# 2. OOP 와 Java 프로그램의 구조

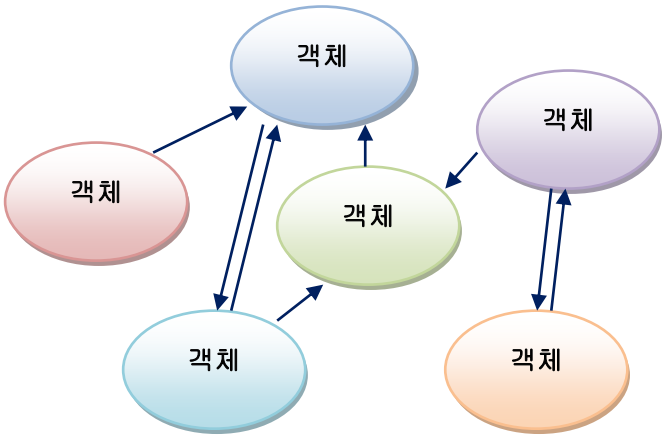
객체지향 프로그래밍(Object-Oriented Programming, OOP) 은 컴퓨터 프로그램을 명령어의 목록으로 보는 시각에서 벗어나 여러 개의 독립된 단위, 즉 "객체"들의 모임으로 파악하려는 것이다. Java는 객체지향 프로그래밍언어로 분류되며 Java라는 언어로 프로그램을 개발한다는 것은 프로그램에서 요구되는 단위로 객체들을 만드는 것을 의미하는데 이러한 객체들에 대하여 데이터와 행위로 나누어 설계한 것이 바로 클래스이다. 그리고 이 클래스라는 것은 Java 프로그램의 기본 구조이다.

## 2.1 객체와 클래스

Java 프로그래밍 언어의 구문과 구조를 익히기 전에 객체와 클래스에 대한 이론적인 지식을 파악하는 것은 객체지향 프로그래밍 언어인 Java를 파악하는데 있어서 많은 도움이 된다. 프로그램에서의 객체에 대한 정의와 객체를 만들기 위한 설계도인 클래스 그리고 객체지향 프로그램의 특징 등에 대하여 소개한다.

### ■ 객체지향 프로그래밍

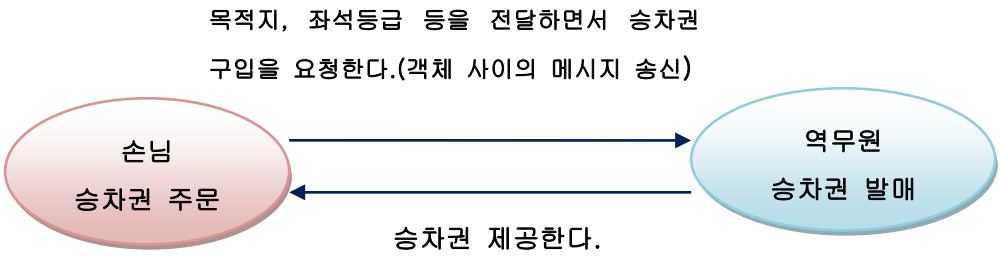
객체지향 프로그래밍(Object-Oriented Programming, OOP)은 컴퓨터 프로그래밍 패러다임의 하로서 **실제 세계를 모델링하여 소프트웨어를 개발하는 방법**이다. 컴퓨터 프로그램을 명령어의 목록으로 보는 시각에서 벗어나 여러 개의 독립된 단위, 즉 "객체"들의 모임으로 파악하고자 하는 것이다. 각각의 객체들은 메시지를 주고받으면서 데이터를 처리하게 된다.




