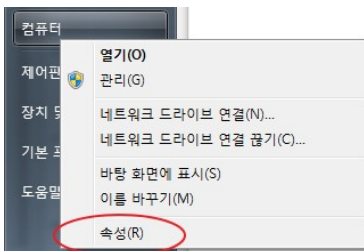
"C:\Program Files\Java\JDK1.7.0_21"

자바를 기본값으로 설치했다면 위와 같은 경로가 될 것이다.

[실습] win7을 기준으로 설정해 보자.

Step1.

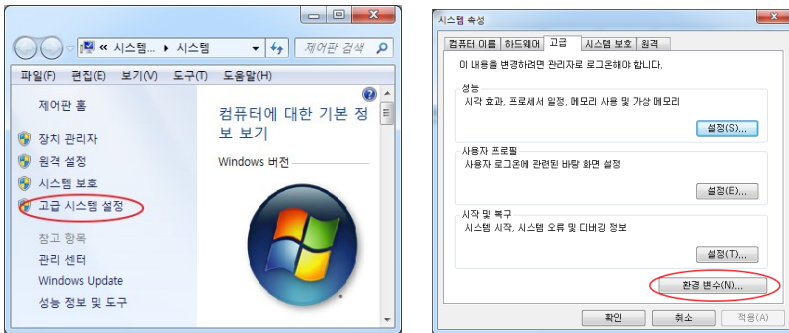
먼저 시작버튼에서 내컴퓨터 아이콘을 찾아 마우스 오른쪽 버튼을 클릭한다. 다음으로 '속성' 메뉴를 클릭한다.



Step2.

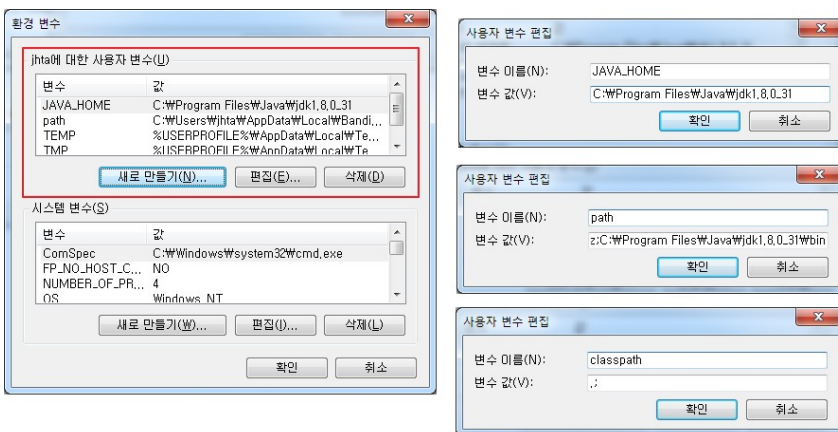
왼쪽 메뉴들중 '고급 시스템 설정'을 선택하고 고급>환경변수를 선택한다.

자바의 개요 1.3 자바 설치



Step3.

환경 변수를 설정 하는 구역이 두 구역으로 나뉘어져 있는데 왼쪽 구역은 현재 사용자에게 관한 환경 변수를 설정하는 것이고 아래쪽 구역은 시스템 전반에 걸쳐 영향을 미치는 환경 변수 내용들이다. 특별한 사유가 없으면 현재 사용자 환경 변수 구역에 CLASSPATH, PATH, JAVA_HOME의 내용을 수정하거나 새로 만들어 작업을 완료한다.



1.4 자바의 기본 구조

1.4.1 기본 구조

자바를 사용하여 무언가를 하려면 반드시 class구조 안에서 해야 하며, 컴파일하여 실행할 수 있는 구조도 class 이다. 따라서 비록 한 문자를 출력하려고 해도 class라는 구조를 갖고 있어야 한다.

```
package 패키지명 ----- 1)
import 풀 네임의 클래스명 ----- 2)
class 클래스명{ //----- 3)
    접근자 멤버 변수; // ----- 4)
    접근자 반환형 메소드(매개 변수){}; //----- 5)
    public static void main(String[] args){ --- 6)
        메인 프로그램 // ----- 7)
    }
}
```

[코드 설명]

1) package

class가 위치하고 있는 장소를 의미한다. 일반적으로 도메인의 역순으로 package명을 기술하는데 이는 class 이름이 동일하더라도 서로 구분될 수 있도록 하기 위한 묵시적인 약속이기도 하다. 동일한 도메인을 갖고 있더라도 class의 목적이나 성격을 구분하기 위해서도 package를 선언할 수 있다. package는 아래와 같은 특징을 갖고 있다.

- 하나의 class 파일엔 하나의 package만 사용할 수 있다.
- 주석을 제외한 코드의 첫줄에 작성해야 한다.
- 저장장치내에서는 하나의 폴더로 만들어 진다.

예를 들어 보자.

개발사의 도메인이 jobtc.kr이라고 가정하면 해당 개발사에서 개발되어 배포되는 클래스들은 모두 kr.jobtc라는 패키지명을 갖는게 원칙이다. 따라서 class를 작성할 때 아래와 같은 방법으로 package명을 작성한다.

```
package kr.jobtc;
```

만약 해당 개발사에서 member 라는 프로젝트로 class를 묶어야 한다면 아래와 같다.

```
package kr.jobtc.member;
```

2) import

현재 작성되고 있는 클래스와 동일한 패키지에 있는 다른 클래스는 이름만 가지고도 사용할 수 있지만, 다른 패키지에 있는 클래스들은 **패키지명.클래스명**처럼 패키지명을 지정해야 사용할 수 있다. 클래스명을 패키지명과 같이 사용하는 것을 풀네임이라 하는데, 코드상에서 클래스명을 항상 풀네임으로 기술하기엔 다소 어려움이 있거나 코드를 분석하는데 방해요소가 된다. 예를 들면 아래와 같다.

```
public kr.jobtc.member.Member getData(kr.jobtc.membe.Member m){ ... }
```

위와 같은 코드를 import 문장과 더불어 사용하면 아래와 같이 단출하게 사용된다.

```
import kr.jobtc.member.Member
...
public Member getData(Member m){ ... }
```

즉, import문장은 외부 패키지에 있는 클래스를 참조하도록 하는 코드이며, 코드상에서 클래스명을 풀네임으로 사용하지 않아도 되도록 해주는 기능을 갖고 있다.

import 문장은 필요한 갯수만큼 사용할 수 있으며 특정 패키지안에 있는 모든 클래스를 참조할 수 있도록 만들 수도 있다. 아래는 kr.jobtc.member 패키지안에 있는 모든 클래스를 참조하도록 지정한 것이다.

```
import kr.jobtc.member.*
```

1) 자바의 모든 실행 코드 중 package와 import문을 제외 하고는 반드시 class 영역 안에 기술되어야 한다. 또한 자바는 기본적으로 class뒤에 쓰여진 클래스명으로 파일명을 지정해야 하며 대부분의 경우 하나의 파일 안에 하나의 클래스를 지정한다. 클래스명은 대소문자 구분 없이 지정할 수는 있으나 일반적으로 대문자로 시작한다.

2) 멤버 변수.

클래스 내부 전 지역에서 사용할 수 있는 변수를 의미한다. 다른 용어로 필드, 인스턴스형 변수, 전역변수라는 이름으로도 사용된다. 자바에서는 필드란 단어로 사용한다.

3) 메서드 .

하나의 클래스에는 여러 개의 메서드(Method)를 기술할 수 있는데 이는 특정 처리가 반복되거나 다른 처리 내용들과 구분이 필요할 때 선언(정의)해서 사용할 수 있다.

4) main() 함수(메서드)

클래스로 만들어진 **프로그램을 시작하는 기능**을 하는 아주 독특한 함수이다. 구조는 위에서 기술된 것과 같이 반드시

```
public static void main(String[] 매개변수){}
```

와 같이 선언해야 하며 이 때 '매개변수'만이 개발자 임의대로 지정할 수 있다. 만약 매개변수 이외의 어떤 값을 변경하면 프로그램을 시작하게 하는 main함수의 기능이 상실되고 일반 함수(메서드)로 처리된다.

5) 프로그램이 시작될 때 처리할 내용을 기술한다. 좀 복잡한 내용이지만 모든 클래스가 4)항에서 기술된 main 함수가 필요한 것은 아니다. main함수는 main함수가 선언된 클래스를 단독으로 실행할 때만 필요하다. 또는 간단히 테스트할 내용이 있는 경우 main함수에 기술해서 테스트 한다. 따라서 5)항에서 기술될 메인 프로그램은 각종 제어 연산과 관련된 내용 보다는 클래스를 생성하는 역할을 하는 것으로 마무리 될 것이다.

[컴파일 및 실행하기]

step 1.

자바 코드를 작성하고 실행해 보기 위해서 편의상 탐색기를 통해 **testjava** 폴더를 만들고 시작해 보자.

step 2.

메모장을 열어 아래의 코드를 오타 없이 작성한 후 step 1에서 만들어 뒀던 **testjava** 폴더에 **Test.java** 파일명으로 저장한다.

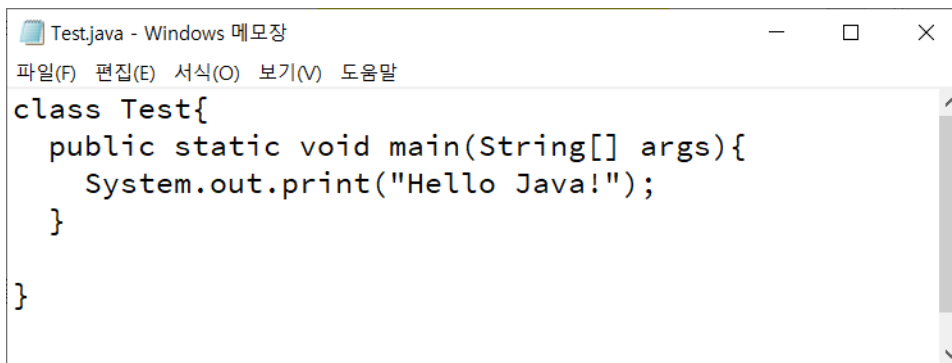


그림 1: c:/testjava/Test.java 코드

step 3.

윈도우의 command 창을 열어 아래와 같은 명령어들을 사용하여 Test.java가 저장되어 있는 폴더로 이동해야 편하다.

```
cd\
cd testjava
dir
```

그러면 아래의 그림처럼 작성된 파일의 목록이 보일 것이다.

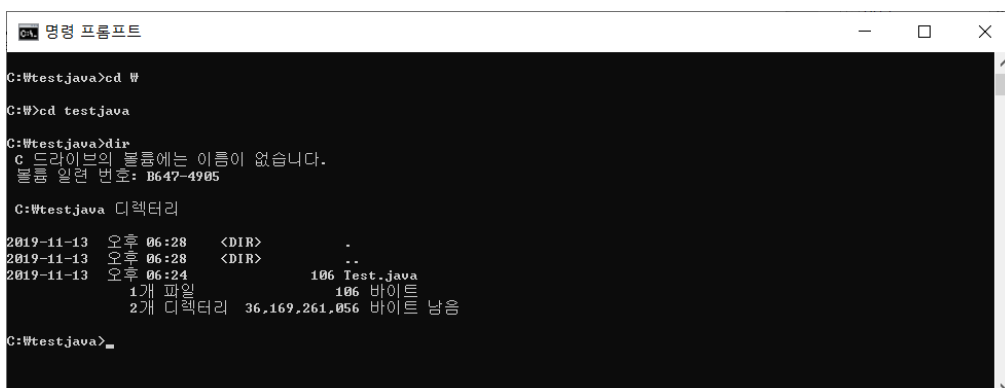


그림 2: 폴더 이동 결과

step 4.

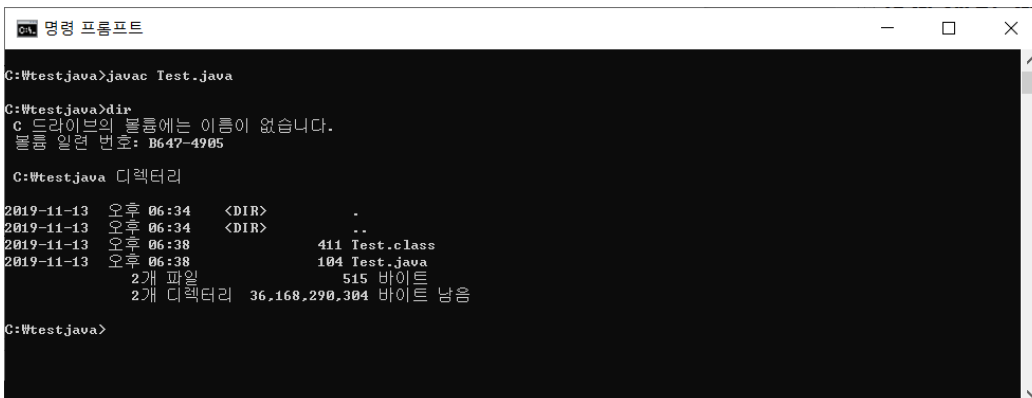
작성된 코드를 컴파일 하고 실행해 보자.

컴파일

```
javac Test.java
dir
```

오탈자가 없다면 아무런 메시지 없이 프롬프트만 나타날 것이다. 만약 오탈자가 있다면 수정 후 다시 컴파일 하면 된다.

오류가 없는 경우 화면



```
C:\Wtest.java>javac Test.java
C:\Wtest.java>dir
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: B647-4905

C:\Wtest.java 디렉터리

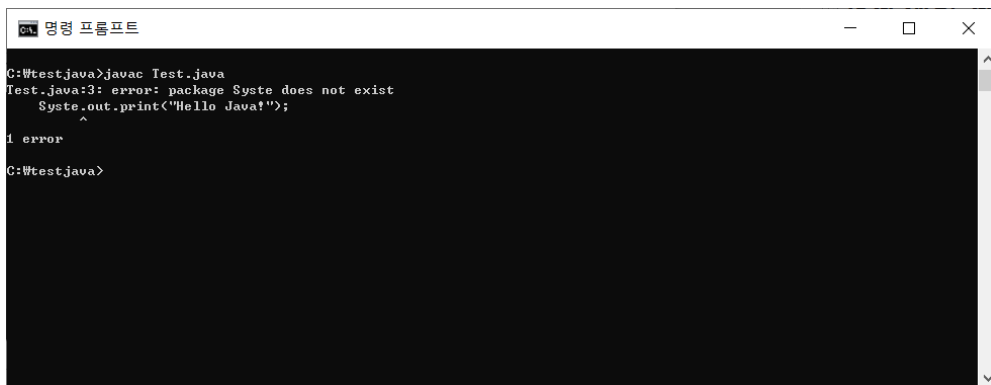
2019-11-13 오후 06:34 <DIR>          .
2019-11-13 오후 06:34 <DIR>          ..
2019-11-13 오후 06:38             411 Test.class
2019-11-13 오후 06:38             104 Test.java
                        2개 파일             515 바이트
                        2개 디렉터리  36,168,290,304 바이트 남음

C:\Wtest.java>
```

그림 3: 오류가 없는 경우

Test.class 란 새로운 파일이 생성되어 있다. 이 파일이 Test.java코드를 컴파일한 자바 실행 파일이다.

오류가 발생한 화면



```
C:\Wtestjava>javac Test.java
Test.java:3: error: package Syste does not exist
    Syste.out.print("Hello Java!");
    ^
1 error
C:\Wtestjava>
```

그림 4: 명령어 오류

탈자 오류



```
C:\Wtestjava>javac Test.java
Test.java:5: error: reached end of file while parsing
    >
    ^
1 error
C:\Wtestjava>
```

그림 5: 불럭 탈자

에러의 종류는 수 도 없이 많으므로 더 이상의 오류 화면은 생략하기로 한다.

실행

java Test

텍스트 편집기등으로 코드를 작성하였다면 해당 파일의 확장자를 'java'로 지정하여 저장한다. 그런 후 컴파일과 실행 과정을 거쳐야 하는데, 만약 작성된 코드가 'c:/testjava/Test.java' 로 저장되었다면

먼저 코드를 javac.exe 파일을 사용하여 컴파일 해야 한다. 먼저 윈도우의 커멘드 창을 열고 c:/testjava로 이동한 후 아래와 같이 컴파일을 진행 한다.

```
javac.exe -d test.java
```

자바에서 컴파일 과정은 작성자가 알아 볼 수 있는 일반 텍스트 파일을 자바가 알아 볼 수 있는 중간코드로 만드는 과정이다.

컴파일이 정상적으로 진행되면 test.class 파일이 동일한 경로에 만들어 진다. 이를 실행하려면 java.exe 파일을 사용하여 실행한다.

```
java test
```

그러면 작성된 test.java 의 실행 결과가 화면에 보일 것이다.

처음 접해보는 유저 입장에서는 대단히 불편하고 복잡해 보이는것이 사실이다. 그러나 일반적으로 텍스트 편집기를 사용하여 개발하는 경우는 거의 없다고 봐도 된다. 이클립스와 같은 개발툴을 사용하게 되면 위와 같은 실행 방법은 거의 대부분 아이콘 하나로 충분하기 때문에 복잡해 할 것이 없을 것이다.

1.5 주석 과 실행문

1.5.1 주석

주석은 실행과는 상관없이 코드의 설명을 나타내는 부분을 의미한다. 컴파일 되지 않으므로 주석이 많다고 해서 컴파일된 파일의 크기가 늘어나거나 하지 않는다. 주석을 처리 하는 방법은 크게 두 가지가 있다.

- // 주석 : //기호 뒤의 내용은 모두 주석 처리 된다. 한줄 주석이라고도 표현한다.
- /* 주석내용 */ : 일반적으로 여러 줄을 동시에 주석 처리 하거나 특정 영역을 주석으로 만들 때 사용한다.

ex 1)

```
// 이곳은 주석입니다.  
// 이곳도 주석입니다.
```

ex 2)

```
/* 이곳도 주석입니다. */  
/*  
이곳도  
주석  
입니다.  
*/
```

주의 사항은 ""안에서는 주석을 사용할 수 없다.

또한 주석은 코드를 설명하는 내용을 담고 있기 때문에 소스 내용이 수정되면 반드시 주석도 수정되어야 한다. 따라서 소스를 수정 하기전에 먼저 수정된 내용에 맞게 주석을 수정한 후 소스를 수정하는 방법을 추천한다.

[주석의 중요성]

단위 프로그램의 크기가 클 수록 나중에 프로그램을 분석하거나 유지 보수가 어려워진다. 이 때 작성되어 있는 주석을 살펴 봄으로써 전체적인 프로그램의 흐름이나 분석을 쉽게 파악할 수 있는 것이다.

또한 주석은 자기 소스의 자신감이라 할 수 있다. 개발자 자신이 작성해 놓은 주석은 후배나 타인이 필요에 의해서 보게 될것이다. 이 때는 이미 주석을 작성한 개발자는 한단계 이상의 것을 보는 위치에 있을 것이기 때문이다.

1.5.2 실행문

주석을 제외한 모든 문장이 실행 문장이 된다. 실행 문장 뒤에는 반드시 ';'로 실행 문장을 구분해야 한다. 한

줄에 여러 개의 실행 문장이 있을 수 있으나 대부분 한 줄에는 하나의 실행 문장 만을 기술한다.

한 줄로 작성했을 때	여러 줄로 분리했을 때
a=10; b=20; c=30;	a=10; b=20; c=30;
a=a+1; b=b+1;	a=a+1; b=b+1;

위의 좌,우 내용은 모두 동일한 내용이다.

1.5.3 실행 영역

자바는 실행할 내용을 기술할 때는 몇 가지 예외를 제외하고는 반드시 실행 영역 안에 기술해야 한다. 실행 영역은 {} 기호로 구분한다. 영역을 구분하는 몇 가지 예를 들어 보자.

- 1) 클래스의 영역을 구분할 때

```
class TestClass { ... }
```

- 2) 메서드 영역을 구분할 때

```
public void prn() { ... }
```

- 3) 특정 명령의 영역을 구분할 때

```
if(a>10){ ... }
```

- 4) 임의의 영역을 구분할 때

```
{ 처리 내용들 }
```

2 변수와 데이터 유형

2.1 변수

2.1.1 변수란?

단어적인 의미는 변하는 수란 의미를 갖고 있다. 즉 고정된 값이 아니라는 의미이다. 따라서 10 은 변수가 되지 못한다. 그 이유는 10이 갖고 있는 절대값은 변하지 않기 때문이다.

2.1.2 프로그램에서 변수란?

변수 자체가 갖고 있는 의미 뿐만 아니라 프로그램에서 변수란 어떤 하나의 값을 저장하고 있는 메모리 위치를 기호화 한 것을 나타낸다.

2.1.3 변수를 정의하는 방법

거창하게 들릴지는 모르지만 실제로 그리 거창한 일은 아니다. 아래와 같은 규칙에 준하여 변수를 정의할 수 있다.

- 영문자, 한글을 사용할 수 있다. 그러나 대부분 한글을 사용하여 변수를 지정하지 않는다.
- 숫자를 사용할 수 있다. 단, 숫자로 시작할 수는 없다.
- 대소문자를 구분한다. 그러나 일반적으로 변수명은 소문자로 시작한다.
- \$, _ 이외의 특수 문자를 사용할 수 없다.
- 공백 문자를 사용할 수 없다.

[변수로 사용할 수 있는 예]

- 영문자로만 이루어진 경우 : a, abc, ABC, Abc, aBC, ...
- 영숫자로 이루어진 경우 : a1, abc123, a123, A123, Abc123, ...

- 특수문자로 시작하는 경우 : \$abc, _abc, ...

[변수로 사용할 수 없는 예]

- 숫자로 시작하는 경우 : 1a, 123abc, ...
- 공백 문자를 사용한 경우 : a b, abc def, adc 123, ...
- 특수 문자를 사용한 경우 : a=b, a/b, a!b, ...

2.1.4 변수 지정 스타일

사용하는 언어에 따라 변수를 지정하는 스타일이 조금씩 다르다. 예를 들어 자바에서는 변수명을 카멜타입(낙타형)이라고 불리는 형식으로 변수를 지정한다. 다른 언어에서는 변수 중간에 '_'를 삽입하여 하나의 변수에 두 가지 이상의 의미가 복합되었을 때 사용한다.

대표적인 예를 들면,

[자바스타일]

- totKor : 국어 합계
- memberScore : 회원의 점수

[타 언어 스타일]

- tot_kor : 국어 합계
- member_score : 회원의 점수

이런 형식들은 호불호가 갈라져 있어 주장하는 바가 서로 다르지만 한 가지 공통점은 하나의 소스에서는 하나의 형식만을 사용하라는 것이다.

2.1.5 변수를 사용하는 목적

굳이 복잡한 형식을 사용하여 변수를 사용하는 이유는 무엇일까??? 한 마디로 말하지만 바로 재 사용성을 늘려주기 위해서이며, 값의 의미를 명확하게 전달 하기 위해서 이다. 즉 생산성이 높아지기 때문이다. 아래의 예를 보자.

변수를 사용하지 않고 3개 과목의 성적을 사용하여 합계와 평균을 계산하여 성적과 함께 총점, 평균값을 출력한다고 가정해 봅시다. (System.out.println() 은 표준 출력 장치에 ()안에 있는 내용을 출력하는 명령)

```
System.out.println(90); // 국어성적
System.out.println(80); // 영어성적
System.out.println(70); // 수학성적
System.out.println((90+80+70)); // 합계
System.out.println((90+80+70)/3.0); // 평균
```

이러한 작업을 통해서 하나의 건수 만을 처리한다면 그리 어렵지도 않은 일이며 변수를 사용하더라도 그리 상상성이 높아 질 거라고 생각이 들지도 않는다. 그러나 두 가지 경우의 수를 생각해 보자.

첫째, 성적 중 하나 이상이 변경되었을 때
국어 성적이 90에서 80으로 수정되었다고 가정하면 위의 프로그램에서는 총점을 구하는 부분과 평균을 구하는 부분 모두를 수정해야 한다. 하나라도 미처 수정하지 못했다면 결과 값은 실리할 수 없게 된다.

둘째, 성적이 여러 개 존재했을 경우
성적의 개수 만큼 출력 문장들이 있어야 한다. 천 명분의 성적이 있다고 생각한다면 상상하기도 싫어진다.

그럼 변수를 사용해서 다시 작성해 보자.

```
kor=90;
eng=80;
mat=70;
tot = kor + eng + mat;
avg = tot/3.0;
System.out.println(kor); // 국어성적
System.out.println(eng); // 영어성적
System.out.println(mat); // 수학성적
System.out.println(tot); // 합계
System.out.println(avg); // 평균
```

위와 같이 내용이 수정된다면 국어 점수 kor 의 값이 어떤 값으로 바뀌든 상관없이 바뀌진 성적 이외의 어떤 내용도 수정할 필요가 없게 된다. 또한 수학 점수 70이 변수 mat에 저장되어 있는 경우 수학 점수임을 유추해서 생각할 수 있겠으나 단지 70이란 값을 보았을 때 70이 어떤 의미인지 알아내기는 소스 내용 전체를 살펴보기 전엔 알기 힘들 것이다.

2.2 리터럴(상수)의 종류와 변수에 저장 하기

2.2.1 리터럴(상수)란?

리터럴(상수)이란 절대값이 바뀌지 않는 값을 의미한다.

2.2.2 리터럴의 종류

- 숫자 리터럴 : 정수와 실수 모두
- 문자 리터럴 : 작은 따옴표 안에 있는 한 글자.
- 문자열 리터럴 : 큰 따옴표 안에 있는 문자.
- 논리형 리터럴 : true, false

2.2.3 변수에 저장하기

리터럴에 리터럴을 저장할 수 없다. 이유는 리터럴을 정의할 때 “절대값이 바뀌지 않는 수”란 의미가 있기 때문이다.

[작성 불가]

- `10 = 200` : 10이란 값이 100으로 바뀔 수 없다.
- `'a' = 100` : 문자 'a'는 a일뿐 100이 될 수 없다.
- `"abc" = "def"` : 문자열 "abc"는 abc 일 뿐이다.
- `true = 123` : true가 갖고 있는 참이라는 의미는 다른 것으로 바꿀 수 없음.

위에서 보는 것과 같이 어떤 값을 변수에 저장하기 위해서는 대입 연산자인 '=' 기호를 사용한다. = 기호 오른쪽의 값을 왼쪽의 변수에 저장하라는 의미이다.

[리터럴 저장 예]

- `a=10` : 상수 10을 변수 a에 저장하라.
- `a='박'` : 문자 '박'을 변수 a에 저장하라.
- `a=3.14` : 실수형 상수 3.14를 변수 a에 저장하라.

- a="박원기" : 문자열 "박원기"를 변수 a에 저장하라.
- a=true : 참을 의미하는 true값을 변수 a에 저장하라
- a=b : 변수 b에 있는 값을 변수 a에 저장하라.

따라서 = 기호 왼쪽엔 반드시 변수가 와야 하며, 오른쪽에는 리터럴이든 변수이든 상관 없다.

2.3 데이터(변수) 유형

위에서 살펴본 것과 같이 변수에는 리터널과 변수값을 저장할 수 있다. 그러나 변수에 어떤 형태의 리터럴이든 저장할 수 있는 것은 아니다. 변수에는 저장할 수 있는 값의 형태가 지정되어 있다. 이를 '데이터형에 종속적이다' 라고 표현한다.

C/C++ 언어의 정수형(int) 타입은 32비트 운영체제에서는 4byte, 64비트 운영체제에서는 8byte로 그 크기가 바뀌지만, 자바의 데이터 유형은 운영체제가 바뀐다고 해서 그 유형이 달라지지 않는다. 즉, int형은 어떤 컴퓨터를 쓰든지 모두 32비트를 갖는다.(단, 운영체제가 아니라 JDK의 종류에 따라 달라진다)

자바는 대소문자를 구별한다. 또한 한글로 변수를 선언해도 상관없지만 실제로는 거의 한글로 변수를 선언해서 쓰는 경우는 없다.

또한 변수의 선언은 {}으로 묶어 있는 어떤 곳에서 선언해도 상관없다. {} 영역 밖에서는 변수를 선언하거나 사용할 수 없다.

2.3.1 변수의 기본형

자바에서는 8가지의 변수의 기본형을 갖고 있다.

데이터(변수) 유 형		예 약 어	범 위
논리형	논리형	boolean	true, false
정수형	문자형	char	16비트 유니코드(0~65535)
	수치형(바이트)	byte	8비트, -128~127
	수치형(16비트 정수)	short	16비트, -32,768~32,767
	수치형(정수)	int	32비트, -2,147,483,648~2,147,483,647
	수치형(64비트 정수)	long	64비트

자바의 개요 2.3 데이터(변수) 유형

실수형	수치형(실수형)	float	32비트
	수치형(64비트 실수형)	double	64비트

이 이외에 자바는 레퍼런스 데이터형을 가지고 있는데 이것은 메모리상의 클래스 객체나 배열의 위치를 가리킨다. 레퍼런스 유형은 배열이나 객체에서 다루도록 하겠다.

문자형 상수는 ' ' 사이에 들어 있는 값이다.

아래와 같은 특수문자도 존재한다.

<code>\n</code>	New Line	<code>\f</code>	Formfeed
<code>\r</code>	Return	<code>\'</code>	Single Quote
<code>\t</code>	Tab	<code>\"</code>	Double Quote
<code>\b</code>	BackSpace	<code>\\</code>	BackSlash

2.3.2 기본형 변수 선언 방법

기본형의 변수는 아래와 같이 선언해서 사용하며, 지정된 변수형 이외의 데이터는 저장 할 수 없다. 하나의 변수에는 하나의 값만이 존재한다.

