Java 프로그래밍 길잡이

✓ 원리를 알면 IT가 맛있다

Java Programming for Beginners

chapter 07.

인터페이스와 추상클래스

❑ 학습목표

- 인터페이스의 용도 및 사용법을 학습한다.
- 추상클래스의 용도 및 사용법을 학습한다.
- 중첩클래스의 종류 및 사용법을 학습한다.

❑ 1] 추상 클래스와 인터페이스

앞에서 배운 상속은 객체지향 프로그래밍의 핵심기능으로서 상속을 적용하면 코드의 재사용 및 다형성, 오버라이딩 메소드등과 같은 객체지향적인 프로그램 기법을 적용할 수 있다. 하지만 강력한 상속을 적용시켜도 하위 클래스에서 부모의 메소드를 상속받아서 사용하지 않고 자신만의 메소드를 작성하여 사용한다면 상속을 사용하는 장점을 얻을 수 없다. 상속은 강제성이 없기 때문이다.

따라서 객체지향 특징인 재사용성 및 유지보수를 향상시키기 위해서 하위 클래스에서 반드시 부모 클래스의 메소드를 사용하게끔 강제할 필요성이 등장하게 되며 자바에서는 인터페이스와 추상 클래스를 통해서 하위 클래스들에게 부모의 메소드를 반드시 사용하게 강제할 수 있다.

강제를 함으로써 통일성 및 일관성이 지켜질 수 있으며 결국에는 재사용성 및 유지보수가 향상되고 관리하기도 쉬워진다. 인터페이스란 용어자체가 '창구 역할'을 하는 것을 지칭하기 때문에 여러 창구(메서드)가 존재하는 것보다는 통일된 창구(부모에서 정의된 메서드)를 통해서 의사 소통하는 것이 더욱 효율적이다.

❑ 1] 추상 클래스 [abstract class]

- body가 없는 메소드를 포함 할수 있는 클래스를 추상클래스라고 한다.
- body가 없는 메소드를 추상 메소드(abstract method)라고 하며 abstract 키워드를 사용하여 다음과 같이 표현된다.

```
public abstract void 메소드명([인자]);
```

- 추상 메소드를 body가 있는 메소드(concrete method)로 만들기 위한 방법은 추상 클래스를 상속받은 하위 클래스에서 오버라이딩 메소드를 통해서 가능하다. (상속 이용)
- 추상 메소드를 하나라도 포함하면 반드시 추상 클래스로 작성해야 되며 abstract 키워드를 사용하여 다음과 같이 표현한다.

```
public abstract  class  클래스명 { }
```

추상 클래스 특징

- 미완성 클래스이다.
- 구성요소는 다음과 같다.
  :인스턴스 변수
  :일반 메소드 (concrete method)
  :생성자
  :추상 메소드 (abstract method)

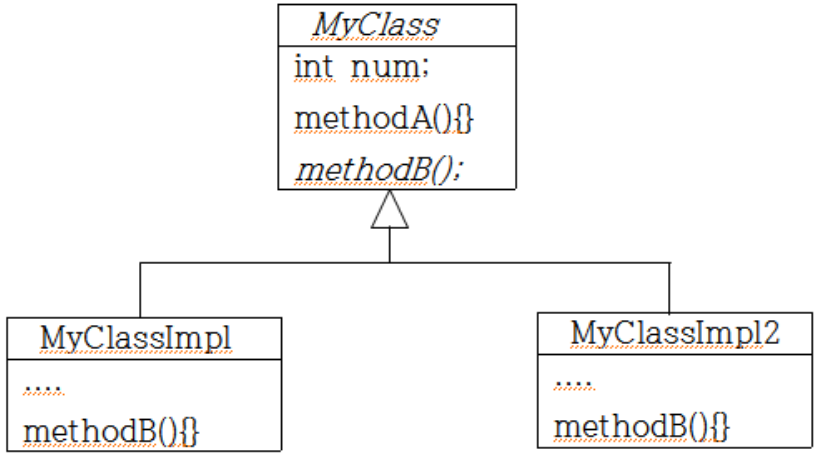
- 추상 메소드를 포함할 수 있기 때문에 객체생성이 불가능하다. 추상 클래스를 사용하기 위해 일반 클래스를 이용하여 추상 메소드를 오버라이딩 한다.
- 추상 클래스도 클래스이기 때문에 단일상속만 지원된다.
- 강제성 및 통일성을 제공한다. 하위 클래스에게 상속의 장점인 부모의 변수와 메소드를 선언 없이 사용할 수도 있고, 또한 특정 메소드만 강제할 수도 있다.

