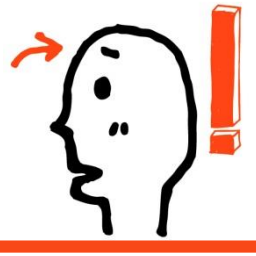
Android Java

13장 클래스의 상속 1: 상속의 기본

클래스의 상속 1: 상속의 기본



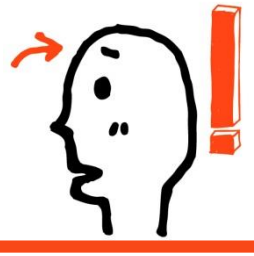
01 상속은 재활용 + 알파

02 상속의 기본문법 이해

03 상속과 접근제어 지시자

04 static 변수(메소드)의 상속과 생성자의 상속에 대한 논의

상속을 알아야 유연한 설계를 할 수 있다



- **상속** : 클래스의 메소드나 속성을 다른 클래스에게 전달해 주는 것
  상속 클래스와 클래스 사이의 관계를 정의하는 것

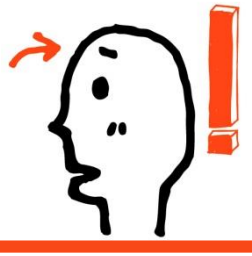
- **부모 클래스(Parent class), 상위 클래스(Super class)** : 자원을 물려주는 클래스

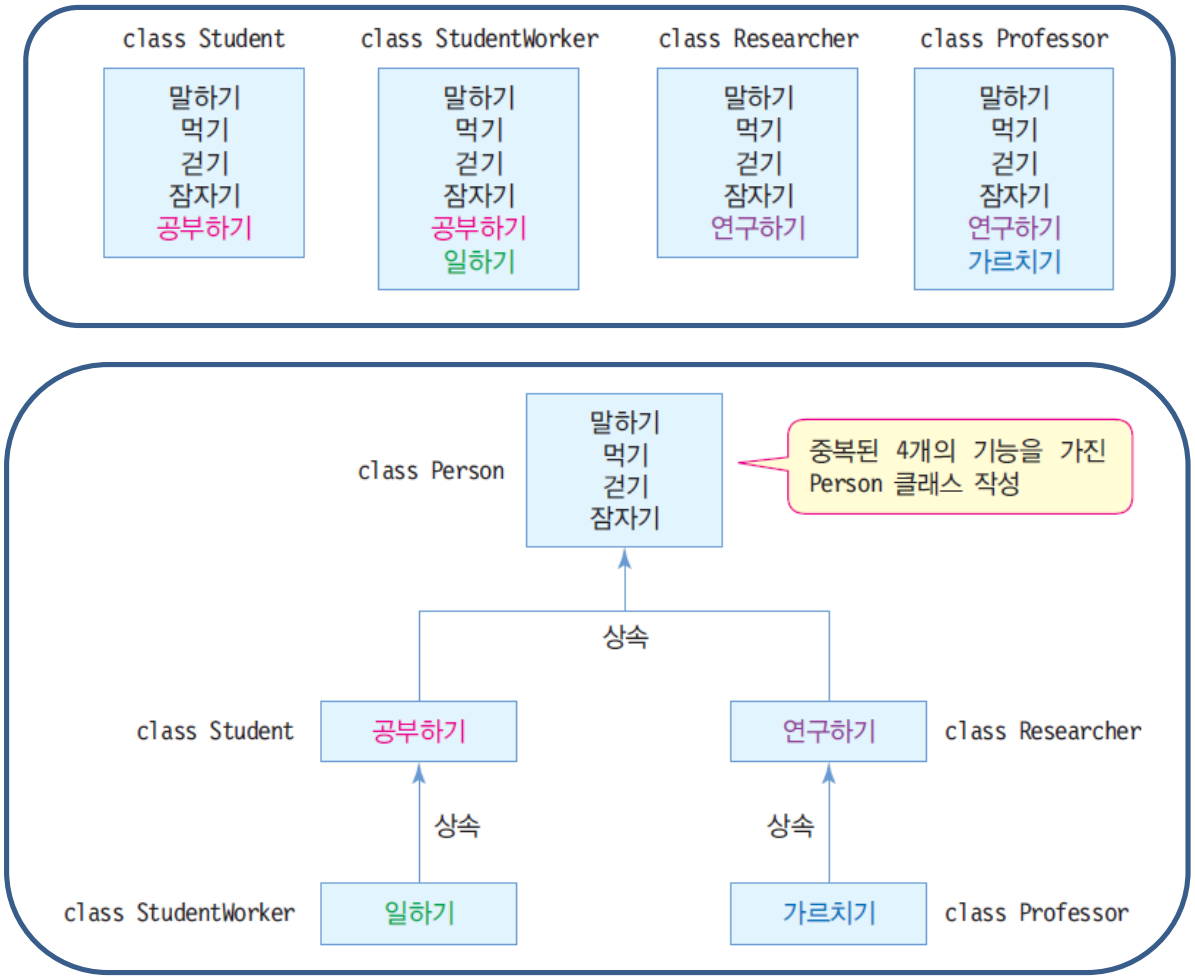
- **자식 클래스(Child class), 하위 클래스(Sub class)** : 물려준 자원을 상속받는 클래스

