PRESENTATION

42주차 주제 인터페이스

~자바 인터페이스~

by mun

인터페이스란?

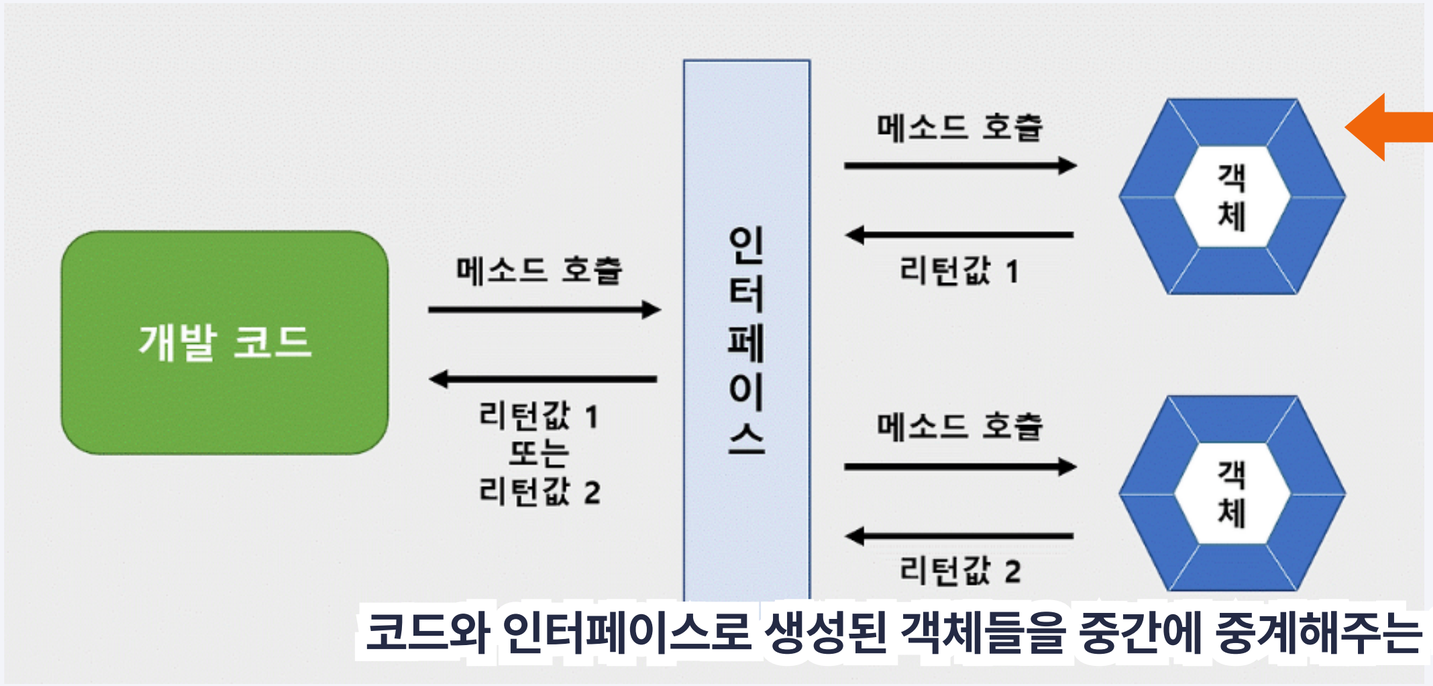
자바 프로그래밍 언어에서 클래스들이 구현해야 하는 동작을 지정하는데 사용되는 추상 자료형

- 클래스들이 필수로 구현해야 하는 추상 자료형
- 객체의 사용방법을 가이드라인 하는 것
- 추상 메서드와 상수로만 이루어져 있음

특징

- 다중 상속 가능
- 추상 메서드와 상수만 사용 가능
- 생성자 사용 불가
- 메서드 오버라이딩 필수

인터페이스를 사용하는 이유



코드와 인터페이스로 생성된 객체들을 중간에 중계해주는 역할

인터페이스로 구현된 객체는
인터페이스에서 정의된 추상 메서드와 동일한
메소드 이름, 매개 타입, 리턴 타입을 가진 실체 메서드를
필수로 가지고 있어야 한다

