

클래스의 상속 2

15-1.

상속을 위한 두 클래스의 관계

상속 관계에 놓은 두 대상의 관찰

상속의 특성

하위 클래스는 상위 클래스의 모든 특성을 지닌다.

거기에 더하여 하위 클래스는 자신만의 추가적인 특성을 더하게 된다.



따라서 다음과 같이 표현 가능!

모바일폰 vs. 스마트폰

모바일폰을 스마트폰이 상속한다.

```
class 스마트폰 extends 모바일폰 {...}
```

이렇듯, 상속 관계에 있는 두 대상은
IS-A 관계를 가져야 한다.

상속과 IS-A 관계

IS-A 관계

.. 은 .. 이다. 의 관계

Life **is a** journey.

ex) 노트북은 컴퓨터이다. 전기자동차는 자동차이다.

IS-A 관계를 갖지 않는 두 클래스가 상속으로 연결되어 있다면, 적절한 상속인지 의심해야 한다.

IS-A 관계의 예

```
class MobilePhone {
    protected String number;    // 전화번호

    public MobilePhone(String num) {
        number = num;
    }
    public void answer() {
        System.out.println("Hi~ from " + number);
    }
}

class SmartPhone extends MobilePhone {
    private String androidVer;    // 안드로이드 운영체제 네임(버전)

    public SmartPhone(String num, String ver) {
        super(num);
        androidVer = ver;
    }
    public void playApp() {
        System.out.println("App is running in " + androidVer);
    }
}
```

```
public static void main(String[] args) {
    SmartPhone phone =
        new SmartPhone("010-555-777", "Nougat");
    phone.answer();    // 전화를 받는다.
    phone.playApp();    // 앱을 선택하고 실행한다.
}
```

명령 프롬프트

```
C:\JavaStudy>java MobileSmartPhone
Hi~ from 010-555-777
App is running in Nougat
C:\JavaStudy>
```

15-2. 메소드 오버라이딩

상위 클래스의 참조변수가 참조할 수 있는 대상의 범위

```
class SmartPhone extends MobilePhone {...}
```

스마트폰은 모바일폰이다.

```
SmartPhone phone = new SmartPhone("010-555-777", "Nougat");
```

따라서 스마트폰 참조변수로 스마트폰 참조 가능하고,

이 역은 성립하지 않음에 주의!

```
MobilePhone phone = new SmartPhone("010-555-777", "Nougat");
```

모바일폰 참조변수로 스마트폰 참조도 가능하다.

참조변수의 참조 가능성 관련 예제

```
class MobilePhone {
    protected String number;

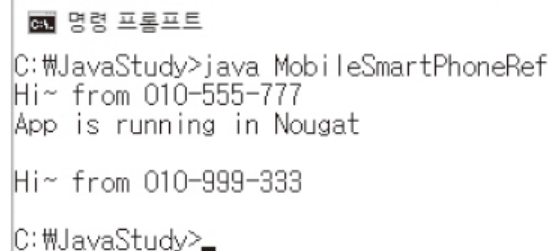
    public MobilePhone(String num) {
        number = num;
    }
    public void answer() {
        System.out.println("Hi~ from " + number);
    }
}

class SmartPhone extends MobilePhone {
    private String androidVer;

    public SmartPhone(String num, String ver) {
        super(num);
        androidVer = ver;
    }
    public void playApp() {
        System.out.println("App is running in " + androidVer);
    }
}
```

```
public static void main(String[] args) {
    SmartPhone ph1 =
        new SmartPhone("010-555-777", "Nougat");
    MobilePhone ph2 =
        new SmartPhone("010-999-333", "Nougat");
    ph1.answer();
    ph1.playApp();
    System.out.println();

    ph2.answer();
    // ph2.playApp();
}
```



```
C:\JavaStudy>java MobileSmartPhoneRef
Hi~ from 010-555-777
App is running in Nougat

Hi~ from 010-999-333
C:\JavaStudy>
```


참조변수의 참조 가능성에 대한 정리

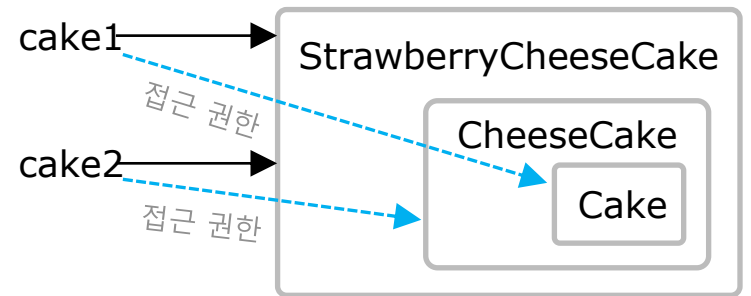
```
class Cake {  
    public void sweet() {...}  
}
```

```
class CheeseCake extends Cake {  
    public void milky() {...}  
}
```

```
class StrawberryCheeseCake extends CheeseCake {  
    public void sour() {...}  
}
```

```
Cake cake1 = new StrawberryCheeseCake();
```

```
CheeseCake cake2 = new StrawberryCheeseCake();
```



StrawberryCheeseCake 인스턴스

참조변수 간 대입과 형 변환

```
class Cake {  
    public void sweet() {...}  
}
```

```
class CheeseCake extends Cake {  
    public void milky() {...}  
}
```

```
CheeseCake ca1 = new CheeseCake();
```

```
Cake ca2 = ca1;    // 가능!
```

```
Cake ca3 = new CheeseCake();
```

```
CheeseCake ca4 = ca3;    // 불가능!
```

이 시점에 컴파일러 및 가상머신은 `ca3`가 참조하는 대상을 `Cake` 인스턴스로 판단한다!

