추상 클래스와 인터페이스 그리고 enum

Java 프로그램의 기본 구조는 클래스이지만 필요에 따라서는 추상클래스, 인터페이스 그리고 열거타입(enum)이라는 클래스와는 다른 구조와 활용 방법을 지원하는 Java 프로그램의 요소가 존재한다.

추상 클래스

새로운 클래스를 만들 때 기존 클래스의 활용 방법은 has-a 관계에 적용되는 객체(인스턴스)생성 구문과 is-a 관계에 적용되는 상속 구문이 있다. 그래서 클래스들은 상속과 객체 생성을 모두 지원하는 것이 일반적이지만 필요에 따라서는 객체 생성은 할 수 없고 상속으로만 활용되는 클래스를 개발하고자 할 때 적용되는 것이 바로 추상클래스이다.

■ 추상 메서드와 추상 클래스의 정의

클래스는 객체를 만들어내기 위한 코드를 담고 있는 설계도라고 할 수 있으며, 추상 클래스는 다음과 같은 활용 방법을 적용해야 하는 **미완성 설계도**이다.

미완성 설계도 → **상속이라는 방법을 사용하여 완성해서 사용해야 한다.**

추상클래스는 객체 생성은 할 수 없으며 상속으로만 활용해야 하는 클래스로서 클래스의 멤버로 추상 메서드가 존재하는 경우에는 반드시 추상 클래스로 만들어야 한다.

▶ 추상 메서드

메서드는 선언부와 구현부로 나뉘며 선언부에는 메서드의 호출 스펙 즉, 메서드명, 매개변수 사양, 리턴값의 타입 등을 정의하는 부분이며 구현부에는 이 메서드가 호출되었을 때 수행하게 되는 코드를 작성한다.

추상 메서드는 메서드의 선언부만 정의되어 있고 구현부가 없는 메서드로서 다음과 같이 메서드의 구현부를 블록({})대신 세미콜론(;)으로 정의해야 하고 메서드의 선언부에 abstract을 지정해주어야 한다. 추상 메서드는 수행 코드가 비어 있는 메서드로서 미완성 메서드라고 할 수 있다.

```
// 주석을 통해 이메서드의 기능이 무엇인지만 알린다.
abstract  리턴값의타입  메서드명([ 매개변수 선언 … ]);
```

메서드를 정의할 때 수행 코드가 비어 있는 미완성 상태의 메서드로 만드는 이유는 상속받는 자손에 따라서 수행코드가 달라져야 하는 메서드를 정의하고자 하는 경우이다. 조상 클래스에서는 메서드의 선언부만 작성해 놓고 "어떠한 기능이다"는 것만 정해 놓고 자손에서 그 기능을 어떻게 수행할 것인지 실제 상속받는 자손 클래스에 따라 구현하도록 할 때 사용되는 것이다.

```
abstract   void   draw() ;
abstract   int   read();
abstract   int   expr(int num1, int num2);
```

▶ 추상 클래스

추상 메서드를 하나라도 가지고 있는 클래스는 반드시 추상 클래스로 정의해야 하며 상속과 추상 메서드의 구현이라는 방법을 통해만 활용될 수 있다. 추상 클래스는 객체 생성은 불가능하고 상속으로만 활용되는 클래스로서 추상 클래스 안에는 0개 이상의 추상 메서드가 존재할 수 있다. 그러므로 추상 클래스를 상속하여 새로운 클래스를 만드는 경우에는 조상이 가지고 있는 추상 메서드들을 존재하는 만큼 모두 오버라이딩하여야 한다.

객체를 생성하는 것은 안되지만 추상 클래스 타입의 변수를 선언하는 것은 가능하며 이러한 참조형 변수에는 다음과 같이 자손 클래스의 객체를 대입하여 사용한다.

```
추상클래스   변수   =   new 자손클래스();
```

추상 클래스를 만드는 목적은 다른 클래스들의 작성에 도움이 되도록 하기 위해서이며 다형성을 통해 그 기능을 완성할 수 있게 된다.

