

Professional Coding Specialist

COS Pro JAVA 1 급

2 강. 기초 문법 정리 2

1. 클래스

1-1. 클래스

1-2. 상속

1-3. 추상 클래스, 인터페이스

과정 소개

COS Pro(Coding Specialist Professional) 시험은 요구사항을 분석하여 프로그램을 설계, 구현하는 능력과 주어진 프로그램을 디버깅하여 수정하는 능력을 평가하는 자격증 시험이며, COS Pro 1 급 자격증은 높은 수준의 프로그래밍 능력을 증명할 수 있다. 이번 시간에는 COS Pro JAVA 1 급 시험 대비를 위한 모의고사를 풀어보기 전에 알아야 하는 JAVA 의 기초 문법을 정리하는 두 번째 시간으로, JAVA 의 클래스에 대한 기본 개념을 정리하고, 추상 클래스, 인터페이스 등 다양한 클래스의 개념들을 알아본다.

학습 목차

1. 클래스
 - 1-1. 클래스
 - 1-2. 상속
 - 1-3. 추상 클래스, 인터페이스

학습 목표

1. 객체 지향 프로그래밍에 대해 이해할 수 있다.
2. 클래스의 기본 개념과 구성요소에 대해서 설명할 수 있다.
3. 정의된 클래스를 기반으로 객체를 생성하여 객체에 대한 메소드를 실행하고, 클래스 멤버를 참조할 수 있다.
4. 정보 은닉, 클래스 or static 변수, 상속, 메소드 오버로딩, 메소드 오버라이딩, 추상 클래스, 인터페이스 등에 대한 개념을 이해하고 설명할 수 있다.

1-1

클래스

1. 객체 지향(Object Oriented) 프로그래밍 언어


1) 특징

- 프로그램을 명령어의 목록으로 보는 시각에서 벗어나 여러 개의 독립된 객체들의 모임으로 구성된 것으로 인식
- 각각의 객체는 메시지를 주고 받고, 데이터 처리


2) '클래스'란?

클래스(Class)	특정 객체를 생성하기 위해 속성(변수)과 기능(메소드)을 정의하는 틀의 역할을 하는 것. 자동차 생산을 예로 들면, 자동차를 만들기 위한 틀(거푸집) 같은 역할을 하는 존재
객체(Object)	클래스를 기반으로 만들어진 실제 사물. 자동차 거푸집을 이용해서 생산된 자동차와 같은 것

예 1) 자동차 클래스를 기반으로 하여 빨간색 차와 노란색 차 객체 생성

자동차 클래스		객체로 생성된 자동차
속성(Attribute)	기능(method)	
모델명	달린다()	
가격	멈춘다()	
색상		

예 2) 버튼 클래스를 기반으로 확인 버튼과 취소 버튼 객체 생성

버튼 클래스		객체로 생성된 버튼
속성(Attribute)	기능(method)	
size	click()	
color	double_click()	
shape		

- ➔ 공통된 속성과 기능을 모아서 클래스를 정의한 뒤 필요에 따라 객체를 생성해서 사용하면 공통적으로 필요한 프로그램 코드를 중복 작성할 필요가 없음

2. 클래스

1) 클래스 정의

클래스(Class) : 틀



- 멤버 변수 : 객체의 속성, 데이터, 각 인스턴스마다 독립된 메모리 공간에 저장
- 생성자(Constructor) : 객체가 생성될 때 실행되며 주로 멤버 변수를 초기화하는데 사용.
생성자의 이름은 클래스 이름과 동일하고 리턴 타입 없음
- 소멸자 : finalize(), 객체가 소멸될 때 실행되며 오버라이드해서 작성 가능
대부분 JVM에 의해 자동 호출, 객체 제거(메모리 관리)를 최적화하는데 사용됨
- 메소드 : 객체의 기능, 행동(method)

2) 인스턴스(Instance) : 클래스를 기반으로 실제 생성된 객체

- 인스턴스 생성 인스턴스 변수 이름 = new 클래스 이름()
- 인스턴스의 멤버 변수 참조 인스턴스 변수 이름.멤버 변수 이름
- 인스턴스의 메소드 실행 인스턴스 변수 이름.메소드 이름()