- 추상 클래스를 통해 객체들 간의 네이밍을 통일할 수 있고 이로 인해 소스의 가독성과 유지보수가 향상
- 확장에는 열려있고 변경에는 닫혀있는 객체 간 결합도(코드 종속성)를 낮춘 유연한 방식의 개발이 가능

https://coding-factory.tistory.com/867

인터페이스 사용법

```
public interface 인터페이스_명{
    //상수 정의
    public static final 자료형 상수_명 = 값;
    //추상메소드 정의
    public 반환형(void, int 등등) 메소드_명(매개변수);
    //default 메소드 정의(자바 8부터 지원)
    public default 반환형 메소드_명(매개변수){
        실행코드 작성 ...
    };
}
```

- 상수
- 추상 메서드
- 디폴트 메서드 (자바8 이후)

반환타입, 메소드명, 매개변수를 동일하게 작성
추상메소드이므로 예외처리는 자유롭게 가능

인터페이스의 필요성 예제

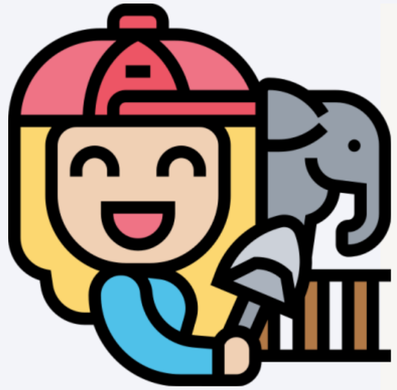
사육사

난 동물원의 사육사!
육식 동물에게 먹이를 주는 역할이야
사자에게는 바나나를,
호랑이에게는 사과를 던져 줘

사자

호랑이

인터페이스의 필요성 예제



```
class ZooKeeper {
    void feed(Tiger tiger) {  // 호랑이가 오면 사과를 던져 준다.
        System.out.println("feed apple");
    }

    void feed(Lion lion) {  // 사자가 오면 바나나를 던져준다.
        System.out.println("feed banana");
    }
}
```

같은 먹이를 주는 feed 메서드여도
매개변수로 오는 동물에 따라 따로 구현

```
public class Sample {
    public static void main(String[] args) {
        ZooKeeper zooKeeper = new ZooKeeper();
        Tiger tiger = new Tiger();
        Lion lion = new Lion();
        zooKeeper.feed(tiger);
        zooKeeper.feed(lion);
    }
}
```

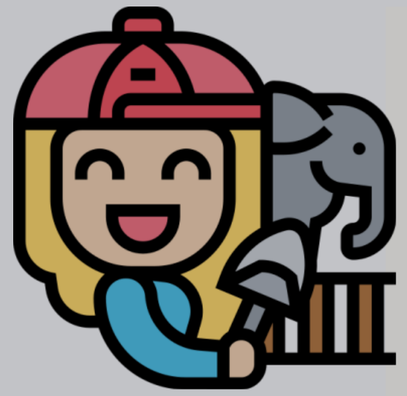
```
class Animal {
    String name;

    void setName(String name) {
        this.name = name;
    }
}

class Tiger extends Animal {
}

class Lion extends Animal {
}
```

인터페이스의 필요성 예제



```
class ZooKeeper {
    void feed(Tiger tiger) {  // 호랑이가 오면 사과를 던져 준다.
        System.out.println("feed apple");
    }

    void feed(Lion lion) {  // 사자가 오면 바나나를 던져준다.
        System.out
    }
}
```

만약 먹이를 줘야 할
육식 동물들이 더 늘어난다면...?

```
public class Sample {
    public static void main(String[] args) {
        ZooKeeper zooKeeper = new ZooKeeper();
        Tiger tiger = new Tiger();
        Lion lion = new Lion();
        zooKeeper.feed(tiger);
        zooKeeper.feed(lion);
    }
}
```

```
class Animal {
    String name;

    void setName(String name) {
        this.name = name;
    }
}

class Tiger extends Animal {
}

class Lion extends Animal {
}
```

