JAVA

- 객체지향 : 상속

클래스의 상속
: 상속의 기본

상속은 재활용 + 알파

상속

상속은 기존에 정의된 클래스에

메소드와 변수를 추가하여

새로운 클래스를 정의 하는 것!

상속의 이유

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

상속의 기본 문법 이해

상속(inheritance)의 정의와 장점

- 상속이란?
  - 기존의 클래스를 재사용해서 새로운 클래스를 작성하는 것.
  - 두 클래스를 조상과 자손으로 관계를 맺어주는 것.
  - **자손은 조상의 모든 멤버를 상속받는다.(생성자, 초기화블럭 제외)**
  - 자손의 멤버개수는 조상보다 적을 수 없다.(같거나 많다.)

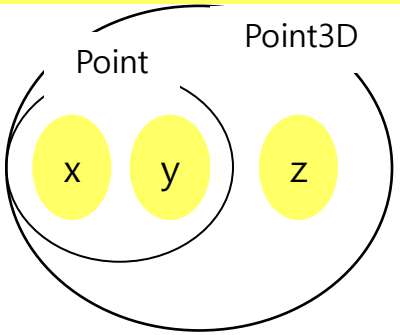
```
class Point {
    int x;
    int y;
}
```

```
class 자손클래스 extends 조상클래스 {
    // ...
}
```

```
class Point3D {
    int x;
    int y;
    int z;
}
```

→

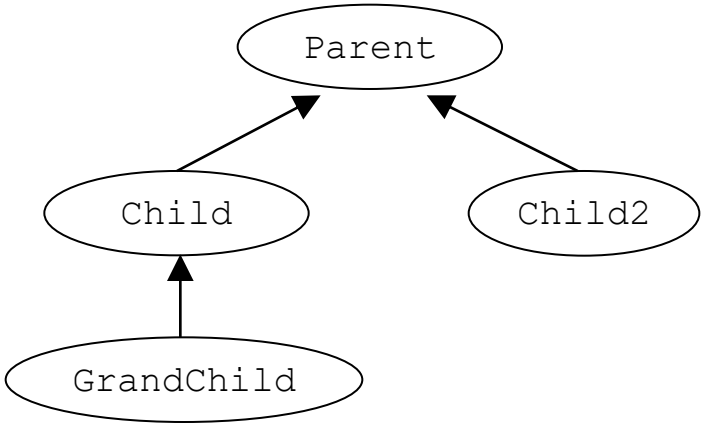
```
class Point3D extends Point {
    int z;
}
```

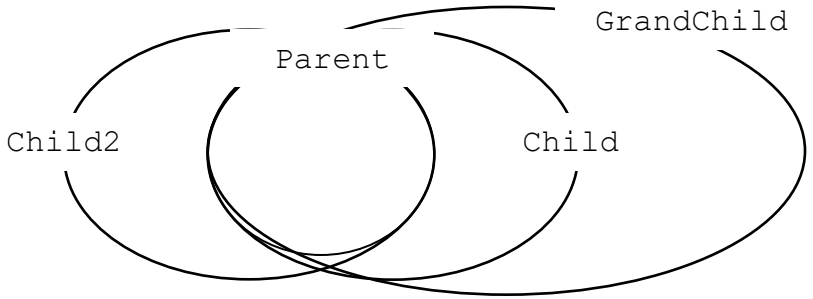


클래스간의 관계 – 상속관계(inheritance)

- 공통부분은 조상에서 관리하고 개별부분은 자손에서 관리한다.
- 조상의 변경은 자손에 영향을 미치지만, 자손의 변경은 조상에 아무런 영향을 미치지 않는다.

```
class Parent {}
class Child extends Parent {}
class Child2 extends Parent {}
class GrandChild extends Child {}
```





상속 관계에 있는 인스턴스의 생성과정

- 여기서 중요한 사실은
  하위클래스의 생성자 내에서 상위클래스의 생성자호출을 통해서
  상위 클래스의 인스턴스 멤버를 초기화 한다는 점이다!
- 하위 클래스의 생성자는 상위클래스의 인스턴스 변수를 초기화할 데이터까지 인자로 전달받아야 한다!
- 하위 클래스의 생성자는 상위 클래스의 생성자 호출을 통해서 상위 클래스의 인스턴스 변수를 초기화한다!
- 키워드 super는 상위 클래스의 생성자 호출에 사용된다.
  Super와 함께 표시된 전달되는 인자의 수와 자료형을 참조하여 호출할 생성자가 결정 된다!

super – 참조변수

- this – 인스턴스 자신을 가리키는 참조변수. 인스턴스의 주소가 저장되어있음
  모든 인스턴스 메서드에 지역변수로 숨겨진 채로 존재
- super – this와 같음. 조상의 멤버와 자신의 멤버를 구별하는 데 사용.

```
class Parent {
    int x=10;
}

class Child extends Parent {
    int x=20;
    void method() {
        System.out.println("x=" + x);
        System.out.println("this.x=" + this.x);
        System.out.println("super.x="+ super.x);
    }
}
```

```
public static void main(String args[]) {
    Child c = new Child();
    c.method();
}
```

반드시 호출되어야 하는 상위 클래스의 생성자

super – 참조변수

- this – 인스턴스 자신을 가리키는 참조변수. 인스턴스의 주소가 저장되어있음
  모든 인스턴스 메서드에 지역변수로 숨겨진 채로 존재
- super – this와 같음. 조상의 멤버와 자신의 멤버를 구별하는 데 사용.

```
class Point {
      int x;
      int y;

      String getLocation() {
            return "x :" + x + ", y :"+ y;
      }
}

class Point3D extends Point {
      int z;
      String getLocation() {       // 오버라이딩
       // return "x :" + x + ", y :"+ y + ", z :" + z;
          return super.getLocation() + ", z :" + z; // 조상의 메서드 호출
      }
}
```

반드시 호출되어야 하는 상위 클래스의 생성자

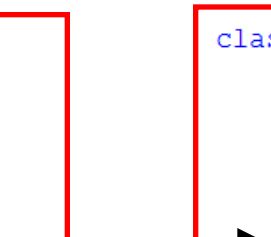
super() – 조상의 생성자

- 자손클래스의 인스턴스를 생성하면, 자손의 멤버와 조상의 멤버가 합쳐진 하나의 인스턴스가 생성된다.