`ca3`가 참조하는 인스턴스의 정확한 클래스 정보는 유지하지 않는다.

참조변수의 참조 가능성: 배열 기반

```
class Cake {  
    public void sweet() {...}  
}
```

```
class CheeseCake extends Cake {  
    public void milky() {...}  
}
```

Cake cake = new CheeseCake(); 가능!

CheeseCake[] cakes = new CheeseCake[10]; 가능!

Cake[] cakes = new CheeseCake[10]; 가능!

상속의 관계가 배열 인스턴스의 참조 관계까지 이어진다.

메소드 오버라이딩1

```
class Cake {  
    public void yummy() {  
        System.out.println("Yummy Cake");  
    }  
}
```

```
class CheeseCake extends Cake {  
    public void yummy() {  
        System.out.println("Yummy Cheese Cake");  
    }  
}
```

← 오버라이딩 관계

CheeseCake의 yummy 메소드가

Cake의 yummy 메소드를 오버라이딩!

```
public static void main(String[] args) {  
    Cake c1 = new CheeseCake();  
    CheeseCake c2 = new CheeseCake();  
  
    c1.yummy();  
    c2.yummy();  
}
```

메소드 오버라이딩 2

```
class Cake {
    public void yummy() {...}
}
class CheeseCake extends Cake {
    public void yummy() {...}    // Cake의 yummy를 오버라이딩
}
class StrawberryCheeseCake extends CheeseCake {
    public void yummy() {...}    // 그리고 CheeseCake의 yummy를 오버라이딩
}

public static void main(String[] args) {
    Cake c1 = new StrawberryCheeseCake();
    CheeseCake c2 = new StrawberryCheeseCake();
    StrawberryCheeseCake c3 = new StrawberryCheeseCake();
    c1.yummy();
    c2.yummy();
}
```

오버라이딩 된 메소드 호출하는 방법

```
class Cake {  
    public void yummy() {  
        System.out.println("Yummy Cake");  
    }  
}  
  
class CheeseCake extends Cake {  
    public void yummy() {  
        super.yummy();  
        System.out.println("Yummy Cheese Cake");  
    }  
  
    public void tasty() {  
        super.yummy();  
        System.out.println("Yummy Tasty Cake");  
    }  
}
```

오버라이딩 된 메소드를 인스턴스 외부에서 호출하는 방법은 없다.

그러나 인스턴스 내부에서는 키워드 `super`를 이용해 호출 가능

인스턴스 변수와 클래스 변수도 오버라이딩이 되는가?

```
class Cake {  
    public int size;  
    ....  
}
```

```
class CheeseCake extends Cake {  
    public int size;  
    ....  
}
```

```
CheeseCake c1 = new CheeseCake();  
c1.size = ... // CheeseCake의 size에 접근
```

```
Cake c2 = new CheeseCake();  
C2.size = ... // Cake의 size에 접근
```

인스턴스 변수는 오버라이딩 되지 않는다. 따라서 참조변수의 형에 따라 접근하는 멤버가 결정된다.

15-3. isinstance 연산자

instanceof 연산자의 기본

```
class Cake {  
}
```

```
class CheeseCake extends Cake {  
}
```

```
class StrawberryCheeseCake extends CheeseCake{  
}
```

```
public static void main(String[] args) {  
    Cake cake = new StrawberryCheeseCake();
```

```
    if(cake instanceof Cake) {...}
```

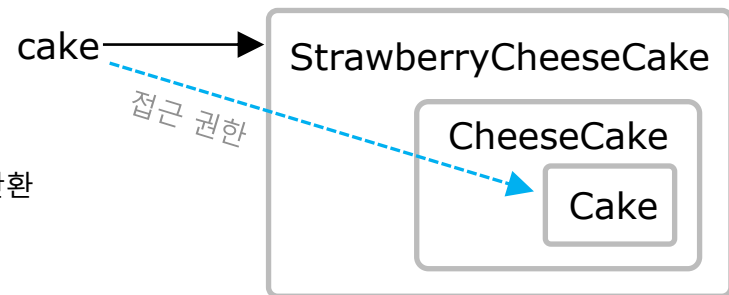
```
    if(cake instanceof CheeseCake) {...}
```

```
    if(cake instanceof StrawberryCheeseCake) {...}  
}
```

if(ref instanceof ClassName)

ref가 ClassName 클래스의 인스턴스를 참조하면 true 반환

ref가 ClassName를 상속하는 클래스의 인스턴스이면 true 반환



Instancedof 연산자의 활용

“상속은 연관된 일련의 클래스들에 대해
공통적인 규약을 정의할 수 있습니다.”

```
class Box {
    public void simpleWrap() {
        System.out.println("Simple Wrapping");
    }
}

class PaperBox extends Box {
    public void paperWrap() {
        System.out.println("Paper Wrapping");
    }
}

class GoldPaperBox extends PaperBox {
    public void goldWrap() {
        System.out.println("Gold Wrapping");
    }
}
```

instanceof 연산자의 사용 예!

그런데 이 예제 코드의 완성도에 점수를 준다면?

```
public static void main(String[] args) {
    Box box1 = new Box();
    PaperBox box2 = new PaperBox();
    GoldPaperBox box3 = new GoldPaperBox();

    wrapBox(box1);
    wrapBox(box2);
    wrapBox(box3);
}

public static void wrapBox(Box box) {
    if (box instanceof GoldPaperBox) {
        ((GoldPaperBox)box).goldWrap();
    }
    else if (box instanceof PaperBox) {
        ((PaperBox)box).paperWrap();
    }
    else {
        box.simpleWrap();
    }
}
```