객체지향 프로그래밍은 실 세계의 현상을 컴퓨터의에 객체로 실현(모델화)함으로써, 컴퓨터를 자연스러운 형태로 사용하여 다양한 문제를 해결하기 위한 프로그램 기법으로서 여기에서 객체

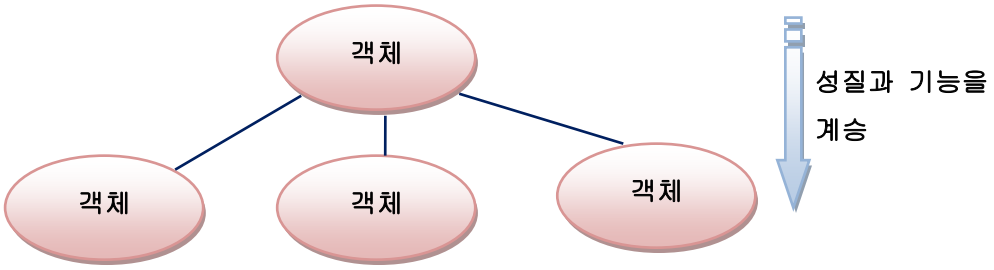
는 실체(데이터)와 그 실체와 관련되는 동작(절차, 방법, 기능)을 모두 포함한다.

기차역에서 승차권 발매의 예를 들면, 실체인 '손님'과 절차인 '승차권 주문'은 하나의 객체이고, 실체인 '역무원'과 절차인 '승차권 발매'도 하나의 객체이다.




어떤 과제를 처리하기 위해 객체 간에는 메시지(지시)를 주고받는다. 메시지를 받은 객체는 요구되는 동작(절차)을 실행한다. 기존 프로그램에서는 동작과 절차를 중심으로 하고 실체는 종속적으로 취급했으나, 객체 지향 프로그램에서는 **실체(데이터)와 동작을 객체로 정의하고 객체 간의 메시지 교환에 주안점을 두어 정보를 처리**한다. 즉, 객체 지향은 과정을 중시하는 절차 중심의 설계가 아니고, 실체(데이터)를 중시하는 설계이다.

객체 지향 프로그램의 또 하나의 중요한 특징은 공통의 성질을 갖는 객체는 객체 등급으로 정의한다는 점인데, 같은 등급에 속하는 객체들은 그들이 받는 메시지에 대하여 비슷하게 반응한다. 객체 등급은 계층화할 수 있으며, 하위 계층의 객체 등급은 상위 등급의 성질과 기능을 계승한다.



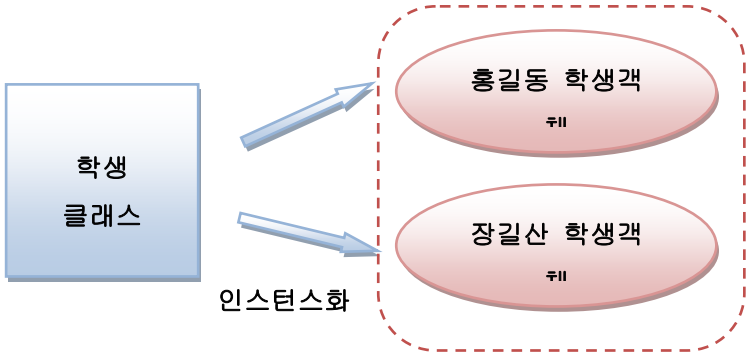

따라서 객체 지향은 시스템의 모듈화, 캡슐화를 촉진하여 복잡화되고 거대화되는 소프트웨어를 사용하기 쉽고, 작성하기 쉬우며, 유지 보수하기 쉬운 방향으로 재구축하는 새로운 기법으로 각

광받고 있다.

■ **클래스**

객체들은 객체들 고유의 데이터와 동작(behavior)을 가지고 있으며 서로 간에 상호 동작을 통해서 작업을 처리하게 된다. 클래스란 **'객체를 정의해놓은 것'** 으로서 **'객체의 설계도 또는 템플릿'** 이라고 정의할 수 있다. 클래스는 객체를 생성하는데 사용되며, 객체는 클래스에 정의된 대로 생성된다.