[변수 선언 방법]

- 변수형 변수명;
- 변수형 변수명 = 초기값;

2.3.3 논리형(boolean)

논리 형태의 값을 저장할 때 사용한다.

[방법]

- boolean 변수명 = 초기값;
- boolean 변수명;

[가능]

- boolean b=true; // 논리형 변수 b를 선언하고 초기값으로 true를 저장

- `boolean a; // 논리형 변수 a 선언`
`a=true; // 논리형 변수 a에 true 저장`

[불가]

- `boolean b='a' ; // 'a'형태가 논리 형태가 아닌 문자형이기 때문`
- `boolean b=123;`
- `boolean b = "park";`
- `boolean a;`
`a=123;`
- `boolean b=true false ; // 두 개의 값을 동시에 저장할 수 없음.`

2.3.4 문자형(char)

숫자, 영문자, 한글, 기타 특수 문자등의 한글자를 저장한다. 또한 정수를 대입할 경우 정수에 해당하는 문자가 대입 된다. 문자형(char)형도 크게 보면 정수형이기 때문에 산술 연산이 가능하다. 그러나 산술연산된 결과 또한 수에 해당하는 문자를 기억하게 된다.

[방법]

- `char 변수명 = 초기값;`
- `char 변수명;`

[가능]

- `char c = '1'; // 문자 1에 해당하는 코드 번호49를 대입`
- `char c = 'A'; // 문자 A에 해당하는 코드 번호 65를 대입`
- `char c = '한'; // 문자 '한'에 해당하는 코드 번호 54620을 대입`

[불가능]

- `char c=""; // 작은 따옴표안에 아무것도 없는 경우 해당 문자가 없으므로 대입 불가.`
- `char c='ab'; // 두 글자는 문자열이기 때문에 대입 불가.`
- `char c=65536 ; // 0~65535까지만 대입 가능.`
- `char c=-10; // 표현 할 수 있는 값의 범위가 아니다.`

2.3.5 byte

1바이트(8bit) 크기의 값을 저장할 수 있는 변수이다. 값의 범위는 -128~ 127까지의 범위의 정수이며 연산 결과가 이를 벗어나면 값이 +,- 를 순환 된다.

[방법]

- `byte b=10;`
- `byte b;`
`b=10;`

[가능]

- `byte b = 0;`
- `byte b = -128;`
- `byte b = 127;`

[불가능]

- `byte b=3.14;` // 실수를 대입할 수 없다.
- `byte b=1000;` // 값의 범위가 아니다.

2.3.6 short

char 형과 동일하게 2byte 크기를 갖고 있지만 값의 범위가 -32,768~ 32,767 까지의 정수 이다. 만약 연산 결과가 허용 값의 범위가 벗어나면 +,- 가 순환 된다.

[방법]

- `short s=100;`
- `short s;`
`s=100;`

[가능]

- `short s=-100;`
- `short s=10000;`
- `short s='1';` // '1'에 해당하는 코드 값 49가 대입 됨.

[불가능]

- `short s="1";` // 다른 타입의 값을 대입할 수 없음.

- `short s=40000;` // 값의 범위가 아님.

2.3.7 int

가장 일반적으로 사용되는 정수 타입이지만 값의 범위가 대략 -21억 ~ 21억 까지 이므로 이 보다 큰 수를 다루는 업무에서는 부적합하다.

[방법]

- `int a=100;`
`int a;`
`a=100;`

[가능]

- `int a=10*10;` // 두 수를 곱한 값이 대입 됨.
- `int a='1';` // 문자 '1'의 코드 값 49가 대입 됨.

[불가능]

- `int a="123";` // 문자열은 대입할 수 없음.
- `int a=22000000000;` // 값의 범위를 벗어남.

2.3.8 long

8byte의 크기로 int 타입보다 더 큰 수를 표현 할 수 있다(대략 구백 이십 이경). 정확한 값의 범위를 알고자 한다면 `Long.MIN_VALUE` 나 `Long.MAX_VALUE` 를 사용하여 값의 범위를 알아 볼 수 있다. long 타입을 정확히 사용하려면 숫자 뒤에 'L' 또는 'l'을 붙여 줘야 한다.

[방법]

- `long a=10L;`
`long a;`
`a=10L;`

[가능]

- `long a='1';` // 문자 '1'의 코드 값 49가 대입 됨.
- `long a=2222222222L;`
- `long a=10;` // int 형의 값은 자동 형 변환(Promotion)되어 저장됨.

[불가능]

- `long a=3.14;` // 실수형의 값은 대입 될 수 없음.
- `long a=2222222222;` // 값의 범위가 벗어난 정수 자체는 표현 불가.

2.3.9 float

4byte크기로 실수를 저장한다. 실수 뒤에 'f' 또는 'F'를 붙여줘야 float타입으로 인식한다. 값의 범위는 $1.4E-45 \sim 3.4028235E38$ 까지 이다. 메모리를 극히 많이 사용하는 구조가 아니거나 유효 자리 수가 더 정밀해야 한다면 double 타입을 권장한다.

[방법]

- `float f = 3.1f`
- `float f = 3.1F`

[가능]

- `float f = 3f;` //자동으로 소숫점이 붙는다.
- `float f = 3;` //자동으로 소숫점이 붙는다.
- `float f = '1';` // 문자 1에 해당하는 49에 소숫점이 붙여 49.0을 대입한다.
- `float f = 'a';` // 문자 a에 해당하는 97에 수숫점이 붙여 97.0을 대입한다.
- `float f = (float)3.14;` // double 타입의 수가 float 타입으로 형 변환되어 대입한다.

[불가능]

- `float f=3.14;` // 자동으로 형 변환되지 않는다.
- `float f = "a";` // 문자열은 숫자로 변환되지 않는다.
- `float f = 3.000000001f;` //유효 자리를 벗어남.

2.3.10 double

8byte의 크기로 값의 범위는 $1.7976931348623157E308 \sim 4.9E-324$ 까지 이다. 일반적으로 표현하는 실수는 double형이라 볼 수 있다. 숫자 끝에 D 또는 d를 붙여 사용하지만 일반적으로 생략한다.

[방법]

- `double d = 0.14D or 0.14d or 0.14`

[가능]

- `double d = 0.14;`

- `double d = 3;` // 자동으로 형 변환
- `double d = 3f;` // 자동으로 형 변환
- `double d = 'A';` // 문자 A에 해당하는 65를 double 타입으로 변경하여 65.0을 대입
- `double d = (double) 3;` // 정수 3을 형 변환하여 3.0 대입
- `double d = 3.14f * 10;` // 연산 결과를 double형을 변환하여 대입.

[불가능]

- `double d = "a";` // 문자열은 숫자로 형 변환 되지 않는다.
- `double d = "10";` // 따옴표로 둘러 쌓여 있으면 값이 숫자형이라도 문자열이 된다.

2.3.11 기타(BigInteger, BigDecimal)

Long 타입보다 큰수라면 BigInteger, double 타입보다 크고 정밀한 수를 처리할 때는 BigDecimal 을 사용한다. math 패키지에 있는 클래스이며 new 연산자를 사용하여 객체를 생성 사용한다.

```
BigInteger v1 = new BigInteger("123456789");
BigInteger v2 = new BigInteger("123456789");
BigInteger v3 = v1.multiply(v2);
```

[주요메서드]

- `v1.add(v2)`
- `v1.subtract(v2)`
- `v1.multiply(v2)`
- `v1.divide(v2)`

2.4 고정 문자열과 가변 문자열

2.4.1 String

자바에서 문자열을 처리하기 위해 사용되는 클래스가 String 타입이다. String 타입은 사용자의 편의성을 위해 new 연산자를 사용하여 문자열을 생성하지 않고 기본형 처럼 대입 연산자를 사용해서 변수에 대입할 수도 있다.

```
String name = "홍길동";
```

```
String name = new String("홍길동");
```

위와 같은 코드는 표면적으로는 동일한 문자열 "홍길동"을 갖고 있으나 자바 내부에서는 서로 다른 처리 방법으로 문자열을 name변수에 대입한다.

[내용 중략]

주요 메서드

메서드명	설명
char charAt(int index)	index에 있는 문자 반환
boolean equals(Object str)	str과 같은 문자열인가 판단
byte[] getBytes() byte[] getBytes(Charset set)	문자열을 byte 배열로 반환.Charset이 있으면 문자열을 해당 문자셋으로 인코딩
int indexOf(String str [, int fromIndex])	str문자열이 시작되는 위치를 반환, fromIndex가 있으면 fromIndex 이후의 위치.
int length()	문자열의 길이를 반환.
String replace(CharSequence target, CharSequence replacement)	target으로 지정된 문자열을 replacement 문자열로 변경
String substring(int beginIndex) String substring(int beginIndex, int endIndex)	beginIndex <= 반환문자열 < endIndex. endIndex가 생략되면 beginIndex 부터 문자열 끝까지를 잘라 반환.
String toLowerCase() String toUpperCase()	문자열을 모두 소문자 또는 대문자로 변환
String trim()	문자열 양옆에 있는 공백을 삭제하여 반환
String valueOf(int I) String valueOf(double d)	파라미터의 값을 모두 문자열로 변환

2.4.2 charAt

주민번호를 사용하여 성별을 구분하여 출력해 보자. 단, 주민번호 뒤자리의 첫글자가 홀수이면 남자이고 짝수이면 여자이다.

```
String jumin = "901213-3234567";
char c = jumin.charAt(7);
int r = Character.getNumericValue(c);
String str = null;
if( r%2==0 ) str = "여자";
else str = "남자";
System.out.println(str); // 남자
```

2.4.3 equals

두 회원의 이름이 동일한지 판별해 보자. 문자열 비교는 '=='으로 하지 않고 반드시 equals()를 사용해야 한다.

```
String name1 = "홍길동";
String name2 = "홍길동";
String name3 = new String("홍길동");

System.out.println( name1 == name2 ); // true
System.out.println( name1 == name3 ); // false

System.out.println(name1.equals(name2)); // true
System.out.println(name1.equals(name3)); // true
```

위의 예와 같이 동일한 문자열이라도 어떤 형식으로 만들어졌는지에 따라 '=='의 연산 결과는 달라지지만 equals()의 처리 결과는 항상 같다.

2.4.4 getBytes

서로 다른 이 기종간의 통신이나 서로 다른 언어로 만들어진 프로그램간의 데이터 통신이 필요할 때, 또는 문자열을 다른 방법으로 인코딩할 때 유용하게 사용될 수 있다. 그러나 상대방의 프로그램도 동일한 방법으로

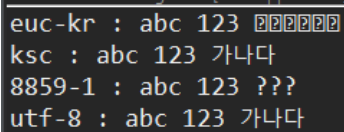
인코딩해야 한다. 또한 예외처리를 반드시 해야 한다.

임의의 문자열을 euc-kr 과 8859-1, ksc5601 인코딩 방법으로 처리하여 표시해 보자.

```
String str = "abc 123 가나다";
byte[] euc = str.getBytes("euc-kr");
byte[] iso = str.getBytes("iso8859-1");
byte[] utf = str.getBytes("utf-8");

System.out.println("euc-kr : " + new String(euc));
System.out.println("ksc : " + new String(euc, "ksc5601")); //ok
System.out.println("8859-1 : " + new String(iso));
System.out.println("utf-8 : " + new String(utf)); //ok
```

[실행결과]



```
euc-kr : abc 123 [garbled]
ksc : abc 123 가나다
8859-1 : abc 123 ???
utf-8 : abc 123 가나다
```

2.4.5 indexOf() | length()

하나의 문자열에서 문자열의 길이와 지정된 문자열의 위치를 표시해 보자.

```
String str1 = "abcdefabc";
String str2 = "abc가나다abc";

int len1 = str1.length();
int len2 = str2.length();

int index1 = str1.indexOf("c");
int index2 = str2.indexOf("나");
int index3 = str2.indexOf("ab", 3);
```

```
System.out.println("length 1 : " + len1);  
System.out.println("length 2 : " + len2);  
System.out.println("index 1 : " + index1);  
System.out.println("index 2 : " + index2);  
System.out.println("index 3 : " + index3);
```

[실행결과]

```
length 1 : 9  
length 2 : 9  
index 1 : 2  
index 2 : 4  
index 3 : 6
```

3 연산자

자바에서 정수형끼리의 연산 결과는 **무조건 정수형**임을 기억하자.

산술 연산시 데이터 형이 큰 쪽으로 결과가 자동 프로모션된다. 예를 들어 실수형과 정수형을 연산하면 자동으로 실수형 결과를 갖는다.

3.1 수치 연산자

수치 연산자에서는 나눈 나머지를 구하는 “%” 만 새로이 알면 될것이다. %(나머지 연산자)는 몫이 정수일 때 까지만 계산한뒤 나머지를 반환한다.

```
System.out.println(10%3); → 1
```

3.2 대입 연산자

연산자	설 명
+=	덧셈을 한 뒤, 대입한다.
-=	뺄셈을 한 뒤, 대입한다.
*=	곱셈을 한 뒤, 대입한다.
/=	나눗셈을 한 뒤, 대입한다.
%=	나머지를 구한 뒤, 대입한다.

3.3 증/감 연산자

자바의 개요 3.3 증/감 연산자

++(1씩 증가), --(1씩 감소) 의 두가지 연산자가 있으며, 이것은 변수의 앞이나 뒤에 붙여서 사용된다. 앞에 붙여지면 값이 대입되기 전에 계산되고 뒤에 붙여 지면 대입된후에 계산된다. 단항으로 연산되었을 때는 증감연산자를 변수 앞에 붙이든, 뒤에 붙이든 결과에 영향이 없지만 다항 연산시에는 연산 결과가 달라진다.

앞에 붙인 경우(전치)	뒤에 붙인 경우(후치)
<pre>int a=10; int b=0; int c = ++a; System.out.println(a); → 11 System.out.println(c); → 11</pre>	<pre>int a=10; int b=0; int c = a++; System.out.println(a); → 11 System.out.println(c); → 10</pre>

또한 후치인 경우 () 사용 여부와 관계없이 계산된 후 무조건 나중에 값이 바뀐다.

```
a=10;
b = 10 + a++ // b=20, a= 11
```

```
a=10;
b = 10 + (a++) // b=20, a=11
```

은 동일한 결과를 낳는다.

3.4 비교 연산자

두 조건을 비교하는 연산자들이며 우리가 알고 있는 통상적인 것과 같다. 단, 두 조건이 같을 경우 "="을 쓰지 않고 "=="를 쓴다는것만 다르다. 또한 두 조건이 같지 않을 경우 "!="을 쓴다.

instanceof 란 비교 연산자도 있는데 이것은 두 클래스의 인스턴스가 같을 경우 참값을 반환한다.

```
class A{ ... }
class P extends A{ ... }

...

A a = new A();
P p = new P();
```

```
System.out.println(a instanceof A); → true
System.out.println(p instanceof A); → true
```

3.5 비트 연산자

- 연산 생략기능이 없다.
- 두개의 값을 비트 단위로 연산한다.
- 종류
 - `&` : 논리곱.
 - `|` : 논리합.
 - `^` : 배타적 논리합. 두개의 값이 서로 다를 때 참

p	q	p & q	p q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

연산 생략기능이란 ?

논리곱인 경우 좌측의 값이 거짓 일때 우측의 값이 어떤 값이든 결과가 거짓이 되기 때문에 우측의 연산을 하지 않는 것을 의미함.

다른예로, 논리합인 경우는 좌측이 값이 참이면 우측의 값이 어떤 상태가 되든 결과가 참이기 때문에 우측의 연산을 하지 않는 기능을 의미함.

3.6 비트 이동 연산자

말 그대로 비트를 좌나 우로 이동시키는 명령어이다.

연산자	설 명
<<	왼쪽으로 정해진 숫자 만큼 이동. 부호 비트는 이동되지 않음
>>	오른쪽으로 정해진 숫자 만큼 이동. 부호 비트는 이동되지 않음
>>>	오른쪽으로 정해진 숫자 만큼 이동. 부호 비트는 이동됨

3.7 논리 연산자

논리형 데이터의 논리값을 논리 연산한다. 이 조건 연산자는 가장 일반적으로 사용되어지는 연산자로 조건문에 가장 많이 쓰인다. 연산 생략 기능이 있다.

연산자	설 명
&&	And 연산. 두 조건이 모두 참이어야 참을 반환한다.
	Or 연산. 두 조건중 하나라도 참이면 참을 반환한다.
!	논리 부정. 참을 거짓으로 거짓을 참으로 바꾼다.

3.8 삼항 연산자

삼항 연산자를 조건 연산자라고도 하며, 조건문을 대신해서 쓰기도 한다.

```
변수=(조건) ? 값1 : 값2;
```

```
예) y=(x>10)? 1:2;
```

x가 10 보다 크면 y에 1이대입되고 그렇지 않으면 2가 대입된다.

3.9 나열형 연산자

','를 찍어 같은 형태의 변수형을 선언할 때 주로 사용된다.

나열형 연산자 사용 전	나열형 연산자 사용 후
<pre>int a=10; int b=20; int c=30;</pre>	<pre>int a=10, b=20, c=30</pre>

4 제어문

제어문이란 프로그램의 흐름을 바꾸거나 특정 영역을 반복 수행하고자 할 때 사용되는 명령어들이다.

4.1 if 문

1) if

```
if (조건) {  
    조건이 참인 경우  
}
```

2) if~else 문

```
if (조건) {  
    조건이 참인 경우  
}else {  
    조건이 거짓인 경우  
}
```

3) 다중 IF문

```
if (조건 1) {  
    조건 1이 참인 경우  
}else if (조건 2){  
    조건 2가 참인 경우  
}else if (조건 3){  
    조건 3이 참인 경우  
}  
...  
else
```



```
{  
    모든 조건이 거짓인 경우  
}
```

[기본예 1] 평균이 `double avg=90` 이 저장되어 있다. if문을 사용하여 평균이 60 이상이면 "합격", 60 미만이면 "불합격"을 출력하도록 프로그램 하시오.

[기본예 2] 두 개의 점수를 더한 평균을 계산하고 평균이 60 이상이면 "합격", 60 미만이면 "불합격"을 출력하도록 프로그램 하시오.

4.2 switch 문

- break 문장이 생략되면 아래에 있는 case문과 OR 조건으로 자동으로 묶여 아래에 있는 case문이 실행된다.
- case 문에 사용되는 값1...값n 들은 범위를 지정할 수 없다.
- default 문장은 case문에 해당되는 내용이 없는 경우 실행된다.

```
switch (변수){  
    case 상수1 : { 실행문1; break; }  
    case 상수2 : { 실행문2; break; }  
    ...  
    default : { 실행문1; break; }  
}
```

4.3 while 문

반복횟수가 일정하지 않거나 예측하기 힘들때 사용되는 반복문이다. while문장의 조건에 따라 해당 블록이 한번도 실행되지 않을 수 있다.

while 내부에서 실행 조건을 변경해 주지 않으면 해당 블록이 무한 반복될 수 있다.

```
while (조건) {  
    실행문;  
}
```

4.4 do~while 문

while문과 유사하나 어떤 경우든 do 블록안에 잇는 실행문이 한번은 실행된다. 즉, do안에 잇는 문장을 실행하고 다시 실행할지를 뒤에 기술된 while문에 의해 판단된다.

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

4.5 for 문

반복문들 중에 가장 많이 사용되는 문장이라 볼 수 있다. 반복 횟수가 일정하거나 정해진 값이 있는 경우 주로 사용된다.

```
for(변수=초기값 ; 조건 ; 증가값) {  
    실행문 ;  
}
```

위의 기본 구조를 아래와 같이 바꾸어 for문의 실행 패턴을 설명해 본다.

```
for( A ; B ; C ){  
    T ;  
}
```

1. A : for문이 시작되면 A가 가장 먼저 실행된다. A는 for문이 종료될 때까지 단 1회만 실행된다.
2. B : A가 실행되고 바로 B가 실행되는데 이는 for문을 실행할지를 판단한다. 만약 B가 참이라면 바로 T를 실행한다.
3. C : T를 실행하고 나서 for문은 C로 넘어 오는데 A에 지정된 제어변수의 값을 바꾸는 영역이다. C에서 제어변수의 값을 바꾼 후 for문을 바로 다시 시작하는 것이 아니라 B로 이동하여 다시 실행할지를 판단하고 B가 참이면 다시 T를 실행한다.
4. B --> T --> C --> B --> T 가 계속 반복된다.

4.6 break 문

각종 반복문에서 반복문을 강제로 종료시키고 해당 블록을 빠져 나오도록 한다.

4.7 continue 문

블록의 처음 실행문으로 이동한다.

4.8 label 사용

반복문을 중첩해서 사용할 때 label을 지정하여 특정 반복문을 break 시키거나 continue를 할 수 있지만 특별한 경우를 제외하고는 **사용을 지양할 필요가 있다**.

```
outer : for ( A ){  
    inner : for( B ){  
        ...  
        break outer;  
    }  
}
```

for문 B안에서 break만을 사용하면 B 부분이 종료되고 for문 A가 다음 회전으로 반복되지만 break outer를 사용하면 for문 A 가 완전 종료된다.

5 배열

거의 모든 언어에서 기본적으로 제공하고 있는 데이터 구조이다. 사용 방법도 거의 유사하기 때문에 굉장히 중요한 자료 구조이다. 배열은 아래와 같은 특징을 갖고 있다.

1. 배열의 개수는 고정적이다.
2. 배열의 요소는 배열명[첨자]와 같은 방법으로 접근한다.
3. 배열의 첨자는 0부터 시작된다.
4. 첨자는 반드시 0을 포함한 양의 정수 이여야 한다.
5. 배열의 길이(갯수)알아내는 속성 : 배열명.length 이다.

5.1 배열 생성

배열이란 같은 종류의 데이터 n 개가 하나의 이름으로 표시되는 데이터의 묶음이라 말할 수 있다. 자바에서는 배열의 사용 방법이 아래와 같은 순서로 정의되고 사용된다.

배열의 선언	메모리 할당	사용
데이터형 변수명[]; or 데이터형[] 변수명;	변수=new 데이터형[실제크기]	변수[첨자]

5.2 배열을 선언하는 여러 가지 방법들

<pre>int[] score = { 10, 20, 30, 40, 50} int[] score = new int[]{ 10, 20, 30, 40, 50 }</pre>	배열이 생성되면서 초기값을 갖게 되며, 배열의 크기는 인수들의 갯수만큼 지정된다.
<pre>int[] score = new int[5] or int[] score; score = new int[5];</pre>	배열의 크기만 지정되고 초기값은 모두 0을 갖는다.

Object[] obj = new Object[5]	Object 타입의 배열이 5개 선언되고 각각의 요소는 null 값이 된다.
int[] score; score = new int[10, 20, 30, 40, 50]	배열 변수만 미리 선언한뒤 나중에 배열에 초기값을 넣는다.

5.3 배열 접근

선언된 배열의 요소는 첨자(INDEX)를 사용하여 접근한다. 첨자는 항상 0보다 크거나 같은 양의 정수이어야 한다. 배열을 선언하고 첨자를 사용하여 값을 대입해 보자.

```
int[] score = new int[5];
score[0] = 100; // 배열 첫번째 요소에 100이 저장된다.
```

배열에 저장된 값을 첨자를 사용하여 가져와 본다.

```
int[] score = new int[]{10, 20, 30, 40, 50 };
int x = score[0]; // x에 10이 대입됨
```

예) 배열에 이름, 주소, 연락처를 저장하고 출력하자

```
1. class ArrayTest{
2.
3.     public static void main(String arg[]){
4.
5.         String str[];
6.         str = new String[3];
7.
8.         str[0]="박원기";
9.         str[1]="대한민국";
10.        str[2]="010-1111-1111";
11.
12.        for(int i=0 ; i<str.length ; ++i){
```

```

13.      System.out.println(str[i]);
14.    }
15.  };
16. };

```

5.4 다차원 배열

배열은 필요에 따라 1차원, 2차원, ...n차원까지 선언하여 데이터를 저장할 수 있다.

int[] score;	정수형 1차원 배열. 행만 존재함.
int[][] score;	정수형 2차원 배열, 행과 열이 존재함.
int[][][] score;	정수형 3차원 배열. 면-행-열 이 존재함.

[] 의 개수에 따라 차원이 나뉘어지며, 여러 가지 요인에 의해 1차원 또는 2차원 배열이 많이 사용된다.

10
20
30
40
50

```

// 정수형 1차원 배열
int[] score = { 10, 20, 30, 40, 50 }
x = score[0] ; // 10
x = score[4] ; // 50

```

10	60
20	70
30	80
40	90
50	100

```

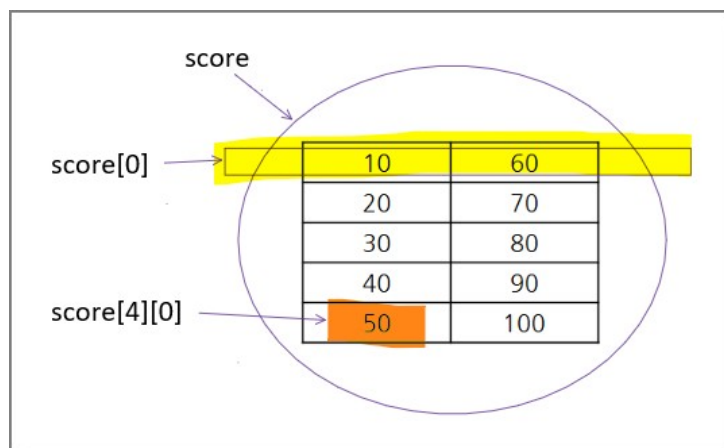
// 정수형 2차원 배열
int[][] score = {
    { 10, 60 },
    { 20, 70 },
    { 30, 80 },
    { 40, 90 },
    { 50, 100 }
}
x = score[0][0]; // 10

```

```
x = score[0][1]; // 60
x = score[1][0]; // 20
```

5.5 배열 대표명

위에서 언급한 것 처럼 배열의 요소를 접근할 때는 첨자를 사용하여 접근한다. 그러나 배열명만을 사용하면 배열 전체를 의미한다. 또한 2차원 이상의 배열인 경우 배열의 첨자를 함만 지정한다면 행 전체를 나타낸다.



- score : 5행 2열의 배열 전체를 의미한다.
- score[0] : 배열의 첫번째 행을 의미한다.
- score[4][0] : 5행 1열의 값 50만을 의미한다.

배열 대표명을 사용한 길이 구하기

- score.length ==> 5행
- score[0].length ==> 2열(10, 60)을 의미함.
- 특정 열만은 지정할 수 없다.

6 클래스와 객체

객체 지향언어인 자바에서 클래스를 이해한다는 것은 자바 언어를 이해한다는 것과 같은 정도로 자바의 모든 구조는 클래스 안에서 이루어진다. 본인의 이름 하나를 출력하더라도 클래스를 선언해야 한다. 아마도 자바를 입문하는 입장에서 본다면 다른 언어들에 비해서 복잡하고 어렵게 느껴지게하는 요소가 될지도 모르겠다. 그러나 자바 언어는 클래스를 선언하는 부분에 일관성이 있어 어떤 프로그램을 만들든지간에 상관없이 클래스를 선언해야 한다. 특히 입문자들은 클래스를 선언하는 것을 프로그램의 영역을 만든다고 생각하면 편할 것이다.

또한 클래스명과 파일명은 일치해야 여러 가지 번거로움을 피할 수 있다. 하나의 파일에 두 개 이상의 클래스를 선언할 수는 있지만 현장에서 거의 사용되지 않는다.

6.1 class 구성 요소

class의 구성 요소를 크게 나누면 필드와 메서드로 이루어져있다. 이제껏 배워왔던 제어문이나 연산식들은 모두 메서드안에서 작성해야 한다. 즉 아래와 같은 코드는 존재할 수 없다.

```
// 오류 코드
class A{
    for(int i=1 ; i<=10 ; i++){
        System.out.println(i);
    }
}
```

즉, 반복문 for는 class의 구성 요소가 아니기 때문이다. 따라서 for문은 하나의 메서드안에서 작성해야 한다. 그렇기 때문에 이제껏 우리는 public static void main(String[] args)라는 메서드안에서 테스트 코드들을 작성한 것이다.

- 필드 : 클래스가 갖고 있어야 하는 값을 저장하는 변수. 멤버 변수라고도 하지만 자바 진영에서는 필드라 부른다.
- 메서드 : 작업을 실행하기 위해서 작성되는 구조이다. 멤버함수, 함수등의 의미와 같지만 이 역시 자바 진영에서는 메서드라 부른다.

6.2 class 정의

- 클래스명은 일반적으로 대문자로 시작하고, 변수명의 만드는 규칙과 일반적으로 같다.
- 접근권한을 지정한다.
- 선언된 class가 객체가 될 때 자동으로 실행되는 생성자 메서드를 작성할 수 있다.

6.2.1 접근 권한자의 종류

접근 권한자	설 명
public	외부에 있는 어떤 클래스든 상관없이 접근하여 사용할 수 있게 한다.
생략	접근 제한자를 생략하는 경우이며, 같은 패키지내에 있는 클래스들만 접근을 허용한다. default나 package라는 의미로 사용함.

6.2.2 클래스 수정자의 종류

접근 제한자와 조합하여 사용될 수 있다.

클래스 수정자	설 명
abstract	추상 클래스로 만들 때 선언한다.
final	상속할 수 없는 클래스를 만들 때 사용한다.

예) 아무나 접근할 수 있는 클래스 Member 선언

```
public class Member{ ... }
```

예) 아무나 접근할 수 있는 추상 클래스 Member 선언

```
public abstract class Member{ ... }
```

예) 같은 패키지내에서만 접근할 수 있는 클래스 Member 선언

```
class Member{ ... }
```

예) 아무나 접근할 수 있지만 상속할 수 없는 클래스 Member 선언

```
public final class Member{ ... }
```

6.3 필드 선언

클래스가 갖고 있어야 하는 값들이 있다면 필드를 선언하여 저장한다. 필드를 멤버변수라고도 하며 클래스 전체 영역에서 사용할 수 있기 때문에 전역형 변수라고도 한다. 클래스 내부에 어떤 곳에서든지 선언할 수 있지만 클래스 상단에 기술하는 것을 원칙으로 한다.

