

자바 기초

April. 30, 2007

객체와 클래스

류 대 우

davidryu@newtc.co.kr

학습 목표

■ 학습 목표

- 객체와 클래스의 개념을 배운다.
- 객체를 만드는 방법과 생성자에 대해 배운다.
- 오버로딩이 무엇인지 배운다.
- Main() 메서드의 구조를 배운다.
- Import 문과 패키지에 대해 배운다.

객체 지향이란?

- Object-Oriented Programming
- 컴퓨터 프로그램을 명령어의 목록으로 보는 시각에서 벗어나 여러 개의 독립된 단위, 즉 "객체"들의 모임으로 파악하고자 하는 것이다. 각각의 객체는 메시지를 주고받고, 데이터를 처리할 수 있다.
- 장점
 - 프로그램을 유연하고 변경이 용이하게 만든다.
 - 더 배우기 쉽게 하고 소프트웨어 개발과 보수를 간편하게 한다.
 - 보다 직관적인 코드 분석을 가능하게 한다.
- 단점
 - 지나친 프로그램의 객체화 경향은 실제 세계의 모습을 그대로 반영하지 못한다

객체 지향 프로그래밍의 요소

- 클래스(Class)
 - 하나의 프로그램 단위를 뜻하는 것
 - 객체 속성과 메서드를 표현하는 방식이며, 자바 프로그램의 기본 집합이다.
- 객체(Object) - 클래스의 인스턴스이다.
- 메서드 (Method) - 클래스로부터 생성된 객체를 사용하는 방법이다.
 - 한 객체의 서브루틴 형태로 객체의 속성을 조작하는데 주로 사용 된다.
 - 객체를 실행하는 명령으로 객체와 외부 세계를 연결하는 인터페이스

객체 지향 프로그래밍의 요소

- Reusability
- 클래스 단위로 프로그램을 작성하면 객체 단위로 묶을 수 있어서 다른 프로그램에서 쉽게 가져다 쓸 수 있음
- B클래스에서 A 클래스를 불러 쓰면, A 클래스에서 정의된 메서드나 변수 등을 그대로 사용하거나 다시 정의해서 사용이 가능한데, 이 때 메모리에 상주하는 A 클래스의 내용을 '인스턴스'(Instance)라고 한다.

객체?

- Object

- 객체는 현실 세계에 존재하는 특정 대상을 표현하는 것이며, 기능이 유사한 것을 모아 놓은 집합체 이다.

- 객체의 이벤트

- 사용자가 외부 세계의 특정 부분을 조작 후 컴퓨터로부터 발생하는 일종의 신호

- 객체의 속성(Property)

- 객체가 갖고 있는 기본적인 특성

- 객체의 메서드

- 아이콘 위에서 마우스 왼쪽 버튼을 누르면 윈도우가 화면에 표시된다.

객체

이벤트

메서드

류 대 우

davidryu@newtc.co.kr

객체 지향의 4가지 개념

- 추상화
 - Abstraction
- 상속
 - Inheritance
- 다형성
 - Polymorphism
- 캡슐화
 - Encapsulation

객체 지향의 4가지 개념

■ 추상화

- Abstraction
- 클래스에 부여할 속성과 행동을 정의하는 것
- 즉, 클래스의 속성과 행동을 하나씩 세분화하여 정의하는 것을 말한다.

객체 지향의 4가지 개념

- 상속

- Inheritance

- 상위 클래스의 속성이나 행동을 하위 클래스가 물려받아서 사용하는 것

객체 지향의 4가지 개념

- 다형성

- Polymorphism

- 같은 이름의 행동이 여러 클래스에서 각각 다르게 정의되어 사용되는 것

객체 지향의 4가지 개념

■ 캡슐화

■ Encapsulation

- 객체의 속성과 행동을 객체 자신만 알고 다른 객체에는 알려주지 않는 것
 - 정보 은닉(Information Hiding)이라고도 함.
- 일반적으로 메서드는 public으로 선언하여 다른 클래스의 접근을 허용한 인터페이스 역할을 하지만,
- 변수는 private로 선언하여 다른 클래스의 접근을 막는 것이 캡슐화의 시작임.