클래스로부터 객체를 만드는 과정을 클래스의 인스턴스화(instantiate)라고 하며, 어떠한 클래스로부터 만들어진 객체를 모두 그 클래스의 인스턴스(instance)라고 한다. 그러므로 객체와 인스턴스는 거의 비슷한 개념으로 사용된다.




**학생 클래스로 부터 생성된 인스턴스들**

프로그래밍에서의 객체는 클래스에 정의된 내용대로 메모리에 생성되어 사용 가능한 상태 및 수행 가능한 상태가 된 것을 뜻한다.

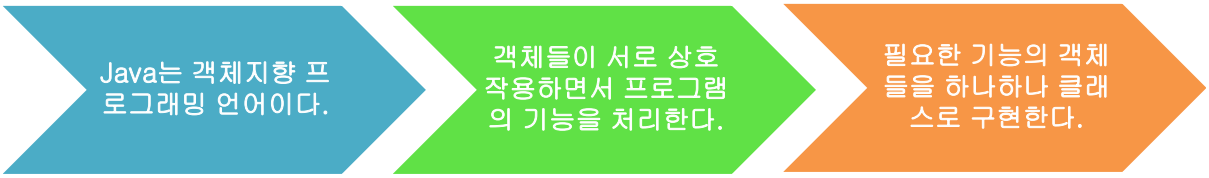
▶ 클래스의 구성 요소

클래스라는 객체를 만드는데 사용되는 설계도 또는 템플릿이다. 클래스를 만들 때는 어떠한 객체에 대한 설계도인가에 따라 객체에서 처리하게 될 데이터들은 변수로 객체에서 데이터를 처리하기 위해 필요한 행위는 메서드로 만든다. 바로 이 변수와 메서드가 클래스를 구성하는 요소들이며 클래스의 멤버라고 한다.

**객체의 데이터 --→ 클래스의 변수**
**객체의 행위 --→ 클래스의 메서드**

## 2.2 Java 프로그램의 구조

Java 프로그램의 기본 단위는 클래스이다. 이유는 Java는 완벽한 객체지향 프로그래밍 언어이기 때문이며 다음에 제시한 시나리오가 적용되기 때문이다.




Java에서의 클래스라는 단어의 의미는 2가지이다. 하나는 **객체를 설계한 프로그램의 구조**이며 다른 하나는 **Java만의 바이트 코드 형식으로 되어 있는 Java의 실행 파일**이다.

Java 프로그램의 소스 코드를 담게되는 파일을 소스파일이라고 하며 소스 파일의 명칭은 어떠한 명칭을 지정하든 확장자는 반드시 **.java** 여야만 한다. 그리고 그 안에는 하나의 클래스 또는 여러 개의 클래스를 정의할 수 있다. 그러나 가급적이면 클래스명과 그 클래스를 담게될 Java 프로그램의 소스 파일명은 대소문자까지 적용하여 동일하게 지정하는 것이 유지보수 측면에서 좋다.

Java 프로그램의 소스 코드는 기본적으로 다음과 같은 순서로 구성된다.

```
[ 패키지 정의 ]
[ import 또는 import static 선언 ]
클래스 정의
[ 클래스 정의 ... ]
```

■ **패키지(Package)**

서로 관련된 클래스와 인터페이스의 묶음을 의미한다. 클래스가 물리적으로 클래스파일(*.class)로 만들어지는 것처럼, 패키지는 물리적으로 폴더 즉 디렉토리로 만들어진이다. 패키지는 서브 패키지를 가질 수 있으며, '.'으로 구분한다. 클래스의 실제 이름(full name)은 패키지명이 포함된 것이다. 즉 String클래스의 실제 명칭은 java.lang.String, Date클래스의 실제 명칭은 java.util.Date이다.