3) 클래스 관련 이름 규칙

- 클래스 이름 규칙
 - 첫 번째 문자는 대문자 : Car
 - 둘 이상의 단어가 묶일 때 새로 시작하는 단어는 대문자 : CarSale
- 메소드와 변수 이름 규칙
 - 첫 번째 문자는 소문자 : addCar
 - 둘 이상의 단어가 묶일 때 새로 시작하는 단어는 대문자

4) 클래스 정의 및 인스턴스 기본 사용법

사용 예 1)

- 클래스 정의하기

```
class Car1{  
    String model="SM3";  
    int price=2000;  
  
    void sale(int price) {  
        this.price=this.price-price;  
    }  
}
```

클래스 Car1 의 멤버 변수 model, price 의 값을 지정

매개변수 price 를 갖는 클래스 Car1 의 메소드 sale() 를 정의

♦ 객체(인스턴스) 생성하고 사용하기

```
Car1 newCar1 = new Car1();  
System.out.println(newCar1.model);  
System.out.println(newCar1.price);  
  
newCar1.price=1500;  
System.out.println(newCar1.price);  
  
newCar1.sale(200);  
System.out.println(newCar1.price);
```

클래스 Car1 을 기반으로 인스턴스 변수 new Car1 생성

인스턴스 변수 newCar1 의 멤버 변수 price 를 1500 으로 변경

인스턴스 변수 newCar1 의 메소드 sale() 를 실행하여 멤버 변수 price 에서 200 을 뺌

<결과>

```
1500  
SM3 1300
```

사용 예 2)

♦ 클래스 정의하기

```
class Car1_1{  
    String model;  
    int price;  
  
    Car1_1(){  
        this.model="SM5";  
        this.price=3000;  
    }  
  
    void sale(int price) {  
        this.price=this.price-price;  
    }  
}
```

Car1_1 클래스의 멤버 변수 model, price 를 선언

클래스 Car1_1 의 생성자(Constructor)를 매개변수가 없는 형태로 정의. 멤버 변수 model, price 의 값을 초기화

매개변수 price 를 갖는 클래스 Car1_1 의 메소드 sale() 를 정의

♦ 객체(인스턴스) 생성하고 사용하기

```
Car1_1 newCar1_1 = new Car1_1();  
newCar1_1.sale(200);  
System.out.println(newCar1_1.model);  
System.out.println(newCar1_1.price);
```

클래스 Car1_1 을 기반으로 새로운 객체(인스턴스) newCar1_1 생성

인스턴스 변수 newCar1_1 의 메소드 sale() 를 실행하여 멤버 변수 price 에서 200 을 뺌

<결과>

```
1500  
SM3 1300
```

※ 참고

- 객체 즉 인스턴스 변수를 생성(new)하면 객체의 주소값이 반환되는 형태로 인스턴스 변수는 참조 변수임
- this 는 자기자신, super 는 부모를 가리킴

5) 생성자(Constructor)

객체가 생성될 때 실행되는 일종의 함수로 주로 멤버 변수를 초기화하는데 사용

클래스 이름과 동일해야 하며 리턴 타입 없음 (값을 반환하지 않음)

매개변수의 타입과 개수가 다른 생성자를 여러 개 선언할 수 있으며, 생성자를 선언하지 않은 경우 디폴트 생성자가 호출

```
class CarCons{
    String model;
    int price;

    public CarCons(String model, int price){
        this.model=model;
        this.price=price;
    }

    void sale(int price) {
        this.price=this.price-price;
    }
}

public class ExamClassConstructor {
    public static void main(String[] args) {
        CarCons carCons1 = new CarCons("Avante",1000);
        System.out.println(carCons1.model);
        System.out.println(carCons1.price);
    }
}
```