- **상속의 이유 !**

  상속을 통해 연관된 일련의 클래스에 대한 공통적인 규약을 정의하고 적용하는데,
  상속의 실질적인 목적이 있다!

상속을 알아야 유연한 설계를 할 수 있다



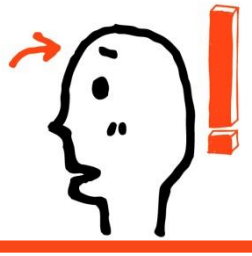


상속이 없는 경우
중복된 멤버를 가진
4 개의 클래스

상속을 이용한
경우 중복이 제거되고
간결해진 클래스 구조

상속은 재활용 + 알파

상속을 알아야 유연한 설계를 할 수 있다



상속 관계임을 알려주는 extends 키워드

- 부모 클래스와 자식 클래스 사이에 상속 관계를 선언할 때 **extends 키워드** 사용
- 자식 클래스 선언부의 클래스 이름 뒤에 extends 키워드를 선언하고 상속받을 부모 클래스의 이름을 입력

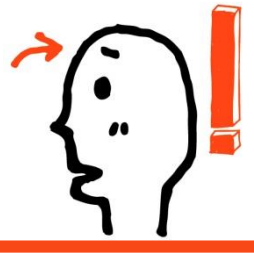
사용법 : [제어자] class [클래스 이름] extends [부모 클래스의 이름]

사용예 :

```
//Parent.java 클래스
public class Parent
{
    //.....
}

//Children.java 클래스
public class Children extends Parent
{
    //.....
}
```

클래스 선언부에서 extends 키워드와 부모 클래스의 이름을 입력하면 됩니다.

상속을 알아야 유연한 설계를 할 수 있다



상속은 언제, 왜 사용할까?

- 객체 지향 프로그래밍에서 상속 기법을 사용하는 이유 : 소스의 재활용
- 재활용되는 소스가 많아진다면 그만큼 개발자의 **편의성**과 **생산성**은 증가

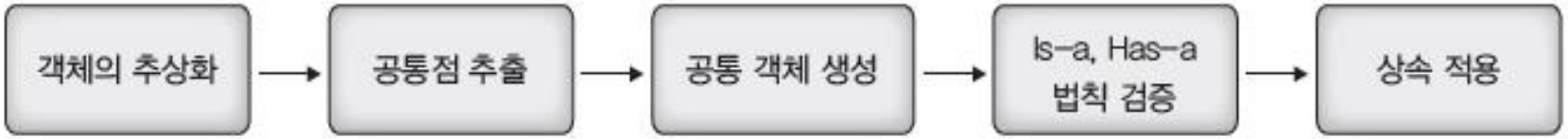
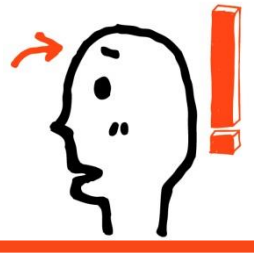