[ 패키지의 장점 ]

- 서로 다른 패키지에는 동일한 명칭의 클래스가 존재할 수 있으므로 충돌을 방지한다.
- 패키지 이름과 함께 클래스 이름을 사용함으로써 클래스를 효율적으로 관리할 수 있다.
- 클래스를 관련이 있는 것끼리 묶어 놓음으로써, 필요한 클래스의 식별이 용이하게 한다.
- 패키지 단위의 접근 권한을 지정할 수 있고, 이를 통하여 클래스만 있는 경우보다 더욱 다양한 접근 권한을 제어할 수 있다.

패키지는 소스파일에 제일 첫 번째 문장(주석 제외)에 다음과 같은 형식으로 한 번만 선언한다. 하나의 소스파일에 둘 이상의 클래스가 포함된 경우, 모두 같은 패키지에 속하게 된다. 모든 클래스는 하나의 패키지에 속하며, 패키지가 선언되지 않은 클래스는 자동적으로 이름없는(unnamed) 패키지 즉, 디폴트 패키지에 속하게 된다.

```
package 패키지명;
```

패키지가 선언된 소스파일에 포함된 클래스와 인터페이스는 동일한 패키지에 속하게 된다. 패키지명을 정할 때는 계층적인 구조를 사용하여 **'패키지명.서브패키지명'**과 같은 형식으로 정의할 수 있다.

package package1.package2.package3; → **package1/package2/package 폴더에 클래스를 생성**

▶ 패키지가 설정된 Java 소스의 컴파일

패키지가 설정된 Java 소스를 컴파일 할 때는 두 가지 방법으로 컴파일한다.

**- 패키지명에 해당하는 폴더들을 만들고 그 안에서 소스를 컴파일한다.**
**- 어느 디렉토리에서 컴파일 하든 패키지를 생성하고자 하는 위치를 -d 옵션과 함께 지정한다.**

```
javac  -d  폴더명  소스명.java
```

-d 옵션은 주어진 폴더명에 해당하는 위치에 package문에 있는 경로대로 디렉토리를 만들라는 것을 의미하며 현재 디렉토리를 의미하는 . 을 지정하면 현재 작업하고 있는 디렉토리에 만들도록 하는 의미이다.
-d 옵션과 폴더명을 사용하면 컴파일한 후 그곳에 .class파일을 소스파일에 정의된 package문

에 기술한 디렉토리를 만들거나 찾아서 저장하게 된다.

다음은 Java 소스에 패키지를 지정하는 다양한 예제이다.

다음에 제시된 Java 클래스에 대한 소스의 명칭은 반드시 Example_PackageTest1.java여야만 한다. Java의 소스명 규칙에는 소스 안에 정의되는 클래스가 public형인 경우에는 반드시 소스명과 클래스명이 동일해야 하기 때문이다. 그리고 소스안에 exam2라는 명칭의 패키지가 선언되어 있으므로 Example_PackageTest1.class는 exam2라는 폴더에 만들어지게 된다.

Example_PackageTest1.java을 컴파일 할 때 다음과 같이 컴파일 명령을 사용하면

```
javac  -d  c:/ocjp/example  Example_PackageTest1.java
```

Example_PackageTest1.class는 c:/ocjp/example/exam2/ 폴더에 생성된다.

```
package exam2;

public class Example_PackageTest1 {

}
```

exam2라는 명칭의 폴더를 찾거나 만들고 Exampe_PackageTest1.class를 생성한다.

패키지 선언은 소스당 한 번만 선언할 수 있다. 다음 예제에서는 하나의 Java 소스에 두 개의 클래스가 정의된 예를 보여주고 있다. 소스안에 정의되는 클래스가 public일 때는 클래스명과 소스명이 동일해야 하므로 하나의 Java 소스에는 두 개 이상의 public 클래스가 존재할 수 없다. 주어진 예제에서는 Exampe_PackageTest2이 public 클래스이고 Exampe_PackageTest3은 public 클래스가 아니므로 동일한 Java 소스에 존재할 수 있다. public은 제어자로서 이 클래스를 클래스가 속한 패키지의 밖에서도 접근할 수 있도록 접근 권한을 공개하는 기능이다. public이 지정되지 않은 클래스는 해당 패키지내에서만 접근할 수 있는 클래스가 된다.

```
package exam2.test;

public class Example_PackageTest2 {

}
class Example_PackageTest3 {

}
```

exam2/test라는 명칭의 폴더를 찾거나 만들고 Exampe_PackageTest2.class와 Exampe_PackageTest3.class를 생성한다.