디폴트 생성자

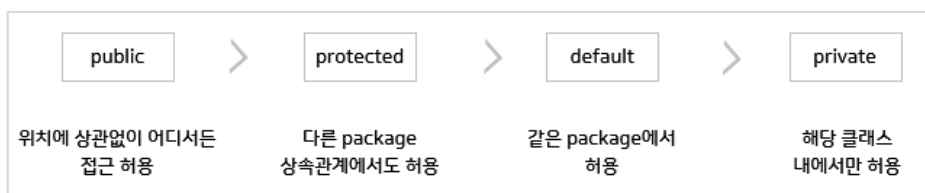
```
public CarCons(){ }
```

```
CarCons carCons1 = new CarCons();
```

6) 패키지(package)

- 패키지 : 클래스 그룹
- 물리적으로 하나의 폴더를 의미하므로 하나의 패키지에 속한 클래스들은 해당 패키지 이름의 폴더 안에 존재
- 패키지는 다른 패키지를 포함할 수 있고, 이때 계층 구조는 마침표(.) 로 구분
- import 명령 : import 패키지명.클래스명; import 패키지명.*;
 - Ex) import hello.ABC; import hello.*;
- java.lang 패키지는 자바의 기본 기능을 수행하기 때문에 별도의 import 없이 사용 (예: System.out.println)
- 이클립스에서는 ctrl + shift + o 를 누르면 자동 import 됨
- 보통 COS PRO 시험 문제에서는 import java.util.* 등이 코드 맨 위에 적혀 있으나, 학습 시 import 확인 필요

7) 접근 제어 지시자(Access Level Modifiers)



- 클래스 정의 : public, default

- 멤버 변수와 메소드 : public, protected, default, private
 - ※ public 클래스는 하나의 소스 파일에 1 개만 허용하며, 파일 이름과 동일해야 함

3. 정보 은닉

1) 기능

- 멤버 변수의 직접적인 접근을 차단
- 메소드에서 허용하는 방법으로만 멤버 변수의 접근을 허용하여 논리적인 오류를 방지하고, 객체의 안정성을 높임

2) 사용 예

```
class Car2{
    private String model;
    private int price;

    Car2(String model, int price){
        this.model=model;
        this.price=price;
    }

    void sale(int price) {
        this.price=this.price-price;
    }

    String getModel() {
        return this.model;
    }

    int getPrice() {
        return this.price;
    }

    void setPrice(int price) {
        this.price=price;
    }
}
```

Car2 클래스를 기반으로 인스턴스를 생성할 때 멤버 변수를 private 로 함으로써 멤버 변수를 클래스 내에서만 접근할 수 있도록 한정

```
Car2 newCar2 = new Car2("Sonata",2500);
newCar2.sale(200);

//System.out.println(newCar2.model);
System.out.println(newCar2.getModel());

//newCar2.price=2000;
newCar2.setPrice(2000);
System.out.println(newCar2.getPrice());
```

멤버 변수가 private 로 되어 있으므로 main()에서 newCar2.model 처럼 인스턴스 변수를 생성하여 멤버 변수를 직접 참조할 수 없음
main()에서 private 로 된 멤버 변수는 직접 참조할 수 없으므로 getModel() 과 같이 주어진 메소드를 사용해야 함

이처럼 접근 제어 지시자와 메소드를 이용하여 멤버 변수를 은닉한 형태를 정보 은닉이라 한다.

4. 클래스 변수, 클래스 메소드

1) 클래스 변수

```
class Car
{
    int instanceVar;      멤버 변수
    static int classVar;  클래스 변수
    void method()
    {
        int localVar;    지역 변수
        ...
    }
}
```

인스턴스 변수

```
Car s1 = new Car();
Car s2 = new Car();
Car s3 = new Car();
```

- ♦ **인스턴스 변수**

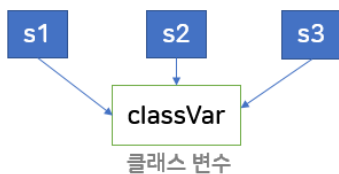
모든 인스턴스에서 따로 생성되므로 공유 불가
인스턴스 생성 시 만들어짐

- ♦ **지역 변수**

클래스 영역 이외에서 선언한 변수
선언된 영역 안에서만 유효 { }

- ♦ **클래스 변수 or static 변수**

선언된 클래스의 모든 인스턴스가 공유
어떤 인스턴스에도 속하지 않고 메모리 공간에 딱 하나만 존재하는 변수
클래스가 메모리에 올라갈 때 만들어짐



인스턴스 이름 or 클래스 이름으로 접근함
s1.classVar++;
Car.classVar++;

