

# 제13장

## 추상클래스와 인터페이스



## 5. 인터페이스(interface)의 개념

- 인터페이스는 일종의 추상클래스이긴 하나 멤버의 주류가 추상메서드이다.
- 다른 의미로는 **객체사용 설명서** 라고도 한다.
- 실제로 구현된 것은 아무 것도 없는 설계도라고 보면 된다.
- 인터페이스의 멤버
  - 상수(static final) - 부수적 개념
  - **추상 메서드 - 인터페이스의 본질**
  - 정적 메서드(JDK1.8)
  - 디폴트 메서드(JDK1.8)
- **생성자가 없다. 하여, 인스턴스를 만들 수 없다.**
- 클래스 작성에 도움을 준다.
- 표준을 제시하여, 그 규칙에 맞게끔 구현하도록 한다.

**ex) 공업표준을 정해놓고, 각 회사의 환경에 맞게끔 사용함.**



## 6. 인터페이스의 작성

- class대신 interface로 사용하는 것 외, 클래스 작성과 동일하다.

```
public interface A {  
    int a = 10;    //static final속성(생략가능)  
    void method(); //abstract 속성(생략가능)  
}
```

- 하지만, 구성요소는 추상 메서드와 상수가 있으며, JDK1.8부터 정적 메서드, 디폴트 메서드가 추가되었다.
- 상수는 static final속성을 지니고 있으며, 추상 메서드는 abstract속성을 가진다. 또한 생략 가능(컴파일러가 알아서 추가해 줌)
- 인터페이스도 클래스와 동일하게 상속이 가능하다. (다중상속 가능)

```
interface B {}  
interface C {}  
public interface A extends B, C {  
}
```

- \* 통상적으로 인터페이스는 ~able(할 수 있는)로 선언되곤 한다.
- \* Object클래스와 같이 최고 조상이 없다.
- \* 아울러, 생성자가 없어서 인스턴스를 생성할 수 없다.

## 7. 인터페이스의 구현

▣ 인터페이스를 구현한다는 것은 클래스를 상속하는 것과 동일하다고 생각하자. 하여, 인터페이스도 일종의 조상이다.(다형성 개념이 적용됨)

▣ 단, **extends**를 사용하지 않고 **implements**를 사용한다.

```
public class HTMLParser implements Parseable {  
    @Override  
    public void parse(String fileName) {  
        System.out.println(fileName + "-HTML 구문 분석 완료!");  
    }  
}
```

▣ 구현클래스는 반드시 인터페이스에 선언되어 있는 추상 메서드를 재정의 해야 한다.

▣ 상속과 동시에 구현도 가능하다.

```
public class HTMLParser extends ParserManager implements Parseable {  
    @Override  
    public void parse(String fileName) {  
        System.out.println(fileName + "-HTML 구문 분석 완료!");  
    }  
}
```

## 8. 인터페이스의 다형성

- 인터페이스 타입의 참조변수로 인터페이스를 구현한 클래스의 인스턴스 참조 가능

```
public static void main(String[] args) {  
    //인터페이스의 필드의 다형성  
    A a = new Member();  
}
```

- 인터페이스를 매개변수 타입으로 설정할 수가 있다. 단, 매개변수로 올 수 있는 것은 인터페이스를 구현한 클래스만 올 수가 있다.(중요)

```
//인터페이스의 매개변수의 다형성  
public void method1(A a) {  
    System.out.println("method1() 호출됨");  
}
```

- 인터페이스를 리턴 타입으로 설정할 수가 있다. 단, 리턴타입으로 올 수 있는 것은 인터페이스를 구현한 클래스만 올 수가 있다.(중요)

```
//인터페이스의 리턴타입의 다형성  
public A method1() {  
    return new Member();  
}
```

## 9. 인터페이스의 장점

■ 개발시간이 상당히 단축된다.

- 인터페이스만 만들어지면, 메서드의 선언부를 이용하여 개발하는 쪽과 구현클래스를 작성하는 쪽 이렇게 **독립적으로 프로그래밍이 가능**해진다.

■ **표준화가 가능**하다.(컬렉션 프레임워크를 배우면 더욱 이해가 빠르다.)

- 인터페이스를 만든다는 것은 구현 클래스에서 모두 메서드를 구현해야 하므로 보다 일관된 개발이 이루어져 정형화가 이루어진다.

■ 서로 관계없는 클래스를 간접적 관계를 만들어 줄 수가 있다.

ex) List계열, Set계열, Map계열 등

## 10. 인터페이스 예제

```
public interface Repairable {}
```

```
class Vehicle {  
}  
  
class Taxi extends Vehicle implements Repairable {  
}  
  
class Bus extends Vehicle implements Repairable {  
}
```

Taxi, Bus 클래스는 Repairable 인터페이스를 구현하고 Vehicle을 상속 받았다. 하지만, Man 클래스는 아무것도 상속, 구현을 하지 않았다.

```
class Man {}
```

```
class CarCenter {  
    public void repair(Repairable r) {  
        if(r instanceof Vehicle) {  
            Vehicle vehicle = (Vehicle)r;  
            /* 수리하는 코드 */  
        }  
    }  
}
```

상기 CarCenter 클래스에 있는 repair(Repairable r) 메서드에서 매개변수로 들어올 수 있는 것은 무엇인가?