클래스 정의하기

- 2강에서 배운 것처럼 클래스는 아래와 같이 선언한다.
 - [클래스 접근 한정자] class 클래스명
 - [extends 슈퍼클래스명] //→ 옵션사항임
 - [implements 인터페이스명] //→ 옵션사항임
 - {
 - [필드 접근 한정자] 멤버 변수;
 - [생성자 접근 한정자][생성자];
 - [메서드 접근 한정자] 메서드;
 - }
- 클래스 이름 시작은 보통 대문자로 시작함
 - 3강 참조

객체 인스턴스 생성

- public class 클래스명{
 - 호출할_클래스명 aa = new 호출할_클래스명();
- }

클래스객체 인스턴스 생성 3단계

1단계 : 클래스 객체 변수 선언

```
ClassTest ct;
```



2단계 : 인스턴스 생성

```
ct = new ClassTest();
```



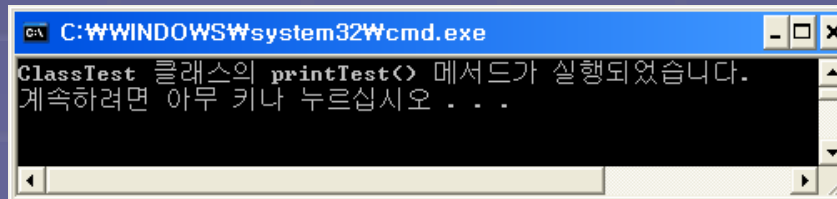
3단계 : 인스턴스를 사용하여 클래스의 멤버에 접근

```
ct.printTest();
```

수행 과제(I)

```
■ public class ClassTestMain
■ {
■     public static void main(String args[]){
■         ClassTest ct = new ClassTest();
■         ct.printTest();
■     }
■ }

■ class ClassTest
■ {
■     public static void printTest(){
■         ■ System.out.println("ClassTest 클래스의 printTest() 메서드가 실행되었습니다.");
■     }
■ }
```



생성자

- Constructor
- 앞서 예를 보면
 - `ClassTest();`와 같은 메서드가 실행되고 있습니다.
 - 이와 같이 다른 클래스를 자기 것처럼 사용하기 위하여 `instance`를 생성하게 되는 데, 이 `instance`를 만들어 주는 것이 생성자입니다.
 - 자바 버추얼머신이 자동으로 제공해 줍니다.

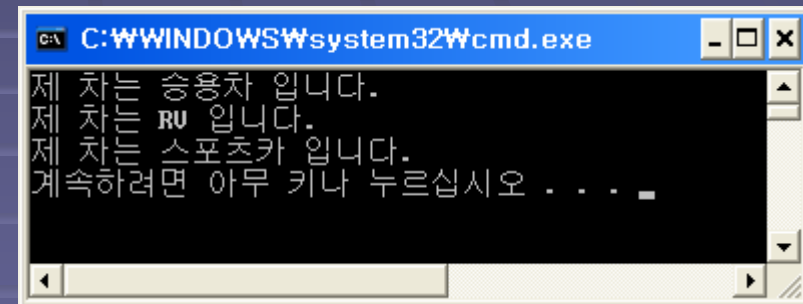
생성자의 특징

- 반환 값이 없다.
- 생성자의 이름과 클래스의 이름이 같다.
- 생성자는 new 연산자에 의해 만들어지고 호출된다.

수행 과제(II)

```
■ public class MyCarExec
■ {
■     public static void main(String args[]){
■         new MyCar("승용차");
■         new MyCar("RV");
■         new MyCar("스포츠카");
■     }
■ }

■ class MyCar
■ {
■     MyCar(String carName){
■         System.out.println("제 차는 "+ carName + " 입니다.");
■     }
■ }
```



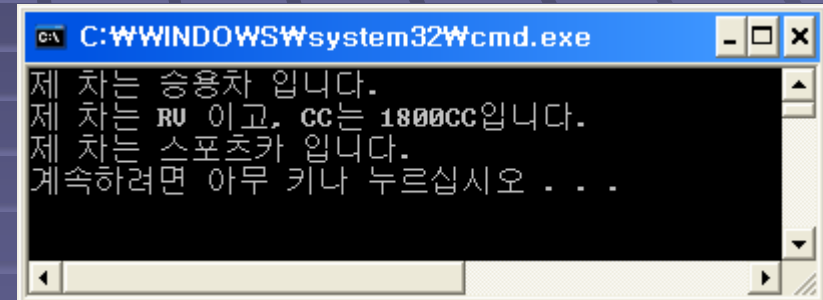
생성자 오버로딩

- 생성자에서 ()안에 매개 변수를 입력하는 부분이 있는 데 ()안의 매개변수가 다르면 서로 다른 생성자로 인식하고 이와 같이 실행하는 데, 이것을 ‘오버로딩 (Overloading)’이라고 합니다.