6.3.1 필드의 구조

```
[ 접근제한자 ] [ 수정자 종류 ] 데이터 유형 필드명 [ = 초기값 ];
```

- 데이터 유형은 변수의 데이터 유형과 동일하다.
- 초기값을 지정하지 않으면 정수형은 0, 실수형은 0.0, 객체형은 null 값이 자동으로 대입된다.
- 접근 제한자, 수정자의 기술 순서는 없다.

6.3.2 접근 제한자

자바의 개요 6.3 필드 선언

클래스와 기능이 동일하며 `protected`와 `private` 형태가 추가된다.

접근 권한자	설 명
<code>public</code>	외부에 있는 어떤 클래스든 상관없이 접근하여 사용할 수 있게 한다.
<code>protected</code>	같은 패키지내에 있거나 상속관계에 있는 클래스들만 접근을 허용한다.
생략	접근 제한자를 생략하는 경우이며, 같은 패키지내에 있는 클래스들만 접근을 허용한다. <code>default</code> 나 <code>package</code> 라는 의미로 사용함.
<code>private</code>	클래스 외부에서는 접근할 수 없다.

6.3.3 수정자의 종류

일반적인 의미는 클래스의 수정자 종류와 동일하지만 `final`의 의미는 클래스에서 사용했을 때와 다르다.

수정자	설 명
<code>final</code>	클래스에 붙어 있는 경우는 클래스가 상속되지 않도록 하는 기능이었으나 필드에 붙어 있으면 값을 변경할 수 없게 한다.
<code>static</code>	<ul style="list-style-type: none">- 정적 필드라 부르며, 이는 클래스가 객체가 되지 않더라도 클래스명으로 접근할 수 있게 한다. 물론 객체명을 통해서도 접근할 수 있다.- 클래스 공통변수라고도 하며 첫번째로 객체가 생성되었을 때만 초기화되고 두번째 부터는 객체가 생성되더라도 변경된 값이 초기화 되지 않는다.

예) 외부에서 접근할 수 없는 정수형 필드 `score` 선언

```
private int score;
```

예) 아무나 쓸 수 있는 문자형 변수 `grade` 선언

```
public char grade;
```

예) 같은 패키지나 상속관계에 있는 클래스에서만 접근할 수 있는 실수형 변수 score 선언

```
protected double score;
```

예) 아무나 클래스명으로 접근할 수 있는 문자열 필드 address 선언

```
public static String address;
```

예) 아무나 접근 가능하고 지정된 값을 바꿀 수 없는 실수형 변수 PI를 선언하고 초기값 3.14 대입

```
public final double PI=3.14;
```

예) 클래스 공통변수를 아무나 접근할 수 있도록 정수형 변수 hp 선언

```
public static int hp;
```

6.4 메서드의 정의

클래스가 단순히 어떤 값을 저장할 것이 아니라 어떤 기능을 하게 하려면 메서드를 정의해야 한다. 메서드 없이는 어떤 기능을 수행할 수 없다. 메서드는 일반 언어의 함수와 그 역할과 선언 방법이 유사하다. 메서드의 종류로는 생성자와 일반 메서드가 있다.

6.4.1 메서드의 종류

메서드	설 명	공통 사항
생성자	<ul style="list-style-type: none">- 클래스가 객체가 되어 메모리에 로드될 때 클래스의 초기화를 담당하는 메서드를 생성자라 부른다. 만약 생성자를 기술하지 않으면 자바 가상머신이 매개변수가 없고 아무런 부가 기능도 없는 생성자를 실행시켜준다.- 반드시 클래스명과 동일해야 한다.- 반환형을 사용할 수 없다.	<ul style="list-style-type: none">- 외부에서 값을 전달 받기 위해 매개변수를 추가할 수 있다.

- 반환형을 사용할 수 없기 때문에 반환값이 없다..

일반 메서드

- 클래스의 기능 구현을 하기 위해 작성된다.
- 반드시 반환형을 기술해야 한다.
- 처리된 결과를 메서드를 호출한 곳으로 전달 할 수 있다.

6.4.2 메서드 구조

```
접근 권한 수정자 반환형 메서드명( [ 매개 변수 ] ) {
    ...
    [ return 리턴값; ]
}
```

- 메서드를 정의할 때 반환형이 void가 아니라면 반드시 return 문을 작성해야 한다.
- 반환형은 자바가 사용할 수 있는 데이터 유형과 클래스 모두 사용할 수 있다.
- 매개 변수는 메서드가 실행될 때 필요한 정보를 외부에서 전달 할 때 사용되며 반환형과 같이 자바 변수형과 클래스들을 필요한 만큼 '변수유형 변수명'을 한 쌍으로 ','로 나열한다.

접근 권한 부분은 필드에 적용할 수 있는 접근 권한과 동일함으로 생략하기로 한다.

6.4.3 수정자의 종류

수정자	설 명
final	클래스가 상속되었을 때 final이 붙은 메서드는 자손 클래스에서 메서드의 내용을 바꿀 수 없게 한다.
static	클래스를 객체로 만들지 않고 클래스명을 사용하여 접근할 수 있는 정적 메서드를 만든다.
abstract	추상 메서드를 만든다. 추상 메서드는 메서드의 body부분을 작성하지 않고 상속받은 자손 클래스에서 해당 메서드를 정의해야 한다.
synchronized	동기화 메서드를 만든다. 동기화 메서드란 동시에 2개 이상의 프로세스가 메서드 내부의 처리 과정을 실행할 수 없게 하는 기능이다. 이를 상호 배타적 동기화라 부른다.

예) 접근권한은 아무나, 반환값은 없고 매개변수도 없는 메서드

```
public void prn() { ... }
```

예) 접근 권한은 아무나, 반환값은 정수형인 메서드

```
public int sum(){
    int x = 0;
    ...
    return x;
}
```

예) 접근권한은 아무나, 반환값은 없고 매개변수가 문자열형 하나를 갖는 메서드

```
public void prn(String str){ ... }
```

예) 접근 권한은 같은 패키지안에서만, 문자열을 반환값으로 갖고 매개변수는 정수형 2개인 메서드

```
void String prn(int x, int y){
    String r = "";
    ...
    return r;
}
```

6.5 접근제한자 수정자 종류별 기술 가능 위치

종 류		클래스	필드	메서드
접근 제한자	public	○	○	○
	protected		○	○

자바의 개요 6.5 접근제한자 수정자 종류별 기술 가능 위치

	default	○	○	○
	private		○	○
수정자	final	○	○	○
	static		○	○
	abstract	○		○
	synchronized			○

* 이 이외에도 transient, volatile 이 있지만 다루지 않는다.

6.5.1 final 수정자 사용 위치에 따른 의미변화 정리

사용위치	의미
클래스	상속 안됨.
메서드	재 작성 안됨
필드	값이 변경되지 않음.

6.5.2 static 수정자 사용 위치에 따른 의미변화 정리

사용위치	의미
메서드	정적 메서드. 클래스를 객체로 만들지 않고도 접근할 수 있음.
필드	클래스 공통변수. 초기화는 한번만.

6.6 class 사용하기

class는 class만으로서로는 사용할 수 없다. 사용하기 위해서는 반드시 메모리에 올려야 하는데 이 올려진 class를 클래스 객체 또는 클래스 인스턴스(Class Instance)라고 한다. class는 new 연산자를 사용하여 메모리에 올리고 추후 해당 객체를 접근하기 위해 객체명에 대입하면 된다.

클래스를 사용하여 객체를 만드는 방법들은 아래와 같다.

1. new 클래스명([매개변수])
2. 클래스명 객체명 = new 클래스명([매개변수])
3. 부모 클래스명 객체명 = new 클래스명([매개변수])
4. 부모 인터페이스명 객체명 = new 클래스명([매개변수])

첫번째 방법은 익명형(Anonymous)으로 객체를 생성한 것이며, 이는 객체명을 지정하지 않았기 때문에 추후 다시 접근할 수 없는 일회성 객체를 생성한 것이다.

두번째 방법은 객체명을 지정하였기 때문에 한번 객체를 생성한 뒤 추후 필요할 때 생성된 객체를 객체명을 통해 접근하여 재 사용할 수 있다.

세번째 방법은 아직 상속의 개념을 배우지 않아 간단히 설명을 하기는 어렵지만, 부모 개체형의 변수명에 자손 객체를 생성하여 대입할 수 있다. 이런 형태를 다형성이라 부르기도 한다.

네번째 방법 또한 인터페이스를 먼저 설명해야 하지만, 세번째 방법과 같은 이유로 가능하다. 이 또한 객체의 다형성이라 부른다.

6.6.1 객체 생성 예

예) Student 클래스가 아래와 같이 정의되었을 때 Student 클래스를 사용하여 객체를 만들고 멤버들을 접근하는 예

[Student 클래스 정의]

```
public class Student {  
    String irum;  
    int score;  
}
```

- Student 클래스는 아무나 접근할 수 있는 클래스이다.
- 문자열형으로 irum, 정수형으로 socre 필드가 선언되어 있다.
- 선언된 필드들은 같은 패키지내에서만 접근이 가능하다.

[StudentTest 클래스 정의]

```
public class StudentTest {  
    public static void main(String[] args) {  
        Student s = new Student();  
        s.irum = "hong";  
        s.score = 100;  
        System.out.println("이름 : " + s.irum);  
        System.out.println("성적 : " + s.score);  
    }  
}
```

- StudentTest 클래스는 아무나 접근할 수 있는 클래스이다.
- 클래스를 컴파일 하고 java StudentTest와 같이 실행하면 main 함수가 자동으로 실행된다.
- main 함수 내부
 - Student클래스 타입의 변수(객체) s를 선언하고 Student 클래스를 생성하여 대입한다.
 - 생성된 s의 필드인 irum과 score에 값을 대입한다.
 - 출력문장을 사용하여 s의 필드값들을 표준 출력장치에 출력한다.

[실행결과]

```
이름 : hong  
성적 : 100
```

예) 필드와 메서드가 함께 있는 클래스 Member를 사용하는 예

[Member 클래스 선언]

```
public class Member {
```

```
public String irum;
public String phone;

public void set(String irum, String phone) {
    this.irum = irum;
    this.phone = phone;
}

public void prn() {
    System.out.println("성명 : " + this.irum);
    System.out.println("연락처 : " + this.phone);
}
}
```

- Member 클래스는 아무나 접근할 수 있는 클래스이다.
- 문자열형으로 irum과 phone 두 개의 필드가 선언되어 있다. 이 두 개의 필드는 아무나 접근할 수 있다.
- set 메서드
 - 아무나 접근할 수 있으며 반환값은 없다.
 - 문자열 타입으로 두 개의 매개변수가 정의되어 있다.
 - 매개변수로 전달 받은 값들을 필드에 저장한다.
- prn 메서드
 - 아무나 접근할 수 있으며 반환값도 없고 외부에서 값을 전달 받을 수 있는 매개변수도 없다.
 - 출력문장을 사용하여 필드에 있는 값들을 표준 출력장치에 출력한다.

[Member 클래스 테스트]

```
public class MemberTest {

    public static void main(String[] args) {
        Member m = new Member();
        m.set("hong", "010-1111-1234");
        m.prn();
    }
}
```

- 아무나 접근할 수 있는 클래스이다.
- 클래스를 컴파일하고 java MemberTest를 실행하면 자동으로 실행되는 main 함수가 선언되어 있다.
- main 함수
 - Member 타입의 변수 m을 선언하고 Member 클래스를 생성하여 대입한다.
 - 생성된 변수 m의 메서드 set을 사용하여 값을 전달한다.
 - m의 메서드 prn을 실행한다.

예) 생성자 메서드를 사용하여 필드를 초기화하는 클래스 Product

[Product 클래스]

```
class Product {
    String code;
    String codeName;
    public Product(String code, String codeName) {
        this.code = code;
        this.codeName = codeName;
    }

    public void prn() {
        System.out.println("code : " + this.code);
        System.out.println("codeName : " + this.codeName);
    }
}
```

- 같은 패키지내에서만 접근할 수 있는 클래스이다.
- 문자열 필드인 code와 codeName이 선언되어 있는데 같은 패키지내에서만 접근할 수 있다.
- 문자열 타입의 매개변수 2개가 있는 생성자가 기술되어 있다.
- 생성자
 - 객체가 생성될 때 전달받은 두 개의 문자열을 각각 필드에 저장하고 있다.
 - 생성자의 접근 권한이 public 이지만 클래스 자체의 접근 권한이 default로 되어 있어 생성자는 클래스의 접근 권한에 영향을 받는다.
- prn 메서드
 - 표준 출력 장치에 필드의 값들을 출력한다.

[Product 테스트 클래스]

```
public class ProductTest {  
  
    public static void main(String[] args) {  
        Product p1 = new Product("a001", "tv");  
        Product p2 = new Product("a002", "computer");  
  
        p1.prn();  
        p2.prn();  
    }  
}
```

- 아무나 접근할 수 있는 클래스이다.
- 컴파일 후 java ProductTest를 실행하면 자동으로 실행되는 main함수가 있다.
- main 함수
 - Pruduct 타입의 변수 p1, p2를 선언하고 Product클래스의 생성자를 통해 값을 전달하여 객체를 생성한 후 p1, p2에 객체를 대입한다.
 - p1과 p2에 있는 메서드 prn을 실행한다. p1과 p2는 독립된 객체이다.

6.7 final 수정자

- 클래스에 사용될 경우 : 더 이상 상속을 허락하지 않겠다라는 의미

- 메소드에 사용될 경우 : 재정의(오버라이딩)을 허락하지 않겠다는 의미
- 변수에 사용될 경우 : 변수가 하나의 상수화 된다.

예1) final 클래스 F를 선언한 뒤 F를 상속받으려 할 때 오류 확인

예2) final void m()를 정의한 후 재정의 하려할 때 오류 확인

예3) final double PI=3.14를 필드로 정의한 후 값을 재설정하러 할 때 오류 확인

6.8 static 제한자

- 클래스에는 사용할 수 없으며 메소드나 멤버 변수에 사용할 수 있다.
- 일종의 전역 변수 역할을 한다.
- 접근할 때 클래스 객체에서는 물론 클래스이름만으로도 접근이 가능하다.
- static으로 선언된 변수의 초기화는 클래스가 인스턴스화 될 때 최초 1회만 실행된다.

예1)

```
public class StaticSum {  
    static int sum(int x, int y) {  
        return x+y;  
    }  
}
```

```
public class StaticSumTest {  
  
    public static void main(String[] args) {  
        int r = StaticSum.sum(10,20);  
        System.out.println(r);  
    }  
}
```

* StaticSum 클래스를 생성하지 않고 바로 클래스명만으로 sum메서드를 사용할 수 있음을 알 수 있다.

* 위와 같은 이유로 자바의 Math 클래스내에 있는 대부분의 메서드들은 Math 클래스명만을 사용하여 바로 사용할 수 있는 것이다.

예2) 클래스 공통변수

```
public class StaticMarine {
    static int HP=1;
    int posX=10;
    int posY=10;
    public String info() {
        String str = String.format("hp:%d, x=%d, y=%d", HP, posX, posY);
        return str;
    }
}
```

```
public class StaticMarineTest {
    public static void main(String[] args) {
        StaticMarine[] m = new StaticMarine[5];
        for(int i=0; i<m.length ; i++) {
            m[i] = new StaticMarine();
        }
        System.out.println("초기값 마린들");
        for(StaticMarine sm : m) {
            System.out.println(sm.info());
        }
        System.out.println("마린1 이동");
        m[0].posX=100;
        System.out.println("이동 결과");
        for(StaticMarine sm : m) {
            System.out.println(sm.info());
        }
        System.out.println("생체연구소 건물 완성");
        System.out.println("마린 HP 증가");
        StaticMarine.HP = 10;
        for(StaticMarine sm : m) {
            System.out.println(sm.info());
        }
    }
}
```

[실행결과]

```
초기값 마린들
hp:1, x=10, y=10
hp:1, x=10, y=10
hp:1, x=10, y=10
hp:1, x=10, y=10
hp:1, x=10, y=10
마린1 이동
이동 결과
hp:1, x=100, y=10
hp:1, x=10, y=10
hp:1, x=10, y=10
hp:1, x=10, y=10
hp:1, x=10, y=10
생체연구소 건물 완성
마린 HP 증가
hp:10, x=100, y=10
hp:10, x=10, y=10
hp:10, x=10, y=10
hp:10, x=10, y=10
hp:10, x=10, y=10
```

6.9 native

다른 언어로 작성한 메소드를 자바 프로그램에서 호출할 수 있도록 해당 메소드를 선언할 때 사용.

6.10 synchronized(동기화)

동시에 두 가지 이상의 작업을 수행하는 멀티 스레드 프로그램에서 유용하게 사용된다. synchronized 가 붙은 메서드는 동시에 접근 할 수 없게 한다. (좌석예약 프로그램을 예로 들자)

7 상속과 구현

상속이란 일반적인 관점에서 보면 부모가 자식에게 무언가를 물려주는 것을 상속이라 표현하는 것처럼 자바에서도 부모 클래스의 것을 자식 클래스에게 물려주는 것을 상속이라 한다.

7.1 재정의(Override)

7.1.1 정의

상위 클래스로 부터 상속 받은 메서드의 내용을 하위 클래스에서 기능을 바꾸거나 기능을 확장하는 것.

7.1.2 재정의의 규칙

- 메서드명과 인수의 종류, 갯수, 순서가 반드시 일치해야 한다.
- 반환형이 일치해야 한다.
- 접근자의 범위는 크거나 같아야 한다.(확대)
- 예외처리의 갯수나 범위는 같거나 적어야 한다.(축소)
- 인스턴스 메서드를 static메서드로 또는 그 반대로 변경할 수 없다.

7.1.3 Overloading 과 Overriding의 차이

구분	Overloading	Overriding
공통	메서드명이 동일해야 한다.	
매개변수	종류, 개수, 순서가 달라야 한다.	반드시 일치해야 한다.
리턴형	상관없다.	일치해야 한다.
접근자	상관없다	접근 범위가 같거나 넓어야 한다.
발생 시점	동일한 클래스내 또는 상속후 발생	상속후에만 발생

7.2 상속 방법

상속 방법에는 크게 세 가지로 나뉜다.

명령어	설명
extends	부모 자식이 모두 같은 유형의 구조로 만들 때
implements	구조가 부모는 interface, 자식은 class 타인 경우
extends, implements	하나의 클래스와 여러 개의 interface를 상속받아 클래스를 만드는 경우

7.2.1 extends

- 부모 자식이 모두 class 유형인 경우 부모 class는 하나만 사용할 수 있다.(단일상속)
- 부모 자식이 모두 interface인 경우 여러 개의 interface 부모를 둘 수 있다.

예 1)

```
class P1 { ... }  
class P2 { ... }  
  
class C1 extends P1 { ... } (O)  
class C1 extends P1, P2 { ... } (X)
```

예 2)

```
interface I1 { ... }  
interface I2 { ... }  
interface I3 extends I1 { ... } (O)  
interface I4 extends I1, I2 { ... } (O)
```

7.2.2 implements

- 한 개 이상의 부모 인터페이스를 지정할 수 있다.
- 부모 인터페이스에 있는 추상 메서드는 자식 클래스에서 반드시 재 정의 해야 한다.

예1)

```
interface I1{ ... }
interface I2{ ...}
class C1 implements I1{ ... } (O)
class C1 implements I1, I2{ ... } (O)
```

예1) 추상 메서드가 있는 interface인 경우

```
interface I1{
    public void input();
}

class C implements I1{
    @Override
    public void input(){ ... }
}
```

7.2.3 extends, implements

하나의 클래스와 여러 개의 interface를 상속받아 클래스를 구성할 수 있다. 이 때도 interface에 있는 추상 메서드는 자손 클래스에서 재정의해야 한다.

예1) JFrame와 ActionListener

```
package chapter07_inheri;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
```

```
import javax.swing.JFrame;

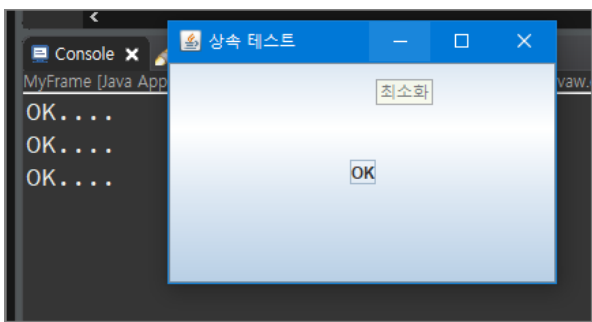
public class MyFrame extends JFrame implements ActionListener{
    JButton btn;

    public MyFrame() {
        btn = new JButton("OK");
        btn.addActionListener(this);
        this.add(btn);
        this.setTitle("상속 테스트");
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("OK....");
    }

    public static void main(String[] args) {
        MyFrame mf = new MyFrame();
        mf.setBounds(100,100,300,200);
        mf.setVisible(true);
    }
}
```

[실행결과]

상속을 통해 간단한 몇줄만으로 윈도우 프레임이 만들어지고 프레임의 최소화 버튼, 이전화면 버튼, 최대화 버튼등이 만들어지고 기능을 할 수 있다는 것을 알 수 있다.



8 추상 클래스와 인터페이스

8.1 추상 클래스와 abstract 수정자

메소드나 클래스 앞에 선언되며 메소드와 클래스에 붙었을 때 각각 다른 의미를 갖는다.

1) 메소드에 붙었을 때

메소드의 프로토타입¹만 정의하고 내용은 정의하지 않을 때 `abstract`를 사용한다. `abstract` 메소드를 사용하려면 하위 클래스에서 오버라이드해서 사용해야 한다.

2) 클래스에 붙었을 때

해당 클래스에 대한 `Object`를 생성할 수 없다는 뜻이다. `abstract` 메소드가 하나라도 포함되어 있거나, 상위 클래스에서 상속받은 메소드 중에서 `abstract` 메소드가 하나라도 포함되어 있으면 그 클래스는 `abstract` 클래스 (추상 클래스)가 되어야 한다.

3) `abstract` 수정자를 사용하는 이유

하위 클래스를 제대로 사용하기 위해서는 해당 메서드를 반드시 구현해야 한다는 것을 명시해 주는 역할을 한다.

4) 선언방법

```
abstract 접근제한자 반환형 메소드명(매개변수);
```

내용부분의 `{ }`는 생략하고 기술하지 않는다.

¹ 프로토타입이란? 메서드를 정의할 때 접근자 메서드명 매개변수만을 작성하는 형태. 즉 `{ }`은 작성하지 않는다.
예) `public void prn(String str);`

예)

먼저 출력에 관한 추상 클래스를 작성한다.

```
abstract class Abstract {
    String name;
    abstract public String prn(); // 오버라이딩해야 함.
}
```

console 출력용으로 사용하기 위해 Abstract 클래스를 상속받아 작성한다.

```
package chapter07_inheri;

class AbstractToConsole extends Abstract {
    // 반드시 재정의해야 할 메서드
    @Override
    public String prn() {
        String str = "Name : " + name;
        return str;
    }
}
```

web 용으로 출력하기 위해 Abstract 클래스를 상속받아 작성한다.

```
public class AbstractToWeb extends Abstract {
    @Override
    public String prn() {
        String str = "<h2>" + name + "</h2>";
        return str;
    }
}
```

* 위의 예를 보면 내용을 출력하기 위한 변수로는 name을, 출력용 메서드는 prn으로 강제하고 있는것을 볼 수 있다.

8.2 interface와 완전 추상

추상 클래스(abstract class)는 일부분의 메서드가 추상 메서드인것에 반해 interface는 모든 메서드가 추상 메서드만 존재한다. interface를 선언할 때 추상 클래스를 정의할 때와 마찬가지로 abstract를 사용해야 하지만 interface는 완전 추상이기 때문에 메서드명 앞에 abstract는 일반적으로 생략하여 사용한다.

interface 안에 기술된 필드들은 기본적으로 상수로 사용된다.

그리고 자바 버전이 올라가면서 interface에도 다양한 패턴들이 존재하지만 본 지면에서는 일단 다양한 패턴들의 사용형태는 배제하도록 하겠다.

예) interface 선언

```
public interface InterfaceTest {  
    int abc=10; //상수  
    public void fun1();  
    public void fun2(int a, int b);  
}
```

```
public class InterfaceImplement implements InterfaceTest {  
    public void fun1() {  
        // abc=100; // 오류  
        System.out.println("fun1");  
        System.out.println(abc);  
    }  
  
    public void fun2(int a, int b) {  
        System.out.println("a=" + a + ", b=" + b);  
    }  
  
    public static void main(String args[]) {  
        InterfaceImplement i = new InterfaceImplement();  
        i.fun1();  
        i.fun2(100, 200);  
    }  
}
```


9 패키지과 import

package는 일종의 폴더를 만드는 것으로 이해해도 될 것 같다. 실제로 package를 만들면 새로운 폴더가 만들어진다. 그러나 package를 만드는 이유는 단순히 소스관리나 폴더를 만드는 것이 목적이 아니라 개발자가 만든 프로그램들이 서로 충돌이 일어나지 않게 하는 일종의 안전 장치중 하나로 사용된다.

예를 들어 날짜를 관리하는 프로그램으로 서로 다른 두명의 개발자가 개발하게 되더라도 같은 이름으로 클래스명을 사용할 가능성이 매우 높을 것이다. Date로 말이다. 그런데 이 클래스가 서로 다른 기능을 갖고 있어 두 클래스가 모두 하나의 class에서 사용하고자 할 때는 package 개념이 없었다면 구분해서 쓸 방법이 없다. 간단한 사례였지만 이 이외에도 각기 다른 클래스들이 하나의 이름으로 되어 있다면 하나의 프로그램에서 사용하려 할 때 수 많은 문제점들을 야기할 수 있을 것이다.

만약 package를 만들어 주었다면 이런 혼란을 쉽게 없었을 있다. 그 이유는 package명은 묵시적으로 도메인의 역순으로 만들기 때문이다.

두 개발자의 도메인이 각각 jobtc.kr , korea.kr 이라면 Date 객체를 생성할 때

```
package kr.jobtc;
class Date{ ... }

package kr.korea;
class Date{ ... }
```

와 같이 작성하게 되면 같은 Date 객체를 사용하더라도

```
kr.jobtc.Date d1 = new kr.jobtc.Date();
kr.korea.Date d2 = new kr.korea.Date();
```

와 같이 서로 구분하여 사용할 수 있게 된다.

9.1 package

위에서 언급한 것처럼 package 명은 도메인의 역순으로 만드는 것이 원칙이며 자바 코드에서 사용할 때 다음과 같은 특징을 갖는다.

- package는 하나만 기술할 수 있다.

- 주석을 제외한 프로그램의 첫줄에 기술해야 한다.

9.2 import

import 는 외부에 있는 클래스를 사용해야 할 경우 사용한다. C의 #include 와 그 역할이 유사하다. 자바의 클래스는 그 가지수가 매우 많아 종류별로 클래스를 묶어 관리하고 있다. 이것을 Package라 부른다.

그중에 사용자가 특별히 선언하지 않고 사용할 수 있는 Package는 "java.lang" 이다. 우리가 익히 사용한 "System.out.print"라는 명령도 실제로는 java.lang안에 들어 있는 하나의 클래스이다.

import를 선언하는 방법은 크게 두가지로 선언한다.

- 1) import 패키지.클래스;
- 2) import 패키지.*;

- 1)번의 경우는 특정 패키지의 클래스만을 삽입하는 경우이고,
- 2)번의 경우는 특정 패키지안에 있는 **모든 클래스만을** 삽입하라는 명령이다. 즉 패키지안에 또 다른 패키지가 존재 한다면 하위 패키지안에 있는 클래스들은 import 되지 않는다.

import를 사용하는 이유는 코드의 가독성을 높이고 외부 패키지에 있는 클래스명을 보다 간결하게 기술 하기 위한 방법이다.

package 에서 언급된 Date 객체를 import를 사용하지 않고 사용한 예

```
kr.jobtc.Date d1 = new kr.jobtc.Date();
```

import를 사용한 예

```
import kr.jobtc.Date;
...
Date d1 = new Date();
```

10 예외 처리

- 오류가 발생할 수 있는 코드에 예외 처리를 할 수 있다.
- 발생한 오류를 코드를 호출한 곳으로 전달할 수 있다.
- 임의로 예외를 발생 시킬 수 있다.

자바에서는 예외를 직접 처리하든지, 직접 처리하지 않을 경우 처리하지 않겠다고 선언해야 한다.
만약 직접 처리하겠다고 선언할 경우 try 문을 사용하고 직접 처리 하지 않을 경우 throws문을 사용해야 한다.

10.1 기본 구조

```
try{
    예외 발생이 예상되는 문장;
}

catch (AException e){
    AException 에 대한 처리;
}
catch (BException e){
    BException 에 대한 처리;
}

[finally{
    무조건 실행되는 부분;
}]
```

- 1) finally 문장은 생략가능하다.
- 2) 예외(Exception)의 종류는 클래스 계층 구조로 분류되어 있다.
- 3) 클래스의 계층 구조로 되어 있기 때문에 상위 계층의 Exception을 위에 기술하면 하위의 Exception은

실행되지 않는다. 따라서 최하위의 Exception을 맨 상위에 기술하는 것이 좋다.

4) finally 문자는 각 catch문에 return 문이 있어도 finally 문장을 수행한후 return 된다. 단, System.exit() 문장이 있는 경우는 finally 문자를 수행하지 않는다.