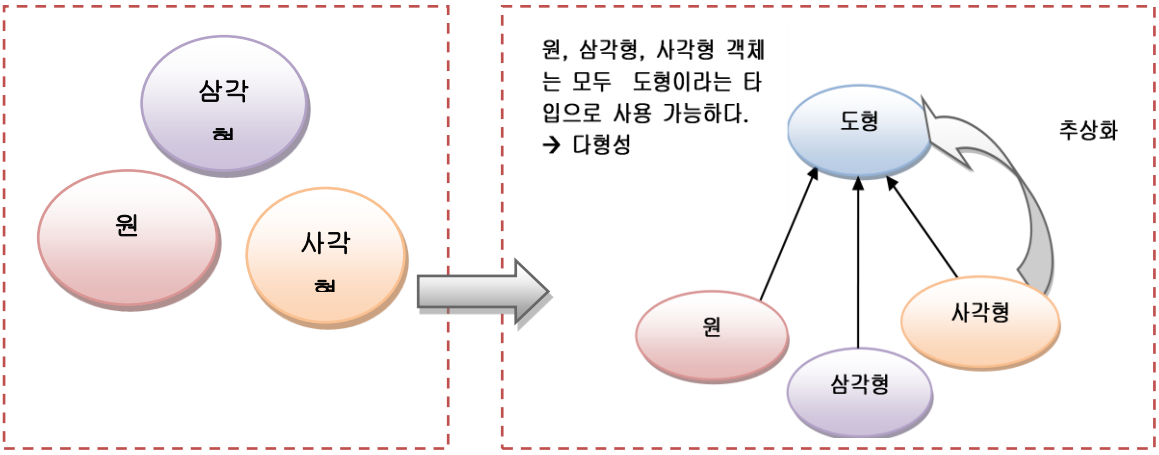
```
abstract class 클래스명 [extends 조상클래스명]{
   // 주석을 통해 이메서드의 기능이 무엇인지만 알린다.
  abstract   리턴값의타입   메서드명([ 매개변수 선언 … ]);
             :
}
```

상속이 자손 클래스를 만드는데 있어서 조상 클래스를 사용하는 것이라면 이와 반대로 추상화는 기존 클래스의 공통 부분을 뽑아내서 조상 클래스를 만드는 것이다.

상속을 통해서 클래스를 구현하고 확장하는 작업을 구체화라고 하며 클래스간의 공통점을 찾아내서 공통의 조상을 만드는 작업을 추상화라고 한다.

▶추상화

다음 그림과 같이 세 개의 객체에 대한 클래스를 설계한다고 할 때 각각의 클래스를 정의해도 되지만 좀 더 객체 지향(객체 지향의 장점을 잘 적용한)적으로 설계한다면 이 클래스들에 공통으로 정의되는 메서드들을 선택하여 도형이라는 클래스를 만들어 이 도형 클래스를 상속하여 구현한다.



이렇게 공통 조상 클래스를 두고 각각의 클래스를 정의함으로써 얻을 수 있는 장점은 원, 삼각형 그리고 사각형 객체를 전달받아서 기능을 수행하는 메서드의 경우에는 다음과 같이 각각의 메서드를 정의하지 않고 동형 타입을 전달받는 메서드 하나로 통일할 수 있다는 것이다. 자손 타입의 객체는 조상 타입으로 전달될 수 있다는, 조상 타입의 변수로 자손의 객체를 참조할 수 있다는 다형성이 적용되기 때문이다.

```
void printInfo(Circle   obj) {
    :
}
void printInfo(Rectangle   obj) {
    :
}
void printInfo(Triangle   obj) {
    :
}
```

```
void printInfo(Shape   obj) {
    :
}
```

각 타입을 매개변수의 타입으로 하는 메서드들을 각각 정의하는 대신 Shape 타입을 전달받는 메서드 하나로 정의 가능

공통으로 다루고자 하는 클래스들의 공통된 메서드들을 모아서 조상 클래스를 만들 때 자손마다 다르게 구현되어야 하는 메서드가 있다면 이러한 메서드를 바로 추상 메서드로 정의하며 이러한 메서드를 가지는 클래스는 추상 클래스로 정의하게 되는 것이다.