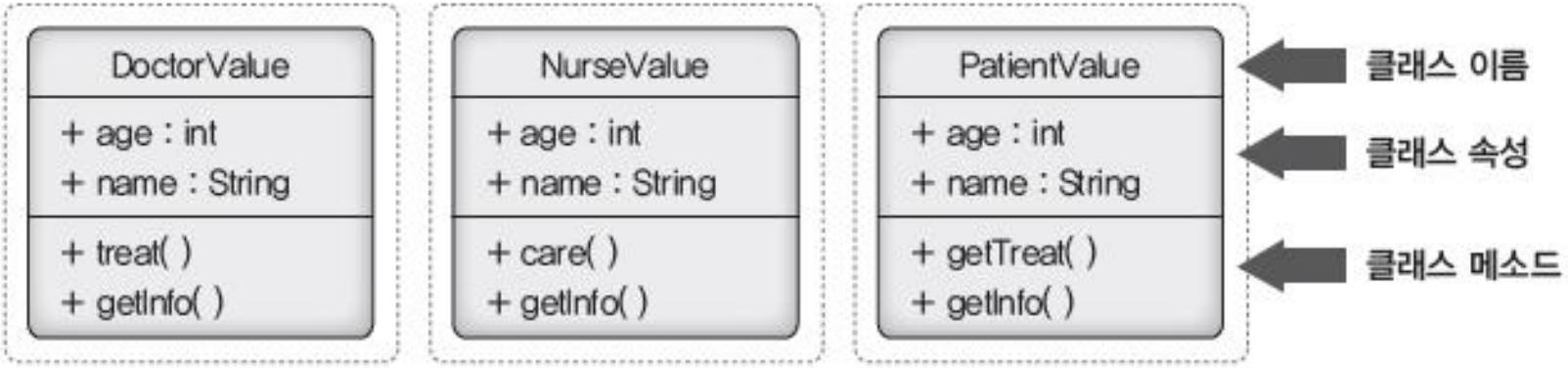
그림 6-1 상속 기법을 적용하는 과정

상속을 알아야 유연한 설계를 할 수 있다



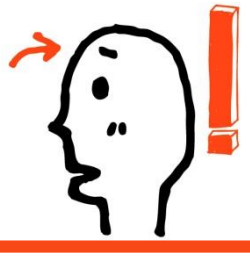
상속은 언제, 왜 사용할까?

1. 개발자 A씨는 종합병원에서 '종합병원 업무 전산화' 시스템을 구축해달라는 의뢰를 받음
2. 개발자 A씨가 '종합병원 업무 전산화' 프로그램을 추상화해보니
   프로그램을 접속하는 사용자에 따른 직업을 표현하기 위한 객체가 필요,
   그 결과 필요한 객체는 '의사' 객체, '간호사' 객체 그리고 '환자' 객체로 정리



△ 그림 6-2 추상화된 클래스의 UML

상속을 알아야 유연한 설계를 할 수 있다



상속은 언제, 왜 사용할까?

- 상속 기법을 적용하기 위해 추상화된 클래스의 속성과 메소드 공통점

공통 클래스 속성 : age 변수, name 변수

공통 메소드 : getInfo( )

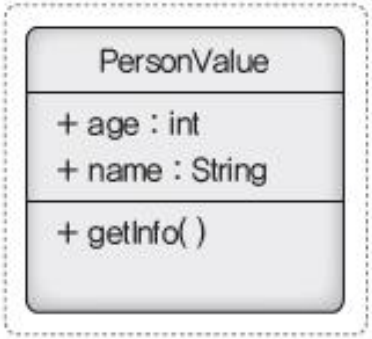
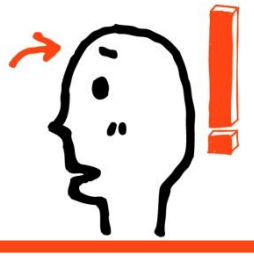


그림 6-3 3개의 객체로부터 뽑은 공통 객체

상속의 가장 기본적인 특성



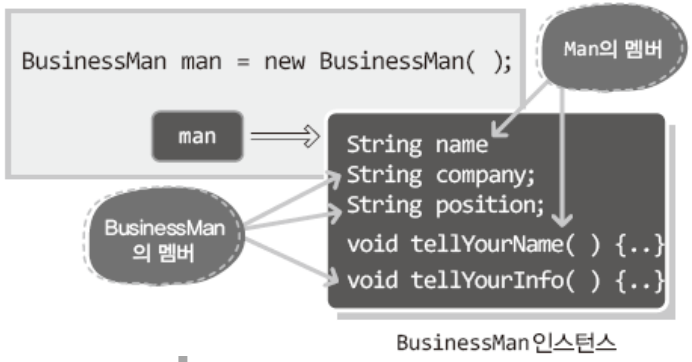
상위 클래스, 기초 클래스

```
class Man
{
    public String name;
    public void tellYourName() {
      System.out.println("My name is "+name); }
}
```

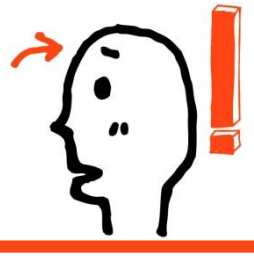
하위 클래스, 유도 클래스

```
class BusinessMan extends Man
{
    public String company;
    public String position;
    public void tellYourInfo()
    {
        System.out.println("My company is "+company);
        System.out.println("My position is "+position);
        tellYourName();
    }
}
```

Man 클래스를 상속했기 때문에 호출가능!



