day13-java

태그	
날짜	@2022년 10월 18일

Interface

- 작업명령서라고도 불린다.

```
Sample.java

package day13_1018;

public class Sample {

  public static void main(String[] args) {
    // 동물원..

    ZooKeeper zooKeeper = new ZooKeeper();
    Tiger tiger = new Tiger();
    Lion lion = new Lion();

    zooKeeper.feed(tiger);
    zooKeeper.feed(lion);
  }

}
```

zooKeeper 객체는 tiger 객체에는 apple을 주고 , lion객체에는 banana를 먹일 수 있게 되었다.

각기 다른 매서드를 오버로딩을 이용해서 호출하여 각각다른 먹이를 줄 수 있다.

현재는 각각의 동물마다 새로운 매서드를 계속해서 작성해서 추가해주어야만 한다.

이렇게 동물이 추가될 때마다 각기 다른 메서드를 계속해서 추가해야 한다면 사육사가 몹시 힘들게 된다.

Interface를 구현함에 따라

```
Predator.java

package poly_exam;

public interface Predator {

}
```

```
Tiger.java
public class Tiger extends Animal implements Predator, Show{
//extends 상속, implements 구현
```

이전에는 각각의 동물 별로 각각의 메서드를 작성해야 했지만 하나의 메서드만으로 수백개, 수천개의 동물을 하나의 메서드로 밥을 줄 수 있게 되었다.

(아직 동물 별로 다른 음식을 줄 수 는 없다. 추후 공부)

```
ZooKeeper.java
void feed(Predator predator) {
    System.out.println("FEED " + predator.getFood());
  }
```

이것은 tiger, lion 객체가 각각 Tiger, Lion 클래스의 자료형이기도 하지만 동시에 Predator 인터페이스를 구현하고 있기 때문에 Predator 인터페이스의 자료형이기 때문이다.

하지만 현재 가장 큰 문제점은 모든 동물을 각각 다른 음식을 먹일수가 없고 그냥 한가지 FOOD로만 먹일수 밖에 없다. 이부분을 수정해보자

```
Predator.java
package day13_1018;

public interface Predator {

    String getFood();


}
```

동물의 종류만큼 feed 메서드가 필요했었는데 인터페이스를 이용하여 구현했더니 하나의 feed메서드 만으로도 구현이 가능해졌다. 여기 중요한 점은 메서드의 개수가 줄었다는 것이 아니라 ZooKeeper 클래스가 동물들의 종류의 의존적이였던 클래스에서 동물들의 종류와는 상관없는 독립적인 클래스가 되었다는 점이다. 이것이 인터페이스의 핵심이다.

컴퓨터의 USB포트를 생각해 보자 연결할 수 있는 기기는 외장하드 프린터 스캐너 헤드셋 등등 무척이나 많다. 하지만 기기와는 상관없이 USB포트에만 연결만 해주면 동작하게 된다. 각각의 기기가 어떻게 연결될지 고민할 필요가 없다. 이점이 인터페이스와 매우 유사한 개념이다.

사육사가 어떤 동물인지 신경 쓸 필요가 없이 그냥 먹이를 던져주는 버튼을 누르기만 하면 알아서 해당 동물에 맞게 각각 다른 먹이가 던져지게 되는 것이다.

실세계	자바세계
컴퓨터	zookeeper
USB허브	predator
하드, 스캐너, 프린터	lion, tiger …

자바 8부터는 인터페이스에 몸통이 있는 메서드를 구현할 수 있게 변화 되었다.

```
1  package day13_1018;
2
3  public interface Predator {
4
5          String getFood();
6
7          default void printFood() {
8              System.out.printf("my food is %s", getFood());
9          }
10 }
11
```

디폴트 메서드는 메서드명 앞에 default 라고 표기한다.

디폴트 메서드를 구현하면 Predator 인터페이스를 구현한 클래스들은(tiger, lion) printFood 메서드를 구현하지 않아도 사용할 수 있다.

또 디폴트 메서드는 오버라이딩도 가능하다. 즉, 실제 클래서에서는 다르게 구현할 수 도 있다.

static 메서드

자바8 부터 사용할 수 있다. 클래스처럼

인터페이스명. 스태틱메서드명

일반 클래스의 스태틱메서드 사용 방식과 완전히 동일하게 인터페이스명으로 메서드에 직접 접근하여 호출 가능하다.

다형성 polymorphism

```
Bouncer.java

package poly_exam;

public class Bouncer {

    void barkAnimal(Animal animal) {
      if(animal instanceof Tiger) {
        System.out.println("어흥");
      }else if(animal instanceof Lion) {
        System.out.println("으르렁");
      }
    }
}
```

```
Sample.java

package poly_exam;

public class Sample {

  public static void main(String[] args) {
    // 동물원..

    ZooKeeper zooKeeper = new ZooKeeper();
```

```
    Tiger tiger = new Tiger();
    Lion lion = new Lion();
    Bouncer bouncer = new Bouncer();

    bouncer.barkAnimal(tiger);
    bouncer.barkAnimal(lion);
  }

}
```

Bouncer 클래스는 동물을 짖게 만든 역할을 한다. Bouncer 클래스의 barkAnimal 메서드는 입력 받은 객체가 Tiger로 생성한 객체인 경우는 “어흥”을 출력하고 Lion으로 생성한 객체가 들어올 경우는 “으르렁”을 출력한다.

(어느 클래스로 생성한 인스턴스인지를 확인하는 명령어가 instanceof 이다.)

만약 당나귀 또는 토끼 또는 돼지 등등 동물을 추가할때 마다 매번 분기를 반복하여 만들어 주어야 한다.

barkable 인터페이스를 작성

```
Barkable.java
public interface Barkable {

  void bark();
}
```

```
Bouncer.java

public class Bouncer {
void barkAnimal(Barkable b) {
      b.bark();
    }
}
```

```
Lion.java
package poly_exam;

public class Lion extends Animal implements BarkablePredator{

  public String getFood() {
    return "banana";
  }
```

```
  public void bark()  {
    System.out.println("으르렁");
  }
}
```

위에서 처럼 여러 인터페이스를 동시에 다중 구현할 수 있다.

lion과 tiger는 predator와 Barkable을 동시에 구현하고 있다. 따라서 predator의 자료형임과 동시에 Barkable의 자료형이기도 하다.

이렇게 하나의 객체가 여러개의 자료형 타입을 가질 수 있는 것을 객체지향에서는 다형성 Polymorphism이라고 부른다.

즉 Tiger클래스는

Tiger tiger = new Tiger();

Animal tiger = new Tiger();

Predator tiger = new Tiger();

Barkable tiger = new Tiger();

Predator로 선언된 tiger객체와 Barkable 로 선언된 tiger객체는 메서드가 서로 다르다는 점이다.

Predator로 선언된 tiger객체는 getFood()메서드가 선언된 Predator인터페이스의 객체이므로 getFood메서드만 호출이 가능하다. 마찬가지로 Barkable로 선언된 tiger 객체는 bark 메서드만 호출이 가능하다.

그렇다면 getFood 메서드와 bark 메서드를 모두 사용하고 싶다면 어떨까

- 두개의 인터페이스를 모두 상속한 새로운 인터페이스를 만들 수 있다.