객체 지향 프로그래밍4_다형성 2

인터페이스(interface)란?

- 일종의 추상클래스. 추상클래스(미완성 설계도)보다 추상화 정도가 높다. 실제 구현된 것이 전혀 없는 기본 설계도
- 추상메서드와 상수만을 멤버로 가지며, 인스턴스 생성 불가, 클래스 작성에 도움을 줄 목적으로 사용된다.
- 미리 정해진 규칙에 맞게 구현하도록 표준을 제시하는 데 사용된다.
- ‘class’대신 ‘interface’를 사용.

```
interface 인터페이스이름 {
      public static final 타입 상수이름 = 값;
      public abstract 메서드이름(매개변수목록);
}
```

- 하지만, 구성요소(멤버)는 추상메서드와 상수만 가능하다.

- 모든 멤버변수는 public static final 이어야 하며, 이를 생략할 수 있다.
- 모든 메서드는 public abstract 이어야 하며, 이를 생략할 수 있다.

```
interface PlayingCard {
      public static final int SPADE = 4;
      final int DIAMOND = 3;      // public static final int DIAMOND = 3;
      static int HEART = 2;       // public static final int HEART = 2;
      int CLOVER = 1;             // public static final int CLOVER = 1;

      public abstract String getCardNumber();
      String getCardKind();  // public abstract String getCardKind();
}
```

인터페이스의 상속과 구현

```
interface Movable {
     /** 지정된 위치(x, y)로 이동하는 기능의 메서드 */
     void move(int x, int y);
}

interface Attackable {
     /** 지정된 대상(u)을 공격하는 기능의 메서드 */
     void attack(Unit u);
}

interface Fightable extends Movable, Attackable { }
```

- 인터페이스도 클래스처럼 상속이 가능.(다중상속 허용)
- 인터페이스는 Object클래스와 같은 최고 조상이 없다.

- 인터페이스를 구현하는 것은 클래스를 상속받는 것과 같다. ‘extends’ 대신 ‘implements’를 사용.
- 인터페이스에 정의된 추상메서드를 완성해야 한다.
- 상속과 구현이 동시에 가능하다.

```
class 클래스이름 implements 인터페이스이름 {
     // 인터페이스에 정의된 추상메서드를 구현해야한다.
}
```

```
class Fighter implements Fightable {
     public void move() { /* 내용 생략*/ }
     public void attack() { /* 내용 생략*/ }
}

interface Fightable {
     void move(int x, int y);
     void attack(Unit u);
}
```

```
abstract class Fighter implements Fightable {
     public void move() { /* 내용 생략*/ }
}
```

```
class Fighter extends Unit implements Fightable {
     public void move(int x, int y) { /* 내용 생략 */}
     public void attack(Unit u) { /* 내용 생략 */}
}
```

인터페이스를 이용한 다형성

- 인터페이스 타입의 변수로 인터페이스를 구현한 클래스의 인스턴스를 참조할 수 있다.

```
class Fighter extends Unit implements Fightable {
     public void move(int x, int y)  { /* 내용 생략 */ }
     public void attack(Fightable f) { /* 내용 생략 */ }
}
```

```
Fighter   f = new Fighter();
Fightable f = new Fighter();
```

- 인터페이스를 메서드의 매개변수 타입으로 지정할 수 있다.

```
void attack(Fightable f) {   // Fightable인터페이스를 구현한 클래스의 인스턴스를
    //...                    // 매개변수로 받는 메서드
}
```

- 인터페이스를 메서드의 리턴타입으로 지정할 수 있다.

```
Fightable method() { // Fightable인터페이스를 구현한 클래스의 인스턴스를 반환
     // ...
     return new Fighter();
}
```

인터페이스의 장점

1. 개발시간을 단축시킬 수 있다.

- 일단 인터페이스가 작성되면, 이를 사용해서 프로그램을 작성하는 것이 가능하다. 메서드를 호출하는 쪽에서는 메서드의 내용에 관계없이 선언부만 알면 되기 때문이다.
- 그리고 동시에 다른 한 쪽에서는 인터페이스를 구현하는 클래스를 작성하도록 하여, 인터페이스를 구현하는 클래스가 작성될 때까지 기다리지 않고도 양쪽에서 동시에 개발을 진행할 수 있다.

2. 표준화가 가능하다.