수행 과제(II)

```
public class MySecondCarExec
{
    public static void main(String args[]){
        new MyCar("승용차");
        new MyCar("RV","1800CC");
        new MyCar("스포츠카");
    }
}

class MyCar
{
    MyCar(String carName){
        System.out.println("제 차는 "+ carName + " 입니다.");
    }
    MyCar(String carName, String type){
        System.out.println("제 차는 "+ carName + " 이고, CC는 "+type+"입니다.");
    }
}
```



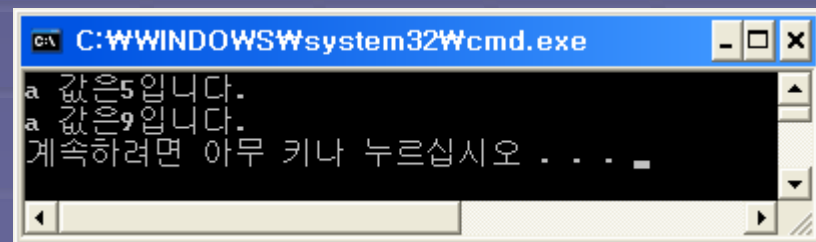
메서드 오버로딩

- 오버로딩은 생성자 뿐만 아니라 일반 메서드에도 적용된다.
- 메서드 이름과 반환 타입이 같고, 매개 변수의 수나 종류가 다를 때 성립된다.

```
public class Overloading
{
    static void add(int i){
        int a = 0;
        a = i+1;
        System.out.println("a 값은"+a+"입니다.");
    }

    static void add(int i, int j){
        int a = 0;
        a = i+j;
        System.out.println("a 값은"+a+"입니다.");
    }

    public static void main(String args[]){
        add(4);
        add(4,5);
    }
}
```

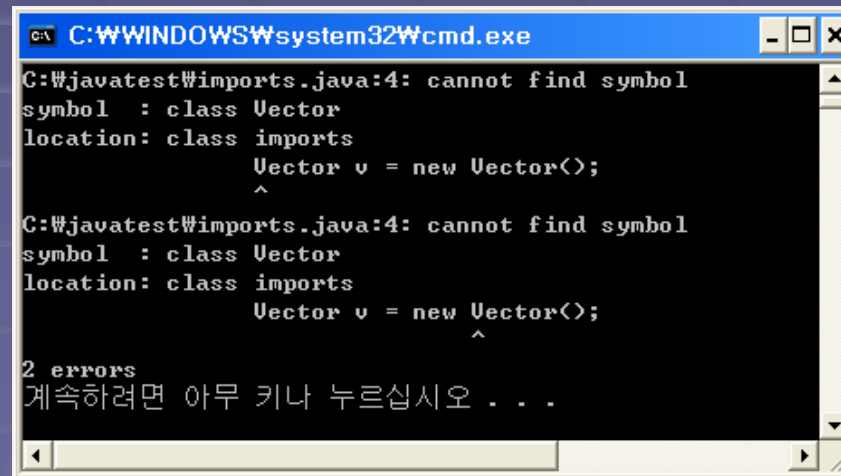


import문

- import문은 여러 폴더에 흩어져 있는 클래스들 가운데 원하는 클래스를 찾아서 사용할 수 있도록 도와주는 키워드이다.
- 자바 버추얼 머신이 원하는 클래스를 찾는 방법
 - 클래스가 저장된 폴더 → 클래스 패스를 설정한 폴더 → 임포트한 클래스
- 사용하는 방법
 - 클래스를 사용하기 전에 먼저 선언한다.
 - `import 패키지명.클래스명;`
 - `import 클래스명;`

import문 사용의 잘못된 예

- public class imports
- {
- public static void main(String args[]){
- Vector v = new Vector();
- }
- }