인터페이스의 필요성 예제

```
class ZooKeeper {
    void feed(Tiger tiger) {
        System.out.println("feed apple");
    }

    void feed(Lion lion) {
        System.out.println("feed banana");
    }

    void feed(Crocodile crocodile) {
        System.out.println("feed strawberry");
    }

    void feed(Leopard leopard) {
        System.out.println("feed orange");
    }
}

(... 생략 ...)
```

클래스가 추가될 때마다 feed 메서드를 추가해야 되어
사육사의 클래스가 복잡해진다

인터페이스의 필요성 예제

인터페이스를 사용한다면

```
interface Predator {
}
```

```
class Tiger extends Animal implements Predator {
}

class Lion extends Animal implements Predator {
}
```

육식 동물 인터페이스를 구현하도록 한다

인터페이스의 필요성 예제

인터페이스를 사용한다면

```
interface Predator {
}
```

```
class Tiger extends Animal implements Predator {
}

class Lion extends Animal implements Predator {
}
```

변경 전

```
class ZooKeeper {
    void feed(Tiger tiger) {
        System.out.println("feed apple");
    }

    void feed(Lion lion) {
        System.out.println("feed banana");
    }
}
```

변경 후

```
class ZooKeeper {
    void feed(Predator predator) {
        System.out.println("feed apple");
    }
}
```

feed 메서드의 입력으로 각 객체를 필요로 하지 않고
Predator라는 인터페이스로 대체할 수 있게 된다

인터페이스의 필요성 예제

인터페이스를 사용한다면

```
interface Predator {
}
```

```
class Tiger extends Animal implements Predator {
}

class Lion extends Animal implements P
}
```

변경 전

```
class ZooKeeper {
    void feed(Tiger tiger) {
        System.out.println("feed apple");
    }

    void feed(Lion lion) {
        System.out.println("feed banana");
    }
}
```

어떤 객체가 들어와도
똑같은 문자열만 반환되는 문제가 존재

변경 후

```
class ZooKeeper {
    void feed(Predator predator) {
        System.out.println("feed apple");
    }
}
```

feed 메서드의 입력으로 각 객체를 필요로 하지 않고
Predator라는 인터페이스로 대체할 수 있게 된다

인터페이스의 필요성 예제

```
interface Predator {
    String getFood();
}
```

getFood() 메서드를 추가

```
class Tiger extends Animal implements Predator {
    public String getFood() {
        return "apple";
    }
}

class Lion extends Animal implements Predator {
    public String getFood() {
        return "banana";
    }
}
```

각 객체에 메서드를 구현하도록 한다

인터페이스의 필요성 예제

```
interface Predator {
    String getFood();
}
```

getFood() 메서드를 추가

```
class Tiger extends Animal implements Predator {
    public String getFood() {
        return "apple";
    }
}

class Lion extends Animal implements Predator {
    public String getFood() {
        return "banana";
    }
}
```

각 객체에 메서드를 구현하도록 한다

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

Predator 인터페이스를 구현한 구현체(Tiger, Lion)의 getFood() 메서드를 호출해 결과를 다르게 줄 수 있다

인터페이스의 필요성 예제

동물에 의존적이던 클래스

```
class ZooKeeper {
    void feed(Tiger tiger) {
        System.out.println("feed apple");
    }

    void feed(Lion lion) {
        System.out.println("feed banana");
    }

    void feed(Crocodile crocodile) {
        System.out.println("feed strawberry");
    }

    void feed(Leopard leopard) {
        System.out.println("feed orange");
    }
}

(... 생략 ...)
```

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

각 동물 클래스들과 상관 없는
독립적인 클래스가 될 수 있다

인터페이스의 특징을 이해하고
잘 활용해 보아요!

감사합니다

참고

- https://ko.wikipedia.org/wiki/인터페이스_(자바)
- https://wikidocs.net/217
- https://coding-factory.tistory.com/867
- https://velog.io/@ung6860/JAVA%EC%9D%B8%ED%84%B0%ED%8E%98%EC%9D%B4%EC%8A%A4Interface%EC%97%90-%EB%8C%80%ED%95%B4-%EC%95%8C%EC%95%84%EB%B3%B4%EC%9E%90-fc81k7rr