- 프로젝트에 사용되는 기본 틀을 인터페이스로 작성한 다음, 개발자들에게 인터페이스를 구현하여 프로그램을 작성하도록 함으로써 보다 일관되고 정형화된 프로그램의 개발이 가능하다.

3. 서로 관계없는 클래스들에게 관계를 맺어 줄 수 있다.

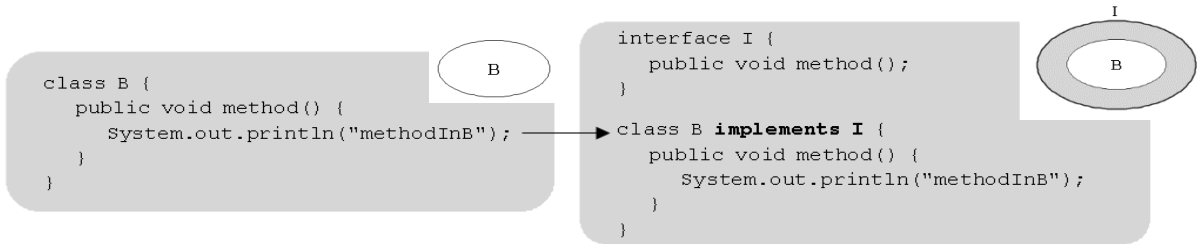
- 서로 상속관계에 있지도 않고, 같은 조상클래스를 가지고 있지 않은 서로 아무런 관계도 없는 클래스들에게 하나의 인터페이스를 공통적으로 구현하도록 함으로써 관계를 맺어 줄 수 있다.

4. 독립적인 프로그래밍이 가능하다.

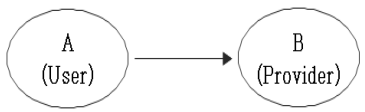
- 인터페이스를 이용하면 클래스의 선언과 구현을 분리시킬 수 있기 때문에 실제구현에 독립적인 프로그램을 작성하는 것이 가능하다.
- 클래스와 클래스간의 직접적인 관계를 인터페이스를 이용해서 간접적인 관계로 변경하면, 한 클래스의 변경이 관련된 다른 클래스에 영향을 미치지 않는 독립적인 프로그래밍이 가능하다.

인터페이스의 이해

- 인터페이스는… 두 대상(객체) 간의 '연결, 대화, 소통'을 돕는 '중간 역할'을 한다.
- 선언(설계)와 구현을 분리시키는 것을 가능하게 한다.



- 사용하는 쪽(User)과 제공하는 쪽(Provider)이 있다. 메서드를 사용(호출)하는 쪽에서는 사용하려는 메서드(Provider)의 선언부만 알면 된다

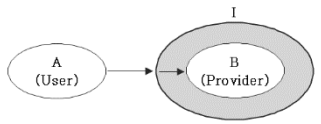



```
class A {
	public void methodA(B b) {
		b.methodB();
	}
}

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();
	}
}
```

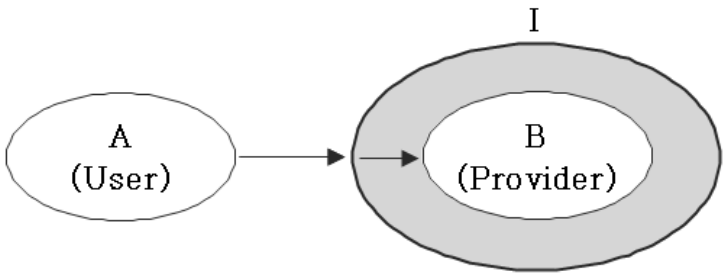



```
interface I { void methodB(); }

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

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

인터페이스의 이해




```
public class Time {
     private int hour;
     private int minute;
     private int second;

     public int getHour() { return hour; }
     public void setHour(int h) {
          if (h < 0 || h > 23) return;
          hour=h;
     }
     public int getMinute() { return minute; }
     public void setMinute(int m) {
          if (m < 0 || m > 59) return;
          minute=m;
     }
     public int getSecond() { return second; }
     public void setSecond(int s) {
          if (s < 0 || s > 59) return;
          second=s;
     }
}
```

```
public interface TimeIntf {
     public int  getHour();
     public void setHour(int h);

     public int  getMinute();
     public void setMinute(int m);

     public int  getSecond();
     public void setSecond(int s);
}
```

디폴트 메서드

- **디폴트 메서드는 인터페이스에 추가된 일반 메서드(인터페이스 원칙 위반)**