2) 클래스 메소드(static 메소드)

```
class Car
{
    static void method()
    {
        ....
    }
}
```

클래스 이름으로 접근
Car.method();

인스턴스 메소드로 할 필요가 없을 때
공통으로 같이 사용할 때
인스턴스를 생성하지 않아도 접근 가능

5. 메소드 오버로딩(Method Overloading)

```
class A
{
    void method(int v1) { .... }
    void method(int v1, int v2) { .... }
    void method(int v1, double v2) { .... }
}
```

- ♦ 조건
 - 같은 이름의 메소드
 - 매개 변수의 타입, 개수가 다른 경우

- 매개 변수의 타입과 개수에 맞는 함수를 자동으로 호출

※ 생성자도 메소드 오버로딩에 속함

1-2

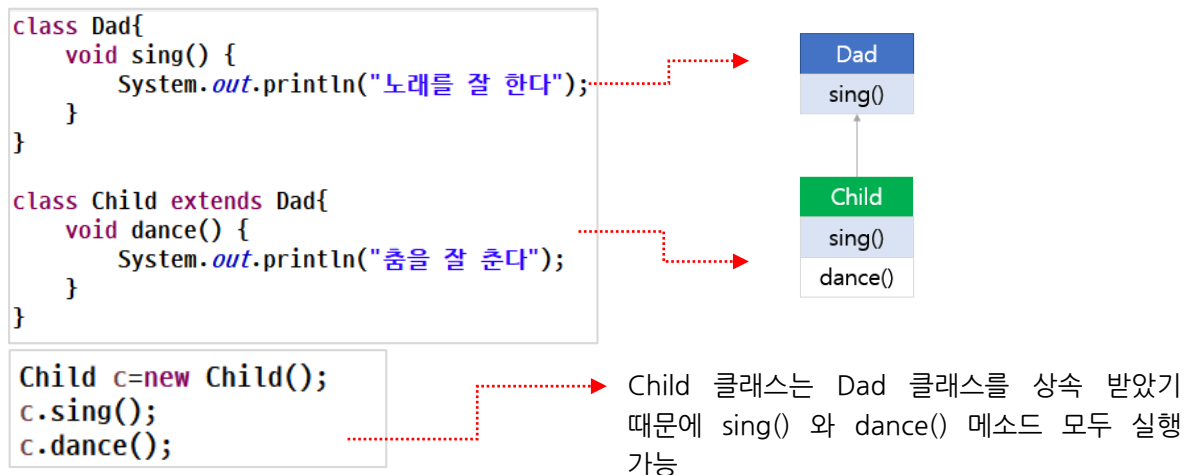
상속(Inheritance)

1. 상속(Inheritance)

1) 의미

- 이미 만들어진 클래스의 멤버 변수와 메소드를 그대로 이어 받아서 필요한 부분만 재정의하거나 추가하여 새로운 클래스를 만드는 것
- 상속을 해 주는 클래스 : 부모 클래스, 슈퍼 클래스, 상위 클래스
- 상속을 받는 클래스 : 자식 클래스, 서브 클래스, 하위 클래스

2) 단일 상속



<결과>

노래를 잘 한다
춤을 잘 춘다

※ 자바의 클래스는 단일 상속만 지원한다.

