

자바 클래스

자바 상속

자바 상속(하위 클래스 및 상위 클래스)

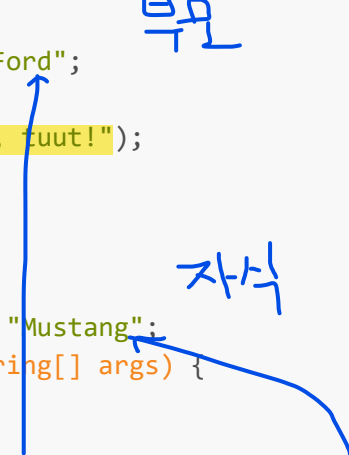
Java에서는 한 클래스에서 다른 클래스로 속성과 메소드를 상속할 수 있습니다. 우리는 "상속 개념"을 두 가지 범주로 분류합니다.

- 하위(sub) 클래스 (자식) - 다른 클래스를 상속받은 클래스
- 상위(super) 클래스 (부모) - 상속하는 클래스

클래스에서 상속하려면 `extends` 키워드를 사용하세요.

아래 예에서 Car클래스(하위 클래스)는 Vehicle클래스(상위 클래스)로부터 속성과 메서드를 상속합니다.

```
class Vehicle {  
    protected String brand = "Ford";  
    public void honk() {  
        System.out.println("Tuut, tuut!");  
    }  
}  
  
class Car extends Vehicle {  
    private String modelName = "Mustang";  
    public static void main(String[] args) {  
        Car myCar = new Car();  
        myCar.honk();  
        System.out.println(myCar.brand + " " + myCar.modelName);  
    }  
}
```



결과 :

Tuut, tuut!
Ford Mustang

Vehicle에서 `protected` 수정자를 발견하셨나요 ?

Vehicle의 브랜드 속성을 `protected` 접근 수정자로 설정했습니다. `private`로 설정하면 Car 클래스에서 접근할 수 없습니다.

"상속"을 사용하는 이유와 시기는 무엇입니까?

- 코드 재사용에 유용합니다. 새 클래스를 생성할 때 기존 클래스의 속성과 메서드를 재사용합니다.

팁: 상속된 메서드를 사용하여 다양한 작업을 수행하는 다음 장인 다형성도 살펴보세요 .

자바 다형성

자바 다형성

다형성은 "다양한 형태"를 의미하며 상속을 통해 서로 관련된 클래스가 많을 때 발생합니다.

상속을 통해 다른 클래스의 속성과 메서드를 상속받을 수 있습니다

다형성은 이러한 방법을 사용하여 다양한 작업을 수행합니다.

이를 통해 우리는 단일 작업을 다양한 방식으로 수행할 수 있습니다.

```
class Animal {  
    public void animalSound() {  
        System.out.println("동물의 울음 소리~~");  
    }  
}  
  
class Pig extends Animal {  
    public void animalSound() {  
        System.out.println("The pig says: 꿀꿀");  
    }  
}  
  
class Dog extends Animal {  
    public void animalSound() {  
        System.out.println("The dog says: 멍멍");  
    }  
}
```

클래스로부터 상속받기 위해 `extends` 키워드를 사용한다는 것을 기억하세요.

이제 두 개체 모두에서 개체를 생성하고 메서드를 호출 할 수 있습니다

```
class Animal {  
    public void animalSound() {  
        System.out.println("동물의 울음 소리~~");  
    }  
}  
  
class Pig extends Animal {  
    public void animalSound() {  
        System.out.println("The pig says: 꿀꿀");  
    }  
}  
  
class Dog extends Animal {  
    public void animalSound() {  
        System.out.println("The dog says: 멍멍");  
    }  
}  
  
class Main {
```

```

    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        Animal myPig = new Pig();
        Animal myDog = new Dog();
        myAnimal.animalSound();
        myPig.animalSound();
        myDog.animalSound();
    }
}

```

결과

동물의 울음 소리~~

The pig says: 꿀꿀

The dog says: 멍멍

상속"과 "다형성"을 사용하는 이유와 시기는 무엇입니까?