❑ 1] 추상 클래스 [abstract class]

- abstract 는 UML 표기법으로 이탤릭체를 사용한다.
- 변수의 데이터 형으로 사용 가능하다.

문법:

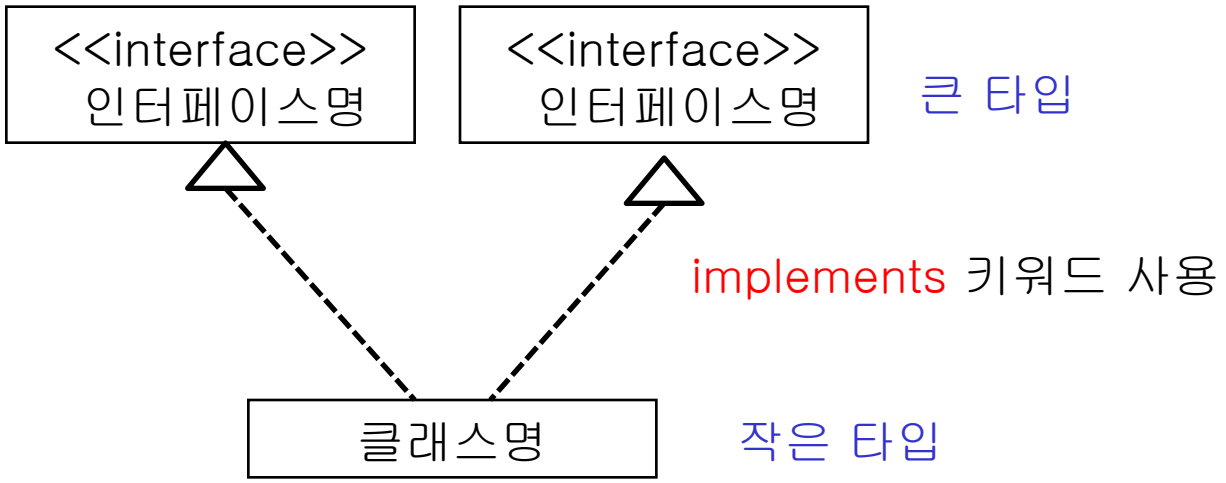
```
public abstract class 클래스명{
        //인스턴스 변수
        // concrete 메소드
       // 생성자
       // 추상메소드 ( abstract method )
 }
```



특징:

- 확장자는 *.java 이다.
- 구성요소는 상수 및 추상메소드(abstract method)만 갖는다.
  (java8에서는 default 메서드와 static 메서드 사용 가능)
  상수는 public static final 이 자동으로 지정된다.
  추상메소드는 public abstract 가 자동으로 지정된다.
- 객체생성불가 (new 사용 불가)
  객체생성을 못하면 구성요소 사용이 불가능하다. 독자적으로 사용 못하고 클래스를 이용해서 사용된다. 따라서 인터페이스와 클래스간에 관계가 이루어진다. (준 상속관계인 구현관계이다.)
  implements 키워드를 이용하여 구현관계를 코드에서 표현한다. 구현관계가 성립되면 클래스는 반드시 인터페이스의 추상 메소드를 concrete 메소드로 구현해야 된다. (오버라이딩 메소드 규칙을 따른다.)
  이것이 인터페이스를 사용하는 이유이다.
  하위클래스에게 특정 메소드를 강제적으로 사용하게끔 하고자 할 때 이다. (강제성과 통일성 확보)

❑ 2) 인터페이스 [interface]



- 다중 구현이 가능하다.
- 타입으로는 인터페이스가 큰 타입이고 클래스가 작은 타입이다.(다형성 적용 가능)
- 인터페이스끼리 상속이 가능하다. (extends 이용 , 다중상속 가능)
- 변수의 데이터형으로 사용 가능.
- 추상 메서드는 UML 표기법으로 이탤릭체로 표현한다.
- 인터페이스의 추상 메서드는 abstract 키워드 생략 가능하다.