다음은 Circle과 Rectangle클래스의 공통 기능으로 Shape 이라는 추상클래스를 정의하고 이 클래스를 상속하여 구현하는 Circle과 Rectangle클래스 예제이다.

```
abstract class Shape {       // 추상 클래스 정의
    String color;
    abstract void draw();    // 구현부분이 없는 추상 메서드 정의
    void setColor(String color) {
        this.color = color;
    }
}
class Circle extends Shape {
    void draw() {            // 조상의 추상 메서드를 오버라이딩한다.
        System.out.println(color + " 원을 그리는 기능");
    }
}
class Rectangle extends Shape {
    void draw() {            // 조상의 추상 메서드를 오버라이딩한다.
        System.out.println(color + " 사각형을 그리는 기능");
    }
}
```

다음은 Shape타입의 변수에 Circle과 Rectangle객체를 대입할 수 있고 메서그 호출 시 Shape타입의 매개변수에 전달할 수 있다는 것을 보여주는 예제이다.

```
Shape obj = new Shape();      // 오류 : 추상클래스는 객체 생성 불가능
Shape obj = new Circle();      // Circle객체는 조상인 Shape타입 변수에 대입 가능
Shape obj = new Rectangle(); // Rectangle객체는 조상인 Shape타입 변수에 대입 가능
printInfo(new Circle());

void printInfo(Shape   obj) {
    :
}
```

인터페이스

인터페이스는 일종의 추상클래스이다. 추상 클래스와 다른 점은 구현이라는 방법으로 상속을 적용하기 때문에 다중 상속이라는 특징을 적용할 수 있다는 것이다. 인터페이스는 규격화된 프로그램 또는 표준화된 프로그램을 개발할 수 있도록 하며 객체와 객체간에 결합도가 낮은 프로그램을 개발할 수 있도록 하여 유지보수성이 좋은 프로그램 개발을 지원한다.

■ 인터페이스의 정의

인터페이스를 작성하는 것은 클래스를 작성하는 것과 크게 다르지 않다. 클래스를 정의할 때는 키워드로 class를 사용하지만 인터페이스를 정의할 때는 interface라는 키워드를 사용하며 멤버로는 상수와 추상 메서드만을 정의할 수 있다.

```
interface 인터페이스명 [extends 조상인터페이스명] {
    public static final 타입 상수이름=값;  // 상수
    public abstract 리턴타입 메서드명([매개변수 선언]);
}
```

인터페이스에 선언되는 변수

public static final 형이어야 하며 명시적 초기화를 하고 있어야 한다. public static final 중에서 하나라도 생략되면 자동으로 추가된다.

인터페이스에 선언되는 메서드

public abstract 형이어야 하며 public abstract 중에서 하나라도 생략되면 자동으로 추가된다.

인터페이스에 선언되는 생성자

인터페이스는 클래스와는 다르게 생성자를 정의하지 않으며 객체 생성은 불가능하다.

인터페이스 상속

기존 인터페이스를 상속하여 새로운 인터페이스 정의하는 것이 가능하며 클래스의 경우와 다르게 여러 인터페이스를 다중 상속하여 새로운 인터페이스를 구현하는 것도 가능하다. 그리고 인터페이스는 클래스의 Object와 같은 최상위 인터페이스라는 것은 존재하지 않는다.

