Object – Oriented Programming

LAB #10. Interfaces & Inner Classes

Interface

- 추상 메소드의 집합

- 일종의 추상 클래스.

- 실제로 구현된 것이 전혀 없는 기본 설계도

- 객체를 생성할 수 없고, 클래스 작성에 도움을 줄 목적으로 사용

- 미리 정해진 규칙에 맞게 구현하도록 표준을 제시하는데 사용

Interface

- 클래스와 비슷한 점
  - 인터페이스에는 여러가지 메소드가 포함될 수 있다.
  - 인터페이스는 .java 확장자를 가진 파일로 작성되며, 인터페이스 이름이 파일 이름과 일치해야 한다.
  - 인터페이스의 byte code는 .class 파일에 있다.

- 클래스와 다른 점
  - 인터페이스는 인스턴스화할 수 없다.
  - 인터페이스에는 생성자가 포함될 수 없다.
  - 인터페이스의 모든 메소드는 추상 메소드이다.
  - 인터페이스에는 인스턴스 변수가 포함될 수 없다. (static final 선언만 가능)

Interface

- 인터페이스 및 모든 메소드는 public 으로 선언해야 함

```
public interface InterfaceName{
        public returnType method1();
        public returnType method2();
        public returnType method3();
}
public class SomeClass implements InterfaceName{
        //implement all the methods
}
```

- 인터페이스는 type 이므로, 인터페이스 type의 매개변수를 사용하여 메소드를 작성할 수 있다.
  - > 이 매개변수는 인터페이스를 implements한 모든 클래스를 인자로 받을 수 있다.

Interface

- ‘class’대신 ‘interface’를 사용한다는 것 외에는 클래스 작성과 동일하다.

```
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();
}
```

Interface

- 인터페이스를 사용하는 것은 클래스를 상속받는 것과 유사하다.
  다만, ‘extends’ 대신 ‘implements’를 사용한다.

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

- 인터페이스에 정의된 추상메서드를 완성해야 한다.

```
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*/ }
}
```

- 상속과 구현이 동시에 가능하다.

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

Interface

- 인터페이스도 extends를 통해 상속이 가능하다.
  - 하지만 인터페이스의 상속은 인터페이스 간에만 가능하다.
  - 클래스들과 달리 인터페이스는 다중 상속이 허용된다.

```java
public interface Ridable extends Movable, Runnable{
    public void ride();
}
```

```java
public interface Movable {
    public void move();
}
```

```java
public interface Runnable {
    public void run();
}
```

```java
public class Car implements Ridable{
    @Override
    public void move() {
        // TODO Auto-generated method stub

    }

    @Override
    public void run() {
        // TODO Auto-generated method stub

    }

    @Override
    public void ride() {
        // TODO Auto-generated method stub

    }

}
```

Interface

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.

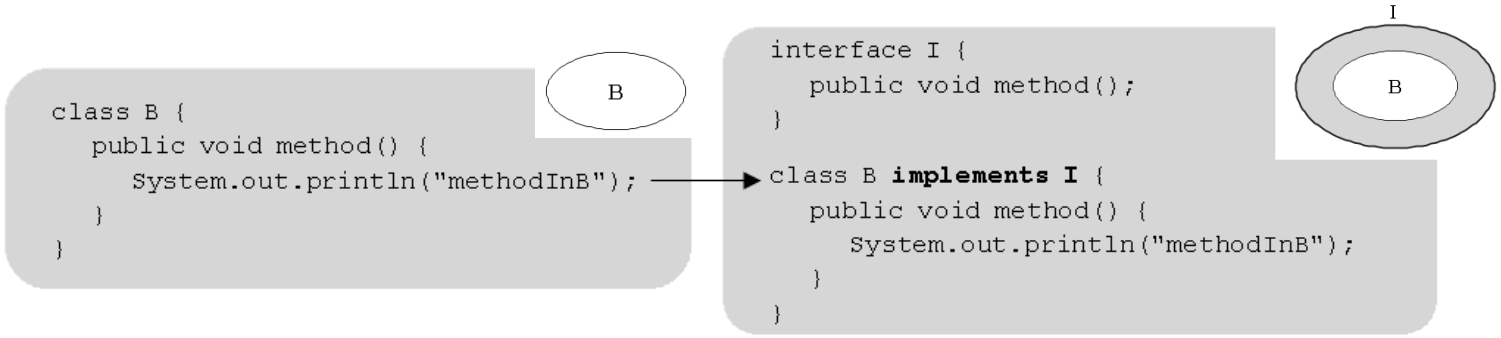
서로 상속관계도 아니고 같은 조상클래스를 가지고 있지 않은 서로 아무런 관계도 없는 클래스들에게 하나의 인터페이스를 공통적으로 구현하도록 함으로써 관계를 맺어 줄 수 있다.