10.2 주요 예외(Exception)의 종류

1) IOException (입출력시 발행하는 예외)

IOException	입출력이 잘못되었을 때 발생
EOFException	입출력시 파일의 끝을 만났을 때 발생
FileNotFoundException	파일이 존재하지 않았을 때 발생

2) RuntimeException (실행시 발생하는 예외)

NullPointerException	클래스 객체에 값을 주지 않았거나 잘못 주었을때
ArrayIndexOutOfBoundsException	배열 첨자를 잘못 주었을때
ArithmeticException	수학적인 처리가 잘못되었을때
NumberFormatException	입력된 문자열을 숫자열로 변환시킬 수 없을때

[Exception 예제]

```
// =====
// NumberFormatException
// 문자열을 숫자로 변환시킬수 없을때
// 작성자 : 박원기
// =====
class NumberFormatExceptionSample{
    public static void main(String[] args)    {
        int i;
        try{
            i=Integer.parseInt(args[0]);
        }
        catch(NumberFormatException e){
```

```
        System.out.println("숫자가 아님..");
    }
    System.out.println(args[0]);
}
}
```

```
// =====
// ArrayIndexOutOfBoundsException
// 배열의 인수가 잘못되었을때
// 작성자 : 박원기
// =====
class ArrayIndexOutOfBoundsExceptionSample{
    public static void main(String[] args){
        String s;
        try{
            s=args[1];
            System.out.println(s);
        }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println("인수가 적습니다...");
        }
    }
}
```

```
// =====
// ArithmeticException
// 연산식이 잘못되었을때
// 작성자 : 박원기
// =====
class ArithmeticExceptionSample{
    public static void main(String[] args){
        int x, y;
        try{
```

```
        x=20;
        y=x/0;
    }
    catch(ArithmeticException e)    {
        System.out.println("연산식에 오류가 있습니다.");
    }
}
```

10.3 예외 던지기

예외를 바로 처리하기 위해서는 try~catch 문장을 사용하지만, 자신의 블록에서 예외를 바로 처리하지 않고 예외가 포함된 메서드를 호출한 곳으로 발생한 예외를 전달하기 위해서는 throws 절을 메서드 선언시 추가해 주고, 메서드를 호출한 문장은 try~catch절 안에 있어야 한다.

```
public void m() throws 예외 종류{ ... }
```

m() 메서드를 호출한 곳에서는 아래와 같이 try절안에 있어야 한다.

```
try{
    m();
}
catch(예외 종류){ ... }
```

[throws 예]

```
class ThrowsExceptionEx{
    int div(int a) throws ArithmeticException {
        int r;
        r = a / 0;
        return r;
    }
}
```

```
void call(){
    try{
        div(10);
    }catch(ArithmeticException ex){
        System.out.println("연산식에 오류가 있습니다.");
    }
}

public static void main(String[] args) {
    ThrowsExceptionEx ag = new ThrowsExceptionEx();
    ag.call();
}
}
```

10.4 임의로 예외 발생시키기

코드를 테스트 하려 하거나, 특정 조건이 되면 예외를 발생시켜 프로세스의 흐름을 차단하거나 변경할 수 있는 방법이 있다. 물론 if문등의 블록으로 처리할 수도 있지만 throw 문장을 사용하면 코드의 가독력을 향상시킬 수 있겠다.

throw 예외종류;

```
public class ThrowExam {

    public void test() {
        String n="A";
        try {
            if(n.equals("A")) throw new Exception();
            System.out.println("run.....");
        }catch(Exception ex) {
            System.out.println("문자열이 A라서 오류 발생...");
        }
    }
}
```

```
}

public static void main(String[] args) {
    ThrowExam t = new ThrowExam();
    t.test();

}

}
```


11 java.lang 패키지

java.lang 패키지는 자바 프로그램에서 기본적으로 사용되어야 하는 클래스들을 포함하고 있으며, import 문장을 사용하지 않고도 사용할 수 있도록 되어있다.

11.1 Object

자바의 모든 클래스의 최고 조상이 되는 클래스이다. 멤버 변수는 없고 11개의 메서드만을 제공한다.

주요 메서드	설 명
protected Object clone()	객체 자신의 복사본을 반환한다.
public boolean equals(Object obj)	객체 자신과 객체 obj가 같은 객체인지 알려준다.
public int hashCode()	자신의 해시코드를 반환한다.
public String toString()	자신의 정보를 문자열로 반환한다.
public void notify() public void notifyAll()	자신을 사용하려고 기다리는 스레드를 깨운다.
public void wait() public void wait(long timeout) public void wait(long timeout, int nanos)	다른 스레드가 notify()나 notifyAll()을 호출할 때까지 현재 스레드를 기다리게 한다. - timeout : 천분의 1초 - nanos : 10 ⁹ 분의 1초
public Class getClass()	자신의 객체 정보를 담고 있는 Class 인스턴스를 반환한다.
protected void finalize()	객체가 소멸될 때 가비지 콜렉션에 의해 자동으로 호출된다.

11.1.1 clone 메서드

- 기본형의 필드값은 복사되지만 참조형 필드의 값은 완전한 복사가 되지 않고 얇은 복사가 된다.
- 배열과 같은 참조형 멤버를 깊은 복사하려면 clone 메서드를 재정의하여 처리해야 한다.
- Cloneable 인터페이스를 구현한 클래스에서만 clone 메서드를 사용할 수 있다.

```
public class Point implements Cloneable{
    int x;
    int y;
    public Object clone(){
        Object o = null;
        try{
            o = super.clone();
        }catch(Exception ex){}
        return o;
    }
}
```

```
public class CloneTest {
    public CloneTest(){
        Point p = new Point();
        p.x = 100;
        p.y = 200;

        Point temp = (Point)p.clone();
        temp.x=50;
        temp.y=50;
        System.out.println("p.x=" + p.x);
        System.out.println("temp.x=" + temp.x);
        System.out.println(p== temp); //false
        System.out.println(p.equals(temp)); //false
    }
    public static void main(String[] args) {
        new CloneTest();
    }
}
```

* 두 객체의 비교값은 모두 false가 나온다. 즉 서로 다른 객체임을 의미한다.

11.1.2 hashCode 메서드

hashCode는 객체를 식별하는 정수값이다. 객체가 생성된 메모리 번지를 기본으로 정수값이 만들어지기 때문에 객체 마다 서로 다른 값을 갖는다.

아래의 코드를 작성한 후 출력결과를 살펴보자.

```
public class hashCodeObject {  
}
```

```
public class hashCodeObjectTest {  
    public static void main(String[] args) {  
        hashCodeObject o1 = new hashCodeObject();  
        hashCodeObject o2 = new hashCodeObject();  
  
        System.out.println(o1.hashCode());  
        System.out.println(o2.hashCode());  
        System.out.println(o1==o2);  
    }  
}
```

[출력결과] 실행할 때 마다 다른 값이 나올 수 있다.

```
15211118594  
1940030785  
false
```

따라서 같은 클래스를 사용하여 객체를 생성하더라도 hashCode 값이 다르기 때문에 같은 객체로 인식하지 않는 것이다.

그렇다면 논리적으로 클래스가 갖고 있는 특정한 값이 같다면 같은 객체로 인식하도록 해 보자. hashCode 메서드를 재정의하여 특정값을 반환하게 하면 될 것이다.

```
public class hashCodeObject {  
    int id;  
  
    public hashCodeObject(int id) {
```

```
        this.id = id;
    }
    @Override
    public int hashCode() {
        return id;
    }

    public String toString() {
        return super.toString() + "," + super.hashCode();
    }
}
```

```
public class HashCodeObjectTest {

    public static void main(String[] args) {
        HashCodeObject o1 = new HashCodeObject(10);
        HashCodeObject o2 = new HashCodeObject(10);
        HashCodeObject o3 = o1;

        System.out.println(o1.hashCode());
        System.out.println(o2.hashCode());
        System.out.println(o3.hashCode());

        System.out.println(o1.toString());
        System.out.println(o2.toString());
        System.out.println(o3.toString());

        System.out.println(o1==o2);
        System.out.println(o1.equals(o2));

        System.out.println(o1==o3);
        System.out.println(o1.equals(o3));

    }
}
```

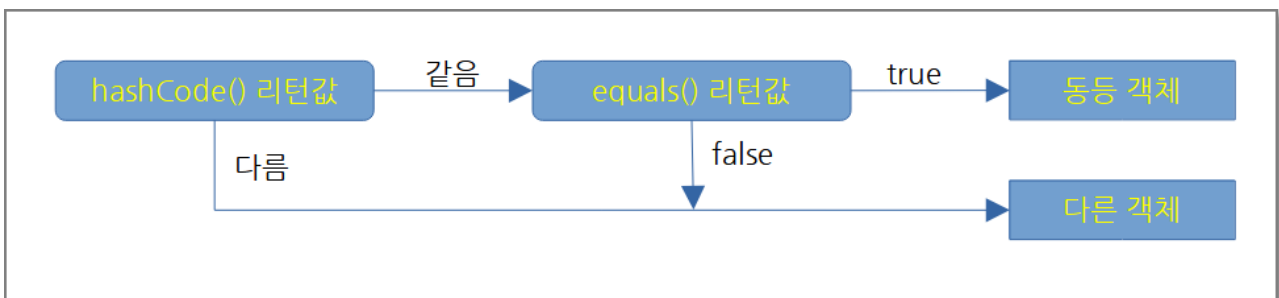
[실행결과]

10

```

10
10
d_class.HashCodeObject@a,1521118594
d_class.HashCodeObject@a,1940030785
d_class.HashCodeObject@a,1521118594
false
false
true
true
    
```

놀랍게도 hashCode값이 동일하면 동일한 객체로 인식될것 같았는데 서로 다른 객체라고 출력된다. 이는 아래의 그림으로 이해될 수 있을 것이다.



HashCodeObject의 hashCode 값은 같았을지 모르지만 부모 객체인 Object의 hashCode값은 다르게 나왔다. 이는 Object 객체가 갖고 있는 equals 메서드를 재정의하지 않았기 때문에 나오는 결과이다.

11.1.3 equals 메서드

객체가 생성되어 저장된 객체의 주소값으로 같은지를 판별하게 된다. 따라서 '=='로 비교하든, equals 메서드를 사용하여 비교하든 같은 값이 나온다.

```

package d_class;

public class EqualsTest {

    public static void main(String[] args) {
        Object o1 = new Object();
        Object o2 = new Object();
    }
}
    
```

```
System.out.println(o1==o2); //false
System.out.println(o1.equals(o2)); //false

Object o3 = o1;
System.out.println(o1==o3); //true
System.out.println(o1.equals(o3)); //true
}
```

따라서 객체가 같고 있는 값들과 같은지를 판별하여 같은 객체인가를 판별하려면 equals 메서드를 재정의하여야 한다.

예) id가 동일하면 동일객체로 판별

```
public class EqualsObject {
    String id;

    @Override
    public boolean equals(Object obj) {
        boolean b = false;
        if(obj instanceof EqualsObject) {
            EqualsObject eo = (EqualsObject) obj;
            if(this.id.equals(eo.id)) {
                b=true;
            }
        }
        return b;
    }
}
```

```
public class EqualsObjectTest {

    public static void main(String[] args) {
        EqualsObject o1 = new EqualsObject();
        EqualsObject o2 = new EqualsObject();
        EqualsObject o3 = new EqualsObject();
    }
}
```

```
o1.id = "hong";
o2.id = "hong";
o3.id = "kim";

System.out.println(o1==o2); //false
System.out.println(o1==o3); //false

System.out.println(o1.equals(o2)); // true
System.out.println(o1.equals(o3)); // false
}
```

equals메서드를 재정의한다고 하더라도 '==' 로 비교할 경우 서로 다른 객체라 판단한다.

따라서 두 객체가 완전 동등 객체임을 비교해야 하는 프로그램이라면 hashCode 메서드와 equals 메서드를 함께 재정의해야 된다.

11.1.4 toString 메서드

- 인스턴스에 대한 정보를 문자열로 제공한다.
- 반환되는 기본 문자열은 자신의 클래스명@hashCode()의 결과 값이다.
- 메서드를 재정의하여 클래스가 기본적으로 출력해야 하는 문자열을 만들어 사용자에게 제공할 수 있다.

11.2 String

고정 문자열을 처리하는 각종 메서드가 있다.

주요 메서드	설명
char charAt(int p)	p 위치의 문자를 하나 반환
byte[] getBytes(encoding)	문자열을 encoding 방식으로 변환(euc-kr, utf-8, ...)
int length()	문자열의 길이를 구함

<code>int indexOf(String str [, int offset])</code>	<code>str</code> 의 위치를 구함. 못찾은경우 -1 반환 <code>offset</code> 이 지정되면 그 이후의 위치부터 찾음.
<code>String[] split(String str, int arg1)</code> <code>String[] split(String regexp)</code>	문자열을 <code>str</code> 이나 정규 표현식으로 나눈다. <code>arg1</code> 이 있으면 <code>arg1</code> 의 갯수만큼만 나눈다.
<code>String substring(int 시작위치[, int 끝위치])</code>	시작위치에서 끝위치전까지의 문자열 반환. 끝 위치가 생략되면 시작위치에서 문자열 끝까지 반환
<code>String trim()</code>	양쪽 공백을 제거한 문자열 반환
<code>String replace(바꿀문자열, 바꿀문자열)</code> <code>String replaceAll(정규식, 바꿀문자열)</code>	문자열을 바꾸어 새로운 문자열 반환

예1)

```
String str = "abc,123.def,456.abc,123"
```

- 1) `str`에서 1이 시작되는 위치를 찾아 출력
- 2) ","를 기준으로 문자열을 쪼개 출력
- 3) 모든 "."을 ","로 바꾸어 출력

예2)

```
String nal = "2022-01-18"
```

- 1) 년월일을 나누어 출력

11.3 StringBuffer | StringBuilder

`String` 을 고정문자열이라 한다고 하면, `StringBuffer`나 `StringBuilder`는 가변 문자열이라 볼 수 있다. 하나의 변수값이 자주 바뀐다면 `String`을 사용하는것 보다 `StringBuffer`나 `StringBuilder`를 사용하는 편이 매우 유리하다. 버퍼의 기본크기는 16 characters이다.

[주요 메서드]

메서드명	설명
<code>append(매개인자)</code>	문자열 끝에 매개인자값을 추가한다.

	매개인자에는 자바가 사용하는 대부분의 자료형이 올 수 있다.
insert(int offset, 매개인자)	offset 위치에 매개인자값을 삽입한다.
delete(int start, int end) deleteCharAt(int index)	start<= x < end 위치에 있는 문자열을 제거 index 위치의 문자값 제거
replace(int start, int end, String str)	start<=x < end 위치에 있는 문자열을 str로 대치
StringBuilder reverse()	문자열의 순서를 바꿔 새로운 StringBuilder로 반환
setCharAt(int index, char ch)	index 의 값을 ch로 변경

예)

```
String name = "이름:홍길동"
String addr = "주소 : 대한민국"
String phone="연락처 : 연기를 내시오"
StringBuilder에 각각의 문자열을 추가한 후 출력.
```

응용) String과 StringBuilder의 속도차

```
package chapter11_api;

public class StringSpeed {
    public StringSpeed(){
        String str = "a";
        long sTime=0, eTime=0, rTime=0, rTime2;

        //-----
        // String
        //-----
        sTime = System.currentTimeMillis();
        for(int i=0;i<99000;i++) str += "a";
        eTime = System.currentTimeMillis();

        rTime = (eTime-sTime);
        System.out.println("Time1 : " + (rTime/1000d) + " 초");
```

```
//-----
//  StringBuilder
//-----

StringBuilder sb = new StringBuilder();
sTime = System.currentTimeMillis();
for(int i=0;i<99000;i++) sb.append("a");
eTime = System.currentTimeMillis();
rTime2 = (eTime-sTime);
System.out.println("Time2 : " + (rTime2/1000d) + " 초");

System.out.println(rTime/rTime2 + " 배");
}
public static void main(String[] args) {
    new StringSpeed();
}
}
```

11.4 StringTokenizer

지정된 문자열을 구분자를 사용하여 token 이라는 조각으로 분리하는 기능.

구분자를 2개 이상의 문자로 지정하더라도 한자씩 사용하여 token으로 분리함.

[주요 생성자]

생성자	설 명
StringTokenizer(String source, String delim)	delim에 들어있는 문자 하나 하나 따로 사용하여 source 문자열을 토큰으로 만든다.
StringTokenizer(String source, String delim, boolean b)	Boolean 값이 true이면 delim 문자들로 토큰이 된다.

[주요 메서드]

메서드	설명
int countTokens()	꺼내지 않고 남아 있는 토큰의 갯수
boolean hasMoreTokens()	남아 있는 토큰이 있으면 참
String nextToken()	토큰을 하나 꺼내옴.

예)

```
import java.util.StringTokenizer;

public class SplitAndStringToken {

    public static void main(String[] args) {
        String str="a/b/c///d-e,f";

        StringTokenizer token = new StringTokenizer(str,"/,-", false);
        System.out.println(token.countTokens());

        System.out.println("token result:");
        while(token.hasMoreElements()){
            System.out.printf("sp[%d]=%s\n",pos,token.nextElement());
        }
    }
}
```

11.4.1 StringTokenizer와 String.split의 차이

- StringTokenizer는 무효의 값은 버려진다.
- String.split은 무효의 값도 배열로 처리된다. 문자열로 쪼갤 수 있다.

11.5 Math

대부분의 수학과 관련한 함수들이 들어 있는 클래스이다. 메서드는 모두 static형이므로 클래스명을 사용하여 바로 사용 가능하다.

[주요 메소드]

메소드명	기능	예
int abs(int a) double abs(double d)	절대값 반환	Math.abs(-10) => 10 Math.abs(-3.14) => 3.14
double ceil(double d)	소수점 이하를 올림값(절상)	Math.ceil(3.01) => 4.0 Math.ceil(-3.01) => -3.0
double floor(double d)	소수점 이하를 내림값(절하)	Math.floor(3.01) Math.floor(-3.01)
int max(int x, int y) double max(double x, double y) int min(int x, int y) double min(double x, double y)	x,y중 큰값 또는 작은값	
double random()	난수값 $0 \leq r < 1$	
long round(double x)	반 올림값	

예1) 예금 이자를 연이율 4.5%를 계산하고 소수점이하는 절상하여 표시

예2) 부가세 10% 가 붙은 최종 총액이 10000원이다. 이를 제품가격과 부가세로 나누어 표시(단 원단위 미만은 부가세에서 절삭)

예2) 1~45 사이의 수중 중복되지 않은 정수 6개를 추출하여 표시

11.6 Wrapper

- 기본형 8가지에 대한 객체를 만들거나 값을 반환할 수 있는 클래스들.

- 데이터의 변환은 자동으로 boxing, unboxing 된다.

종류 : Byte, Character, Short, Integer, Long, Float, Double, Boolean

중요 메서드	설명
Byte.parseByte(str) Short.parseShort(str) Integer.parseInt(str) Long.parseLong(str) Float.parseFloat(str) Double.parseDouble(str)	문자열 str를 해당 타입의 숫자로 바꾸어준다.
int Character.getNumericValue(Char c)	c 값을 정수형으로 반환

예) 매개변수가 문자열로 되어 있는 숫자 x,y를 전달 받았다. x,y를 더한 값을 출력

12 기본 API 클래스

12.1 Format 클래스

클래스명	설명
DecimalFormat	천단위 분리기호나 소숫점자리를 지정
MessageFormat	DM 등 메시지를 발송할 때의 포맷 지정
SimpleDateFormat	날짜 형식 지정.
ChoiceFormat	범위내에 있는 값을 선택.

12.1.1 DecimalFormat

- 천단위 분리기호를 만들어 준다.
- 소수점 이하의 자리수가 부족하면 반 올림된다.

주요 기호	설 명
0	빈자리는 0으로 채움
#	빈자리는 공백으로 채움
E	지수 표현
;	음수 양수를 구분하여 표현
%	백분율로 표시
\u00a4	통화기호

```
DecimalFormat df = new DecimalFormat("\u00a4 ##,###,###.###");
String str = df.format(1234667.489999); //₩ 1,234,667.49
```

12.1.2 MessageFormat

DM과 같은 일정한 문자열에 특정 값들이 변경될 때 사용됨.

```
String str = " name : {0} \n age : {1} \n address : {2}";
String args[] = {"hong", "18", "남원"};
String r = MessageFormat.format(str, args);
```

12.1.3 SimpleDateFormat

날짜 데이터를 일정한 형식으로 바꾸어 준다.

형식	설명
yyyy,MM,dd	년월일
hh:mm:ss	시분초. ss:천분의 1초 HH : 24시각제
E	요일

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd(E) HH:mm:ss:SS");
String str = sdf.format(new Date()); // 2021-10-26(화) 22:30:31:707
```

12.1.4 ChoiceFormat

- 체크할 값이 특정 영역이면 그에 해당하는 문자를 리턴함.
- 영역을 지정할 값은 오름차순으로 정렬되어 있어야 함.

```
double limit[] = {60,70,80,90}; // 오름차순으로 정렬되어 있어야 함
String grade[] = {"D","C","B", "A"};
int myscore[] = {88,70,99,55,66,45,55,88,77,88};
```

```
ChoiceFormat cf = new ChoiceFormat(limit, grade);
for(int a : myscore){
    System.out.println(a + " : " +cf.format(a));
}
```

[실행결과]

```
88 : B
70 : C
99 : A
55 : D
66 : D
45 : D
55 : D
88 : B
77 : C
88 : B
```

12.2 Pattern 클래스

문자열을 효과적으로 검색하거나 일정한 패턴의 문자열인지 판별하기 위해 사용.

주요 기호	의 미
[]	하나의 문자. [abc] → a,b,c 이면 참
[^]	문자가 아니면 참. [^abc] → a,b,c 가 아니면 참
*	0개 이상
+	1개 이상
?	0 또는 1개
.	한개 문자
\d	[0-9]의 의미

\w	[a-zA-Z_0-9]의 의미
()	그룹
{n}, {n,m}, {n,}	n~m개 까지. {n,} → n개 이상

[이메일 테스트]

```
String email = "\\w+@\\w+\\.\\w+(\\.\\w+)?";

String[] test = {
    "hipwg@", "hipwg@naver", "hipwg@naver.com",
    "hipwg@naver.co.kr", "hipwg@naver.com.co.kr", "@naver.com"
};

for(String s : test) {
    boolean b = Pattern.matches(email, s);
    System.out.println(s + " >>> " + b);
}
```

[실행결과]

```
hipwg@ >>> false
hipwg@naver >>> false
hipwg@naver.com >>> true
hipwg@naver.co.kr >>> true
hipwg@naver.com.co.kr >>> false
@naver.com >>> false
```

[연락처 테스트]

```
String phone = "\\d{2,3}-\\d{3,4}-\\d{4}$";
String[] test = {
    "02-12-1234", "02-123-1234", "02-1234-123",
    "0321-123-1234", "010-6351-3491"
};

for(String s : test) {
    boolean b = Pattern.matches(phone, s);
    System.out.println(s + " >>> " + b);
}
```

[실행결과]

```
02-12-1234 >>> false
02-123-1234 >>> true
02-1234-123 >>> false
0321-123-1234 >>> false
010-6351-3491 >>> true
```

13 중첩 클래스

클래스 안에 클래스를 선언하여 사용하는 방법이다. 주로 이벤트를 처리할 목적이나 간단한 쓰레드를 처리할 목적으로 중첩 클래스를 사용하지만 그 외의 사용은 최대한 사용하지 않는편이 좋다. 그 이유는 클래스가 갖고 있는 일반적인 객체의 특성을 대부분 사용을 어렵게 하거나 특성이 회손되기 때문이다.

[중첩클래스의 유형]

유형	설명
인스턴스형	<pre>class A{ class B{ ... } }</pre>
정적형	<pre>class A{ static class B{ ... } }</pre>
로컬형	<pre>class A{ void method(){ class B{ ... } } }</pre>

예) 이벤트를 처리를 내부 클래스를 만들어 처리하고 있다.

```
import java.awt.*;
import java.awt.event.*;

class innerClass4 extends Frame {
    Button b = new Button("exit");
    myActionListener myAction = new myActionListener();
}
```

```
public innerClass4() {
    setLayout(null);
    setSize(400, 400);
    b.setBounds(100, 100, 100, 100);
    b.addActionListener(myAction);
    add(b);
}

public static void main(String args[]) {
    innerClass4 ic4 = new innerClass4();
    ic4.setVisible(true);
}

// method type inner class
class myActionListener implements ActionListener{
    public void actionPerformed(ActionEvent ev) {
        System.exit(0);
    }
}
}
```

14 컬렉션

자바에서 사용하는 자료 구조의 한 방법으로 배열의 단점을 보완하고 검색이나 정렬등의 편의 기능들이 추가됨.

14.1 종류

- List
- Set
- Map
- Properties

14.2 주요기능 비교

구분	LIST	SET	MAP
C(Create)	add	add	put
R(Read)	get	Iterator	get, keySet, values
U(Update)	set	remover → add	put
D>Delete)	remove		
기타	- 순서있음 - 중복가능 - 배열 구조와 유사	- 순서 없음 - 중복 안됨	- KEY : 중복안됨 - VALUE : 중복됨.

14.3 List

- 순서를 유지하고 자료의 중복 저장이 가능하다.
- 배열과 가장 유사한 구조이다.
- 주요 클래스
 - ArrayList, Vector

[메서드]

메서드명	설명
add(E)	요소를 추가한다.
set(Index , E)	Index 위치의 값을 E로 수정한다.
get(Index)	Index 위치의 값을 가져온다.
remove(Index) remove(E)	Index 위치나 E의 값을 제거한다.
boolean contains(E)	E의 요소가 있는지 판별한다.
boolean containsAll(List)	List요소의 부분 집합인가를 체크
List<E> retainAll(List<E>)	두 List의 교집합을 구함.

[예] 개나리, 진달래, 코스모스, 장미, 백합, 아카시아를 List에 저장하고 저장된 값을 출력하시오.

[예2] 교집합 구하기

retainAll을 사용하여 교집합을 구할 수 있다.

```
Integer[] a = {1,2,3,4,5,6,7,8,9};
Integer[] b = {1,4,5,6,10,11,12};

//aa 의 값을 변경하려면 반드시 배열값을 new ArrayList를 따로 생성하여 추가해야 함.

List<Integer> aa = new ArrayList<>(Arrays.asList(a) );
List<Integer> bb = new ArrayList(Arrays.asList(b) );
System.out.println("aa=" + aa);
System.out.println("bb=" + bb);

aa.retainAll(bb);
System.out.println("retain aa =" + aa.toString());
```

[실행결과]

```
aa=[1, 2, 3, 4, 5, 6, 7, 8, 9]
bb=[1, 4, 5, 6, 10, 11, 12]
retain aa =[1, 4, 5, 6]
```

[예] 부분집합 확인하기

containsAll 을 활용하여 부분 집합인가를 체크할 수 있다.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

// containsAll을 사용한 부분집합 체크
public class contains {

    public static void main(String[] args) {
        Integer[] aa = {1,2,3,4,5};
        Integer[] bb = {3,4,5,6,7};
        Integer[] cc = {1,2,3};

        // 배열을 List로 생성
        List<Integer> aaList = new ArrayList<>(Arrays.asList(aa));
        List<Integer> bbList = new ArrayList<>(Arrays.asList(bb));
        List<Integer> ccList = new ArrayList<>(Arrays.asList(cc));

        // cc가 aa의 부분집합인가를 체크
        boolean cBoolean = aaList.containsAll(ccList);
        if(cBoolean) {
            System.out.println("cc는 aa의 부분집합");
        }else {
            System.out.println("cc는 aa의 부분집합이 아님.");
        }

        // bb가 aa의 부분집합인가 체크
        boolean bBoolean = aaList.containsAll(bbList);
        if(bBoolean) {
            System.out.println("bb는 aa의 부분집합");
        }else {
            System.out.println("bb는 aa의 부분집합이 아님.");
        }
    }
}
```

```
}  
  
}
```

[실행결과]

```
cc는 aa의 부분집합  
bb는 aa의 부분집합이 아님.
```

[예] 차집합 구하기

removeAll을 활용하여 차집합을 구할 수 있다.

```
package chapter15.collection.list;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
  
// 두 리스트의 교집합 구하기  
public class removeAll {  
  
    public static void main(String[] args) {  
        Integer[] a = {1,2,3,4,5,6,7,8,9};  
        Integer[] b = {1,4,5,6,10,11,12};  
  
        //aa 의 값을 변경하려면 반드시 배열값을 new ArrayList를 따로 생성하여 추가해야 함.  
  
        List<Integer> aa = new ArrayList<>(Arrays.asList(a) );  
        List<Integer> bb = new ArrayList(Arrays.asList(b) );  
        System.out.println("aa=" + aa);  
        System.out.println("bb=" + bb);  
  
        aa.removeAll(bb);  
        System.out.println("차집합 =" + aa.toString());  
    }  
}
```



```
}  
}
```

[실행결과]

```
aa=[1, 2, 3, 4, 5, 6, 7, 8, 9]  
bb=[1, 4, 5, 6, 10, 11, 12]  
차집합=[2, 3, 7, 8, 9]
```

14.4 Set

- 중복저장되지 않고 순서를 유지하지 않는다.
- 주요 클래스
 - HashSet, Hashtable

[주요 메서드]

메서드명	설명
add(E)	E를 추가한다.
remove(index) remove(E)	index위치값이나 E를 제거한다.
iterator()	값을 나열형으로 가져온다.
boolean contains(E)	E의 값이 있는지 판별한다.