▶ 패키지가 설정된 Java 클래스의 실행

패키지가 설정된 Java 클래스를 실행시킬 때는 클래스명에 패키지명을 붙여서 지정해야 한다. 패키지명을 확장한 클래스명을 풀네임이라고 하며 다음과 같이 **패키지명.클래스명** 형식으로 사용한다.

```
java exam2.Example_PackageTest1
java exam2.test.Example_PackageTest2
```

■ import와 import static

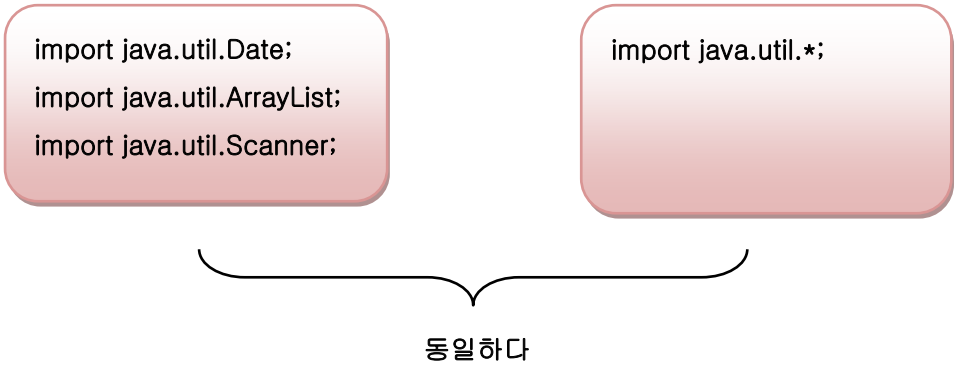
새로운 Java 클래스의 소스를 만들 때 동일 패키지의 클래스를 사용하는 경우에는 클래스명만 사용해도 되지만 패키지화된 클래스를 사용하는 경우에는 다음과 같이 일일이 패키지명을 붙여주는 풀네임 형식을 사용해야 한다. String 클래스의 경우에는 패키지명을 생략할 수 있는데 String 클래스는 java.lang 패키지에 속하는 클래스이기 때문이다. java.lang 패키지는 Java 에서 유일하게 자동 import되는 패키지이다.

```
package exam2;
public class Example_ImportTest1 {
    public static void main(String args[]) {
        java.util.Date obj = new java.util.Date();
        java.text.DateFormat format = java.text.DateFormat.getInstance();
```

사용하려는 클래스명마다 풀네임을 사용하는 것은 번거로운 일이다. 이러한 경우에 사용하는 구문이 바로 import 선언이다. 사용할 클래스가 속한 패키지를 지정하는데 사용하며 import문을 사용하면 클래스를 사용할 때 패키지명을 생략할 수 있다.
위의 소스를 다음과 같이 바꿀 수 있다.

```
package exam2;
import java.util.Date;              // import java.util.*; 로 사용해도 된다.
import java.text.DateFormat;    // import java.text.*; 로 사용해도 된다.
public class Example_ ImportTest2 {
    public static void main(String args[]) {
        Date obj = new Date();
        DateFormat format = DateFormat.getInstance();
                    :
```

import 선언은 패키지 선언문의 뒤에 그리고 클래스를 정의하기 전에 구현해야 하며 필요없는 경우에는 생략 가능하다. import 선언은 클래스 단위로 해야 하며, 특정 패키지에서 여러 개의 클래스를 사용하는 경우에는 메타 문자인 '*' 기호를 대신 사용하는 것도 가능하다.