- 코드 재사용에 유용합니다. 새 클래스를 생성할 때 기존 클래스의 속성과 메서드를 재사용합니다.

자바 내부 클래스

자바 내부 클래스

- Java에서는 클래스(클래스 내의 클래스)를 중첩하는 것도 가능합니다.
- 중첩 클래스의 목적은 함께 속한 클래스를 그룹화하여 코드를 더 읽기 쉽고 유지 관리하기 쉽게 만드는 것입니다.

내부 클래스에 액세스하려면 외부 클래스의 객체를 만든 다음 내부 클래스의 객체를 만듭니다.

```

class OuterClass {
    int x = 10;

    class InnerClass {
        int y = 5;
    }
}

public class Main {
    public static void main(String[] args) {
        OuterClass myOuter = new OuterClass();
        OuterClass.InnerClass myInner = myOuter.new InnerClass();
        System.out.println(myInner.y + myOuter.x); // 5 + 10
    }
}

```

결과 : 15

private 내부 클래스

일반 클래스와 달리 내부 클래스는 `private` 또는 `protected` 일 수 있습니다. 외부 객체가 내부 클래스에 액세스하지 못하게 하려면 클래스를 `private`를 사용해서 선언하세요 .

```
class OuterClass {
    int x = 10;

    private class InnerClass {
        int y = 5;
    }
}

public class Main {
    public static void main(String[] args) {
        OuterClass myOuter = new OuterClass();
        OuterClass.InnerClass myInner = myOuter.new InnerClass();
        System.out.println(myInner.y + myOuter.x);
    }
}
```

외부 클래스에서 비공개 내부 클래스에 액세스하려고 하면 오류가 발생합니다.

```
Main.java:13: error: OuterClass.InnerClass has private access in OuterClass
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
```

static 내부 클래스

static 내부 클래스는 외부 클래스의 객체를 생성하지 않고도 액세스할 수 있다는 의미입니다.

```
class OuterClass {
    int x = 10;

    static class InnerClass {
        int y = 5;
    }
}

public class Main {
    public static void main(String[] args) {
        OuterClass.InnerClass myInner = new OuterClass.InnerClass();
        System.out.println(myInner.y);
    }
}
```

결과 :

5

참고 : static 속성 및 메서드와 마찬가지로 static 내부(Inner) 클래스는 외부(Outer) 클래스의 멤버에 액세스할 수 없습니다.

내부 클래스에서 외부 클래스에 액세스

내부 클래스의 한 가지 장점은 외부 클래스의 속성과 메서드에 액세스할 수 있다는 것입니다.

```
class OuterClass {
    int x = 10;

    class InnerClass {
        public int myInnerMethod() {
            return x;
        }
    }
}

public class Main {
    public static void main(String[] args) {
        OuterClass myOuter = new OuterClass();
        OuterClass.InnerClass myInner = myOuter.new InnerClass();
        System.out.println(myInner.myInnerMethod());
    }
}
```

결과

10

추상화

추상화 클래스 추상화 메소드

- 데이터 추상화는 특정 세부 정보를 숨기고 필수 정보만 사용자에게 표시하는 프로세스입니다.
- 추상화는 추상 클래스나 인터페이스를 사용하여 달성할 수 있습니다.

abstract 키워드는 클래스와 메서드에 사용되는 비액세스 수정자입니다.

- Abstract class : 객체를 생성하는 데 사용할 수 없는 제한된 클래스입니다(액세스하려면 다른 클래스에서 상속되어야 함).
- Abstract method : 추상 클래스에서만 사용할 수 있으며 본문이 없습니다. 본문은 하위 클래스(상속됨)에 의해 제공됩니다.