[예] List에 저장된 값들중 중복을 제거하여 하나의 Set을 만들 수 있다.

```
package chapter15.collection.set;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

// 두 리스트의합집합에서 중복값 제거
public class DupleRemove {

    public static void main(String[] args) {
        Integer[] a = {1,2,3,4,5,6,7,8,9};
        Integer[] b = {1,4,5,6,10,11,12};

        //aa 의 값을 변경하려면 반드시 배열값을 new ArrayList를 따로 생성하여 추가해야 함.

        List<Integer> aa = new ArrayList<>(Arrays.asList(a));
        List<Integer> bb = new ArrayList(Arrays.asList(b));
        System.out.println("aa=" + aa);
        System.out.println("bb=" + bb);

        aa.addAll(bb);
        System.out.println("aa+bb = " + aa.toString());

        Set<Integer> set = new HashSet<>();
        set.addAll(aa);
        System.out.println("remove duple = " + set);

    }
}
```

[실행결과]

```
aa=[1, 2, 3, 4, 5, 6, 7, 8, 9]
bb=[1, 4, 5, 6, 10, 11, 12]
aa+bb = [1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 4, 5, 6, 10, 11, 12]
remove duple = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

14.5 Map

- Key, Values가 한쌍으로 저장된다.
- Key는 중복되지 않고 저장 순서가 유지되지 않는다.
- 동일한 Key값으로 값을 저장하면 기존값이 수정된다.
- Map.Entry를 사용하여 키와 값을 한쌍으로 처리할 수 있다.
- Value는 중복저장된다.
 - 주요 메서드 : HashMap, TreeMap, SortMap

[주요 메서드]

메소드명	설명
put(K, V)	K,V를 한쌍으로 저장한다.
get(K)	키값이 K인 값을 가져온다.
remove(K)	키값이 K인 값을 제거한다.

```
package chapter15.collection;

import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
```

```
public class HashMapEx {
    public HashMapEx(){
        HashMap<String, String> hm =
            new HashMap<String, String>();
        //입력
        hm.put("a1", "aaaa");
        hm.put("a2", "bbbb");
        hm.put("a3", "cccc");
        System.out.println("입력결과 : " + hm.toString());
        // 수정
        hm.put("a2", "dddd");
        System.out.println("수정결과 : " + hm.toString());
        // 삭제
        hm.remove("a2");
        System.out.println("삭제결과 : " + hm.toString());
        // key값을 기준으로 출력
        Set<String> set = hm.keySet();
        Iterator<String> it = set.iterator();
        System.out.println("key 값을 기준으로 출력");
        while(it.hasNext()){
            System.out.println(hm.get(it.next()));
        }
        // value값을 기준으로 출력
        Collection<String> c = hm.values();
        Iterator<String> it2 = c.iterator();
        System.out.println("value 값을 기준으로 출력");
        while(it2.hasNext()){
            System.out.println(it2.next());
        }

        //Entity
        Set s = hm.entrySet();
        Iterator it3 = s.iterator();
        while(it3.hasNext()){
            Map.Entry en = (Map.Entry<String, String>)it3.next();
            System.out.println(en.getKey()+ " : " + en.getValue());
        }
    }
    public static void main(String[] args) {
        new HashMapEx();
    }
}
```

```
}
}
```

14.6 Properties

- 키값=값 형태로 저장된다.
- 키와 값은 모두 문자열 형태이다.

[주요 메서드]

메서드명	설명
load(InputStream)	스트림을 통해 읽어 들인다.
store(OutputStream)	스트림을 통해 저장한다.

```
package chapter15.collection;

import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.net.URLDecoder;
import java.util.Properties;

public class PropertiesEx {

    public void test1() {
        //String path = PropertiesEx.class.getResource("myprop.properties").getPath();
        String path = "newProp.properties";
        Properties prop = null;
        try {
            path = URLDecoder.decode(path, "utf-8"); // 한글 복원
            System.out.println("파일 경로 : " + path);
        }
    }
}
```

```
    FileReader reader = new FileReader(path);
    prop = new Properties();
    prop.load(reader);

    String id = prop.getProperty("id");
    String irum = prop.getProperty("irum");
    String address = prop.getProperty("address");
    String phone = prop.getProperty("phone");

    System.out.println("id : " + id);
    System.out.println("irum : " + irum);
    System.out.println("address : " + address);
    System.out.println("phone : " + phone);

} catch (Exception ex) {
    ex.printStackTrace();
}

}

public void test2() {
    File f = new File("newProp.properties");
    if( !f.exists())
        try {
            f.createNewFile();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    Properties prop = new Properties();
    prop.setProperty("id", "a001");
    prop.setProperty("irum", "홍길동");
    prop.setProperty("phone", "010-6351-3491");

    try {
        prop.setProperty("address", new String("대한민국".getBytes(), "utf-8"));
        String path = URLDecoder.decode(f.getPath(), "utf-8");
        FileWriter fw = new FileWriter(path);
        prop.store(fw, "");
        fw.close();
    } catch (Exception ex) {}
}
```

```
public static void main(String[] args) {  
    PropertiesEx p = new PropertiesEx();  
    p.test2();  
    p.test1();  
}  
}
```

15 Thread

동시에 2개 이상의 작업을 동시에 작업하려 할때 다중처리 방법들을 사용해야 하는데, 자바에서는 Thread를 통해 다중 처리를 수행한다. 자바에서 Thread를 2개 이상의 Thread를 동시에 만들어 실행할 수 있도록 멀티스레드 기능을 지원한다.

[멀티 태스킹과 멀티 쓰레드의 차이점]

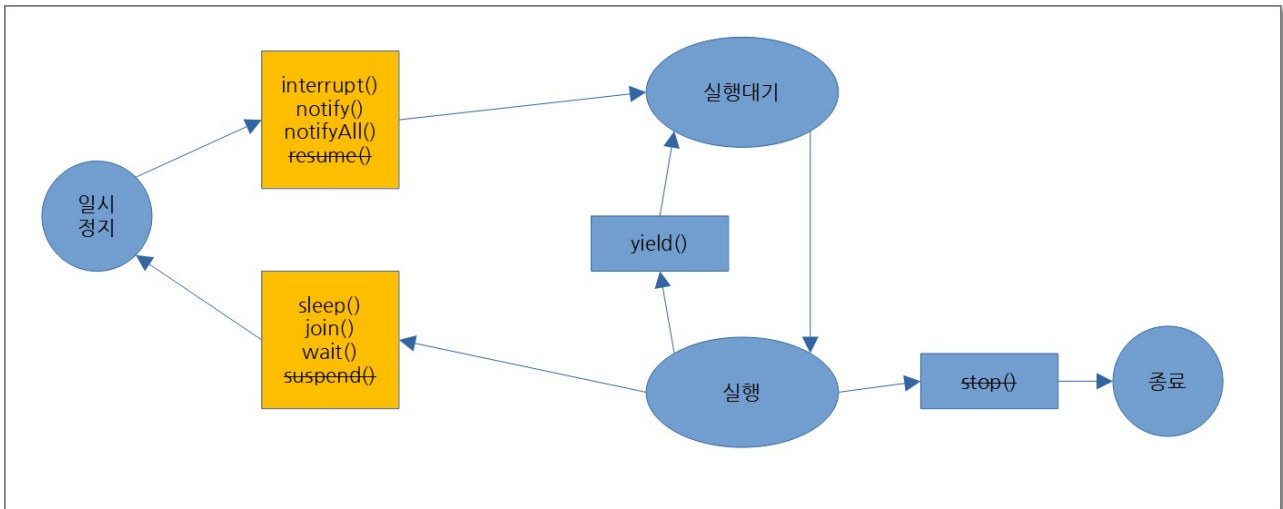
멀티 태스킹 : 하나 이상의 프로세서를 이용하여 여러개의 프로그램을 동시에 실행하는 것.

멀티 쓰레드 : 하나의 프로그램에서 여러개의 일을 동시에 실행하게 하는 것.

15.1 Thread 생성 방법

Thread를 상속했을 때	Runnable을 구현하였을 때
<pre>class A extends Thread{ @Override public void run(){ ... } }</pre>	<pre>class A implements Runnable{ @Override public void run(){ ... } }</pre>
<pre>A a = new A(); a.start()</pre>	<pre>A a = new A(); Thread t = new Thread(a); t.start()</pre>

15.2 Thread Life Cycle



15.3 Daemon Thread

- 메인 스레드가 종료되면 작업 스레드를 무조건 종료하게 하는 방법
- `setDaemon(true)`를 통해 데몬 스레드를 만들 수 있다.

```

package chapter12_thread;

public class DaemonThred extends Thread{

    @Override
    public void run() {
        try{
            for(int i=1 ; i<=100 ; i++){
                System.out.printf("%5d",i);
                if(i%10==0) System.out.println();
                Thread.sleep(30);
            }
            System.out.println("작업 스레드 종료");
        }catch(Exception ex){

        }
        System.out.println(this.getName() + " 스레드 종료"); }
    
```

```
public static void main(String[] args) {
    DaemonThred dt = new DaemonThred();
    dt.setDaemon(true);
    dt.start();

    try {
        Thread.sleep(2000);
    } catch (Exception ex) {}

    System.out.println("\n\n프로그램 완전 종료\n\n");
}
}
```

[실행결과]

```
 1  2  3  4  5  6  7  8  9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62

프로그램 완전 종료

63
```

그림 6: 데몬 쓰레드 결과

```
 1  2  3  4  5  6  7  8  9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63

프로그램 완전 종료

64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100

작업 쓰레드 종료
Thread-0 쓰레드 종료
```

그림 7: 일반 쓰레드 결과

15.4 join()하기

작업쓰레드가 종료되기전에 메인 쓰레드를 종료하지 못하게 하는 방법

```
class JoinThread extends Thread{
```

```
public void run(){
    try{
        for(int i=1 ; i<=100 ; i++){
            System.out.printf("%5d",i);
            if(i%10==0) System.out.println();
            Thread.sleep(30);
        }

    }catch(Exception ex){

    }
    System.out.println(this.getName() + " 스레드 종료");
}

}

public class JoinTest {

    public static void main(String[] args) {
        JoinThread jt = new JoinThread();
        try{
            jt.start();
            // join()을 하지 않으면 호출된 JoinThread가 종료되기 전에
            // 메인 프로그램이 먼저 종료된다.

            jt.join();
            System.out.println("스레드 종료후 실행...");
        }catch(Exception ex){

        }
        System.out.println("프로그램 완전 종료!!!!");

    }

}
```

[실행결과]

자바의 개요 15.4 join()하기

```
  1   2   3   4   5   6   7   8   9  10
11  12  13  14  15  16  17  18  19  20
21  22  23  24  25  26  27  28  29  30
31  32  33  34  35  36  37  38  39  40
41  42  43  44  45  46  47  48  49  50
51  52  53  54  55  56  57  58  59  60
61  62  63  64  65  66  67  68  69  70
71  72  73  74  75  76  77  78  79  80
81  82  83  84  85  86  87  88  89  90
91  92  93  94  95  96  97  98  99 100
Thread-0 스레드 종료
스레드 종료후 실행...
프로그램 완전 종료!!!!
```

16 람다식

1990년대에 디자인되었지만 주목 받지 못하다가 최근들어 다시 부각되고 있는 함수적 프로그래밍 기법이다. 함수적 프로그래밍 기법은 병행처리나, 이벤트 지향 프로그래밍에 적합하다고 알려져 있다. 자바는 8버전부터 람다식을 지원하고 있다. 람다식의 장점은 자바 코드가 간결해 지고, 컬렉션 요소를 필터링하거나 매핑해서 원하는 결과를 손쉽게 집계할 수 있기 때문이다.

람다식은 매개변수를 가진 코드 블록이지만 런타임시에는 익명 구현 객체를 생성하여 실행된다.

16.1 기본 문법

```
(매개변수 타입 매개변수, ...) -> { 처리내용; [return 값] }
```

- 매개변수가 없는 경우 ()는 반드시 사용해야 한다.

```
()->{ ... }
```

- 매개변수 타입은 실행시 대입되는 값에 따라 자동 인식 될 수 있기 때문에 기술하지 않아도 된다.

```
(변수형 변수, ...) ->{ ... }
```

```
(변수, ...) -> { ... }
```

- 매개변수가 하나이고 실행문장도 한개라면 (), {}를 생략할 수 있다.

```
a -> System.out.print(a)
```

- { }안에 return문장만 있다면 return 키워드와 {}를 생략할 수 있다.

```
(x,y) -> { return x+y; }
```

```
(x,y) -> x+y
```

기타 사용예는 Stream 항목에서 살펴 보도록 하자.

17 스트림

Stream은 자바8버전 부터 추가된 기능으로 배열이나 Collections의 저장 요소를 하나씩 참조하여 랴다식으로 처리할 수 있도록 도와주는 반복자이다.

예를 들어 List에 저장된 데이터를 출력하기 위해 일반적으로 아래와 같은 방법으로 사용해 왔다.

```
List<String> list = new ArrayList<String>();
list.add("강아지");
list.add("호랑이");
list.add("고양이");
list.add("망아지");

for( String s : list) {
    System.out.println(s);
}
```

위의 코드 스트림으로 바꾸어 표현하면 아래와 같이 된다.

```
List<String> list = new ArrayList<String>();
list.add("강아지");
list.add("호랑이");
list.add("고양이");
list.add("망아지");

// lambda
Stream<String> stream = list.stream();
stream.forEach( x -> System.out.println(x) );
```

또는

```
List<String> list = new ArrayList<String>();
list.add("강아지");
list.add("호랑이");
```

```
list.add("고양이");
list.add("망아지");

// list인 경우 stream 없이 바로 forEach 사용가능
list.forEach( x -> System.out.println(x) );
```

17.1 배열을 스트림으로 반복하기

```
String[] names = {"hong", "kim", "lee", "park"};

// 기존 코드 방식
for(String s : names) {
    System.out.println(s);
}

System.out.println("-----");
// 스트림 방식
Stream<String> stream = Arrays.stream(names);
stream.forEach(x-> System.out.println(x));
```

17.2 Map 구조를 스트림으로 반복하기

```
Map<String, String> map = new HashMap<>();
map.put("a001", "kim");
map.put("a002", "lee");
map.put("a003", "park");
map.put("a004", "hong");

// 기존 방식
Set<Map.Entry<String, String>> set = map.entrySet();
Iterator<Map.Entry<String, String>> iter = set.iterator();
while(iter.hasNext()) {
    Map.Entry<String, String> entry = iter.next();
```

```
System.out.println("key:" + entry.getKey());
System.out.println("value : " + entry.getValue());
System.out.println("-----");
}

// 스트림
map.forEach((key,value)->{
    System.out.println(key);
    System.out.println(value);
});
```

17.3 map()

출력시 모두 대문자로 출력하고자 할 때 등과 같이 출력 컬렉션을 매핑하거나 변경할 때 사용한다.

```
List<String> list = Arrays.asList("abc", "Abc", "aBC", "abC");
Stream stream = list.stream().map(x ->x.toUpperCase());
stream.forEach(x -> System.out.print(x + ", "));
```

17.4 distinct()

스트림내에 있는 값들중 중복값을 제거한다.

```
List<Integer> list = Arrays.asList(1,2,3,4,3,4,5,6);
Stream stream = list.stream();
stream.distinct().forEach(x -> System.out.print(x)); //123456
```

17.5 filter()

컬렉션에 있는 데이터를 조건에 맞는 것만을 골라낼 수 있다. filter 메서드는 boolean 결과를 리턴하는 람다 표현식이 필요하다.

```
List<Integer> list = Arrays.asList(1,2,3,4,3,4,5,6);
```



```
Stream stream = list.stream().filter(x -> x>4);
stream.forEach(x -> System.out.print(x+ " "));
```

17.6 sorted()

컬렉션의 값들을 정렬하여 출력

```
List<Integer> list = Arrays.asList(6,5,8,9,4,3,2,1,7);
Stream stream = list.stream().sorted();
stream.forEach(x -> System.out.print(x+ " "));
```

17.7 match()

컬렉션의 조건들이 특정 조건에 만족하는지 조사한다.

[종류]

- allMatch() : 모든 조건이 만족할 때 참
- anyMatch() : 하나의 조건만 만족해도 참
- noneMatch() : 모든 조건이 만족하지 않으면 참

```
List<Integer> list = Arrays.asList(6,5,8,9,4,3,2,1,7);
boolean b1 = list.stream().allMatch(x -> x%2==0);
System.out.println("모두 짝수인가 ? " + b1); // false

Stream stream = list.stream();
boolean b2 = list.stream().allMatch(x -> x<10);
System.out.println("모두 10 보다 작은가 ? " + b2); // true
```

17.8 기본 집계

종류 : sum(), count(), average(), max(), min()

예1) sum

```
int[] su = {1,2,3,4,5};
long hap = Arrays.stream(su).sum();
System.out.println("hap : " + hap); //15

hap = Arrays.stream(su).filter(x->x<4).sum();
System.out.println("hap2 : " + hap); //6
```

예2) count

```
int[] su = {1,2,3,4,5};
long cnt = Arrays.stream(su).count();
System.out.println("cnt : " + cnt); //5

cnt = Arrays.stream(su).filter(x->x<4).count();
System.out.println("hap2 : " + cnt); //3
```

17.9 reduce()

스크림내부의 값을 변경하지 않고 다양한 형태로 집계할 수 있는 기능이다.

```
Stream<Integer> stream = Stream.of(1,2,3,4,5);
Optional<Integer> opt = stream.reduce((x,y)->x+y);
opt.ifPresent(s -> System.out.println(s));
```

17.10 collect()

컬렉션에 저장된 데이터를 특정 기준에 따라 분류하여 새로운 Collection으로 반환한다.

```
Integer[] su = {1,2,3,4,5,6,7,8,9};  
List<Integer> list = Arrays.stream(su).filter(x->x>5)  
    .collect(Collectors.toList());  
  
list.stream().forEach(x -> System.out.println(x));
```

18 JDBC

Java Database Connectivity의 약자로 자바 표준 데이터베이스 인터페이스를 뜻하며 자바 프로그램과 데이터베이스를 연결하는 하나의 통로를 의미한다.

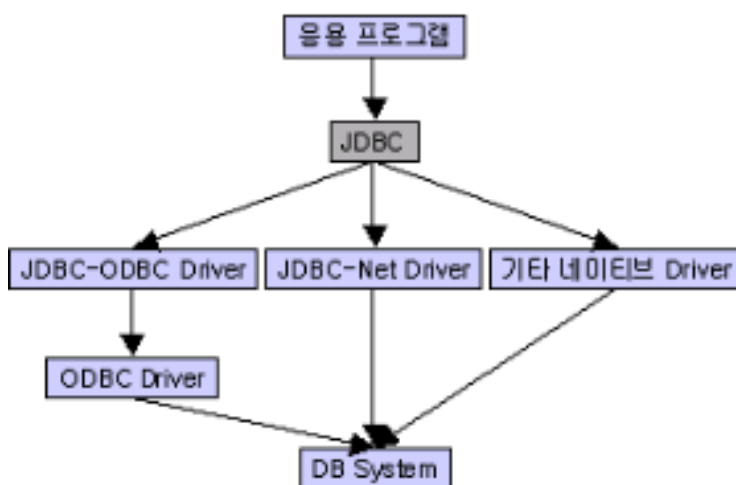
Interface :

인터페이스는 몸체가 없는 메서드들로 이루어진 클래스를 뜻하며 자바에서는 다중 상속을 구현하기 위해 많은 부분을 인터페이스 형태로 클래스를 지원한다.

JDBC도 일종의 데이터베이스의 연결통로로 사용되는 인터페이스로 대부분의 데이터 베이스 회사들은 Sun사로부터 JDBC인터페이스를 제공받아 자신들의 데이터 베이스에 맞게 기능을 부여한 것을 말한다.

따라서 사용자들은 JDBC의 내부 구조를 알지 못하더라도 interface만 알면 데이터 베이스를 조작할 수 있는 것이다. 이러한 데이터베이스 인터페이스의 종류로는 JDBC, ODBC등이 있다.

18.1 JDBC 드라이버의 종류



1) JDBC-ODBC Driver

윈도우 계열에서만 사용가능하며 각각의 클라이언트 컴퓨터에 ODBC 드라이버가 사전에 설치되어 있어야 한다. JDBC-ODBC 브릿지라고도 불린다.

2) JDBC-Net Driver

네트워크를 기반으로 하는 순수 자바 기반 드라이버로 JDBC와 데이터베이스 사이에 미들웨어를 두어 처리하는 방식이다.

특정 데이터베이스에 종속되지 않는다. 개별적인 데이터베이스 서버가 그 프로토콜을 받아 자체적으로 번역하여 실행한다. JDBC의 종류중 가장 유연성이 있는 드라이버라 할 수 있다. 그러나 미들웨어를 추가 설치해야 한다는 단점이 있다.

3) Native Driver

네트워크를 기반으로 하는 순수 자바 기반 드라이버로, 드라이버는 데이터베이스 업체에서 직접 제작하여 배포한다. 대부분의 JDBC 드라이버가 이에 속한다고 볼 수 있다.

데이터의 처리속도 및 관련 내용은 각 드라이버 마다 최적화 되어 있다고 볼 수 있으며, 클라이언트에는 추가적인 라이브러리나 애플리케이션이 필요없다는 장점이 있는 반면, 데이터베이스의 종류가 바뀌면 사용 드라이버도 변경되어야 한다는 단점이 있다.

4) Native-API Driver

JDBC 명령을 데이터베이스에서 사용하는 프로토콜에 맞춰 변환한후 전달하는 방식이다. 각 데이터베이스별로 특화된 기능을 최대한 사용할 수 있다는 장점이 있는 반면, 각 클라이언트마다 데이터베이스 제작사에서 제공하는 별도의 드라이버가 설치되어 있어야 한다는 단점이 있다.

18.2 드라이버의 등록

Driver등록은 크게 두가지 방법을 제시한다.

1) Class.forName() 메소드를 이용하는 방법

드라이버를 직접 선택해서 로딩하고자 할때

- Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); // MS-SQL
- Class.forName("postgresql.Driver"); // PostgreSQL
- Class.forName("oracle.jdbc.driver.OracleDriver"); // Oracle DB
- Class.forName("com.mysql.cj.jdbc.Driver"); // My-SQL mysql용 드라이버

2) DriverManager.registerDriver() 메소드를 이용하는 방법

특정 드라이버를 레지스터에 등록하고자 할때

- DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());
- DriverManager.registerDriver(new postgresql.Driver());
- DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
- DriverManager.registerDriver(new org.gjt.mm.mysql.Driver());

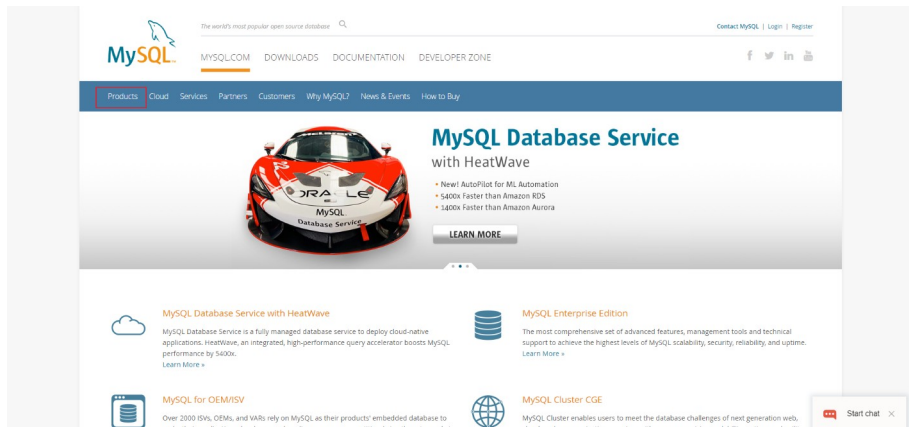
18.3 MySQL 드라이버 등록 하기

16.3.1MS-Windows에서 MySQL JDBC 드라이버 설치하기

Mysql과 Java는 현재 시스템에 설치되어있어야 한다. Java 는 c:\java에 설치되었다고 가정한다.

1) 드라이버 다운로드 www.mysql.com 에 접속하여 적당한 드라이버를 다운로드한다. 파일명은 mysql-connector-java-버전-stable.zip 와 유사할 것이다.

자바의 개요 18.3 MySQL 드라이버 등록 하기



mysql.com 사이트를 방문한 뒤, 메뉴에서 Products를 선택한다.

	MySQL Standard Edition	MySQL Enterprise Edition	MySQL Cluster CGE
Annual Subscription ^{2,3,4,5}	USD 2,000	USD 5,000	USD 10,000
product-metric	Buy Now	Buy Now	Buy Now
Oracle Premier Support³			
24x7 Support	✓	✓	✓
Unlimited Support Incidents	✓	✓	✓
Knowledge Base	✓	✓	✓
Maintenance Releases	✓	✓	✓
MySQL Consultative Support	✓	✓	✓
MySQL Features			
MySQL Database Server	✓	✓	✓
MySQL Document Store		✓	✓
MySQL Connectors	✓	✓	✓
MySQL Replication	✓	✓	✓
MySQL Router		✓	✓
MySQL Partitioning		✓	✓
MySQL Shell		✓	✓
MySQL Workbench ¹	✓	✓	✓
Storage Engine: MyISAM	✓	✓	✓
Storage Engine: InnoDB	✓	✓	✓

MYSQL Connectors를 선택한다.

자바의 개요 18.3 MySQL 드라이버 등록 하기

<div data-bbox="220 338 1074 882"><p>Developed by MySQL</p><table><tr><td>ADO.NET Driver for MySQL (Connector/NET)</td><td>Download</td></tr><tr><td>ODBC Driver for MySQL (Connector/ODBC)</td><td>Download</td></tr><tr><td>JDBC Driver for MySQL (Connector/J)</td><td>Download</td></tr><tr><td>Node.js Driver for MySQL (Connector/Node.js)</td><td>Download</td></tr><tr><td>Python Driver for MySQL (Connector/Python)</td><td>Download</td></tr><tr><td>C++ Driver for MySQL (Connector/C++)</td><td>Download</td></tr><tr><td>C Driver for MySQL (Connector/C)</td><td>Download</td></tr><tr><td>C API for MySQL (mysqlclient)</td><td>Download</td></tr></table></div>	ADO.NET Driver for MySQL (Connector/NET)	Download	ODBC Driver for MySQL (Connector/ODBC)	Download	JDBC Driver for MySQL (Connector/J)	Download	Node.js Driver for MySQL (Connector/Node.js)	Download	Python Driver for MySQL (Connector/Python)	Download	C++ Driver for MySQL (Connector/C++)	Download	C Driver for MySQL (Connector/C)	Download	C API for MySQL (mysqlclient)	Download	<p>JDBC Driver 항목의 Download 링크를 클릭한다.</p>
ADO.NET Driver for MySQL (Connector/NET)	Download																
ODBC Driver for MySQL (Connector/ODBC)	Download																
JDBC Driver for MySQL (Connector/J)	Download																
Node.js Driver for MySQL (Connector/Node.js)	Download																
Python Driver for MySQL (Connector/Python)	Download																
C++ Driver for MySQL (Connector/C++)	Download																
C Driver for MySQL (Connector/C)	Download																
C API for MySQL (mysqlclient)	Download																
<div data-bbox="183 981 1099 1608"><p>MySQL Community Downloads</p><p>Connector/J</p><div><p>General Availability (GA) Releases</p><p>Archives</p></div><p>Connector/J 8.0.27</p><p>Select Operating System:</p><p>Microsoft Windows</p><p>Recommended Windows Download:</p><div><p>MySQL Installer for Windows</p><p>All MySQL Products. For All Windows Platforms. In One Package.</p><p>Starting with MySQL 5.6 the MySQL Installer package replaces the standalone MSI packages.</p><p>Windows (x86, 32 & 64-bit), MySQL Installer MSI</p><p>Go to Download Page ></p></div></div>	<p>Archives 탭을 선택한다.</p>																

자바의 개요 18.3 MySQL 드라이버 등록 하기

MySQL Product Archives

MySQL Connector/J (Archived Versions)

Please note that these are old versions. New releases will have recent bug fixes and features!
To download the latest release of MySQL Connector/J, please visit MySQL Downloads.

Product Version: 8.0.26
Operating System: Platform Independent

Platform Independent (Architecture Independent), Compressed TAR Archive (mysql-connector-java-8.0.26.tar.gz)	Jun 8, 2021	4.0M	Download
Platform Independent (Architecture Independent), ZIP Archive (mysql-connector-java-8.0.26.zip)	Jun 8, 2021	4.7M	Download

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

MySQL open source software is provided under the GPL License.

다운로드하려는 버전과 Platform independent를 선택한 압축 파일을 선택한다.

2) 다운로드한 파일의 압축을 해제하고

3) 압축 해제된 디렉토리 안의 파일 중 mysql-connector-java-버전-stable-bin.jar 파일을 c:\java\lib 디렉토리에 복사한다. (다른 파일은 복사하지 않아도 된다)

4) 시스템 등록정보 -> 사용자 변수 항에 "CLASSPATH"를 편집하거나 추가한다. ;c:\java\lib\mysql-connector-java-버전-stable-bin.jar

5) 시스템을 재 부팅하고 CLASSPATH를 확인한다. c:\>echo %classpath%

* CLASSPATH에 라이브러리를 등록하지 않고, 이클립스를 사용하는 경우 JDBC 라이브러리 파일을 프로젝트내에 복사한 후 프로젝트의 Build Path에서 라이브러리를 등록해야 한다.

18.4 MySQL의 실행권한

데이터베이스를 원격지에서 자유롭게 실행하려면 데이터베이스나 테이블에 실행권한을 부여하여야 한다. 특정 DB의 테이블까지 사용자별로 사용권한을 따로 부여할수도 있지만 수업범위 밖에 있기 때문에 test 데이터베이스의 모든 테이블에 관한 모든 실행권한을 얻도록 해 보자.

윈도우용이든, Linux용이든 그 사용방법은 동일하다.

1) DB를 root권한으로 실행하자.