서브 클래스/슈퍼 클래스의 생성자 호출과 실행



질문 1 서브 클래스의 인스턴스가 생성될 때 서브 클래스의 생성자와 슈퍼 클래스의 생성자가 모두 실행되는가? 아니면 서브 클래스의 생성자만 실행되는가?

답 둘 다 실행된다. 생성자는 인스턴스를 초기화할 목적으로 사용되므로 서브 클래스의 생성자는 서브 클래스 내의 멤버를 초기화하거나 필요한 초기화 작업을 수행할 필요가 있고, 슈퍼 클래스의 생성자는 슈퍼 클래스의 멤버를 초기화하거나 필요한 초기화 작업을 수행할 필요가 있기 때문이다.

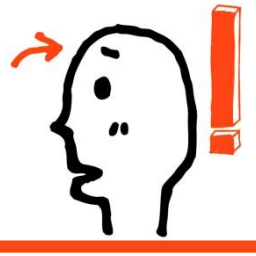
질문 2 서브 클래스의 인스턴스가 생성될 때 서브 클래스의 생성자와 슈퍼 클래스의 생성자의 실행 순서는 어떻게 되는가?

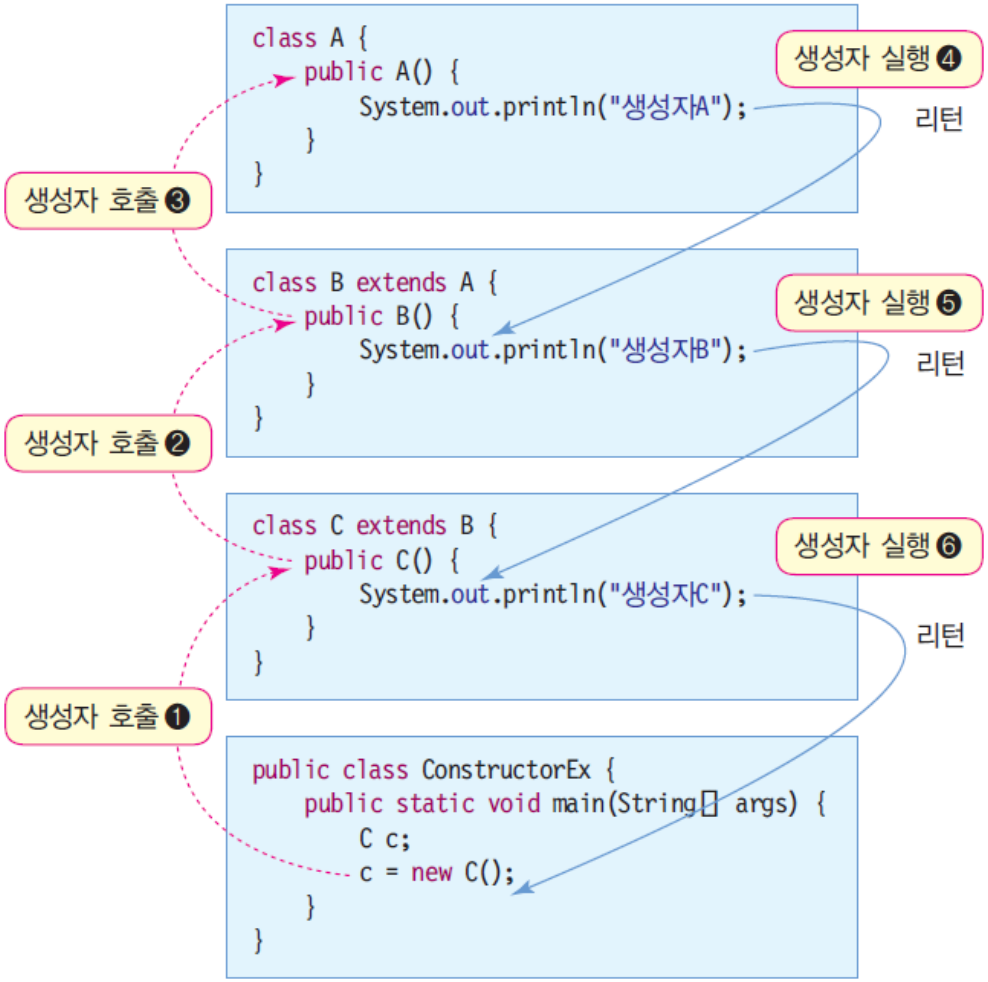
답 슈퍼 클래스의 생성자가 먼저 실행된 후 서브 클래스의 생성자가 실행된다.