▶ import static 문

다른 클래스에 정의되는 변수나 메서드를 사용할 때는 해당 클래스를 객체 생성하고 생성된 객체를 통해서 멤버연산자(.)으로 사용한다. 그러나 클래스에 정의된 변수나 메서드가 static 형인 경우에는 객체 생성없이 **클래스명.메서드명, 클래스명.변수명**과 같이 클래스명에 멤버연산자를 사용하여 활용하게 된다.

그런데 Java SE 5.0 부터는 이렇게 다른 클래스에 존재하는 static 형 멤버들을 사용할 때 클래스명을 생략하고 필요한 멤버만을 사용할 수 있게 되었으며 이 때 사용되는 구문이 바로 import static 선언이다.

import static 선언을 할 때는 풀네임 형식으로 클래스명을 사용하고 이 뒤에 클래스명없이 사용하고자 하는 멤버의 명칭을 지정하거나 메타문자 '*' 기호를 사용한다.

```
import static 패키지명.클래스명.*;          // 주어진 클래스의 모든 static멤버
import static 패키지명.클래스명.변수명;
import static 패키지명.클래스명.메서드명;
```

static 변수의 명칭이나 static 메서드의 명칭 대신에 메타문자 '*' 기호를 사용하게 되면 이 클래스가 가지고 있는 모든 static 멤버에 적용되는 결과가 된다. 메서드명을 지정할 때는 괄호를 생략하고 메서드명만을 지정한다.

다음에 제시된 import static 문을 사용하지 않은 예제와 import static 문을 사용한 예제를

비교해 본다. Example_ImportTest3에서는 DateFormat클래스의 getInstance()메서드, System클래스의 out변수 그리고 Math클래스의 PI변수와 random()메서드를 각각 클래스명을 지정하여 사용하고 있는 것을 볼 수 있다. 그러나 Example_ImportTest4에서는 import static문을 추가하고 getInstance()메서드, out변수 그리고 PI변수와 random()메서드를 각각 클래스명을 생략하고 사용하는 것을 볼 수 있다.

```
package exam2;
import java.util.*;
import java.text.*;
public class Example_ImportTest1 {
    public static void main(String args[]) {
        DateFormat format =
DateFormat.getInstance();
        System.out.println("날짜와 시간 정보");
        System.out.println(format.format(new Date()));
        System.out.println(Math.PI);
        System.out.println(M
    }
}
```