```
mysql -u root -p
```

2) 권한을 부여할 DB를 오픈하자.

```
use test ; // 사용할 DB명이 test 이다.
```

3) 권한을 부여해 보자.

```
grant all privileges on test.* to park identified by '1111';
```

grant all -> 모든 권한(읽기, 쓰기, 수정, 삭제...) on test.* -> test DB안의 모든 테이블 to park -> 유저 park에게 identified by '1111' -> 암호 '1111'로 접근하도록 허용

18.4.1 MySQL JDBC 드라이버 테스트

```
package jdbc;

import java.sql.Connection;
import java.sql.DriverManager;

public class DriverTest {

    public DriverTest() {
        String driver = "com.mysql.cj.jdbc.Driver";
        String path = "jdbc:mysql://localhost:3306/mysql";
        String user = "root";
        String pwd = "1111";

        try {
            Class.forName(driver);
            System.out.println("driver loading ok.....");
            Connection conn = DriverManager.getConnection(path, user, pwd);
            System.out.println("connection ok.....");
        }
    }
}
```

```
        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
    }

}

public static void main(String[] args) {
    new DriverTest();
}
}
```

18.5 오라클 11gXE 드라이버 등록하기

오라클 사이트에서 무료로 드라이버를 다운로드 받아 설치한다. 그러나 11gXE 버전에서는 배포판안에 JDBC 드라이버가 포함되어 있기 때문에 설치된 경로에 가서 드라이버를 찾아 자바와 관련된 경로에 복사하는 것으로 드라이버 설치의 마무리된다.

18.5.1 드라이버 복사

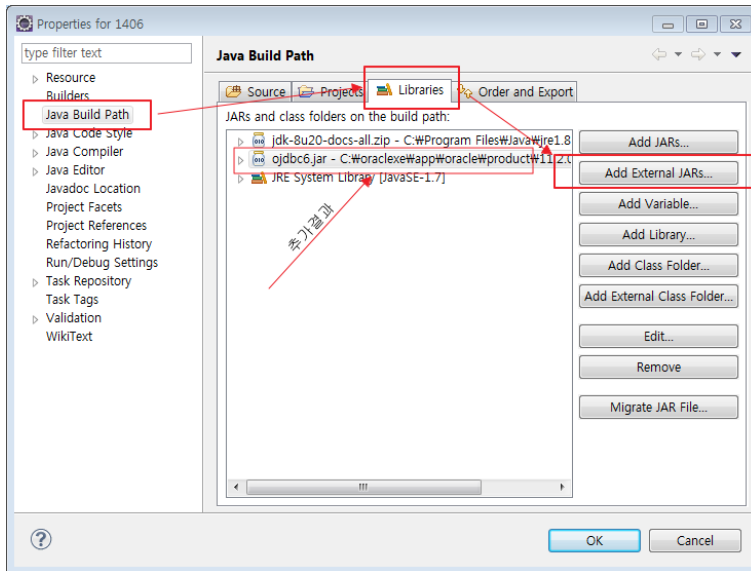
오라클이 설치되어 있는 경로가 c:/oraclexe 라면 드라이버 위치는 아래와 같다.

c:/oraclexe/app/oracle/product/버전/server/jdbc/lib/ojdbc6.jar

자바를 사용하는 다른 프로그램도 해당 드라이버를 사용하도록 하려면 위의 파일을 아래의 자바 경로에 복사한다.

[자바홈]/jre/lib/ext

만약 이클립스에서만 JDBC 드라이버를 사용하도록 한다면 이클립스에서 해당 드라이버를 프로젝트에 추가만 하면 바로 사용할 수 있다.



18.5.2 procedure 호출

오라클의 procedure 나 function을 호출할 수 있다. 그러나 대부분 function은 sql문장 내부에서 응용되어 사용되곤 경우가 많으므로 procedure를 호출하는 방법을 기술 하겠다.

18.5.2.1 단순변수 *return* 사용

Step 1

먼저 sql에서 단순 변수를 return 하는 procedure를 작성한다.

```
/* 자바에서 호출되는 프로시저(단순변수 리턴) */  
create or replace procedure JCallScalar  
  (v1 in number, v2 in number, v3 out number)  
is
```

자바의 개요 18.5 오라클 11gXE 드라이버 등록하기

```
begin
    v3 := v1 + v2;
end;
```

위와 같이 정의된 procedure를 sql에서 직접 테스트 하려면 아래와 같이 테스트 할 수 있다.

```
/* 단순 변수를 리턴하는 프로시저 테스트 */
declare
    v3 number;
begin
    JCallScalar(101,310, v3);
    dbms_output.put_line('v3:' || v3);
end;
```

v1 => 101, v2=>310 값이 입력되고 처리된 내용이 v3에 저장된다.

Step 2.

(Step 1)에서 만들어진 procedure를 자바 코드에서는 prepareCall()를 사용하여 CallableStatement를 생성한 후 procedure의 IN 모드의 값들은 setXXX(index, value)형태로 지정하고 OUT 모드의 값들은 registerOutParameter(index, Type) 형태로 지정한다.

```
package r_jdbc;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Types;

public class ProcedureCall {
    Connection conn ;
    public ProcedureCall(){
        conn = new DBConn().getConn();
    }
}
```

```
/*
 * 작성할 procedure
 * create or replace procedure JCallScalar
 *   (v1 in number, v2 in number, v3 out number)
 *   is
 *   begin
 *     v3 := v1 + v2;
 *   end;
 */
String sql = "{call JCallScalar(?,?,?)}";
try {
    CallableStatement cstmt = conn.prepareCall(sql);
    cstmt.setInt(1, 111);
    cstmt.setInt(2, 222);
    cstmt.registerOutParameter(3, Types.INTEGER);

    cstmt.executeQuery();
    int a = cstmt.getInt(3);
    System.out.println(a);

} catch (SQLException e) {
    e.printStackTrace();
}

}

public static void main(String[] args) {
    ProcedureCall pc = new ProcedureCall();
}
}
```

18.5.2.2 Cursor return 사용

Step 1.

2건 이상의 데이터를 procedure를 통해 전달 받기 위해서는 procedure의 out 모드 형태를 sys_refcursor 타입으로 지정해야 한다.

```
/* 호출되는 프로시저(Cursor) */
create or replace procedure JCallScalarRs
  (dept in emp3.department_id%type , rs out sys_refcursor)
is
```

```
begin
  open rs for
    select * from emp3 where department_id = dept;
end;
```

위의 프로시저를 Sql Tools등에서 테스트하려면 아래와 같이 테스트 할 수 있다.

```
declare
  c sys_refcursor;
  r emp3%rowtype;

begin
  JCallScalarRs(80,c);

  loop
    fetch c into r;
    exit when c%notfound;
    dbms_output.put_line(r.first_name);
  end loop;

end;
```

Step 2.

자바 코드를 사용하여 2건 이상의 데이터를 procedure를 통해 가져와 처리해 보자.

```
package r_jdbc;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Types;

public class ProcedureCallResultSet {
  Connection conn ;

  public ProcedureCallResultSet(){
    conn = new DBConn().getConn();
  }
}
```

```
String sql = "{call JCallScalarRs(?,?)}";
try {
    CallableStatement cstmt = conn.prepareCall(sql);
    cstmt.setInt(1, 80);
    cstmt.registerOutParameter(2, oracle.jdbc.OracleTypes.CURSOR);

    cstmt.executeQuery();
    ResultSet rs = (ResultSet)cstmt.getObject(2); // 두번째 파라미터가 out mode이다.

    while(rs.next()){
        System.out.println(rs.getString("first_name") + "-" + rs.getInt("salary"));
    }

} catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

}

public static void main(String[] args) {
    new ProcedureCallResultSet();
}

}
```

18.6 JDBC 사용 절차

18.6.1 드라이버 로드

데이터베이스를 접속하려면 가장 먼저 해야할 일이 데이터베이스 드라이버를 로드해야 한다. Class.forName()을 사용하여 로드해 보자.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```


18.6.2 데이터베이스 연결

데이터베이스를 연결하려면 연결주소, 사용자 계정명, 암호의 정보를 사용하여 DriverManager 를 사용한 Connection 객체를 생성함으로 연결되어 진다.

Connection 객체를 만드는 방법은 다음과 같다.

```
Connection conn = DriverManager.getConnection(연결주소, 사용자계정명, 암호);
```

이때 연결주소를 지정하는 방법은 데이터베이스의 종류 마다 다르므로 해당 데이터베이스의 API를 참조하기 바란다.

오라클은 아래와 같은 형식으로 연결 주소를 만든다.

JDBC:oracle:thin:@IP:Portj

위와 같은 주소를 문자열로 만들어 연결해야 하기 때문에 아래와 같이 정리 될 수 있다.

```
String db_url = "JDBC:oracle:thin:@127.0.0.1:1521";// 연결주소
```

IP주소는 현재 자바가 실행되는 컴퓨터에 오라클이 설치되어 있다고 가정한 것이며, 포트번호는 1521이 오라클의 기본값이다.

[예]

```
String url = "jdbc:oracle:thin:@127.0.0.1:1521:xe";
String id = "hr";
String pwd = "hr";
Connection conn = DriverManager.getConnection(url, id, pwd);
```

18.6.3 Statement/PreparedStatement 객체 생성

연결된 데이터베이스에 SQL문을 실행할 수 있게 해주는 단계이다. 연결된 Connection객체를 사용하여 두 개 중 하나의 객체를 생성한다. 방법은 아래와 같이 Statement클래스와 PreparedStatement를 사용할 수 있다. 일반적인 SQL문장을 사용하는 경우에는 Statement 보다 PreparedStatement클래스를 사용하는 편이 편리한점이 많다. 간단한 예를 들어 보자.

아이디가 "a001" 이고 연락처가 "123-1234"이며 서울에 사는 남자를 조회한다고 가정해서 데이터를 조회해 보자.

[Statement 를 사용하는 경우]

```
String mid = "a001";
String phone="123-1234";
String gen = "남자";
String address = "서울";

String sql = "select * from member where mid= " + mid + " and phone=" + phone + " and genger = " +
gen + " and address = " + address + """;

Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql);
```

와 같이 조회해야 한다. 즉, 문자열과 변수를 모두 "+"연산자로 더해주어야 하며, 문자열의 값 양쪽에는 작은 따옴표로 묶어주어야 한다. 오타자의 가능성도 높지만 SQL문장 자체를 이해하기 힘들다.

그럼 PreparedStatement를 사용했을 때 SQL문장이 어떻게 바뀌는지 알아보자.

[PreparedStatement를 사용하는 경우]

```
String mid = "a001";
String phone="123-1234";
String gen = "남자";
String address = "서울";

String sql = "select * from member where mid= ? and phone=? and genger=? and address =?";
```

```
PreparedStatement stmt = conn.prepareStatement(sql);
stmt.setString(1, mid);
stmt.setString(2, phone);
stmt.setString(3, gen);
stmt.setString(4, address);
ResultSet rs = stmt.executeQuery();
```

SQL문장 자체가 매우 간단하게 바뀌었다. 대신 PreparedStatement 객체를 sql을 매개변수로 만든 후 setString()메서드를 통해 나중에 값을 ? 에 전달하고있다.

stmt.setString(1,mid)에서 1은 ?의 위치를 나타내는 index역활을 한다. setString()이외에 setInt()와 같이 자바에서 제공하는 8가지 유형에 해당하는 메서드들 모두 제공하고 있다.

18.6.4 SQL 실행

Statement가 만들어졌다면 실제로 sql문장을 실행해야 한다. sql문장은 하나의 String 타입으로 정의하며 대표적으로 executeQuery()와 executeUpdate()를 통해 실행할 수 있다.

1) executeQuery()

select 문을 실행할 때 사용되며 반환형은 ResultSet 형이다.

[예]

```
ResultSet rs = stmt.executeQuery();
```

2) executeUpdate()

update, delete, insert 등과 같은 명령을 실행할 때 사용되며 반환형은 실행된 행수값이다.

[예]

```
int n = stmt.executeUpdate();
```

18.6.5 Statement/PreparedStatement 닫기

마지막 단계에서는 사용된 Statement와 Connection 객체를 닫아준다. 만약 닫아주지 않으면 오라클의 자원이

고갈되어 DB 서버에 접속할 수 없게 된다.

```
stmt.close();  
pstmt.close();  
conn.close();
```

18.6.6 ResultSet

select 문장에 의해 실행된 결과를 저장하는 객체이다.

[ResultSet의 행이동 주요 메소드]

- first() 처음으로
- next() 다음으로
- last() 마지막으로
- previous() 이전으로
- isFirst() 처음인가
- isLast() 마지막인가

그러나 일반적으로 행을 위아래로 스크롤 가능하도록 하려면 Statement를 생성할 때 옵션을 추가로 지정해 주어야 한다.

```
PreparedStatement stmt =  
    conn.prepareStatement(sql , TYPE, CONCURRENCY);
```

[ResultSet TYPE 값]

- TYPE_FORWARD_ONLY : 결과 셋이 스크롤되지 않는다 .(기본값)
- TYPE_SCROLL_INSENSITIVE : 결과 셋은 스크롤되나 데이터베이스 변화에는 반응하지 않는다 .

- TYPE_SCROLL_SENSITIVE : 결과 셋은 스크롤되고 데이터베이스 변화에 반응한다 .

[ResultSet concurrency 값]

- CONCUR_READ_ONLY
- CONCUR_UPDATABLE

[예] 데이터베이스의 변환에 반응하면서 읽기전용으로 PreparedStatement를 만드시오.

```
PreparedStatement stmt =  
    conn.prepareStatement(sql , ResultSet.TYPE_SCROLL_SENSITIVE,  
        ResultSetCONCUR_READ_ONLY);
```

[데이터 가져오기]

```
public void select() throws Exception{  
    String sql = "select mid, email, phone from member";  
    PreparedStatement pst =  
        conn.prepareStatement(sql);  
    ResultSet rs = pst.executeQuery();  
  
    while(rs.next()){  
        System.out.println(  
            rs.getString("mid") + "-" +  
            rs.getString("email") + "-" +  
            rs.getString("phone" ) );  
    }  
    pst.close();  
}
```

- getXXX(필드명) 필드의 데이터 유형에 맞게 get명령이 따로 존재하며, 필드의 유형과 동일해야 한다.

updateXXX(필드명, 데이터) getXXX()와 유사하며 단지 데이터를 DB에 update한다.

만약 SQL의 처리 결과를 저장할 필요가 없는 insert, delete, update등의 SQL문장은 executeUpdate(sql) 명령을 사용한다.

```
stmt.executeUpdate(sql); // 지정된 sql문장을 바로 실행한다.
```

18.7 한글 코드 변환

사용하는 플랫폼이나 사용하는 개발언어에 따라 기본적으로 셋팅되어 있는 한글 코드값이 다를 경우가 있다. 이때 코드값을 적절하게 변형하여 사용해야 하는데 String클래스의 `getBytes()` 메소드를 사용하면 편할 것이다.

```
String str="박원기";  
String s=new String(str.getBytes("8859_1"),"utf-8");
```

위의 코드는 "박원기"란 문자를 8859_1 코드셋으로 받아들이고 utf-8로 변환하여 새로운 스트링 `s`를 만들게 된다. 위의 코드를 활용하여 애플리케이션에서 DB로, DB에서 애플리케이션으로 한글을 처리할 때 사용할 수 있는 메소드를 만들어 보자.

데이터 베이스 사용코드 : 8859_1

응용 애플리케이션 사용코드 : utf-8 일 경우

```
// 한글 데이터를 데이터 베이스로부터 가져올때  
public String fromDB(String str) throws Exception{  
    String s=new String(str.getBytes("8859_1"),"utf-8");  
    return s;  
}
```

```
// 한글 데이터를 데이터 베이스에 저장할때  
public String toDB(String str) throws Exception{  
    String s=new String(str.getBytes("utf-8"),"8859_1");  
    return s;  
}
```

18.8 CRUD 처리하기

CRUD는 일반적으로 Create, Read, Update, Delete를 말하는 것이지만 database에서는 insert, select, update, delete 명령어를 이야기 한다고 볼 수 있다. 이 들 기능은 sql 문장을 사용하여 처리하는데 처리 절차는 아래와 같다. 기본 예시는 mysql을 기준으로 작성되어 있지만, 드라이버명만 오라클로 변경한다해도 소스는 그대로를 오라클상에서 테스트해 볼 수 있을 것이다.

- 1) 드라이버 등록
- 2) sql 문장 작성
- 3) PreparedStatement or Statement 생성
- 4) sql 문장 실행
- 5) sql 문장이 select문장이면 ResultSet 을 반환하고, insert, update, delete인 경우 적용된 행수를 반환

[고객테이블]

고객 테이블이 아래와 같이 환경에서 생성되어 있다.

계정 : hong

DB명 : lecture

```
CREATE TABLE custom(  
  id varchar(10),  
  name varchar(30),  
  phone varchar(20),  
  addr varchar(40),  
  point int(4)  
);
```

18.8.1 class 기본 골격

mysql 연결에 필요한 내용을 필드로 선언해 두고 생성자에서 드라이버를 메모리에 동적으로딩해 두었다.

```
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;
```

```
public class CRUD {
    Connection conn;
    PreparedStatement ps;
    ResultSet rs;

    String driver = "com.mysql.cj.jdbc.Driver";
    String path = "jdbc:mysql://localhost:3306/lecture";
    String user = "hong";
    String pwd = "1111";
    public CRUD() {

        try {
            Class.forName(driver);
            System.out.println("driver loading ok.....");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        CRUD crud = new CRUD();
    }
}
```

위코드를 실행하면 이클립스 콘솔창에 'driver loading ok' 문자열이 출력되면 mysql 드라이버가 정상적으로 메모리에 올려진 것이라 볼 수 있다.

18.8.2 Create(insert)

custom 테이블의 필드에 저장될 각각의 값을 지정하여 테이블에 저장하는 코드이다. create() 메서드에 기능을 분리하였다.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
```



```
import java.sql.ResultSet;
import java.sql.SQLException;

public class CRUD {
    ...
    public void create() {
        String id = "a001";
        String name="홍길동";
        String phone = "010-1234-1234";
        String addr = "korea";
        int point = 100;

        String sql = "insert into custom(id, name, phone, addr, point) "
            + " values(?,?,?,?,?)";

        try {
            conn = DriverManager.getConnection(path, user, pwd);
            conn.setAutoCommit(false);
            ps = conn.prepareStatement(sql);
            ps.setString(1, id);
            ps.setString(2, name);
            ps.setString(3, phone);
            ps.setString(4, addr);
            ps.setInt(5, point);

            int r = ps.executeUpdate();
            if(r>0) {
                conn.commit();
                System.out.println("저장됨");
            }else {
                System.out.println("저장중 오류 발생");
            }
            ps.close();
            conn.close();

        }catch(Exception ex) {
            try {
                conn.rollback();
            } catch (SQLException e) {
```

```
        e.printStackTrace();
    }
    ex.printStackTrace();
}

}

public static void main(String[] args) {
    CRUD crud = new CRUD();
    crud.create();
}
}
```

18.8.3 Read(select)

name 컬럼에 '길'이 들어간 모든 데이터를 찾아 콘솔에 출력하는 예이다.

```
public void read() {
    String find="%길%"; // '길'이 포함된 데이터
    String sql = "select * from custom where name like ? ";
    try {
        conn = DriverManager.getConnection(path, user, pwd);
        ps = conn.prepareStatement(sql);
        ps.setString(1, find);
        rs = ps.executeQuery();
        while(rs.next()) {
            System.out.println(rs.getString("id"));
            System.out.println(rs.getString("name"));
            System.out.println(rs.getString("phone"));
            System.out.println(rs.getString("addr"));
            System.out.println(rs.getInt("point"));
            System.out.println("-----");
        }
        rs.close();
    }
```

```
        ps.close();
        conn.close();

    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

18.8.4 Update(update)

id가 a001인 데이터의 point를 90으로 수정하는 예.

```
public void update() {
    String find="a001"; // id가 a001인 데이터
    String sql = "update custom set point=90 where id = ?";
    try {
        conn = DriverManager.getConnection(path, user, pwd);
        ps = conn.prepareStatement(sql);
        ps.setString(1, find);
        conn.setAutoCommit(false);
        int cnt = ps.executeUpdate();
        if(cnt>0) {
            System.out.println("수정됨");
            conn.commit();
        } else {
            System.out.println("수정중 오류 발생");
            conn.rollback();
        }
        ps.close();
        conn.close();

    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

18.8.5 Delete(delete)

id가 a001인 정보 삭제

```
public void delete() {
    String find="a001"; // id가 a001인 데이터
    String sql = "delete from custom where id = ?";
    try {
        conn = DriverManager.getConnection(path, user, pwd);
        ps = conn.prepareStatement(sql);
        ps.setString(1, find);
        conn.setAutoCommit(false);
        int cnt = ps.executeUpdate();
        if(cnt>0) {
            System.out.println("삭제됨");
            conn.commit();
        }else {
            System.out.println("삭제중 오류 발생");
            conn.rollback();
        }
        ps.close();
        conn.close();
    }catch(Exception ex) {
        ex.printStackTrace();
    }
}
```

19 입출력 스트림

19.1 스트림의 이해

스트림이란 하드 디스크나 네트워크상으로 데이터가 흘러 들어가고 나오는 것을 지칭한다.

1. 스트림은 FIFO(First In First Out)의 구조이다. 따라서 순차적 접근밖에 허용하지 않으며 스트림 내의 특정한 위치의 데이터를 무작위로 읽거나, 쓰는 것이 원칙적으로 금지되어 있다.
2. 스트림은 단방향이다. 읽기 스트림과 쓰기 스트림이 따로 존재한다.
3. 스트림은 지연될 수 있다. 스트림은 순차적으로 데이터가 처리되기 때문에 처리작업 하나가 무한 루프에 빠지면 뒤따라 데이터들 또한 무한히 기다려야 하는 상황이 되버린다.
4. 스트림은 유연하다. 스트림을 생성할 때 입출력을 실행하는 장치명만 바꾸어 주면 나머지 프로그램들은 거의 변경할 필요 없이 표준입출력장치, 파일장치, 네트워크장치 등등을 구현할 수 있다.

19.2 스트림의 구분

자바에 존재하는 기본적인 스트림만 약 40여개의 종류가 있다. 이를 분류하면

1. 스트림이 다루는 데이터형이 바이트와 문자로 구분하여 분류하며,
2. 단순한 입/출력인가, 데이터 조작인가에 따라 분류한다.
3. 바이트를 다루는 스트림 : `InputStream / OutputStream class`
4. 문자를 다루는 스트림 : `Reader / Writer class`
5. 보조 스트림 : `FileReader / FileWriter`

19.3 byte 스트림

입출력을 byte단위로 처리할 수 있는 기능을 갖고 있는 스트림이다. byte단위로 처리하기 때문에 한글등과

같은 문자의 처리는 정상적으로 처리되지 않는다.

19.3.1 InputStream

모든 입출력기능을 수행하기 위해 필요한 스트림이다. 대표적인 메서드는 아래와 같다.

메서드	기능
int read()	입력 스트림으로 부터 1 바이트를 읽어 반환
int read(byte[] b)	바이트를 읽어 배열에 저장하고 읽은 바이트의 갯수를 반환
int read(byte[], int offset, int length)	지정된 시작위치(offset)에서 길이(length) 만큼 읽어 배열에 저장하고, 읽은 크기를 반환
void close()	입력 스트림을 종료하고 자원을 반납한다.

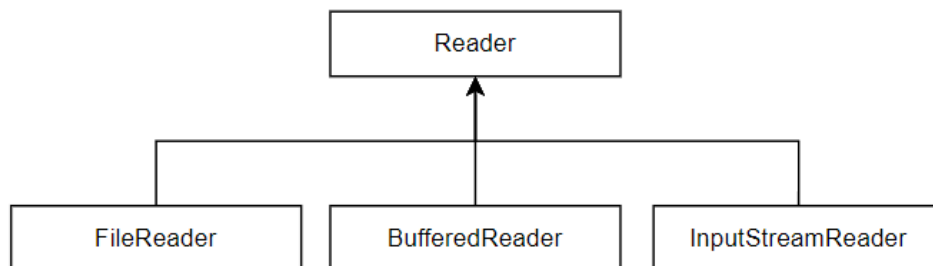
19.3.2 OutputStream

메서드	기능
void write(int b)	출력 스트림으로 1byte을 보낸다.
void write(byte[] b)	주어진 byte배열에 있는 값을 보낸다.
void write(byte[] b, int offset, int length)	byte 배열에 있는 값중 offset에서 부터 length만큼 보낸다.
void close()	출력 스트림을 닫는다.

19.4 문자기반 스트림

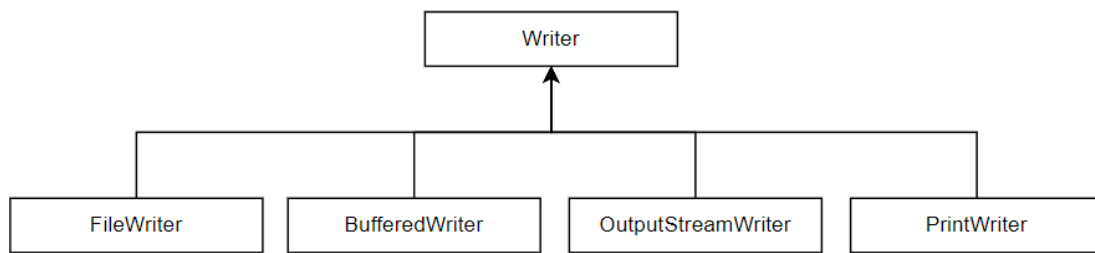
19.4.1 Reader

Reader 추상 클래스를 최상위 클래스로 사용한다.



메서드	기능
<code>int read()</code>	입력 스트림으로 부터 1 문자를 읽어 반환
<code>int read(char[] c)</code>	문자를 읽어 배열에 저장하고 읽은 문자의 갯수를 반환
<code>int read(char[], int offset, int length)</code>	지정된 시작위치(offset)에서 길이(length) 만큼 읽어 배열에 저장하고, 읽은 크기를 반환
<code>void close()</code>	입력 스트림을 종료하고 자원을 반납한다.

19.4.2 Writer



메서드	기능
<code>void write(int c)</code>	출력 스트림에 한문자를 출력
<code>void write(char[] buf)</code>	배열에 있는 모든 문자를 보냄
<code>void write(char[] buf, int off, int len)</code>	배열에서 off에서 부터 len개를 보냄
<code>void write(String s)</code>	문자열을 보냄
<code>void write(String s, int off, int len)</code>	문자열에서 off에서 부터 len개를 보냄
<code>void flush()</code>	버퍼에 있는 모든 문자를 보냄
<code>void close()</code>	출력 스트림을 닫음.

예) 변수에 저장된 문자열을 파일에 저장하고 메모장들을 사용하여 확인해 보자.

예) 텍스트 파일을 읽어 콘솔창에 출력해 보자.

```

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

public class FileStreamEx {
    String fileName = "info.txt";
    
```



```
public void write() {
    try {
        String data = "하이 adb 123 !@# 화이팅~";
        File file = new File(fileName);
        FileOutputStream fos = new FileOutputStream(file);
        OutputStreamWriter osw = new OutputStreamWriter(fos);
        osw.write(data);

        osw.close();
        fos.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

public void append() {
    try {
        String data = "하이 adb 123 !@# 화이팅~";
        File file = new File(fileName);
        FileOutputStream fos = new FileOutputStream(file, true);
        OutputStreamWriter osw = new OutputStreamWriter(fos);
        osw.write(data);

        osw.close();
        fos.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

public void read() {
    try {
        char[] readData = new char[1024];
        StringBuilder sb = new StringBuilder();
        File file = new File(fileName);
        FileInputStream fis = new FileInputStream(file);
        InputStreamReader isr = new InputStreamReader(fis);

        int len = -1;
```

```
        while( (len = isr.read(readData)) != -1) {
            String temp = new String(readData,0, len);
            sb.append(temp);
        }
        System.out.println(sb.toString());

        isr.close();
        fis.close();

    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

public static void main(String[] args) {
    FileStreamEx f = new FileStreamEx();
    //f.write();
    //f.append();
    f.read();
}
}
```

19.5 파일 입출력

문자 기반 스트림의 보조 스트림으로 파일 입출력에 사용되는 스트림이다.

- FileReader
- FileWriter

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
```

```
public class FileReadWriteEx {

    String fileName = "FileReadWriteText.txt";

    public void writer() {
        try {
            String data = "hi...\n하이...\n123\n!@#$";
            FileWriter fw = new FileWriter(fileName);
            fw.write(data);
            fw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void append() {
        try {
            String data = "\nhi...\n하이...\n123\n!@#$\n";
            FileWriter fw = new FileWriter(fileName, true);
            fw.write(data);
            fw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void read() {
        try{
            String data = "";
            FileReader fr = new FileReader(fileName);
            BufferedReader br = new BufferedReader(fr);

            while( (data = br.readLine() ) != null) {
                System.out.println(data);
            }

        }catch(Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

```

    }
}

public static void main(String[] args) {
    FileReaderWriterEx frw = new FileReaderWriterEx();
    //frw.writer();
    frw.append();
    frw.read();
}
}

```

19.6 Object 스트림

자바 클래스로 만들어진 객체를 송수신할 수 있는 스트림이다. Object 스트림을 사용하여 객체를 송수신 하려면 객체는 직렬화(Serializable)되어 있어야 한다. 또한 객체를 송수신할 때 직렬화와 역직렬화 과정을 거치는데 이 때 객체를 식별해 주어야 하는데 이 변수명이 serialVersionUID 이다. 이 변수를 클래스의 정적 필드로 추가해 주어야 한다.