- new에 의해 서브 클래스의 객체가 생성될 때
  - 슈퍼클래스 생성자와 서브 클래스 생성자 모두 실행됨
  - 호출 순서
    - 서브 클래스의 생성자가 먼저 호출, 서브 클래스의 생성자는 실행 전 슈퍼 클래스 생성자 호출
  - 실행 순서
    - 슈퍼 클래스의 생성자가 먼저 실행된 후 서브 클래스의 생성자 실행

01

슈퍼클래스와 서브 클래스의 생성자간의 호출 및 실행 관계

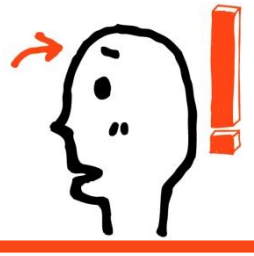




예상 실행 결과는 ?

생성자A
생성자B
생성자C

서브 클래스의 기본 생성자



서브 클래스의 생성자가
슈퍼 클래스의 생성자를
선택하지 않은 경우

```
class A {
    public A() {
        System.out.println("생성자A");
    }
    public A(int x) {
    .....
    }
}
```

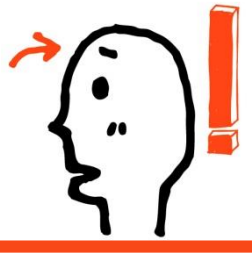
서브클래스의 생성자가 기본 생성자인 경우, 컴파일러는 자동으로 슈퍼클래스의 기본 생성자와 짝을 맺음

```
class B extends A {
    public B() {
        System.out.println("생성자B");
    }
}
```

```
public class ConstructorEx2 {
    public static void main(String[] args) {
        B b;
        b = new B(); // 생성자 호출
    }
}
```

생성자A
생성자B

슈퍼 클래스에 기본 생성자가 없어 오류 난 경우



```
class A {
    public A(int x) {
        System.out.println("생성자A");
    }
}
```

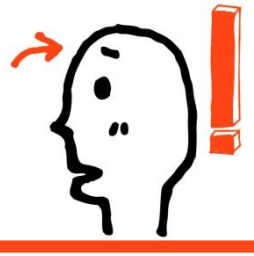
컴파일러가 public B()에 대한 짝을 찾을 수 없음

```
class B extends A {
    public B() {
        System.out.println("생성자B");
    }
}
```

```
public class ConstructorEx2 {
    public static void main(String[] args) {
        B b;
        b = new B();
    }
}
```

컴파일러에 의해 "Implicit super constructor A() is undefined.
Must explicitly invoke another constructor" 오류 발생

서브 클래스의 매개 변수를 가진 생성자 경우



서브 클래스의 생성자가
슈퍼 클래스의 생성자를
선택하지 않은 경우

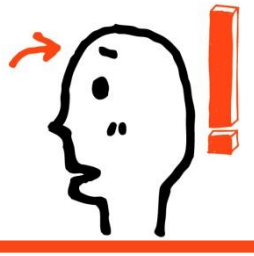
```
class A {
    public A() {
        System.out.println("생성자A");
    }
    public A(int x) {
        System.out.println("매개변수생성자A");
    }
}
```

```
class B extends A {
    public B() {
        System.out.println("생성자B");
    }
    public B(int x) {
        System.out.println("매개변수생성자B");
    }
}
```

```
public class ConstructorEx3 {
    public static void main(String[] args) {
        B b;
        b = new B(5);
    }
}
```

생성자A
매개변수생성자B

super()를 이용한 사례



```
class A {
    public A() {
        System.out.println("생성자A");
    }
    public A(int x) {
        System.out.println("매개변수생성자A" + x);
    }
}
```