다음은 인터페이스의 정의 예이다. 상수 선언, 메서드 정의 모두 각각에 대해서 정해진 제어자가 생략되면 자동적으로 추가된다.

```
interface   Printable {
    public static final int   PRINT_TYPE1 = 1;
    static final int   PRINT_TYPE2 = 2;        // 자동으로 public 이 추가됨
    final int   PRINT_TYPE3 = 3;               // 자동으로 public staticl 이 추가됨
    int   PRINT_TYPE4 = 4;                     // 자동으로 public static final 이 추가됨

    public abstract void print(int type);
    abstract void setPage(int su);             // 자동으로 public 이 추가됨
    boolean isAvailable();                     // 자동으로 public abstract 이 추가됨
}
```

다음은 기존 인터페이스를 상속하여 새로운 인터페이스를 정의하는 예이다.

```
interface   Inter1 {
    void expr1();
}
```

```
interface   Inter2 {
    void expr2();
}
```

```
interface   Inter3   extends Inter1, Inter2{
    void expr3();
}
```

위의 그림과 같이 인터페이스가 정의되어 있는 경우 Inter3이 구현하는 클래스는 expr1(), expr2() 그리고 expr3()를 모두 오버라이딩 하여 구현해야 하며 하나라도 구현하지 않으면 생성되는 클래스는 추상 클래스로 정의해야 한다.

■ 인터페이스의 활용

인터페이스는 객체 생성은 될 수 없지만 인터페이스 타입의 변수는 선언할 수 있다. 인터페이스 타입의 변수에는 인터페이스를 구현하고 있는 자손 클래스의 객체를 대입하여 참조하도록 한다. 인터페이스의 활용에서도 다형성 구문을 기반으로 한다.
인터페이스는 추상 클래스와 동일하게 상속이라는 방법으로만 활용할 수 있다. 새로운 클래스를 만들 때 기존 클래스를 상속하는 것은 extends 절을 사용하지만 인터페이스를 상속하여 구현하는 경우에는 implements절을 사용한다.

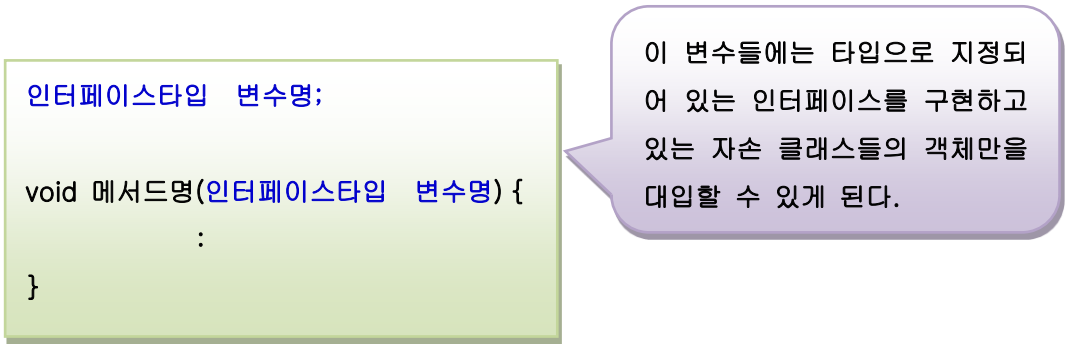
```
class  클래스명  implements  인터페이스명[, 인터페이스명 ..] {
                :
}
```

implements절이라는 또 다른 절을 사용하므로 클래스를 상속하면서 인터페이스들을 추가로 상속하여 기능을 추가할 수 있다. 인터페이스를 상속한 클래스의 경우에는 인터페이스에서 정의하고 있는 추상 메서드들을 오버라이딩하여 기능을 구현해야 하며 하나라도 오버라이딩을 하지 않으면 추상 클래스가 되어야 한다. 인터페이스를 상속한다는 것은 인터페이스에 정의된 추상 메서드를 구현하고 인터페이스 타입의 역할도 추가로 지원한다는 것을 뜻한다.

```
class Triangle extends Shape implements Printable{
    void draw() {              // 조상의 추상 메서드를 오버라이딩한다.
        System.out.println("삼각형을 그리는 기능");
    }
    public void print() {
        :
    }
    // 인터페이스의 메서드들은 자동으로 public 이 되므로 오버라이딩할 때
    // 반드시 public 제어자를 지정해야 한다.
    public void setPage(int page) {    …
    }
    public boolean isAvailable() {    …
    }
}
```

인터페이스를 상속하여 구현하는 클래스는 객체 생성되어 인터페이스 타입의 변수에 대입되어 참조될 수 있다. 즉, 인터페이스의 경우에도 자손의 객체는 조상 타입으로 참조될 수 있다는 참조형 변수의 다형성이 적용된다.