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.

Interface

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



- 인터페이스를 이해하려면 먼저 두 가지를 기억하자.
  - 클래스를 사용하는 쪽(User)과 클래스를 제공하는 쪽(Provider)이 있다.
  - 메서드를 사용(호출)하는 쪽(User)에서는 사용하려는 메서드(Provider)의 선언부만 알면 된다.

Interface

- 직접적인 관계의 두 클래스(A-B)

```
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());
	}
}
```

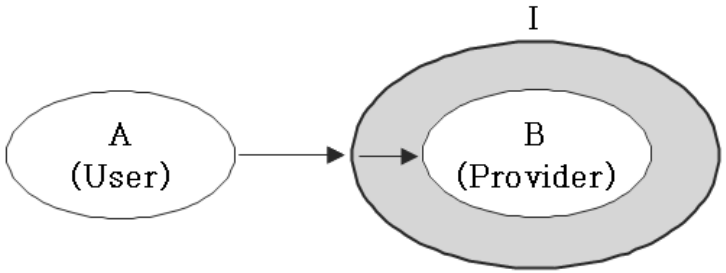
- 간접적인 관계의 두 클래스(A-I-B)

```
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");
	}
}
```



Comparable Interface

- java.lang 패키지에 있으므로, 모든 프로그램에서 자동으로 사용할 수 있다.

- 구현이 필요한 메소드가 있다.

  -> public int compareTo(Object other)

  - compareTo를 사용하여 기준에 따라 같은 타입의 객체를 비교할 수 있음

Comparable Interface

- int compareTo(Object other)
  - 호출하는 객체가 다른 매개변수보다 "앞에 올 때"(작을 때) 음수를 반환
  - 호출하는 객체가 다른 매개변수와 "동일할 때"(같을 때) 0을 반환
  - 호출하는 객체가 다른 매개변수를 "뒤따를 때"(클 때) 양수를 반환

  - 호출하는 클래스와 parameter의 타입이 다를 경우,
    ClassCastException이 throw됨

Comparable Interface

- **Example**

```
public int compareTo(Object obj){
        if (obj == null) throw new NullPointerException("Object is null");
        if (!this.getClass().equals(obj.getClass())) throw new ClassCastException("Object not of the
                same type");
        Car toCompare = (Car) obj;
        if (this.velocity > toCompare.velocity) return -1;
        if (this.velocity == toCompare.velocity) return 0;
        else return 1;
}
```

Using Comparable

- Sort
- Java의 Arrays 클래스는 배열을 정렬하는 데 사용할 수 있는 static sort() 메소드가 포함되어 있음
- 배열을 정렬하려면 비교 가능한 객체만 있어야한다.
- 객체를 정렬하려면 객체를 서로 비교할 수 있어야 한다.
- ->Comparable 인터페이스를 사용하여 객체를 비교할 수 있다.

Comparable interface

```
public class Fruit implements Comparable{
    private String fruitName;
    private int quantity;

    public Fruit(String fruitName, int quantity) {
        this.fruitName = fruitName;
        this.quantity = quantity;
    }

    public String getFruitName() {
        return fruitName;
    }

    public void setFruitName(String fruitName) {
        this.fruitName = fruitName;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    @Override
    public int compareTo(Object compareFruit) {
        int compareQuantity = ((Fruit)compareFruit).getQuantity();
        return this.quantity - compareQuantity;
    }
}
```

Comparable interface

```
import java.util.Arrays;

public class SortFruitObject {

    public static void main(String[] args) {
        Fruit[] fruits = new Fruit[4];

        Fruit pineapple = new Fruit("Pineapple", 70);
        Fruit apple = new Fruit("Apple", 100);
        Fruit orange = new Fruit("Orange", 80);
        Fruit banana = new Fruit("Banana", 90);

        fruits[0] = pineapple;
        fruits[1] = apple;
        fruits[2] = orange;
        fruits[3] = banana;

        Arrays.sort(fruits);

        int i=0;
        for(Fruit f : fruits)
            System.out.println("fruits " + ++i + f.getFruitName() +
                    ", Quantity : " + f.getQuantity());
    }

}
```

☐ return this.quantity - compareQuantity

--------- Output ---------

Fruits 1 : Pineapple, Quantity : 70
Fruits 2 : Orange, Quantity : 80
Fruits 3 : Banana, Quantity : 90
Fruits 4 : Apple, Quantity : 100

☐ return compareQuantity - this.quantity

--------- Output ---------

Fruits 1 : Apple, Quantity : 100
Fruits 2 : Banana, Quantity : 90
Fruits 3 : Orange, Quantity : 80
Fruits 4 : Pineapple, Quantity : 70

Inner Class

- 클래스 안에 선언된 클래스

- 특정 클래스 내에서만 주로 사용되는 클래스를 내부 클래스로 선언한다.