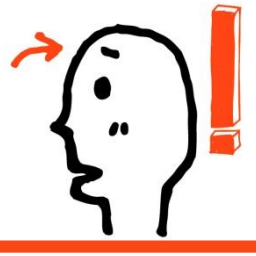
```
class B extends A {
    public B() {
        System.out.println("생성자B");
    }
    public B(int x) {
        super(x);
        System.out.println("매개변수생성자B" + x);
    }
}
```

super(); 라고 하면 A() 호출

```
public class ConstructorEx4 {
    public static void main(String[] args) {
        B b;
        b = new B(5);
    }
}
```

매개변수생성자A5
매개변수생성자B5

상속 관계에 있는 인스턴스의 생성 예



```
class Man
{
    private String name;
    public Man(String name) { this.name=name;}
    public void tellYourName() { System.out.println("My name is "+name); }
}

class BusinessMan extends Man
{
    private String company;        // 회사이름
    private String position;       // 직급
    public BusinessMan(String name, String company, String position)
    {
        super(name);          // 상위 클래스의 생성자 호출문
        this.company=company;
        this.position=position;
    }
    public void tellYourInfo()
    {
        System.out.println("My company is "+company);
        System.out.println("My position is "+position);
        tellYourName();     // Man 클래스를 상속했기 때문에 호출 가능!
    }
}
```

외부에서 호출하는 것은 BusinessMan 클래스의 생성자이니, 이 생성자가 부모 클래스의 인스턴스 변수를 초기화할 의무를 지닌다.

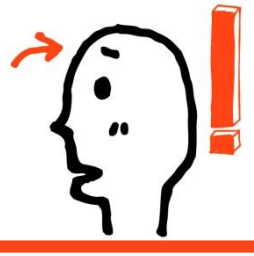
BusinessMan 인스턴스 생성시 초기화해야 할 인스턴스 변수는 다음과 같다.