메서드	기능
void writeObject(Object obj)	obj를 출력한다.
Object readObject()	Object를 읽어 반환한다.

예) 직렬화된 클래스를 구성하고 Object 스트림을 사용하여 파일에 저장하고 읽어 보자.

```

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;

```

```
public class ObjectOutputStreamEx {

    List<Data> list = new ArrayList<>();
    String fileName = "object.obj";

    public void write() {
        try {
            FileOutputStream fos = new FileOutputStream(fileName);
            ObjectOutputStream oos = new ObjectOutputStream(fos);

            Data d1 = new Data("a001", "kim", "korea", "010-1", 90);
            Data d2 = new Data("a002", "lee", "korea", "010-2", 80);
            Data d3 = new Data("a003", "park", "korea", "010-3", 70);

            list.add(d1);
            list.add(d2);
            list.add(d3);

            oos.writeObject(list);

            oos.close();
            fos.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public void read() {
        try {
            FileInputStream fis = new FileInputStream(fileName);
            ObjectInputStream ois = new ObjectInputStream(fis);

            List<Data> readData = (List<Data>) ois.readObject();

            for(Data d : readData) {
                System.out.println(d.toString());
            }
        }
    }
}
```

```
        ois.close();
        fis.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

public static void main(String[] args) {
    ObjectStreamEx ex = new ObjectStreamEx();
    ex.write();
    ex.read();
}
}
```

20 네트워크

자바는 언어의 설계 당시부터 이기종간의 네트워크를 할 수 있도록 고안된 언어이므로 네트워크 관련 프로그램을 작성할 때 매우 단순하고 편리하게 작업 할 수 있게 설계되어 있다.

네트워크 프로그램을 작성할때 기본이 되는 요소: IP주소, 포트번호, 프로토콜

IP 주소 : 마치 서버 컴퓨터의 전화 번호와 같은 역할을 하며 대부분이 이 IP번호로 서버의 주소를 외우기 힘들기 때문에 문자화되어 있는 주소를 사용한다. 이러한 문자화 된 주소를 Domain이라 부른다.

포트번호 : 하나의 컴퓨터에 여러 개의 통신 채널이 열려져 있을때 (예:웹, FTP, Mail, Telnet) 송수신되는 데이터들을 해당 서버들에 배분해야 하는데 이때 포트번호를 사용하여 데이터를 배분한다.

이러한 의미는 같은 웹서버를 연결하더라도 포트번호가 다르면 데이터는 송수신되지 않는다는 뜻이된다.

프로토콜 : 통신규약을 뜻한다. 프로토콜이 다르면 통신의 내용을 서로 알아 들을수 없다. 데이터를 송수신 하는 방법, 송수신된 데이터를 해석하는 방법등을 정의해 놓고 서로 이약속대로 H/W와 S/W를 만들게 된다. (http, ftp, telnet, tcp, ip, u에 ...)

그러나, 실제로 네트워크 프로그램을 작성하려면 위의 기본 개념아래에 다음과 같은 기법을 구사하여 프로그램 하여야 한다.

- 소켓
- 데이터 직렬화(객체 송수신)
- 스트림
- 쓰레드

20.1 TCP/IP

프로토콜에 관련된 이야기를 하자면 너무 방대하고 길기 때문에 현재 가장 많이 사용되어지고 있는 TCP와

UDP 프로토콜에 관한 이야기를 짧게 특성만 짚어보고 넘어가도록 하겠다.

20.1.1 TCP

TCP(Transport Control Protocol)의 약자이다. TCP는 대부분의 네트워크 서비스 애플리케이션들이 사용하고 있는 프로토콜이다.

20.1.2 TCP의 특징

- 1) 연결지향적 : 한번 연결하면 강제로 끊기전에는 연결된 상태로 유지된다.
- 2) 신뢰성 : 데이터를 전송한후 수신할 네트워크 시스템이 데이터를 정상적으로 수신했다는 신호를 주지 않으면 자동으로 데이터를 재전송한다. 또한 데이터를 전송한 순서대로 수신하지 않았더라고 수신된 데이터의 순서를 자동으로 재 배열하는 기능까지 가지고 있다.

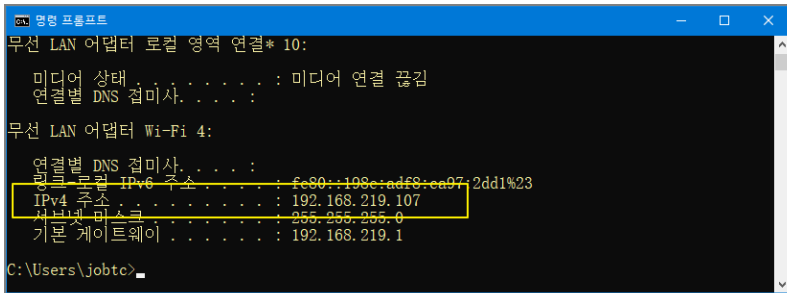
20.1.3 주요 클래스

클래스	설명
Socket	클라이언트 소켓으로 서버에 접속하기 위해 사용한다.
ServerSocket	소켓통신을 하기 위한 서버를 만든다. 클라이언트가 접속하게 되면 반환되는 클래스는 Socket이다.

20.1.4 IP

IP(Internet Protocol)의 약자로, NIC(Network Interface Card)당 하나의 IP가 할당되는데 이 IP는 컴퓨터는 네트워크상에서 하나의 컴퓨터를 구별해 주는 유일한 값을 갖게된다. 윈도우를 기준으로 네트워크가 연결된 컴퓨터에서 컴퓨터에 할당된 IP를 확인하려면 명령 프롬프트창을 열고 아래와 같은 명령을 실행하면 볼 수 있다.

ipconfig



* IPv4에 표시된 IP주소는 각자 다를 수 있다.

20.2 UDP

비연결 지향의 특징을 갖는 대표적인 프로토콜이며 멀티 미디어와 같은 대량의 정보를 전송할 때 유리하다. 전송 순서가 수신 수서와 다를 수 있기 때문에 이러한 조치는 별도로 해주어야 하며, 수신된 데이터에 오류가 있는 경우에도 재 전송 요청을 따로 해 주어야 한다.

20.2.1 udp와 tcp의 비교

UDP 프로토콜은 TCP 프로토콜과는 다르게 연결지향성이나 신뢰성이 없는 프로토콜이다. 따라서 데이터의 신뢰성 보다는 속도가 더 중요할 수 있는 멀티미디어 정보의 송수신, 게임, 채팅등의 분야에 보다 많이 사용되고 있다.

항 목	TCP	UDP
연결 지향성	O	X
신뢰성	O	X
데이터 정렬	O	X
Byte 단위 전송	O	X
양방향 전송	O	O
전송 속도	느림	빠름
대표적인서비스	터미널 애플레이터 FTP	DHCP, SNMP,

20.2.2 upd와 관련된 클래스

- DatagramSocket
- DatagramPacket

20.2.3 DatagramSocket 클래스

DatagramSocket은 일반 소켓과 달리 DatagramPacket형의 데이터만 송수신되며, 서버 소켓과 클라이언트 소켓의 구분이 없다. 즉 하나의 DatagramSocket을 사용하여 서버의 역할과 클라이언트의 역할을 구현한다.

또한 특이한 점은 DatagramSocket은 접속 설정 등의 작업 절차가 필요 없으며, 동시에 다른 주소와 포트로 데이터를 전송할 수 있다. 이것은 전송되려는 주소나 포트가 소켓의 연결정보에 있는 것이 아니라 패킷안에 포함되어 있기 때문이다.

udp 포트 번호는 tcp 포트번호와 독립적으로 사용되기 때문에 특정 포트 번호를 tcp가 사용하더라도 udp에서 그 번호를 사용할 수 있다.

주요 생성자	DatagramSocket()	
	DatagramSocket(int port)	
	DatagramSocket(int port, InetAddress addr)	
주요 메서드	void connect(InetAddress addr, int port)	지정된 주소와 포트를 사용하여 접속한다.
	void disconnect()	접속을 해제한다.
	void send(DatagramPacket p)	패킷을 전송한다.
	void receive(DatagramPacket p)	패킷을 수신한다.

20.2.4 DatagramPacket 클래스

DatagramPacket 클래스는 바이트 데이터를 보내기 위한 Wrapper 클래스이다. DatagramPacket은 송신과 수신은 형태가 서로 다르다. 수신시 만들어지는 DatagramPacket은 단지 데이터만을 받아 오기위한 형태로 만들어지며, 송신시에는 상대방의 주소와 포트 그리고 데이터를 담을 수 있는 형태로 만들어져야 한다.

주요 생성자	DatagramPacket(byte[] data, int length)	수신 할때
	DatagramPacket(byte[] data, int length, InetAddress addr, int port)	송신 할때
주요 메서드	void setAddress(InetAddress addr)	목적지 주소
	void setPort(int port)	목적지 포트
	void setData(byte buff[])	패킷의 실제 데이터
	void setLength(int leng)	데이터 길이

20.3 InetAddress

- 인터넷상의 IP주소를 객체 모델링한 클래스로 ip와 관련된 여러 정보를 가져온다.
- 생성자가 없다.
- 반드시 예외처리를 하여야 한다.

20.3.1 InetAddress 생성

```
try{
    InetAddress address = InetAddress.getByName(String str); // or
    InetAddress address = InetAddress.getLocalHost(String str);
}
catch(UnknownHostException e) { ... }
```

str 은 도메인명이나 IP 주소를 쓴다.

InetAddress.getByName() : 원격지의 InetAddress 정보를 가져온다.

InetAddress.getLocalHost() : 현재 프로그램이 실행되고 있는 컴퓨터의 InetAddress 정보를 가져온다.

20.3.2 주요 메소드

- public String getHostName() : 호스트 이름을 반환한다.
- public String getHostAddress() : ip주소를 문자열로 반환한다.

```
import java.net.InetAddress;
import java.net.UnknownHostException;

public class Net1 {

    public static void main(String[] args) {
        try{
            System.out.println("Local Host -----");
            InetAddress n1=InetAddress.getLocalHost();

            System.out.println( n1.getHostName() );
            System.out.println( n1.getHostAddress() );

        }
        catch (UnknownHostException e){}

    }
}
```

[실행결과]

```
Local Host -----
DESKTOP-CAK38TS
192.168.56.1
```

컴퓨터의 이름과 ip주소는 각자 다를 것이다.

20.4 URL (Uniform Resource Locator)

- URL 클래스와 입출력 스트림을 사용하면 URL로 표시되는 인터넷 자원을 쉽게 접근할 수 있다.
- 우리가 웹에서 일반적으로 사용하는 인터넷 주소는 아래와 같은 구조로 이루어져 있다.
- `http://www.jobtc.kr:8080/java/Begin#sub` 와 같은 주소가 아래와 같이 분해되어 설명될 수 있다.

프로토콜	서버	도메인	포트번호	파 일	섹 션
http://	www	jobtc.kr	8080	/java/Begin.java	#sub

20.4.1 URL 생성

```
try{
    URL url = new URL(String u);
    URL url = new URL(String protocol, String host, String file);
    URL url = new URL(String protocol, String host, int Port, String file) ;
}
catch(MalformedURLException ee){ ...}
```

20.4.2 주요 메서드

<code>public String getProtocol()</code>	프로토콜을 얻는다.
<code>public String getHost()</code>	호스트(서버의 종류)를 얻는다.
<code>public int getPort()</code>	포트번호를 얻는다.
<code>public String getFile()</code>	파일을 가져온다.
<code>public String getRef()</code>	섹션 번호를 가져온다.
<code>public InputStream openStream()</code>	지정된 URL과 접속이 자동으로 이루어지고 그 결과로

InputStream이 반환된다. 이 InputStream을 사용하여 해당 파일을 읽을 수 있다

20.5 소켓 프로그램 만들기

소켓에 대한 짧은 얘기

현재 소켓용 프로그램이다 라고 말하면 대부분이 TCP/IP 프로토콜을 사용하는 네트워크 프로그램을 지칭합니다. TCP/IP가 네트워크 프로그램의 표준화가 되었다는 말입니다. 네트워크로 데이터를 전송하면 먼저 데이터는 Packet이라는 작은 조각으로 나뉘어져 전송되고 송신하는 쪽은 수신된 Packet의 작은 조각들을 다시 복원하는 과정을 밟아야 합니다. 이러한 동작을 해 주는 것이 바로 TCP/IP 프로토콜입니다.

실제로 이러한 역할을 하는 단계는 매우 복잡한 단계를 거치게 되지만 소켓 API를 사용하게 되면 하위의 복잡한 프로토콜과 상관없이 유연하고 편하게 프로그램을 작성할 수 있습니다. 자바에서는 소켓작성에 필요한 함수들과 클래스를 java.net.Socket 패키지에 담아 제공합니다. 우리는 이 패키지를 사용하기 위해

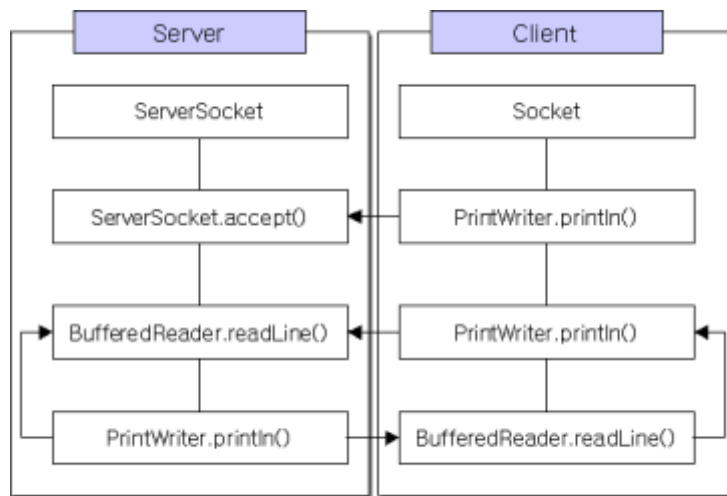
```
import java.net.*;
```

를 사용하여 소켓 뿐만 아니라 네트워크에 필요한 모든 클래스를 가져오기 위해 java.net.*를 import합니다.

클라이언트와 서버

소켓용 프로그램은 대부분이 클라이언트와 서버 개념으로 작성되어 집니다. 클라이언트는 접속을 시도하는 각 사용자를 뜻하고, 서버는 접속을 받아들이고 정보를 제공하는 컴퓨터나 사람을 뜻하죠. 쉽게 이야기 하면 이 글을 보고 계시는 여러분은 모두 클라이언트 입장에서 보시게 되는 거고, 저는 정보를 제공했기 때문에 서버의 입장이 되는 겁니다. 소켓용 프로그램은 거의 대부분이 클라이언트용과 서버용을 따로 만들어 서로 통신하게 하고 서버는 클라이언트들을 하나로 묶을 수 있는 기능이 있어 클라이언트들을 제어하고 통제하기도 합니다.

20.5.1 서버와 클라이언트 구조도



20.5.2 서버 소켓 송수신 절차

1) ServerSocket 클래스의 인스턴스를 사용하여 클라이언트로 부터의 접속을 기다리기 위해 accept()를 시도한다.

```
ServerSocket serverSocket = new ServerSocket(4444);
socket = serverSocket.accept();
```

2) 클라이언트로부터 데이터를 수신하기 위해 대기한다.(블록킹타입일 경우)

```
InputStream input = socket.getInputStream();
InputStreamReader reader = new InputStreamReader(input);
BufferedReader receive = new BufferedReader(reader);
String str = receive.readLine();
```

3) 수신된 정보를 클라이언트에게 송신한다.

```
OutputStream output = socket.getOutputStream();
OutputStreamWriter writer= new OutputStreamWriter(output); ;
PrintWriter send = new PrintWriter(writer,true);
send.println("메시지");
```

20.5.3 클라이언트 송수신 절차

1) 서버에 접속을 시도한다.(출력 스트림을 생성하는 즉시 해당 서버에 접속을 시도한다.)

```
Socket clientSocket = new Socket(getCodeBase().getHost(),4444);
OutputStream output = clientSocket.getOutputStream();
OutputStreamWriter writer= new OutputStreamWriter(output);
PrintWriter send = new PrintWriter(writer, true);
```

2) 서버로부터 데이터를 수신한다.

```
InputStream input = clientSocket.getInputStream();
InputStreamReader reader = new InputStreamReader(input);
BufferedReader receive = new BufferedReader(reader);
String str = receive.readLine();
```

3) 서버로 데이터를 송신한다.

```
send.println("메시지");
```

20.6 소켓 예제

20.7 애코 서버/클라이언트 만들기

20.7.1 애코 서버

```
import java.net.*;
import java.io.*;
```



```
public class EchoServer1{
    public static void main(String[] args){
        ServerSocket serverSocket = null;
        Socket      socket      = null;

        InputStream input;
        InputStreamReader reader;
        BufferedReader receive;

        try{
            System.out.println("서버시작...클라이언트 접속대기...");
            serverSocket = new ServerSocket(4444);
            socket = serverSocket.accept();

            input = socket.getInputStream();
            reader = new InputStreamReader(input);
            receive = new BufferedReader(reader);

            String str="";

            while(true){
                str=receive.readLine();

                System.out.println("receive:" + str);

                if(str.equals("end") ) {
                    break;
                }
            }

        }
        catch ( IOException e){
            e.printStackTrace();
        }
    }
}
```

```
}  
}
```

20.7.2 애코 클라이언트

```
import java.io.*;  
import java.net.*;  
  
public class SocketClientEx{  
  
    Socket clientSocket;  
  
    OutputStream    output;  
    OutputStreamWriter writer;  
    PrintWriter     send;  
  
    // keyBoard Input  
    InputStreamReader keyInput= new InputStreamReader(System.in);  
    BufferedReader   keyBuffer = new BufferedReader(keyInput);  
  
    public void connectProcess() {  
        try{  
            clientSocket = new Socket("localhost",4444);  
            output = clientSocket.getOutputStream();  
            writer  = new OutputStreamWriter(output);  
            send   = new PrintWriter(writer,true);  
  
            String str;  
            str="connect ....ok";  
            send.println(str);  

```

```
    }
    catch (Exception ex){
        System.out.println("연결 오류....");
        System.exit(0);
    }
}

public void sendProcess() {
    String str="";
    try{
        while(!str.equals("end")){
            System.out.print("input Send Data : ");
            str = keyBuffer.readLine();
            send.println(str);
            send.flush();
        }
    }
    catch (Exception e){
        e.printStackTrace();
    }
}

public static void main(String args[]){
    SocketClientEx ss=new SocketClientEx();
    ss.connectProcess();
    ss.sendProcess();
}
}
```

[실행결과]

서버 화면

클라이언트 화면

20.8 계산식 전달후 계산 결과 리턴 받기 예제

서버

```
import java.io.*;
import java.net.*;
public class HapServer {

    String msg = null;
    String array[] = null;

    InputStream is = null;
    InputStreamReader isr = null;
    BufferedReader br = null;

    OutputStream os = null;
    OutputStreamWriter osw = null;
    PrintWriter pw = null;

    ServerSocket server = null;
    Socket soc = null;

    public HapServer(){
        int a=0, b=0;
        double sum = 0;
        char op=' ';
```

```
System.out.println("Hap server start....");
try{
    server = new ServerSocket(12345);
    soc = server.accept();

    is = soc.getInputStream();
    isr = new InputStreamReader(is);
    br = new BufferedReader(isr);

    os = soc.getOutputStream();
    osw = new OutputStreamWriter(os);
    pw = new PrintWriter(osw, true);

    while(true){
        msg = br.readLine();
        if(msg.equals("exit")) break;
        array = msg.split("[+/,*, -]");
        a = Integer.parseInt(array[0]);
        b = Integer.parseInt(array[1]);

        if(msg.indexOf('+') > 0 ) {op='+';sum = a+b;}
        if(msg.indexOf('-') > 0 ) {op='-';sum = a-b;}
        if(msg.indexOf('/') > 0 ) {op='/';sum = a/(1.0*b);}
        if(msg.indexOf('*') > 0 ) {op='*';sum = a*b;}

        msg = a + " " + op + " " + b + " = " + sum;

        pw.println(msg);
        System.out.println(msg);
    }
}catch(Exception ex){
    System.out.println(ex.toString());
}
}

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

```
HapServer hs = new HapServer();
}
}
```

클라이언트

```
import java.io.*;
import java.net.*;

public class HapClient {
    String msg = null;

    InputStream is = null;
    InputStreamReader isr = null;
    BufferedReader br = null;

    OutputStream os = null;
    OutputStreamWriter osw = null;
    PrintWriter pw = null;

    Socket soc = null;

    InputStreamReader isrKey = null;
    BufferedReader brKey = null;

    public HapClient(){
        try{
            soc = new Socket("127.0.0.1", 12345);

            os = soc.getOutputStream();
            osw = new OutputStreamWriter(os);
            pw = new PrintWriter(osw, true);

            is = soc.getInputStream();
            isr = new InputStreamReader(is);
```

```
br = new BufferedReader(isr);

isrKey = new InputStreamReader(System.in);
brKey = new BufferedReader(isrKey);

String msg=null;
String receiveMsg=null;
while(true){
    System.out.print("msg(x,y) :");
    msg = brKey.readLine();

    pw.println(msg);
    if(msg.equals("exit")) break;
    receiveMsg = br.readLine();
    System.out.println(receiveMsg);
}

}catch(Exception ex){
    System.out.println(ex.toString());
}
}

public static void main(String args[]){
    HapClient dd = new HapClient();
}

}
```

[실행결과]

서버화면

클라이언트 화면

20.9 다중 접속 소켓 프로그램

[서버]

```
1.import java.io.*;
2.import java.net.*;
3.import java.util.*;
4.
5.public class ObjectServer {
6.    ServerSocket server = null;
7.    Socket socket = null;
8.
9.    Hashtable clientList = new Hashtable(10);
10.
11.    public ObjectServer(){
12.        try{
13.            System.out.println("object server start");
14.            server = new ServerSocket(12345);
15.            while(true){
16.                socket = server.accept(); //blocking mode
17.                System.out.println("object client connect");
18.
19.                Clients c = new Clients(socket);
20.            }
21.
22.        }catch(Exception ex){
23.            ex.printStackTrace();
24.        }
25.    }
26.
27.    // hashtable에 저장되어 있는 모든 유저 전달.
```



```
28. public void userListSend(Clients c) throws Exception{
29.     System.out.println("user list send");
30.     Enumeration e = clientList.keys();
31.     String userName=null;
32.     while(e.hasMoreElements()){
33.         userName = (String)e.nextElement();
34.         ObjectData data = new ObjectData("userList", userName,null);
35.         c.send.writeObject(data);
36.         c.send.flush();
37.     }
38. }
39.
40. // 새로운 사용자가 로그인 하였을때 새로운 사용자 전달.
41. public void userAddSend(ObjectData d) throws Exception{
42.     System.out.println("add user send");
43.     Enumeration e = clientList.elements();
44.     while(e.hasMoreElements()){
45.         Clients c = (Clients)e.nextElement();
46.         ObjectData data = new ObjectData("new",d.userName, d.msg);
47.         c.send.writeObject(data);
48.         c.send.flush();
49.     }
50. }
51.
52. // key -> hashtable의 키값.(사용자명)
53. // hashtable에 있는 client를
54. public void logoutProcess(ObjectData d) throws Exception{
55.     clientList.remove(d.userName);
56.     System.out.println("client cnt : " + clientList.size());
57.
58.     Enumeration e = clientList.elements();
59.     while(e.hasMoreElements()){
60.         Clients c = (Clients)e.nextElement();
61.         ObjectData data = new ObjectData("exit",d.userName,null);
62.         c.send.writeObject(data);
63.         c.send.flush();
```

```
64.     }
65.
66.   }
67.
68.   public void sendMsgProcess(ObjectData data) throws Exception{
69.       System.out.println(data.msg);
70.
71.       Enumeration e = clientList.elements();
72.       while(e.hasMoreElements()){
73.           Clients c = (Clients)e.nextElement();
74.           c.send.writeObject(data);
75.           c.send.flush();
76.       }
77.
78.   }
79.
80.   // 접속된 클라이언트들의 Stream저장.
81.   class Clients extends Thread{
82.       Socket soc = null;
83.       InputStream is = null;
84.       ObjectInputStream receive = null;
85.
86.       OutputStream os = null;
87.       ObjectOutputStream send = null;
88.       boolean threadFlag = true;
89.
90.
91.       public Clients(Socket s){
92.           try{
93.
94.               soc = s;
95.
96.               is = soc.getInputStream();
97.               receive = new ObjectInputStream(is);
98.
99.
```

```
100.         os = soc.getOutputStream();
101.         send = new ObjectOutputStream(os);
102.
103.         start();
104.
105.     }catch(Exception ex){
106.         System.out.println("thread start error \n" );
107.         ex.printStackTrace();
108.     }
109. }
110.
111. public void run(){ // 무한 순환하면서 데이터 수신
112.     System.out.println("data receive...");
113.     try{
114.         while(threadFlag){
115.             ObjectData d = (ObjectData)receive.readObject();
116.
117.             if(d.command.equals("login")){
118.                 userListSend(this);
119.                 System.out.println("user name" + d.userName);
120.                 clientList.put(d.userName,this);
121.                 System.out.println("client cnt :" + clientList.size());
122.
123.                 userAddSend(d);
124.             }
125.             // 접속종료.
126.             if(d.command.equals("exit")){
127.                 logoutProcess(d);
128.                 Thread.sleep(100);
129.                 //threadFlag=false;
130.             }
131.             if(d.command.equals("msg")) sendMsgProcess(d);
132.
133.             Thread.sleep(500);
134.         }
135.     }catch(Exception ex){
```

```
136.         ex.printStackTrace();
137.     }
138. }
139.
140. }
141.
142. public static void main(String args[]){
143.     ObjectServer os = new ObjectServer();
144.
145. }
146.
147.
148.}
```

[클라이언트]

```
1.import java.awt.*;
2.import java.io.*;
3.import java.net.*;
4.import java.awt.event.*;
5.import java.awt.Dimension;
6.import java.awt.Rectangle;
7.import java.awt.TextField;
8.import java.awt.List;
9.import java.awt.Label;
10.
11.public class ObjectClient extends Frame implements Runnable{
12.
13.     private static final long serialVersionUID = 1L;
14.     private Label label = null;
15.     private TextField serverIP = null;
16.     private Button connectBtn = null;
17.     private Button disconnectBtn = null;
18.     private TextArea doc = null;
```

```
19. private TextField sendMsg = null;
20. private Button sendBtn = null;
21.
22. private Socket soc = null;
23. private OutputStream os = null; // @jve:decl-index=0:
24. private ObjectOutputStream send = null;
25.
26. private InputStream is = null;
27. private ObjectInputStream receive = null;
28.
29. private TextField userName = null;
30. private List userList = null;
31. private Label label1 = null;
32.
33. // user add member
34. boolean threadFlag = true;
35. /**
36.  * This is the default constructor
37.  */
38. public ObjectClient() {
39.     super();
40.     initialize();
41. }
42.
43. /**
44.  * This method initializes this
45.  *
46.  * @return void
47.  */
48. private void initialize() {
49.     label1 = new Label();
50.     label1.setBounds(new Rectangle(247, 40, 62, 28));
51.     label1.setText("사용자명");
52.     label = new Label();
53.     label.setBounds(new Rectangle(5, 36, 60, 42));
54.     label.setText("연결서버");
```

```
55.    this.setLayout(null);
56.    this.setSize(628, 350);
57.    this.setTitle("Frame");
58.
59.    this.add(label, null);
60.    this.add(getServerIP(), null);
61.    this.add(getConnectBtn(), null);
62.    this.add(getDisConnectBtn(), null);
63.    this.add(getDoc(), null);
64.    this.add(getSendMsg(), null);
65.    this.add(getSendBtn(), null);
66.    this.add(getUserName(), null);
67.    this.add(getUserList(), null);
68.    this.add(label1, null);
69.    this.addWindowListener(new java.awt.event.WindowAdapter() {
70.        public void windowClosing(java.awt.event.WindowEvent e) {
71.            disconnectProcess();
72.            System.exit(0);
73.        }
74.    });
75. }
76.
77. /**
78.  * This method initializes serverIP
79.  *
80.  * @return java.awt.TextField
81.  */
82. private TextField getServerIP() {
83.     if (serverIP == null) {
84.         serverIP = new TextField();
85.         serverIP.setText("192.168.123.199");
86.         serverIP.setBounds(new Rectangle(68, 39, 161, 28));
87.         serverIP.addKeyListener(new java.awt.event.KeyAdapter() {
88.             public void keyPressed(java.awt.event.KeyEvent e) {
89.                 if(e.getKeyCode() == KeyEvent.VK_ENTER) connectProcess();
90.             }
91.         });
92.     }
93. }
```

```
91.     });
92.   }
93.   return serverIP;
94. }
95.
96. /**
97.  * This method initializes connectBtn
98.  *
99.  * @return java.awt.Button
100. */
101. private Button getConnectBtn() {
102.   if (connectBtn == null) {
103.     connectBtn = new Button();
104.     connectBtn.setBounds(new Rectangle(425, 39, 82, 26));
105.     connectBtn.setLabel("서버연결");
106.     connectBtn.addActionListener(new java.awt.event.ActionListener() {
107.       public void actionPerformed(java.awt.event.ActionEvent e) {
108.         if(serverIP.getText() != null){
109.           connectProcess();
110.         }
111.       }
112.     });
113.   }
114.   return connectBtn;
115. }
116.
117. /**
118.  * This method initializes disConnectBtn
119.  *
120.  * @return java.awt.Button
121. */
122. private Button getDisConnectBtn() {
123.   if (disConnectBtn == null) {
124.     disConnectBtn = new Button();
125.     disConnectBtn.setBounds(new Rectangle(515, 39, 80, 27));
126.     disConnectBtn.setLabel("연결해제");
```