추상 클래스에는 추상 메서드와 일반 메서드가 모두 있을 수 있습니다.

```
// 추상 클래스
abstract class Animal {
    // 추상 메소드 (본문이 없습니다)
    public abstract void animalSound();
    // 일반 메소드
    public void sleep() {
```

재정의

```

        System.out.println("Zzz");
    }
}

// 자식클래스 (Animal 상속받음)
class Pig extends Animal {
    public void animalSound() {
        // 추상 메소드 이므로 반드시 설정해야 함
        System.out.println("돼지 울음소리: 꿀꿀~");
    }
}

class Main {
    public static void main(String[] args) {
        Pig myPig = new Pig(); // 객체 생성
        myPig.animalSound(); // 돼지 울음소리: 꿀꿀~
        myPig.sleep(); // Zzz
    }
}

```

인터페이스(Interface)

인터페이스

- Java에서 추상화를 달성하는 또 다른 방법은 인터페이스를 사용하는 것입니다.
- 인터페이스는 빈 본문으로 관련 메서드를 그룹화하는 데 사용되는 완전히 "추상 클래스"입니다.

```

// 인터페이스
interface Animal {
    public void animalSound(); // 인터페이스 메소드 (본문이 비었습니다)
    public void run(); // 인터페이스 메소드 (본문이 비었습니다)
}

```

인터페이스 메서드에 접근하려면 인터페이스가 (extends 대신) **implements 키워드를 사용하여** 다른 클래스에 의해 **"구현"**(상속된 것과 유사하게)되어야 합니다.

인터페이스 메소드의 본문은 "implement" 클래스에서 제공됩니다.

```

// 인터페이스
interface Animal {
    public void animalSound();
    public void sleep();
}

// Pig는 Animal 인터페이스(interface)를 구현(implements)합니다
class Pig implements Animal {
    public void animalSound() {
        // animalSound()의 본문은 여기에 제공됩니다.
        System.out.println("돼지 울음소리 : 꿀꿀~");
    }
}

```

```

    }
    public void sleep() {
        // sleep()의 본문은 여기에 제공됩니다.
        System.out.println("Zzz");
    }
}

class Main {
    public static void main(String[] args) {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();
        myPig.sleep();
    }
}

```

결과:

돼지 울음소리 : 꿀꿀~

Zzz

- 추상 클래스와 마찬가지로 인터페이스를 사용하여 객체를 생성할 수 없습니다
- 위 예에서는 Main Class에서 Animal클래스와 관련된 객체를 생성할 수 없습니다.
- 인터페이스 메소드에는 본문이 없습니다. 본문은 "구현" 클래스에 의해 제공됩니다.
- 인터페이스 구현 시 해당 메서드를 모두 재정의해야 합니다.
- 인터페이스 메소드는 기본적으로 추상적이고 공개적입니다.
- 인터페이스 속성은 기본적으로 public, static 및 final입니다.
- 인터페이스는 생성자를 포함할 수 없습니다(객체를 생성하는 데 사용할 수 없습니다).

인터페이스를 사용하는 이유와 시기

1. 보안을 달성하려면 특정 세부 정보를 숨기고 개체(인터페이스)의 중요한 세부 정보만 표시합니다.
2. Java는 "다중 상속"(클래스는 하나의 상속클래스에서만 상속할 수 있음)을 지원하지 않습니다.
3. 그러나 클래스가 여러 인터페이스를 구현할 수 있으므로 인터페이스를 사용하여 이를 달성할 수 있습니다.
4. 여러 인터페이스를 구현하려면 심표로 구분하세요.

다중 인터페이스

여러 인터페이스를 구현하려면 심표로 구분하세요.

```

interface FirstInterface {
    public void myMethod(); // 인터페이스 method
}

interface SecondInterface {
    public void myOtherMethod(); // 인터페이스 method
}

class DemoClass implements FirstInterface, SecondInterface {
    public void myMethod() {
        System.out.println("첫번째 인터페이스 출력..");
    }
}

```

```
    }  
    public void myOtherMethod() {  
        System.out.println("두번째 인터페이스 출력...");  
    }  
}  
  
class Main {  
    public static void main(String[] args) {  
        DemoClass myObj = new DemoClass();  
        myObj.myMethod(); // 첫번째 인터페이스 출력..  
        myObj.myOtherMethod(); // 두번째 인터페이스 출력..  
    }  
}
```

결과

첫번째 인터페이스 출력..

두번째 인터페이스 출력...