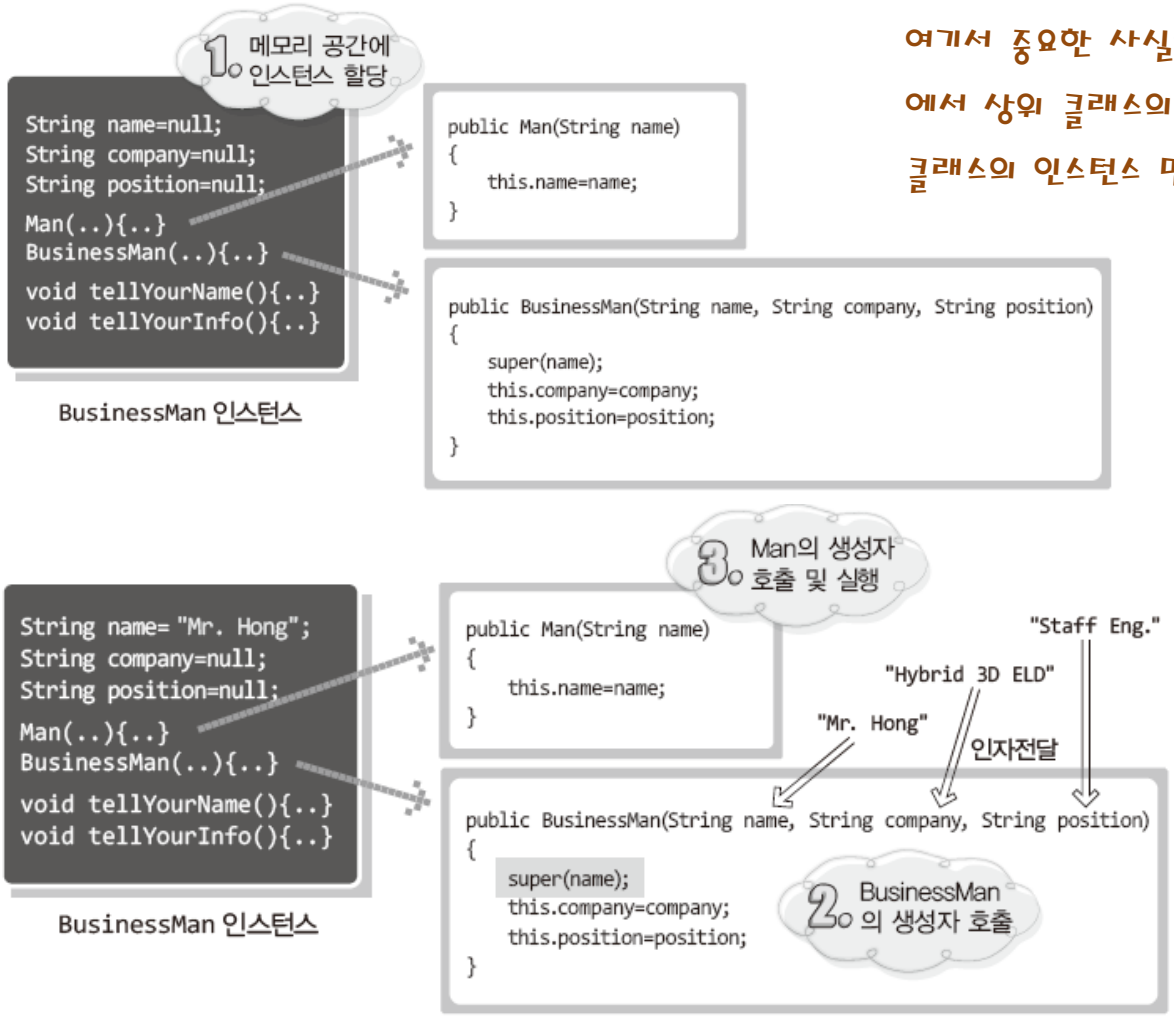
name, company, position

name을 인자로 전달받는 상위 클래스의 생성자를 호출하겠다. 클래스 Man 클래스 BusinessMan

```
public static void main(String[] args)
{
    BusinessMan man1
        =new BusinessMan("Mr. Hong", "Hybrid 3D ELD", "Staff Eng.");
    BusinessMan man2
        =new BusinessMan("Mr. Lee", "Hybrid 3D ELD", "Assist Eng.");

    System.out.println("First man info.....");
    man1.tellYourName();
    man1.tellYourInfo();
    System.out.println("Second man info.....");
    man2.tellYourInfo();
}
```

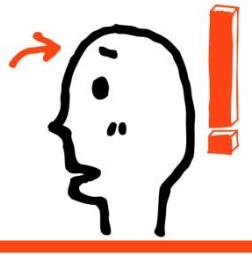
상속관계에 있는 인스턴스의 생성과정 1~3

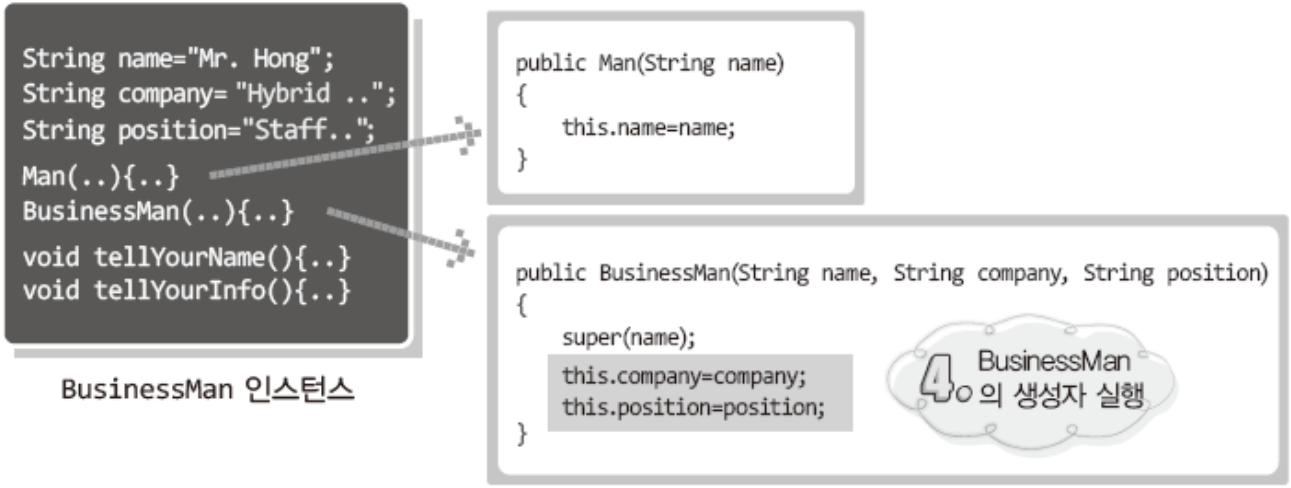




여기서 중요한 사실은 하위 클래스의 생성자 내에서 상위 클래스의 생성자 호출을 통해서 상위 클래스의 인스턴스 멤버를 초기화 한다는 점이다!