2. 메소드 오버라이딩(Method Overriding)

1) 의미

- 자식 클래스에서 부모 클래스에 있는 메소드를 다시 정의하여 부모 클래스에 있는 메소드와 다른 기능을 하도록 하는 것. 일종의 기능 upgrade
- 만일 자식 클래스에서 메소드 오버라이딩을 했을 때 오버라이드를 한 부모의 메소드를 사용하고 싶다면 자식 클래스에서 super.메소드() 를 이용하여 호출

2) 메소드 오버라이딩의 예

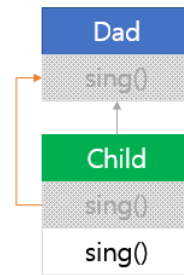
```
class Dad2{
    void sing() {
        System.out.println("노래를 잘한다");
    }
}

class Child2 extends Dad2{
    void sing() {
        System.out.println("랩을 잘한다");
    }
}

public class Exam0verriding {
    public static void main(String[] args) {

        Child2 c1=new Child2();
        c1.sing();

        Dad2 c2=new Child2();
        c2.sing();
    }
}
```



메소드 오버라이딩은 부모의 메소드를 가리는 것

Child 클래스에서 sing() 메소드를 오버라이딩 함으로써 Dad 클래스의 sing() 메소드를 가리고 Child 클래스에서 재정의한 sing() 메소드의 내용으로 명령이 실행

<결과>

랩을 잘한다
랩을 잘한다

※ 메소드 오버로딩 vs 메소드 오버라이딩

- 메소드 오버로딩 : 같은 이름의 매개변수가 다른 메소드를 정의하는 것
 - 메소드 이름 동일
 - 매개 변수의 개수, 타입이 달라야 함
 - 반환 타입은 무관
- 메소드 오버라이딩 : 상속받은 메소드 내용을 변경하는 것

1-3

추상 클래스, 인터페이스

1. 추상 클래스(Abstract Class)

```
abstract class Shape{  
    abstract double area();  
}  
  
class RectShape extends Shape{  
    int r;  
  
    RectShape(int r){  
        this.r=r;  
    }  
  
    double area() {  
        return r*r;  
    }  
}  
  
RectShape rect1 = new RectShape(5);  
System.out.println(rect1.area());
```

Shape : 추상 클래스
area() : 추상 메소드

1) 추상 클래스(Abstract Class)

구체적으로 구현되지 않고 이런 클래스를 만들 것이라고 정의만 해 놓은 것
추상 클래스는 new 연산자로 객체를 생성할 수 없음

2) 추상 메소드(Abstract Method)

함수의 몸체가 없음

※ 추상 메소드를 가진 모든 클래스를 추상 클래스라 한다.

추상 클래스로부터 상속받는 경우 추상 메소드를 구현해야 한다.

2. 인터페이스(Interface) - 다중 상속

인터페이스는 추상 메소드 집합으로 구현된 것이 하나도 없는 설계도, 껍데기

인터페이스는 인스턴스 생성이 불가능

인터페이스를 상속하는 것은 상속이 아닌 구현 implementation

인터페이스를 통해 다중 상속의 모습을 가지나 실제로 완전한 다중 상속은 아님

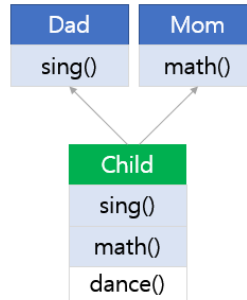
```
interface Dad1{
    public void sing1() ;
}

interface Mom1{
    public void math1();
}

class Child1 implements Dad1,Mom1{
    @Override
    public void sing1() {
        System.out.println("노래를 잘 부른다");
    }
    @Override
    public void math1() {
        System.out.println("수학을 잘 한다");
    }

    void dance1() {
        System.out.println("춤을 잘 춘다");
    }
}
```

```
Child1 c=new Child1();
c.sing1();
c.math1();
c.dance1();
```



※ 인터페이스를 상속한 경우 반드시 추상 메소드를 구현해야 한다.

※ 추상 클래스와 인터페이스 차이 : 추상 클래스는 일반 클래스인데 추상 메소드를 가지고 있는 것(멤버 변수나 생성자 등이 있음), 인터페이스는 추상 메소드만 가지고 있는 것

※ 어노테이션 : @Override

자바에서 어노테이션은 코드를 부연 설명

어노테이션은 어떠한 코드에 대해 컴파일러에게 특정 부분을 설명해 주는 역할(@Override)

컴파일 에러가 발생하는 것은 아님

코드에 대한 부가 설명 및 컴파일 단계에서 도움을 주는 역할