  (추상 클래스의 추상 메서드는 abstract 키워드 생략 불가)

❑ 2) 인터페이스 [interface]

문법:

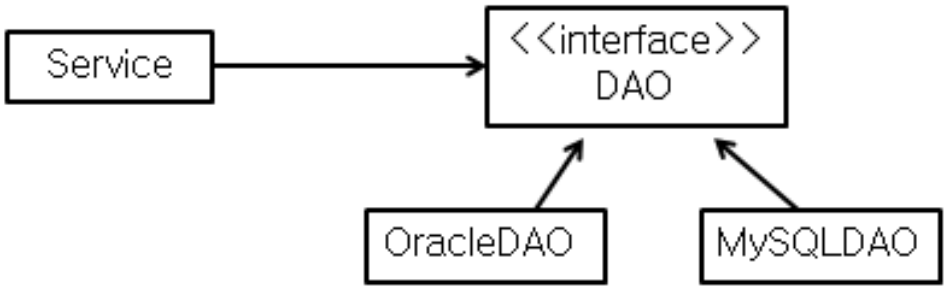
```
public  interface   인터페이스명 {
    //  public static final 로 지정한 상수
    //  public abstract 로 지정한 추상메소드
  }
```

예>

```
public interface Flyer{
        public void  takeoff();
        pubilc  abstract void  fly();
        public void  land();
    }
    public class  Bird  implements  Flyer {
        public void  takeoff(){}
        pubilc void  fly(){}
        public void  land(){}
    }
```

❑ 2) 디커플링(decoupling)

- 인터페이스를 이용한 디커플링(decoupling)
  - 커플링이란 한쌍을 일컫는 말이다. 동사로는 '두 대상을 결합한다'라는 의미로 사용된다.
  - 프로그래밍에서 디커플링이란 '모듈 사이의 연결고리(의존관계)를 인터페이스를 사용하여 두 모듈의 의존성을 감소하여 개발 및 유지보수가 용이하도록 처리하는 방법'을 의미한다. 즉, 모듈사이를 인터페이스로 연결하고 구현체를 분리하는 것을 말한다.
  - 객체들간에 결속력이 강하면, 그 프로그램의 구성 요소중 하나를 수정하게 되면 그와 연관된 모든 구성 요소를 새로 수정해야 된다. 그러나 인터페이스를 이용한 설계가 잘 된 경우에는 수정하고자 하는 요소만 수정하여 재배포 하면 된다.



❑ 2) 디커플링(decoupling)

- 유지보수가 어려운 설계

```
1.  class OracleDao  {
2.      public void insert(){System.out.println("OracleDao");}
3.  }
4.
5.  class MysqlDao  {
6.      public void insert(){System.out.println("MysqlDao");}
7.  }
8.
9.  class Service {
10.     public OracleDao dao;
        // Service에서 사용하는 dao가 MysqlDao 로 변경되면 Service클래스의 내용도 변경된다.
11.
12.     public void execute() {
13.         dao = new OracleDao();
14.         dao.insert();
15.     }
16. }
```

❑ 2) 디커플링(decoupling)

- 인터페이스 기준 설계

```
1.  interface Dao {
2.      void insert();
3.  }
4.
5.  class OracleDao implements Dao {
6.      @Override
7.      public void insert(){System.out.println("OracleDao");}
8.  }
9.
10. class MysqlDao implements Dao {
11.     @Override
12.     public void insert(){System.out.println("MysqlDao");}
13. }
14.
15. class Service {
16.     // Service에서 Dao인터페이스를 사용
17.     //어떤 구현체(OracleDao MysqlDao)를 사용 할것인지 Servcie에서 결정하지 않음
18.     private Dao dao;
19.
20.     public void execute() {
21.         dao.insert();
22.     }
23.
24.     public void setDao(Dao dao) {
25.         this.dao = dao;
26.     }
27. }
```

```
29. class MainTest {
30.     public static void main(String[] args) {
31.         Service service = new Service();
32.         service.setDao(new OracleDao()); // service.setDao(new MysqlDao());
33.         // 사용하는 시점에 어떤 구현체(OracleDao MysqlDao)를 사용할것인지가 결정된다.
34.         service.insert();
35.     }
36. }
```

❑ 3] 중첩 클래스 [nested class]

정의:

클래스 안에 또 다른 클래스가 정의될 수 있다.
이런 형태의 클래스를 '중첩 클래스' 라고 한다.

