

Extends 상속

date. 2021_01_15

접근 제어자(access modifier)

제어자	public	protected	default	private
같은 클래스				
같은 패키지				
자손 클래스				
전체				

*public : 접근 제한이 전혀~ 없음!

*protected : 같은 패키지 안에서 접근 가능!

다른 패키지의 자손 클래스에서도 접근이 가능!

*default : 같은 패키지 안에서만 접근 가능!

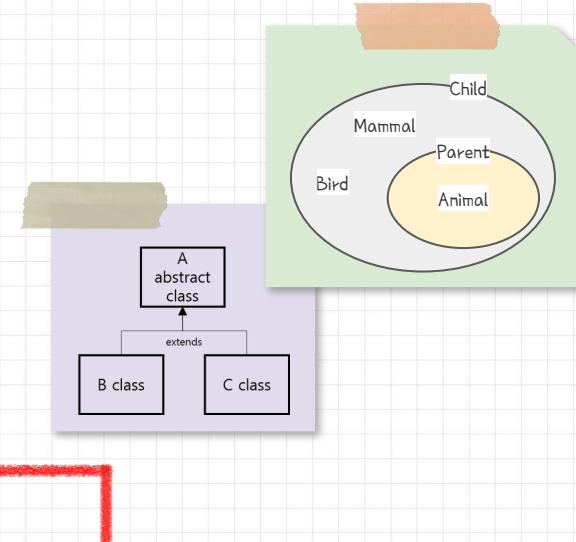
*private : 같은 클래스 안에서만 접근 가능!

```
public class Animal {  
    protected int age;  
    protected String food, speak;
```

```
public Animal(int age, String food, String speak){  
    // 부모 클래스에서 생성자 선언을 했다면  
    // 자식 클래스에서도 무조건 생성자를 선언하자.  
    this.age = age;  
    this.food = food;  
    this.speak = speak;  
}
```

```
public class Mammal extends Animal{  
    protected int legs;  
  
    public Mammal  
        (int age, String food, String crying, int legs) {  
  
        // 부모 클래스의 생성자를 그대로 쓰고  
        // 자식 클래스에서 생성자가 추가 되었다면 추가하자.  
        super(age, food, crying);  
        this.legs = legs;  
    }  
}
```

```
public class Bird extends Animal{  
    protected int wings;  
  
    public Bird  
        (int age, String food, String crying, int wings) {  
  
        // 부모 클래스의 생성자 먼저 쓰고 그 뒤에 추가된 생성자를 추가한다.  
        // 부모 클래스의 생성자를 넘겨주는 방식은 super()를 사용!  
        super(age, food, crying);  
        this.wings = wings;  
    }  
}
```



```
public class Monkey extends Mammal implements Behavior{
```

```
    private String hip;

    public Monkey
        (int age, String food, String speak, int legs, String hip) {
            super(age, food, speak, legs);
            this.hip = hip;
        }
```

```
    public String getHip() {
        return hip;
    }
```

@Override
// 부모클래스로부터 상속받은 method의 내용을 변경하는 것.

// 그대로 사용하기도 하지만 자식클래스에 맞게 변경해서 사용할 수도 있다.

```
    public void eat() {
        System.out.printf
            ("원숭이는 %s를 먹는다.\n", food); }
```

@Override
public void walk() {
 System.out.printf
 ("원숭이는 %s엉덩이를 썰룩거리며 걷는다.\n", hip); }

@Override
public void speak() {
 System.out.printf
 ("원숭이가 %s하며 운다.\n", speak); }

```
public interface Behavior {
    // 인터페이스를 사용할때는
    // implements 선언해야 한다.
    public void eat();
    public void walk();
    public void speak();}
```

```
public class Animal {
    protected int age;
    protected String food, speak;
```

```
    public Animal(int age, String food, String speak) {
        this.age = age;
        this.food = food;
        this.speak = speak;
    }
```

```
    public static void main(String[] args) {
```

```
        Parrot pa = new Parrot
            (2, "모이", "짹짹", 2, "무지개색");
```

```
        pa.eat();
        pa.speak();
        pa.walk();
        pa.feather();
```

```
        Tiger ti = new Tiger
            (5, "고기", "어흥", "줄무늬");
```

```
        ti.eat();
        ti.speak();
        ti.walk();
```

```
        Monkey mo = new Monkey
            (12, "바나나", "우끼끼", "빨간색");
```

```
        mo.eat();
        mo.speak();
        mo.walk();
        mo.hip();
```

< 인터페이스를 사용하면 좋은점 ! >

표준화가 가능하다.

개발시간을 단축 시킬 수 있다.

아무런 관계가 없는 클래스에도 인터페이스를 구현하여 관계를 만들어줄 수 있다.

출력결과

새는 모이를 먹는다.

새는 짹짹하고 운다.

새는 날개 2개로 하늘을 난다.

앵무새 날개는 무지개색이다.

호랑이는 고기를 먹는다.

호랑이는 어흥하고 운다.

호랑이는 줄무늬를 뿜내며 걷는다.

원숭이는 바나나를 먹는다.

원숭이가 우끼끼하며 운다.

원숭이는 유연한 4개의 다리로 나무를 올랐다.

원숭이는 엉덩이는 빨간색이다.

ArrayList

date. 2021_01_15

HashSet

```
import java.util.ArrayList;
// import 후에 사용할 수 있다.
public class ArrayListTest {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();
        // String 타입의 ArrayList 할당, 변수명은 list

        list.add("MILK");
        list.add("BREAD");
        list.add("BUTTER");
        // 변수명.add("element") = ArrayList에 element 추가

        list.add(1, "APPLE");
        list.add(2, "GRAPE");
        // 변수명.add(index, "element") = ArrayList index 번째에 e

        for(int i = 0; i < list.size(); i++) {
            // ArrayList의 데이터를 대입한 만큼?
            System.out.println(list.get(i));
            // ArrayList의 (i) 번째를 출력한다.
        }

        System.out.println("*****");
        list.remove(3);
        // ArrayList의 3번째 데이터를 지운다.

        for(int i = 0; i < list.size(); i++) {
            System.out.println(list.get(i));
        }
    }
}
```

```
import java.util.HashSet;
import java.util.Set;

public class HashSetTest {
    public static void main(String[] args) {
        Set<String> s1 = new HashSet<String>();
        Set<String> s2 = new HashSet<String>();

        Set<String> union = new HashSet<String>(s1);
        union.addAll(s2);

        Set<String> intersection = new HashSet<String>(s1);
        intersection.retainAll(s2);

        System.out.println("합집합 = " + union);
        System.out.println("교집합 = " + intersection);

        HashSet<String> hs = new HashSet<String>();
        String[] sample = {
            "안녕", "하하", "호호", "크크", "키키", "켈켈"
        };

        for (String s : sample) {
            // Set은 값이 중복되면 add가 되지 않고 false를 리턴
            if (!hs.add(s)) {
                System.out.println("중복!");
            }
        }

        System.out.println(hs.size() + " 출력: " + hs);
    }
}
```

Q. `for(String s : sample)` 이해를 못했습니다.