- GUI 애플리케이션(AWT, Swing) 이벤트처리에 주로 사용된다.

```
class A {
    ...
}

class B {
    ...
}
```

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

- 장점 : 내부 클래스에서 외부 클래스의 멤버들을 쉽게 접근할 수 있다.
  코드의 복잡성을 줄일 수 있다. (캡슐화)

Inner Class

- 내부 클래스의 종류

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 {}
   }
}
```

Class with an Inner Class

```
public class BankAccount{
    private class Money{
        private long dollars;
        private int cents;

        public Money(String stringAmount) {
            abortOnNull(stringAmount);
            int length = stringAmount.length();
            dollars = Long.parseLong(stringAmount.substring(0, length - 3));
            cents = Integer.parseInt(stringAmount.substring(length - 2, length));
        }

        public String getAmount() {
            if (cents > 9)
                return (dollars + "." + cents);
            else
                return (dollars + ".0" + cents);
        }

        public void addIn(Money secondAmount) {
            abortOnNull(secondAmount);
            int newCents = (cents + secondAmount.cents)%100;
            long carry = (cents + secondAmount.cents)/100;
            cents = newCents;
            dollars = dollars + secondAmount.dollars + carry;
        }
        private void abortOnNull(Object o) {
            if (o == null) {
                System.out.println("Unexpected null argumnet.");
                System.exit(0);
            }
        }
    }

    private Money balance;

    public BankAccount() {
        balance = new Money("0.00");
    }

    public String getBalance() {
        return balance.getAmount();
    }

    public void makeDeposit(String depositAmount) {
        balance.addIn(new Money(depositAmount));
    }

    public void closeAccount() {
        balance.dollars = 0;
        balance.cents = 0;
    }
}
```

Class Money 끝

Static Inner Classes

- 일반적인 inner class는 inner class의 객체와 inner class 객체를 생성한 outer class 간의 연관성을 가진다

  ->이것은 inner class 정의가 outer class의 인스턴스 변수를 참조하거나 메소드를 호출할 수 있도록 함

- Inner class가 static 이어야 하는 상황은 다르다.
  - Inner class의 객체가 outer class의 static 메소드 내에서 생성되는 경우 (디자인 패턴)
  - Inner class에 static 멤버가 있어야하는 경우
    - Outer class의 인스턴스 변수는 참조할 수 없음
    - Outer class의 non-static 메소드는 호출할 수 없음

Anonymous Classes

- 이름이 없는 일회용 클래스. 단 하나의 객체만 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
   }
}
```

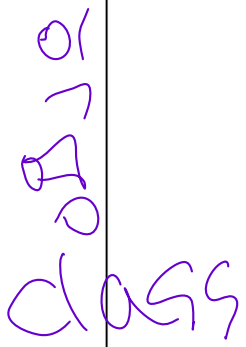
```
InnerEx6.class
InnerEx6$1.class ← 익명클래스
InnerEx6$2.class ← 익명클래스
InnerEx6$3.class ← 익명클래스
```

Anonymous Classes

```
public class Program {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        NumberCarrier anObject = new NumberCarrier()
                {
                    private int number;
                    public void setNumber(int value)
                    {
                        number = value;
                    }
                    public int getNumber()
                    {
                        return number;
                    }
                };

        NumberCarrier anotherObject = new NumberCarrier()
                {
                    private int number;
                    public void setNumber(int value)
                    {
                        number = 2*value;
                    }
                    public int getNumber()
                    {
                        return number;
                    }
                };
```




```
        anObject.setNumber(42);
        anotherObject.setNumber(42);
        showNumber(anObject);
        showNumber(anotherObject);
        System.out.println("End of program.");
    }

    public static void showNumber(NumberCarrier o)
    {
        System.out.println(o.getNumber());
    }

}
```

```
public interface NumberCarrier {
    public void setNumber(int value);
    public int getNumber();
}
```