```

127.     disconnectBtn.addActionListener(new java.awt.event.ActionListener() {
128.         public void actionPerformed(java.awt.event.ActionEvent e) {
129.             disconnectProcess();
130.         }
131.     });
132. }
133. return disconnectBtn;
134. }
135.
136. /**
137.  * This method initializes doc
138.  *
139.  * @return java.awt.TextArea
140.  */
141. private TextArea getDoc() {
142.     if (doc == null) {
143.         doc = new TextArea();
144.         doc.setBounds(new Rectangle(159, 89, 448, 207));
145.     }
146.     return doc;
147. }
148.
149. /**
150.  * This method initializes sendMsg
151.  *
152.  * @return java.awt.TextField
153.  */
154. private TextField getSendMsg() {
155.     if (sendMsg == null) {
156.         sendMsg = new TextField();
157.         sendMsg.setBounds(new Rectangle(32, 305, 493, 26));
158.         sendMsg.addKeyListener(new java.awt.event.KeyAdapter() {
159.             public void keyPressed(java.awt.event.KeyEvent e) {
160.                 if(e.getKeyCode() == KeyEvent.VK_ENTER) sendMsgProcess();
161.             }
162.         });

```



```

163.     }
164.     return sendMsg;
165. }
166.
167. /**
168.  * This method initializes sendBtn
169.  *
170.  * @return java.awt.Button
171.  */
172. private Button getSendBtn() {
173.     if (sendBtn == null) {
174.         sendBtn = new Button();
175.         sendBtn.setBounds(new Rectangle(529, 305, 78, 29));
176.         sendBtn.setLabel("전송");
177.         sendBtn.addActionListener(new java.awt.event.ActionListener() {
178.             public void actionPerformed(java.awt.event.ActionEvent e) {
179.                 sendMsgProcess();
180.             }
181.         });
182.     }
183.     return sendBtn;
184. }
185.
186. public void connectProcess(){
187.     try{
188.         soc = new Socket(serverIP.getText(), 12345);
189.
190.         os = soc.getOutputStream();
191.         send = new ObjectOutputStream(os);
192.
193.         is = soc.getInputStream();
194.         receive = new ObjectInputStream(is);
195.
196.         userList.removeAll();
197.         ObjectData data = new ObjectData("login",userName.getText(), " 님이 입장");;
198.

```

```
199.     threadFlag = true;
200.     Thread t = new Thread(this);
201.
202.     send.writeObject(data);
203.     send.flush();
204.
205.
206.     t.start();
207.
208. }catch(Exception ex){System.out.println("error 2 : " + ex.toString());}
209. }
210.
211. public void sendMsgProcess(){
212.     try{
213.         ObjectData data = new ObjectData("msg",userName.getText(), sendMsg.getText());
214.         send.writeObject(data);
215.         send.flush();
216.         System.out.println("send msg");
217.     }catch(Exception ex){
218.         System.out.println("send error \n" + ex.toString());
219.     }
220.
221. }
222.
223. public void disconnectProcess(){
224.     try{
225.         if(soc != null){
226.             ObjectData data = new ObjectData("exit", userName.getText(), null);
227.             send.writeObject(data);
228.
229.             send.close();
230.             receive.close();
231.
232.             is.close();
233.             os.close();
234.
```

```
235.         userList.removeAll();
236.
237.         Thread.sleep(100);
238.         threadFlag = false;
239.
240.     }
241. }catch(Exception ex){}
242. }
243.
244. public void run(){
245.     System.out.println("thead start...");
246.
247.     try{
248.         while(threadFlag){
249.             ObjectData data = (ObjectData)receive.readObject();
250.
251.             if(data.command.equals("userList")){
252.                 userList.add(data.userName);// 사용자 이름
253.             }
254.             else if(data.command.equals("new")){
255.                 userList.add(data.userName);// 사용자 이름
256.                 doc.append(data.userName + " " + data.msg + "\r\n");
257.             }
258.             else if(data.command.equals("msg")){
259.                 doc.append(data.userName + ":" + data.msg);
260.                 doc.append("\n");
261.                 Thread.sleep(500);
262.             }
263.             else if(data.command.equals("exit")){
264.                 doc.append(data.userName + " 님이 로그아웃하셨습니다\r\n");
265.                 userList.remove(data.userName);
266.             }
267.
268.         }
269.     }catch(Exception ex){System.out.println("error 1" + ex.toString());}
270. }
```

```
271.
272.  /**
273.   * This method initializes userName
274.   *
275.   * @return java.awt.TextField
276.   */
277.  private TextField getUserName() {
278.      if (userName == null) {
279.          userName = new TextField();
280.          userName.setBounds(new Rectangle(313, 39, 104, 29));
281.      }
282.      return userName;
283.  }
284.
285.  /**
286.   * This method initializes userList
287.   *
288.   * @return java.awt.List
289.   */
290.  private List getUserList() {
291.      if (userList == null) {
292.          userList = new List();
293.          userList.setBounds(new Rectangle(30, 89, 123, 207));
294.      }
295.      return userList;
296.  }
297.
298.  public static void main(String args[]){
299.      ObjectClient ct2 = new ObjectClient();
300.      ct2.setVisible(true);
301.  }
302.
303.
304.} // @jve:decl-index=0:visual-constraint="64,23"
```

21 JSON 데이터 처리

데이터를 구조화하기 위해 XML을 사용했었으나 이젠 JSON을 사용하는것이 보편화된것 같다. JSON은 JavaScript Object Notation 의 약자로 서로 다른 프로그래밍 언어 간에 데이터를 교환하기 위한 표기법으로 읽고 쓰기 쉽도록 고안해낸 데이터 표기법이다.

21.1 표현 방법

대부분의 언어에서 친숙하게 사용되고 있는 구조로 이루어져 있으며 크게 2가지의 표현 방법을 사용하고 있다. 데이터 값들은 기본적으로 문자열이어야 한다.

- { 이름:값, ... }
- [배열]
- 혼합형

예) {key:value }

```
{ "id" : "a001", "mName" : "hong", "addr" : "대한민국", "phone": "010-1", "point" : 100 }
```

예) 배열

```
[ "송아지", "망아지", "고양이", "강아지" ]
```

예) 혼합형

```
{ "mnt" : [ "백두산", "금강산", "한라산" ] }  
[ { "id" : "a001", "mName" : "kim" }, { "id" : "a002", "mName" : "lee" } ]
```

JSON을 처리하기 위한 여러 가지 라이브러리들이 존재한다. 그중에서 JSON.simple 라이브러리를 사용하는 방법에 대해 알아 보자.

21.2 JSON.simple library

[라이브러리 설치]

mvnrepository.com 사이트를 방문하여 'json simple'로 검색하여 관련 라이브러리를 다운로드 받는다.

[주요 클래스]

클래스명	설명
JSONObject	JSON 객체를 Map 형태로 처리
JSONArray	JSON 객체를 List 형태로 처리
JSONParser	JSON 데이터를 분석하기 위한 클래스
JSONValue	JSON 데이터를 다루기 위한 클래스

[특징]

- 숫자형은 Long 타입으로 변환된다.
- JSON 데이터를 처리하기 위해 Map과 List 컬렉션을 사용한다.
- 파일에 저장할 수 있는 기능을 제공한다.
- 처리 속도는 대용량이나 다량에서 적당한 성능을 발휘한다.

[JSON 문자열을 Java Object로 변환]

```
// -----  
out("1. json array to json object");  
  
String jsonInt = "[1,2,3,4,5]";  
JSONArray jArray = (JSONArray)jParser.parse(jsonInt);  
out("json int", jArray.toJSONString());  
for(int i=0 ; i<jArray.size() ; i++) {
```

```
    Long s = (Long)jArray.get(i);
    out(i, s);
}

// -----
out("2. json string to json object");
String jsonString = "[" + "'a','b','c', 'd', 'e'"]";
jsonString = jsonString.replaceAll("'", "\\'"); //편의상
jArray = (JSONArray)jParser.parse(jsonString);
out("json string", jArray.toJSONString());
for(int i=0 ; i<jArray.size() ; i++) {
    String s = (String)jArray.get(i);
    out(i, s);
}

// -----
out("3. json map string to json object");
String jsonMap = "{" + "'name': 'hong', 'age': '20'" + "}";
jsonMap = jsonMap.replaceAll("'", "\\'");
jObject = (JSONObject)jParser.parse(jsonMap);
out("json map " , jObject.toJSONString());
out("name " , jObject.get("name"));
out("age", jObject.get("age"));

// -----
out("4. json map list to json object");

String jsonMapList = "[" + "{" + "'name': 'hong' , 'addr': 'seoul'" + "}, {" + "'name' : 'kim', 'addr' : 'busan'" + "}]";
jsonMapList = jsonMapList.replaceAll("'", "\\'");

jArray = (JSONArray)jParser.parse(jsonMapList);
out("json map list string : " + jArray.toJSONString() );
for(Object o : jArray){
    JSONObject job = (JSONObject) o;
    out("name" , job.get("name"));
    out("addr" , job.get("addr"));
    out("---", "");
}
}
```



```
// -----
out("5. json map{<K : []} to json object");

jsonString = "{ 'names' : ['a','b','c','d'] }";
jsonString = jsonString.replaceAll("'", "\"");
jObject = (JSONObject) jParser.parse(jsonString);
out("json{k:[]}", jsonString);
jArray = (JSONArray)jObject.get("names");
for(int i=0 ; i<jArray.size() ; i++){
    out(i, jArray.get(i));
}

// -----
out("6. json map 2 :  [{k:[]}, {k:[]}]");
jsonString = "[{ 'names' : ['a','b','c'] }, { 'ages' : [7,6,5,4,3,2] }]";
jsonString = jsonString.replaceAll("'", "\"");
out("<br/>json string : ",jsonString);

jArray = (JSONArray)jParser.parse(jsonString);

for(int i=0 ; i<jArray.size() ; i++) {
    jObject = (JSONObject)jArray.get(i);
    Iterator iter= jObject.keySet().iterator();
    while(iter.hasNext()) {
        String key = (String)iter.next();
        out("key ", key);
        JSONArray values = (JSONArray)jObject.get(key);
        for(int j=0 ; j<values.size() ; j++) {
            out("values", values.get(j));
        }
    }
}
}
```

[실행결과]

```
1. json array to json object :
json int : [1,2,3,4,5]
0 : 1
```

```
1 : 2
2 : 3
3 : 4
4 : 5

2. json string to json object :
json string : ["a","b","c","d","e"]
0 : a
1 : b
2 : c
3 : d
4 : e

3. json map string to json object :
json map   : {"name":"hong","age":"20"}
name      : hong
age       : 20

4. json map list to json object :

json map list string : [{"name":"hong","addr":"seoul"},
{"name":"kim","addr":"busan"}] :
name : hong
addr : seoul
--- :
name : kim
addr : busan
--- :

5. json map{<K : []} to json object :
json{k:[]} : {"names" : ["a","b","c","d"]}
0 : a
1 : b
2 : c
```

```
3 : d

6. json map 2 : [{k:[]}, {k:[]} ] :
<br/>json string : : [{"names" : ["a","b","c"]}, {"ages" : [7,6,5,4,3,2]}]
key : names
values : a
values : b
values : c
key : ages
values : 7
values : 6
values : 5
values : 4
values : 3
values : 2
```

[Java Object를 JSON 문자열로 변환]

```
// json object to key:value
JSONObject jobj = new JSONObject();
jobj.put("id", "a001");
jobj.put("mName", "hong");
jobj.put("addr", "대한민국");

String jsonString = jobj.toJSONString();
System.out.println(jsonString);

// json object to array
JSONArray jarray = new JSONArray();
jarray.add("kim");
jarray.add("lee");
jarray.add("park");
jarray.add("hong");

jsonString = jarray.toJSONString();
```

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

jarray = new JSONArray();
jarray.add(jobj);
jarray.add(jobj);
jarray.add(jobj);
jarray.add(jobj);
jsonString = jarray.toJSONString();
System.out.println(jsonString);

// json object to {key : [values]}
jobj = new JSONObject();
jobj.put("names", jarray);

jsonString = jobj.toJSONString();
System.out.println(jsonString);
```

[실행결과]

```
[{"id":"a001","mName":"hong","addr":"대한민국"}
["kim","lee","park","hong"]
[{"id":"a001","mName":"hong","addr":"대한민국"},{"id":"a001","mName":"hong","addr":"대한민국"},
{"id":"a001","mName":"hong","addr":"대한민국"},{"id":"a001","mName":"hong","addr":"대한민국"}]
{"names":[{"id":"a001","mName":"hong","addr":"대한민국"},
{"id":"a001","mName":"hong","addr":"대한민국"},{"id":"a001","mName":"hong","addr":"대한민국"},
{"id":"a001","mName":"hong","addr":"대한민국"}]}
```

22 TIP

22.1 이클립스에서 MySQL 연결하기

다양한 Database 클라이언트 유무료 툴들이 존재하지만, 간단한 쿼리 작업등을 하기 위해 굳이 별도의 클라이언트 툴을 설치하기 싫다면 이클립스 기능 중 하나인 Data Management를 사용하여 데이터베이스 연결하고 쿼리를 실행할 수 있습니다. 현재 사용하고 있는 대부분의 DBMS를 연결할 수 있지만, 본 지면에서는 MySQL을 예로 들어 보겠습니다.

물론 사용하려는 MySQL은 로컬이든 원격지든 설치되어 있고, 사용권한이 있는 계정은 있어야 합니다. 또한 사용하려는 버전의 JDBC 드라이버도 이클립스를 사용하고 있는 로컬 컴퓨터에 다운로드되어 있어야 합니다.

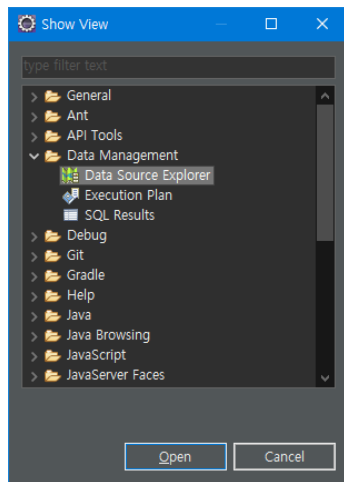
따라서 본 지면에서는 JDBC 다운로드 부터 진행하도록 하겠습니다.

22.1.1 JDBC 드라이버 다운 로드

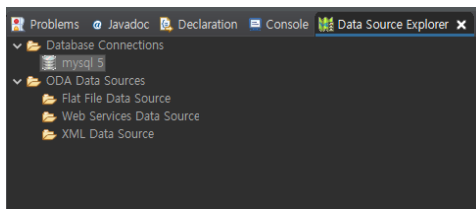
다운로드 및 설치는 MySQL 드라이버 등록하기를 참고하자.

22.1.2 Connection 작업

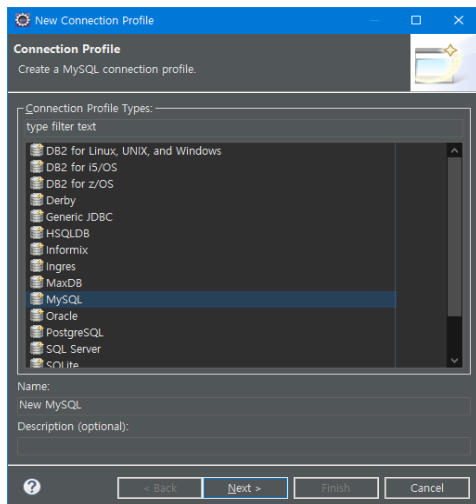
이클립스에서 Database를 연결하려면 JDBC 드라이버 파일이 필요하다.



Window>Show View > Other>Data Management >Data Source Explorer

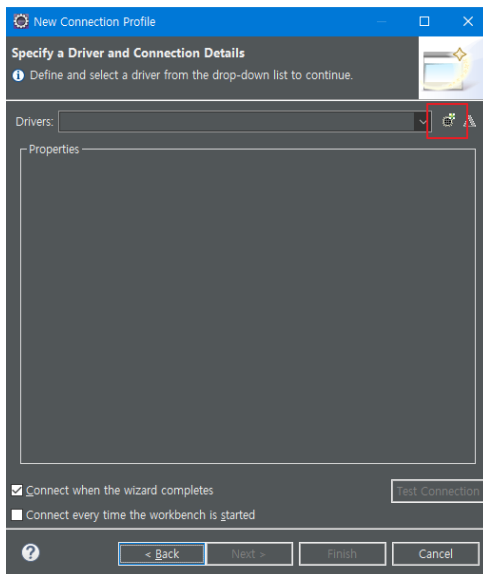


뷰가 새롭게 추가되며, Database Connections에 마우스 우클릭한다.

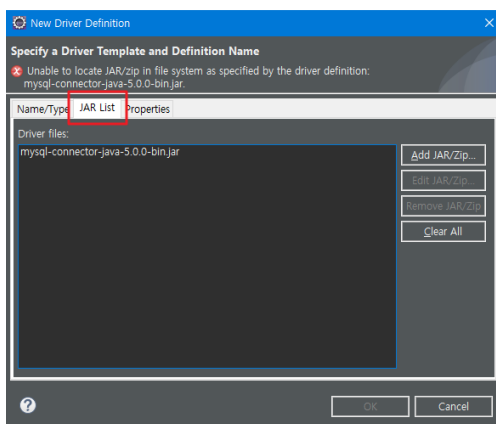


사용하려는 Database 종류를 선택하여 아이템을 선택한다.

자바의 개요 22.1 이클립스에서 MySQL 연결하기

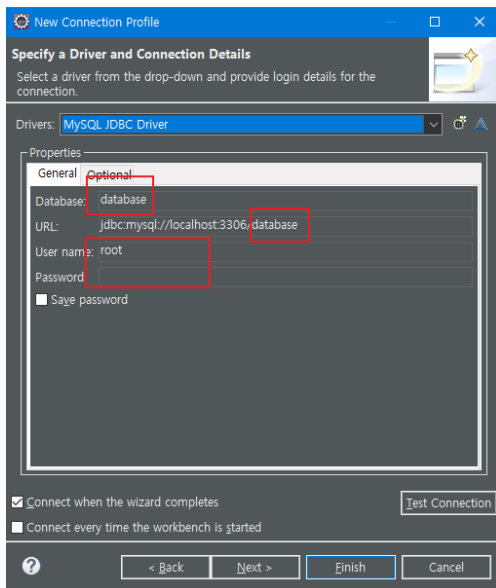


Database에 따른 드라이버 파일이 필요하다. 붉은 사각형 영역을 클릭하여 드라이버를 선택해야 한다.



대부분 사용자가 다운로드한 드라이버가 첫번째 탭인 Name/Type에 없을 것이다. 두번째 탭인 JAR List로 이동하여 다운로드한 JDBC 드라이버를 선택한다.

자바의 개요 22.1 이클립스에서 MySQL 연결하기



붉은 사각형 영역을 사용하려는 데이터 베이스에 맞게 수정한다.

ex) root 사용자인 경우

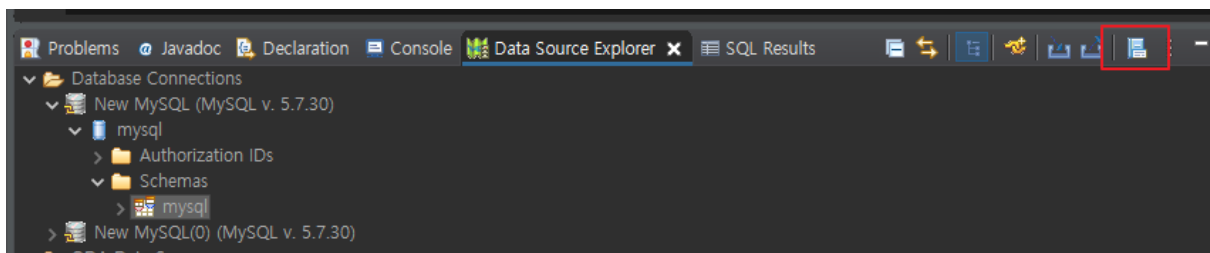
Database : **mysql**

URL : jdbc:mysql://localhost:3306/**mysql**

User Name : **root**

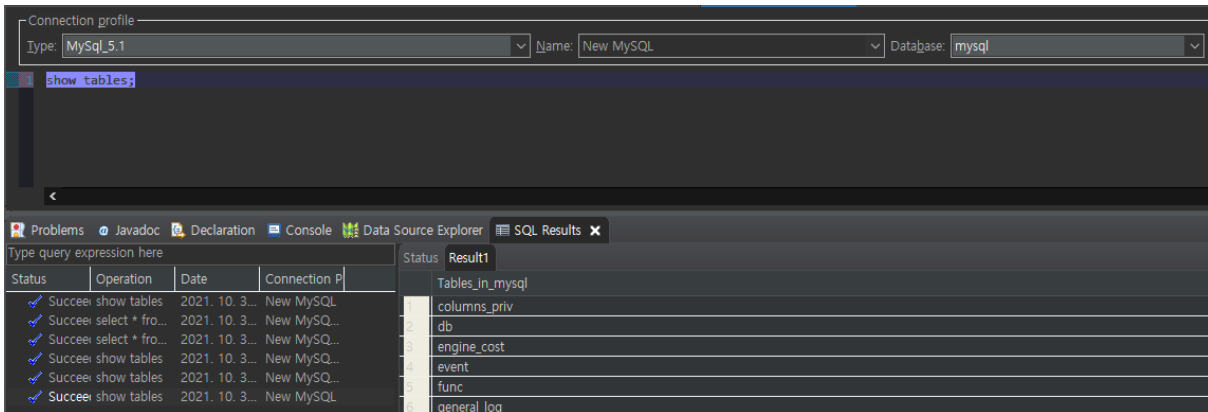
Password : **XXX**

22.1.3 SQL 실행하기



아이콘을 클릭하면 아래와 같은 창이 뜨는데 실행할 쿼리를 입력한 뒤 alt+s 키를 누르면 결과창이 나타난다. 쿼리문에서 마우스 우클릭하면 보다 상세한 옵션 메뉴들을 볼 수 있다.

자바의 개요 22.1 이클립스에서 MySQL 연결하기



22.2 Swing에서 Control key 조합

```
textArea.addKeyListener(new KeyAdapter() {  
    @Override  
    public void keyPressed(KeyEvent e) {  
        if( e.getKeyCode() != KeyEvent.VK_ENTER || e.getModifiers() !=  
        KeyEvent.CTRL_MASK ) return;  
        ...  
    }  
})
```

위와 같이 Control키는 KeyEvent의 getModifiers() 메서드와 KeyEvent.CTRL_MASK를 사용하여 인식하게 한다.

22.3 Linux 에서 MySQL JDBC 드라이버 설치하기

자바는 /usr/local/java에 설치되었다고 가정한다.

1) 드라이버 다운로드 www.mysql.com 에 접속하여 적당한 드라이버를 다운로드한다. 파일명은 mysql-connector-java-버전-stable.tar.gz 와 유사할 것이다.

- 2) 다운로드한 파일의 압축을 해제하고 `tar xvfz mysql-connector-java-버전-stable.tar.gz`
- 3) 압축 해제된 디렉토리 안의 파일 중 `mysql-connector-java-버전-stable-bin.jar` 파일을 `/usr/local/java/lib` 디렉토리에 복사한다. (다른 파일은 복사하지 않아도 된다)
- 4) `/etc/profile` 에 “CLASSPATH”를 편집하거나 추가한다. `export CLASSPATH=./usr/local/java/lib/mysql-connector-java-버전-stable-bin.jar`
- 5) CLASSPATH를 확인한다. `$ source profile $ echo $CLASSPATH`

22.4 SQLite RDBMS 사용하기

임베디드용 DB중 하나이며 안드로이드, ios에서 공통으로 사용할 수 있는 DB이다. 서버 개념이 없으며 원격 연결은 지원하지 않는다. 라이브러리가 C로 만들어져 있기 때문에 굉장히 다양한 언어를 지원하고 있다.

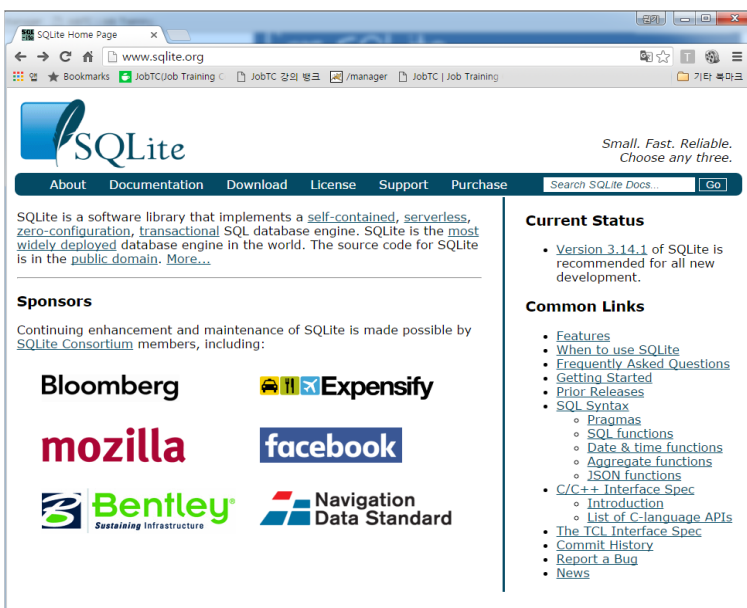
관련 파일 다운로드

Sqlite DB : <http://www.sqlite.org/>

GUI tool : <http://sqlitebrowser.org/>

JDBC Driver : <https://bitbucket.org/xerial/sqlite-jdbc/downloads>

22.4.1 Sqlite 설치



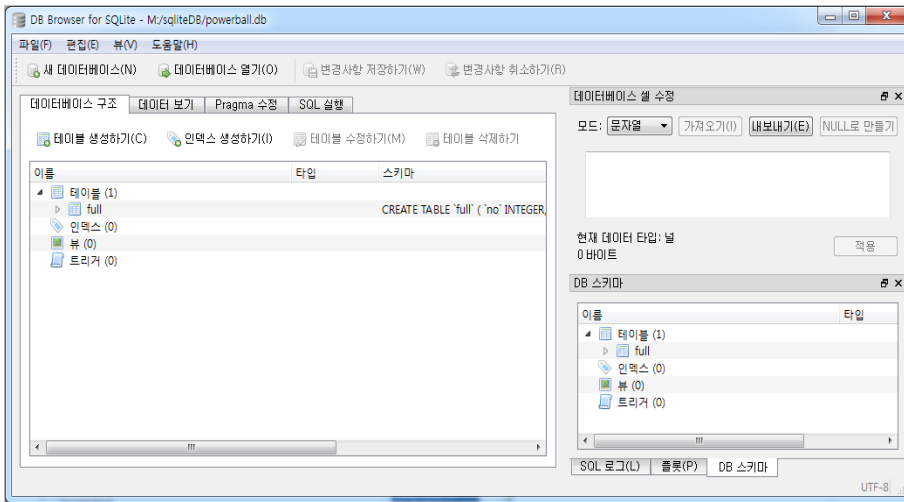
사용자의 컴퓨터 환경에 맞는 파일을 다운로드 받아 적당한 위치에 압축을 해제 하는 것으로 설치는 완료 된다.

22.4.2 GUI tool 설치



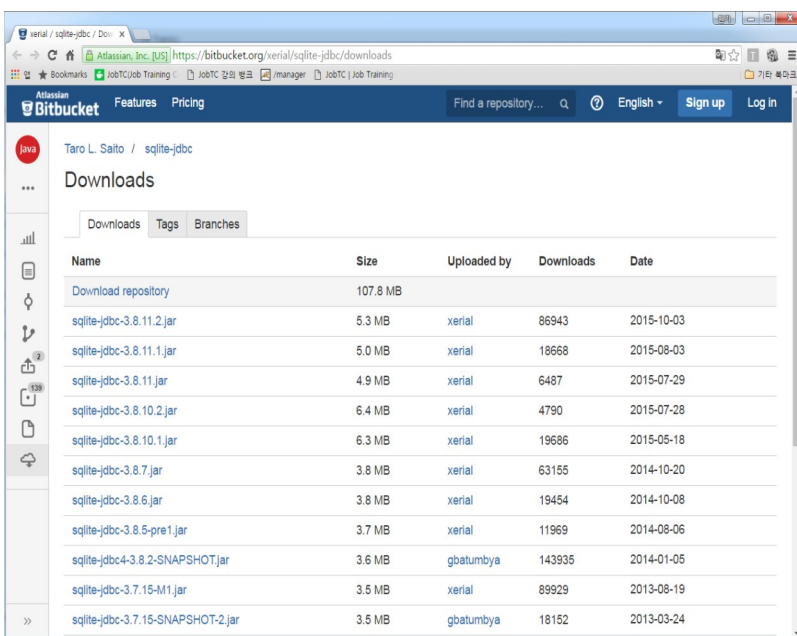
GUI Tool 실행 화면

자바의 개요 22.4 SQLite RDBMS 사용하기



22.4.3 JDBC Driver

일반적으로 드라이버는 자바 설치 경로의 ext에 복사해 두면 여러모로 편리하게 사용된다. 따라서 Sqlite JDBC Driver도 ext 경로에 복사해 두자.



22.4.4 DB 및 Table 생성

위에서 설치한 DB Browser for Sqlite를 사용하여 DB와 Table을 생성해 보자.

새 데이터베이스 메뉴를 선택하면 경로와 파일명을 지정하게 되는데 이때 파일명이 데이터 베이스명이 되며, 확장자는 자동으로 '.db'가 붙게 된다.