지원되는 기능에 관계없이 정해진 스펙의 메스드들을 멤버로 갖는 클래스들의 객체들만을 처리하고자 하는 경우에는 정해진 스펙의 메서드들을 인터페이스로 정의한 다음 이 인터페이스 타입의 변수를 선언하여 객체를 전달받도록 구현하면 그 자리에는 해당 인터페이스를 구현하는 자손의 객체들만 전달될 수 있으므로 요구하는 스펙의 메서드들을 멤버로 갖는 클래스들의 객체만을 전달받고자 하는 요구 사항을 만족하게 된다.



열거타입(enum)

열거타입은 한정된 데이터 값만을 저장할 수 있는 타입을 정의하는데 사용되는 또 다른 Java 프로그램의 구조로서 상수 정의를 목적으로 구현하는 클래스이다. 데이터 값을 비교할 때 타입까지 적용하고자 할 때 사용되며 열거타입으로 선언된 변수에는 열거 타입에 정의된 상수값 만을 데이터 값으로 저장할 수 있다. Java SE 5.0 에서 새로이 추가된 구문으로 OCJP 6.0시험에서 다양한 형태의 문제에서 열거타입이 출제되고 있다.

.

■ **열거타입의 도입 배경**

프로그램에서 처리하고자 하는 데이터들 중에는 한정된 값만을 다루게 되는 경우가 있다. 요일에 대한 데이터의 경우 월,화,수...일 처럼 7개의 값을, 계절에 대한 데이터의 경우 봄, 여름, 가을 겨울이라는 4개의 값을 갖는 처리하게 된다. 이런한 데이터들은 정수나 문자 또는 문자열 리터럴을 이용해서 표현하거나 상수를 선언해서 사용하기도 하였다.

```
public class Season{
    public final static int SPRING = 1;
    public final static int SUMMER = 2;
    public final static int FALL = 3;
    public final static int WINTER = 4;
}
```

Season클래스에 선언되어 있는 SUMMER상수를 참조하기 위한 방법은 다음과 같다.

System.out.println(Season.SUMMER); → 2 가 출력된다.

Season클래스의 소스 내용을 파악하지 못했다면 2 라는 숫자가 어떠한 계절을 의미하는지 알 수 없다. 하여 Java SE 5.0에서는 이러한 단점을 보완하기 위해 열거타입이 Java의 기본 구문으로 추가되었다.

열거타입은 클래스나 인터페이스처럼 직접 만들어서 사용해야 하며 클래스로 취급되기 때문에 열거타입의 이름도 클래스명의 작성 관례에 따라 만들어야 한다.

■ 열거타입의 정의와 사용

열거타입을 정의하는 형식은 다음과 같다.

```
enum 열거타입이름{
    열거상수리스트(콤마로 구분)
}
```

예를 들어 사계절을 열거타입으로 표현하면 다음과 같다.

```
enum Season{
    SPRING, SUMMER, FAIL, WINTER
}
```

```
Season   info = Season.SUMMER;
System.out.println(Season.SPRING);
```

클래스에 종속적인 열거타입을 정의할 수도 있다

```
class 클래스명 {
    enum 열거타입이름{
        열거상수리스트(콤마로 구분)
    }
    :
}
```

클래스의 멤버로 정의되는 열거타입은 정적 내부클래스로 만들어지므로 외부에서 클래스에 종속적인 열거타입을 사용하려면 **클래스명.열거타입이름.열거상수이름** 으로 사용한다.

```
class Tour  {
    enum Season{
        SPRING, SUMMER, FALL, WINTER
    }
}
```

```
Tour.Season   info = Tour.Season.SUMMER;
System.out.println(Tour.Season.SPRING); // SPRING 이 출력된다.
```