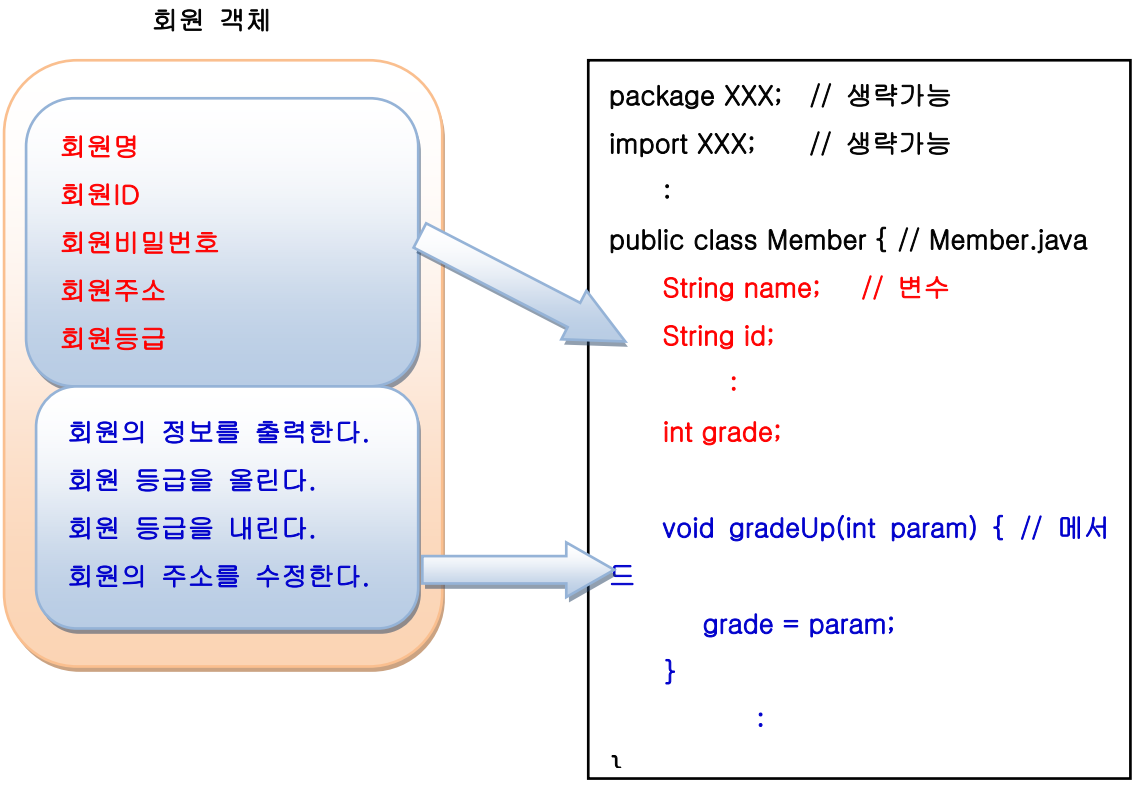
```
package exam2;

import java.util.*;
import java.text.*;
import static  java.text.DateFormat.*;
import static java.lang.System.out;
import static java.lang.Math.*;

public class Example_ImportTest2 {
    public static void main(String args[]) {
        DateFormat format = getInstance();
        out.println("날짜와 시간 정보");
        out.println(format.format(new Date()));
        out.println(PI);
        out.println(random());
    }
}
```

이 클래스들이 가지고 있는 static 멤버들을 import 한다.