상속관계에 있는 인스턴스의 생성과정 1~3



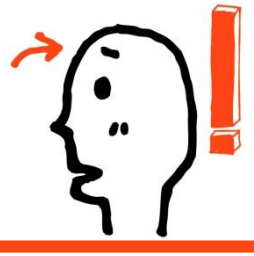


결론 1! 하위 클래스의 생성자는 상위 클래스의 인스턴스 변수를 초기화 할 데이터까지 인자로 전달받아야 한다!

결론 2! 하위 클래스의 생성자는 상위 클래스의 생성자 호출을 통해서 상위 클래스의 인스턴스 변수를 초기화 한다!

결론 3! 키워드 super는 상위 클래스의 생성자 호출에 사용된다. super와 함께 표시된 전달되는 인자의 수와 자료형을 참조하여 호출할 생성자가 결정된다!

반드시 호출되어야 하는 상위 클래스의 생성자



```
class AAA
{
    int num1;
}

class BBB extends AAA
{
    int num2;
}
```

AAA() { }

자동으로 삽입되는 디폴트 생성자의 형태

BBB() { super(); }

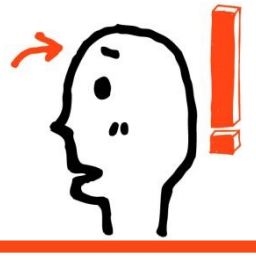
```
class AAA
{
    int num1;
}

class BBB extends AAA
{
    int num2;
    BBB() { num2=0; }
}
```

결과적으로 어떠한 형태로건(프로그래머가 직접 삽입하건, 컴파일러가 삽입하건) 상위 클래스의 생성자는 반드시 호출이 이뤄진다!

자동으로 삽입되는 상위 클래스의 생성자 호출문!

super();

protected 지시자



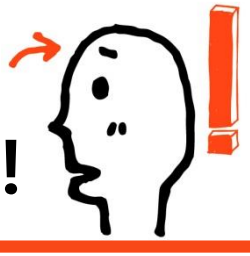
지시자	클래스 내부	동일 패키지	상속받은 클래스	이외의 영역
private	●	×	×	×
default	●	●	×	×
protected	●	●	●	×
public	●	●	●	●

default 패키지로 묶인 두 클래스

```
class AAA
{
    int num1;
    protected int num2;
}

class BBB extends AAA
{
    BBB()
    {
        num1=10;     // AAA 클래스의 default 멤버에 접근
        num2=20;     // BBB 클래스의 protected 멤버에 접근
    }
}
```

이 둘은 상속관계에 앞서 동일 패키지로 묶인 BBB 클래스에 의해 접근 가능!

private 멤버도 상속은 된다. 다만 접근만 불가능!

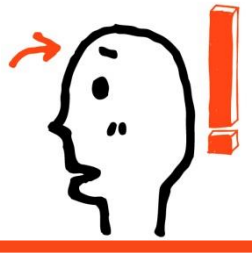


```
class Accumulator
{
    private int val;
    Accumulator(int init){ val=init; }
    protected void accumulate(int num)
    {
        if(num<0)
            return;
        val+=num;
    }
    protected int getAccVal(){return val;}
}
```

private 멤버도 상속이 된다! 다만, 함께 상속된 다른 메소드를 통해서 간접 접근을 해야만 한다!

```
class SavingAccount extends Accumulator
{
    public SavingAccount(int initDep)
    {
        super(initDep);
    }
    public void saveMoney(int money)
    {
        accumulate(money);
    }
    public void showSavedMoney()
    {
        System.out.print("지금까지의 누적금액 : ");
        System.out.println(getAccVal());
    }
}
```

static 변수도 상속이 되나요?



static 변수는 접근의 허용여부와 관계가 있다. 따라서 다음과 같이 질문을 해야 옳다!

"상위 클래스의 static 변수에 하위 클래스도 그냥 이름만으로 접근이 가능한가요?"

이 질문에 대한 답은 YES!

```
class Adder
{
    public static int val=0;
    public void add(int num) { val+=num; }
}

class AdderFriend extends Adder
{
    public void friendAdd(int num) { val+=num; }
    public void showVal() { System.out.println(val); }
}
```

상위 클래스의 static 변수에 이름으로 직접 접근이 가능!

THANK YOU

실무에서 알아야 할 기술은 따로 있다! 자바를 다루는 기술