또한, Man 클래스는 왜 안되는가?

중요 : 매개변수가 인터페이스 타입이라는 것은 해당 인터페이스를 구현한 클래스만이 매개변수로 들어올 수 있다는 것이다.

## 11. 인터페이스의 이해

▣ 인터페이스는 두 대상 사이의 중간에서 소통, 대화를 하게끔 하는 역할을 한다.

ex) 버튼, 윈도우 (Gui) 등..

▣ 아울러, 메서드의 선언과 분리를 가능하게 한다.

```
class A {    직접적 관계 (A, B)
    public void methodA(B b) {
        b.methodB();
    }
} class B가 작성되어 있어야 하는 조건 발생
```

```
class B {
    public void methodB() {
        System.out.println("methodB()");
    }
}
```

```
class InterfaceTest {
    public static void main(String args[]) {
        A a = new A();
        a.methodA(new B());
    }
}
```

```
class A {
    public void methodA(I i) {
        i.methodB();
    }
} interface만 선언되면 각각 개발 가능하므로 시간이 단축됨.
```

```
interface I { void methodB(); }

class B implements I {
    public void methodB() {
        System.out.println("methodB()");
    }
}
```

```
class C implements I {
    public void methodB() {
        System.out.println("methodB() in C");
    }
}
```



## 12. default 메서드

### ■ 디폴트 메서드의 등장(JDK1.8)

- 등장 이유 : 인터페이스가 나온지 시간이 많이 흘렀다. 그만큼 인터페이스의 수정이 불가피해진 것이다. 하지만, 인터페이스에 추상메서드를 추가 혹은 수정, 삭제가 이루어진다면, 과연 어떻게 될까? 그렇다. 해당 인터페이스를 구현한 클래스들은 다 컴파일 예외가 발생할 것이다. 하여, 그러한 불편함을 개선하기 위해서 등장한 것이 바로 구현부가 있는 디폴트 메서드가 등장하여, 필요하다면 재정의할 해서 사용하면 될 것이다.

```
public default void method() {}
```

- 1) JDK 1.8에서 추가된 인터페이스의 새로운 멤버이다.
- 2) 실행 불특을 가지고 있는 메서드이다.
- 3) default 제어자를 반드시 붙여야 한다.
- 4) 기본적으로 public 접근 제한이다.
- 5) 구현클래스에서 오버라이딩도 가능하다.

## 13. 정적(static)메서드

### ▣ 정적 메서드의 등장(JDK1.8)

- 등장 이유 : 원래, 정적메서드는 인스턴스와는 별개이므로, 멤버로 등장하는데 걸림들은 전혀 없었지만, 자바를 좀 더 배우기 쉽게 하기 위해서, **단지 규칙을 지키기 위해** **했던 것이다.** JDK 1.8에서 새로 추가된 인터페이스의 새로운 멤버이다.

```
public interface I {  
  
    public static void change() {  
        System.out.println("바꿉니다.");  
    }  
}
```

## 14. 익명 구현 객체(무명 클래스)-anonymous class

▣ 명시적으로 구현클래스를 작성하지 않고, **바로 구현 객체를 얻는 방법**이다.

- **이름이 없는 클래스라서 익명이라고 한다.**

- 인터페이스의 추상 메서드를 익명구현객체라고 해도 다시 재정의할 하여야 한다.

- 추가적으로, 필드와 메서드를 선언 가능하나 익명 구현객체 안에서만 사용 가능하다.

\* 외부에서 접근 불가

- 인터페이스 참조변수로 접근 역시 불가능하다.(원래 타입이 없다. 객체생성 불가)

```
public interface Soundable {  
    public void sound();  
}
```

```
sa1.sound();
```

```
Soundable sa1 = new Soundable() { 원타입이 인터페이스 타입이란 것에 주목을 하자  
    @Override  
    public void sound() {  
        String str = "냄비";  
        System.out.println("첫번째 익명구현객체 : " + str);  
        System.out.println(str + "에 물을 담습니다.");  
    }  
};
```

## 15. 익명 자손 객체(무명 클래스) - anonymous class

▣ 명시적으로 자손클래스를 작성하지 않고, **바로 자손 객체를 얻는 방법**이다.

- **이름이 없는 클래스라서 익명이라고 한다.**

- 원 타입이 조상클래스 타입이라서, 통상 익명 자손 객체는 조상의 메서드를 오버라이딩하여 잠시 사용할 용도로 사용한다.(UI이벤트 처리나 스레드 객체를 간편히 생성)

- 추가적으로, 필드와 메서드를 선언 가능하나 익명 자손 객체 안에서만 사용 가능하다.

\* 외부에서 접근 불가

- 조상타입 참조변수로 접근 역시 불가능하다.(원래타입에 없다.)

```
Person field = new Person() { 원타입이 클래스 타입이란 것에 주목을 하자.
    @Override
    public void wake() {
        System.out.println("6시에 일어납니다");
    }
};
```

감사합니다.