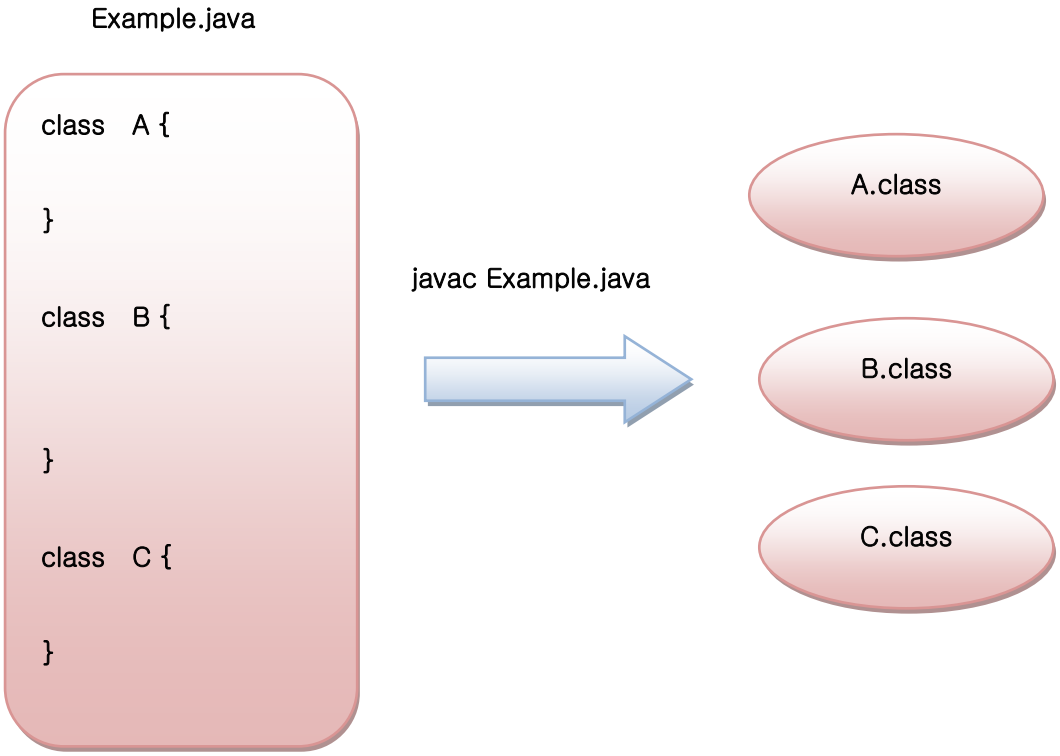
■ 클래스의 정의

클래스를 만들 때는 이 클래스가 어떠한 객체에 대한 설계도(템플릿) 역할을 하게 될 것인지를 고려하여 객체의 데이터는 클래스의 변수로 객에의 행위는 메서드로 정의한다.

예를 들어 회원 객체에 대한 클래스를 만든다면 다음 그림과 같은 방식의 구현이 적용된다.




필요에 따라서는 하나의 Java 소스에 여러 개의 클래스를 정의할 수 있다. 다만 하나의 클래스만을 public으로 지정할 수 있다. Java 컴파일러가 주어진 Java 소스를 컴파일할 때는 소스 단위로 클래스 파일을 만드는 것이 아니며 소스에 정의된 클래스 단위로 클래스 파일을 만들게 되므로 다음과 같이 하나의 Java 소스에 여러 개의 클래스가 정의되어 있는 경우에는 여러 개의 클래스 파일이 만들어진다.

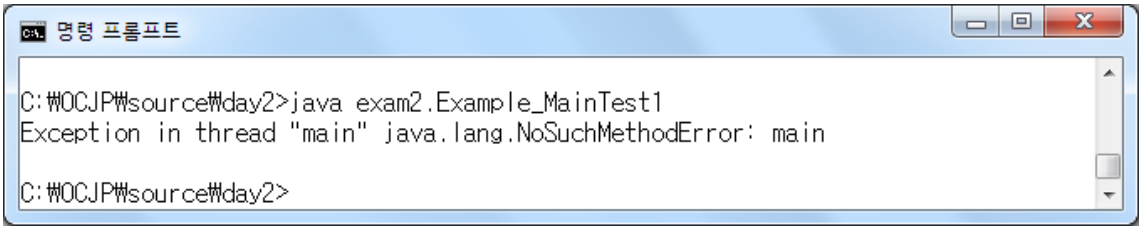



클래스의 정의 그리고 활용과 관련된 내용은 7일차에서 자세히 학습한다.

■ main() 메서드

main() 메서드는 JVM이 Java 프로그램을 수행시킬 때 제일 먼저 호출하는 메서드로서 정해진 사양의 메서드이다. main() 메서드를 가지고 있는 클래스를 메인클래스 또는 어플리케이션 클래스라고 한다.

Java 인터프리터 명령어인 java를 통해서 Java 프로그램을 실행시킬 때는 반드시 메인 클래스명을 파라미터로 지정해 주어야 한다. 그렇지 않으면 다음과 같은 실행 오류가 발생한다.




java 명령어는 파라미터로 주어진 메인 클래스인 exam2.Example_MainTest1 클래스를 찾아서 JVM 영역에 로딩한 다음 제일 먼저 호출하는 메서드가 바로 다음과 같은 형식으로 정의되어 있

는 main() 메서드이다.

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

여기에서 붉은 색으로 표현한 부분은 생략할 수도 없고 수정 할 수도 없다. main() 메서드의 매개변수는 반드시 String 타입의 배열 변수여야 하며 변수명은 다른 명칭으로 변경해도 되지만 관례적으로 args를 사용하고 있다. main() 메서드의 사양에 대한 자세항 내용은 6일차와 7일차에서 학습한다.