실습

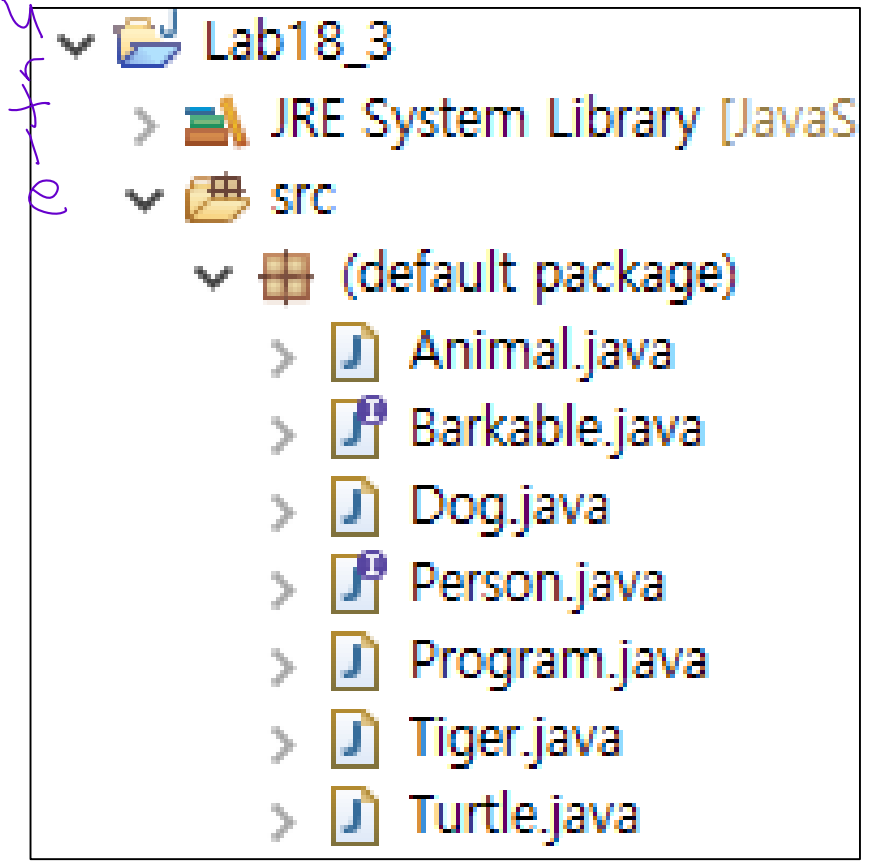
- 다음과 같은 프로그램을 작성한다.

1) Dog, Tiger, Turtle은 Animal을 상속받는다.

2) Dog, Tiger은 Barkable 인터페이스를 Implements한다.

3) Barkable의 bark 함수는 각 동물의 울음소리를 반환한다.
   Dog = “멍멍”
   Tiger = “어흥”

4) Animal의 이름 속성은 각 동물의 이름으로 초기화한다. (Dog : “개” / Tiger : “호랑이“ / Turtle : “거북이”)



실습

4) Person은 익명 클래스를 이용하여 main함수에서 구현한다.

- 속성으로 (-hp:int) 변수를 구현하고 100으로 초기화한다.
- control함수에서는 Tiger를 매개변수로 전달할 경우에는 hp를 80 감소시키고 Dog를 매개변수로 전달할 경우에는 hp를 10을 감소시키며 감소 후에는 각각 “OO을 제압하였습니다.”라는 String을 출력 (hint : getClass() or instanceof 연산자)
- showInfo에서는 사람의 체력을 출력할 것

5) showResult에서는 animal이 짖을 수 있는 동물이면 해당 동물의 정보를 출력할 것

```
1번째 동물 : 개
1번째 동물 울음소리 : 멍멍
개를 제압하였습니다.
사람 HP : 90
2번째 동물 : 호랑이
2번째 동물 울음소리 : 어흥
호랑이를 제압하였습니다.
사람 HP : 10
```

실습

```
public class Program {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Dog dog = new Dog();
        Tiger tiger = new Tiger();
        Turtle turtle = new Turtle();

        Animal[] animal = new Animal[3];
        animal[0] = dog;
        animal[1] = tiger;
        animal[2] = turtle;

        Person person = new Person() {
        };

        showResult(animal, person);

    }

    private static void showResult(Animal[] animals, Person p) {

    }
}
```

실습

```java
public class Animal {
    private String name;

    public Animal(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

```java
public interface Barkable {
    public String bark();
}
```

Dog tiger tutle은 알아서구현

```java
public interface Person {
    public void control(Barkable b);
    public void showInfo();
}
```

10주차 실습/과제 관련 질문: minah741@naver.com (신민아 조교)