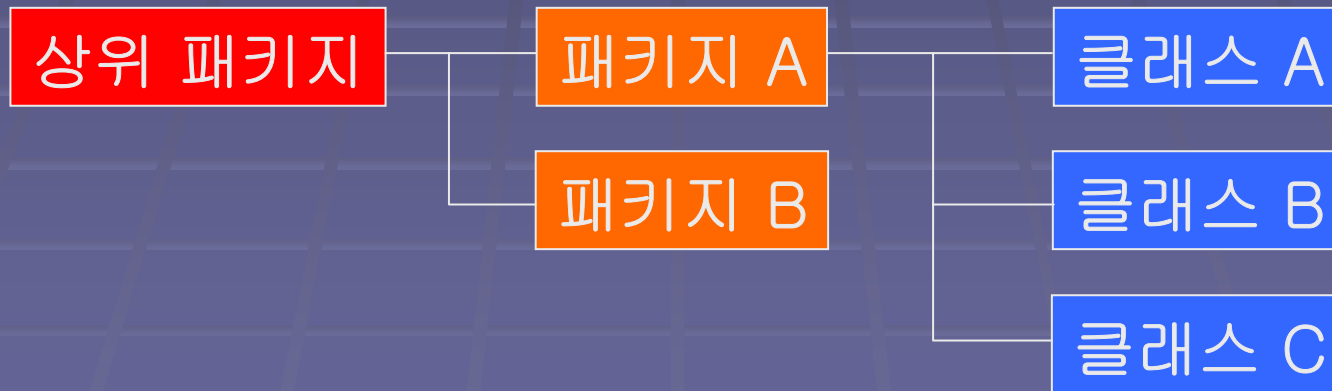
```
C:\WINDOWS\system32\cmd.exe
C:\javatest\imports.java:4: cannot find symbol
symbol : class Vector
location: class imports
    Vector v = new Vector();
                    ^
C:\javatest\imports.java:4: cannot find symbol
symbol : class Vector
location: class imports
    Vector v = new Vector();
                    ^
2 errors
계속하려면 아무 키나 누르십시오 . . .
```

Import문 사용의 잘된 예

- `import java.util.Vector;`
- `public class imports`
- `{`
- `public static void main(String args[]){`
- `Vector v = new Vector();`
- `}`
- `}`

패키지

- 수 많은 비슷한 기능을 하는 클래스들을 묶어서 관리함으로써 더 효율적인 프로그래밍을 한다.

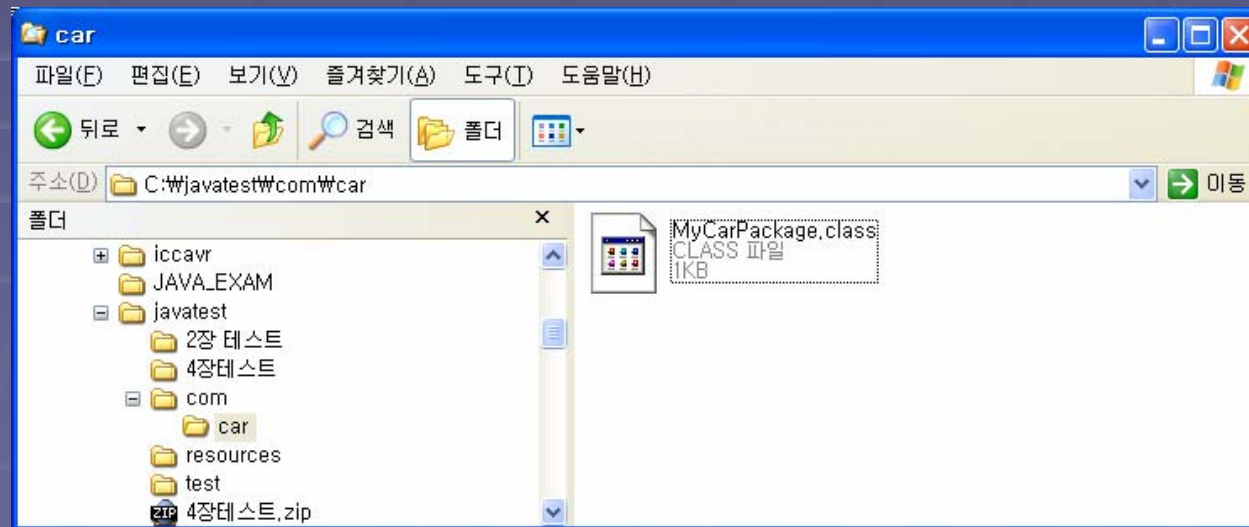


나만의 패키지 만들기

- Package 상위패키지명.패키지명;
- Package 패키지명;
- package com.car
- public class MyCarPackage
- {
- public MyCarPackage(String carName){
 - System.out.println("제 차는 "+ carName +" 입니다.");
- }
- public MyCarPackage(String carName, String type){
 - System.out.println("제 차는 "+ carName +" 이고, CC는 "+type+"입니다.");
- }
- }

나만의 패키지 컴파일

- `Javac -d [경로][소스파일]`
- 즉, `javac -d . MyCarPackage.java`
- 여기서 `.`은 컴파일하는 현재폴더임



import문 없이 패키지의 클래스 사용하기

- java.lang의 클래스들은 기본 클래스
이므로 컴파일할 때 자동으로 로드된
다. String 클래스와 같은 경우
- 전체 경로를 설정하여 실행하는 경우
 - Ex) `java.util.Vector v = new java.util.Vector();`