```
interface MyInterface {
   void method();
   void newMethod(); // 추상 메서드
}
```



```
interface MyInterface {
   void method();
   default void newMethod(){}
}
```

- **디폴트 메서드가 기존의 메서드와 충돌하는 경우 아래와 같이 해결**

1. **여러 인터페이스의 디폴트 메서드 간의 충돌**
   – 인터페이스를 구현한 클래스에서 디폴트 메서드를 오버라이딩해야 한다.

2. **디폴트 메서드와 조상 클래스의 메서드 간의 충돌**
   – 조상 클래스의 메서드가 상속되고, 디폴트 메서드는 무시된다.

내부 클래스(inner class)란?

- **클래스 안에 선언된 클래스, - 특정 클래스 내에서만 주로 사용되는 클래스를 내부 클래스로 선언한다.**
- **GUI어플리케이션(AWT, Swing)의 이벤트처리에 주로 사용된다.**
- **내부 클래스의 장점**
  - 내부 클래스에서 외부 클래스의 멤버들을 쉽게 접근할 수 있다.
  - 코드의 복잡성을 줄일 수 있다.(캡슐화)
- **내부 클래스의 종류와 특징**
  - 내부 클래스의 종류는 변수의 선언위치에 따른 종류와 동일하다.
  - 유효범위와 성질도 변수와 유사하므로 비교해보면 이해하기 쉽다.

```
class A {
    ...
}

class B {
    ...
}
```



```
class A {   // 외부 클래스
    ...
    class B {  // 내부 클래스
        ...
    }
    ...
}
```

내부 클래스	특 징
인스턴스 클래스 (instance class)	외부 클래스의 멤버변수 선언위치에 선언하며, 외부 클래스의 인스턴스멤버처럼 다루어진다. 주로 외부 클래스의 인스턴스멤버들과 관련된 작업에 사용될 목적으로 선언된다.
스태틱 클래스 (static class)	외부 클래스의 멤버변수 선언위치에 선언하며, 외부 클래스의 static멤버처럼 다루어진다. 주로 외부 클래스의 static멤버, 특히 static메서드에서 사용될 목적으로 선언된다.
지역 클래스 (local class)	외부 클래스의 메서드나 초기화블럭 안에 선언하며, 선언된 영역 내부에서만 사용될 수 있다.
익명 클래스 (anonymous class)	클래스의 선언과 객체의 생성을 동시에 하는 이름없는 클래스(일회용)

```
class Outer {
    int iv = 0;
    static int cv =  0;

    void myMethod() {
        int lv = 0;
    }
}
```



```
class Outer {
    class InstanceInner {}
    static class StaticInner {}

    void myMethod() {
        class LocalInner {}
    }
}
```

익명 클래스(anonymous class)

- 이름이 없는 일회용 클래스. 단 하나의 객체만을 생성할 수 있다.

```
new 조상클래스이름() {
   // 멤버 선언
}

   또는

new 구현인터페이스이름() {
   // 멤버 선언
}
```

[예제10-6]/ch10/InnerEx6.java

```
class InnerEx6 {
   Object iv = new Object(){ void method(){} };         // 익명클래스
   static Object cv = new Object(){ void method(){} };// 익명클래스

   void myMethod() {
       Object lv = new Object(){ void method(){} };    // 익명클래스
   }
}
```

```
InnerEx6.class
InnerEx6$1.class ← 익명클래스
InnerEx6$2.class ← 익명클래스
InnerEx6$3.class ← 익명클래스
```

```
import java.awt.*;
import java.awt.event.*;

class  InnerEx7{
   public static void main(String[] args) {
       Button b = new Button("Start");
       b.addActionListener(new EventHandler());
   }
}

class EventHandler implements ActionListener {
   public void actionPerformed(ActionEvent e) {
       System.out.println("ActionEvent occurred!!!");
   }
}
```

```
import java.awt.*;
import java.awt.event.*;

class  InnerEx8 {
   public static void main(String[] args) {
       Button b = new Button("Start");
       b.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    System.out.println("ActionEvent occurred!!!");
                }
            } // 익명 클래스의 끝
       );
   } // main메서드의 끝
} // InnerEx8클래스의 끝
```