■ values() 메서드와 valueof() 메서드

values()메서드와 valueof() 메서드는 열거타입이 컴파일될 때 자동으로 생성되는 클래스 메서드들이다. values()메서드는 열거타입에 속하는 모든 열거 상수들을 배열에 담아서 리턴하고 valueOf()메서드는 열거상수의 이름을 문자열로 넘겨주면 그에 해당하는 열거상수를 리턴하는 메서드이다.

```
enum Season{
    SPRING, SUMMER, FALL, WINTER
}

class SeasonTest {
    public static void main(String args[]) {
        Season day[] = Season.values();
        for(Season value   : day)
        System.out.println(value);      // SPRING, SUMMER, FALL, WINTER 출력

        Season season = Season.valueOf("SUMMER");
        System.out.println(season);      //SUMMER 출력
    }
}
```

■ 열거상수를 다른 값과 매핑하기

열거타입에 정의되는 상수들은 상수이름을 값으로 자동초기화 하지만 열거상수를 다른 값과 매핑하기 위해서는 상수 뒤에 괄호와 함께 값을 설정할 수 있다. 그런데 이 때에는

- 설정하려는 값을 저장하기 위한 private final 형 멤버변수
- 값을 설정할 수 있는 생성자 메서드
- 값을 리턴하는 메서드

를 같이 추가해 주어야 한다.
다음은 열거타입 Season에 정의되는 상수들 SPRING, SUMMER, FALL 그리고 WINTER 에 각각 "봄", "여름", "가을" 그리고 "겨울" 을 값으로 매핑하도록 구현하고 있는 열거타입 정의 예제이다.

```
enum Season{
    SPRING("봄"), SUMMER("여름"), FALL("가을"),   WINTER("겨울")
    private final String name;
    Season(String name){
        this.name = name;
    }
    String returnName(){
        return name;
    }
}
```

.

학습정리

- 추상클래스는 객체 생성은 할 수 없으며 상속으로만 활용해야 하는 클래스로서 클래스의 멤버로 추상 메서드가 존재하는 경우에는 반드시 추상 클래스로 만들어야 한다.

- 추상 메서드는 메서드의 선언부만 정의되어 있고 구현부가 없는 메서드로서 다음과 같이 메서드의 구현부를 블록({})대신 세미콜론(;)으로 정의해야 하고 메서드의 선언부에 abstract 을 지정해주어야 한다.

- 상속이 자손 클래스를 만드는데 있어서 조상 클래스를 사용하는 것이라면 이와 반대로 추상화는 기존 클래스의 공통 부분을 뽑아내서 조상 클래스를 만드는 것이다.

- 인터페이스는 일종의 추상클래스이다. 추상 클래스와 다른 점은 구현이라는 방법으로 상속을 적용하기 때문에 다중 상속이라는 특징을 적용할 수 있다는 것이다.

- 인터페이스를 정의할 때는 interface라는 키워드를 사용하며 멤버로는 상수와 추상 메서드만을 정의할 수 있다.

- 인터페이스는 객체 생성은 될 수 없지만 인터페이스 타입의 변수는 선언할 수 있다. 인터페이스 타입의 변수에는 인터페이스를 구현하고 있는 자손 클래스의 객체를 대입하여 참조하도록 한다.

- 인터페이스는 추상 클래스와 동일하게 상속이라는 방법으로만 활용할 수 있다. 새로운 클래스를 만들 때 기존 클래스를 상속하는 것은 extends절을 사용하지만 인터페이스를 상속하여 구현하는 경우에는 implements절을 사용한다.

- 열거타입은 한정된 데이터 값만을 저장할 수 있는 타입을 정의하는데 사용되는 또 다른 Java 프로그램의 구조로서 상수 정의를 목적으로 구현하는 클래스이다.

- 열거타입으로 선언된 변수에는 열거 타입에 정의된 상수값만을 데이터 값으로 저장할 수 있다.