문법:

```
public class  Outer{

    class  Inner{

     }//end  Inner

}//end Outer
```

❑ 3] 중첩 클래스 [nested class]

특징:

- Inner 클래스는 Outer클래스의 멤버처럼 동작된다.

즉, Outer클래스가 생성되어야 Inner클래스를 사용할 수 있다.

결국 Outer클래스를 통해서 Inner 클래스에 접근할 수 있다는 것이다.

- Inner 클래스는 Outer 클래스의 멤버(변수/메소드)를 마치 자신의 멤버처럼 사용 가능하다. (private 일지라도…)
- Inner 클래스 안에는 static 변수를 사용할 수 없다. 단, static Inner 클래스에서는 사용 가능하다.
- 소스파일을 컴파일 하면 Outer$Inner.class 형식으로 클래스파일이 생성된다.
- Inner 클래스가 Outer 클래스 안에 정의되는 위치에 따라서 4가지 종류가 제공된다.

종류	설명
member	**Outer 클래스의 멤버변수나 메소드처럼 클래스가 정의된 경우이다.**
local	**Outer 클래스의 특정 메소드안에서 클래스가 정의된 경우이다.**
static	**static 키워드를 이용해서 클래스가 정의된 경우이다.**
anonymous	**익명 클래스를 이용해서 클래스가 정의된 경우이다.**

❑ 3) member Inner 클래스

특징:

Outer 클래스의 멤버처럼 클래스가 정의된 경우이다.

즉, 객체를 생성해야만 사용할 수 있는 멤버들과 같은 형태이기 때문에 반드시 Outer 클래스를 객체생성 후에 Inner 클래스를 사용할 수 있다.

```
public class Outer {
..
    class Inner{
      …
     }//end Inner
}//end Outer
```

```
Outer xxx = new Outer();

Outer.Inner yyy = xxx.new Inner();
```

❑ 3) local Inner 클래스

특징:

Outer 클래스의 메소드 안에서 정의된 경우이다.

메소드안에서 정의 되었기 때문에 로컬변수 처럼 동작된다. 메소드가 호출될때 생성되고 메소드가 종료될 때 삭제된다.

```
public class Outer {
..
   public void outerMethod(){
....     int x = 10;  // 상수화
         class Inner{
...
         }//end Inner
    }//end outerMethod
}//end Outer
```

Inner 클래스에서 접근 가능한 변수는 Outer클래스의 멤버변수와 상수만 가능하다. outerMethod내의 로컬변수는 접근만 가능하고 수정 불가(상수화)

Inner클래스의 객체생성은 outerMethod 메소드내에서 한다.

❑ 3) static Inner 클래스

특징:

static 을 이용하여 inner클래스를 정의한 경우이다.

일반 Inner클래스는 static변수를 포함할 수 없지만 static Inner클래스는 가능하다. 반면에 Outer클래스의 멤버변수는 접근이 불가능하다.

```
public class Outer {
..
    static class Inner{
      …
     }//end Inner
}//end Outer
```

```
Outer.Inner yyy = new Outer.Inner();
```

❑ 3) anonymous Inner 클래스

특징:

이름이 없는 Inner 클래스를 의미한다.

일반적으로 인터페이스 또는 추상클래스를 구현하는 클래스를 이용할 때 주로 사용된다. (GUI 이벤트 처리시 많이 사용 됨)

예> // Flyer는 인터페이스

```
Flyer  f =  new Flyer(){
    public void takeoff(){}
    public void fly(){}
    public void land(){}
};
f.takeoff();
f.fly();
f.land();
```

Thank you