- 조상의 멤버들도 초기화되어야 하기 때문에 자손의 생성자의 첫 문장에서 조상의 생성자를 호출해야 한다.

Object클래스를 제외한 모든 클래스의 생성자 첫 줄에는 생성자(같은 클래스의 다른 생성자 또는 조상의 생성자)를 호출해야한다.
그렇지 않으면 컴파일러가 자동적으로 'super();'를 생성자의 첫 줄에 삽입한다.

```
class Point {
    int x;
    int y;

    Point() {
        this(0,0);
    }

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```



```
class Point extends Object {
    int x;
    int y;

    Point() {
        this(0,0);
    }

    Point(int x, int y) {
        super(); // Object();
        this.x = x;
        this.y = y;
    }
}
```

예제

문제 1.
상관관계에 놓여있는 클래스의 생성자 정의 및 호출방식에 대해 알아보았다.
이 내용을 바탕으로 다음 클래스에 적절한 생성자를 넣어보자.
그리고 이의 확인을 위한 maon메서드로 적절히 정의 해 보자.

```
Class Car {
        int gasolinegauge;
}
Class HybridCar extends Car {
        int electronicGauge;
}
Class HybridWaterCar  extends HybridCar {
        int waterGauge;
        public void showCurrentGauge() {
                System.out.println("잔여 가솔린 : " + gasolinegauge);
                System.out.println("잔여 가솔린 : " + electronicGauge);
                System.out.println("잔여 가솔린 : " + waterGauge);
        }
}
```

Car 클래스는 가솔린 차량을 표현한 것이고, HybridCar 클래스는 가솔린과 전기로 동작하는 차량, HybridWaterCar 클래스는 가솔린, 전기, 물 모두 사용 가능한 가상의 차를 표현.

static 변수(메소드)의 상속과 생성자의 상속에 대한 논의

static 변수도 상속이 되나요?

static 변수는 접근의 허용 여부와 관계가 있다.

따라서 다음과 같이 질문을 해야 옳다!

"상위클래스의static 변수에 하위 클래스도 그냥 이름만으로 접근이 가능 한가요?"

이 질문에 대한 답은 **YES!**

static 변수도 상속이 되나요?

```
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);
        }
}
```

static 변수도 상속이 되나요?

```
class StaticInheritance
{
        public static void main(String[] args)
        {
                Adder ad=new Adder();
                AdderFriend af=new AdderFriend();
                ad.add(1);
                af.friendAdd(3);
                AdderFriend.val+=5;
                af.showVal();
        }
}
```

상속을 위한 관계

클래스간의 관계 – 포함관계(composite)

- 포함(composite)이란?
  - 한 클래스의 멤버변수로 다른 클래스를 선언하는 것
  - 작은 단위의 클래스를 먼저 만들고, 이 들을 조합해서 하나의 커다란 클래스를 만든다.

```
class Circle {
    int x; // 원점의 x좌표
    int y; // 원점의 y좌표
    int r; // 반지름(radius)
}
```

```
class Circle {
    Point c = new Point(); // 원점
    int r; // 반지름(radius)
}
```

```
class Point {
    int x;
    int y;
}
```

```
class Car {
    Engine e = new Engine(); // 엔진
    Door[] d = new Door[4];  // 문, 문의 개수를 넷으로 가정하고 배열로 처리했다.
    //...
}
```

클래스간의 관계결정하기 – 상속 vs. 포함

- 가능한 한 많은 관계를 맺어주어 재사용성을 높이고 관리하기 쉽게 한다.

-'is-a'와 'has-a'를 가지고 문장을 만들어 본다.

원(Circle)은 점(Point)**이다**. - Circle **is a** Point.
원(Circle)은 점(Point)을 **가지고 있다**. - Circle **has a** Point.

상속관계 - '~은 ~이다.(is-a)'
포함관계 - '~은 ~을 가지고 있다.(has-a)'

```
class Point {
    int x;
    int y;
}
```

```
class Circle extends Point{
    int r; // 반지름(radius)
}
```

```
class Circle {
    Point c = new Point(); // 원점
    int r; // 반지름(radius)
}
```

상속을 위한 기본 조건인 IS-A 관계의 성립

상속관계에 있는 두 클래스 사이에는 IS-A 관계가 성립해야 한다.
IS-A 관계가 성립하지 않는 경우에는 상속의 타당성을 면밀히 검토해야 한다.
IS-A 이외에 HAS-A 관계도 상속으로 표현 가능하다. 그러나 HAS-A를 대신 해서Composition 관계를 유지하는 것이 보다 적절한 경우가 많다.

전화기 → 무선 전화기
컴퓨터 → 노트북 컴퓨터

경찰은 총을 가지고 있다.

무선전화기는 일종의 전화기이다.
노트북 컴퓨터는 일종의 컴퓨터이다.

경찰 has a 총.

무선전화기 is a 전화기.
노트북 컴퓨터 is a 컴퓨터.

상속은 강한 연결고리를 형성한다. 때문에 총을 소유하지 않는 경찰, 또는 총이 아닌 경찰봉을 소유하는 경찰등 다양한 표현에 한계를 보인다는 단점이 있다!

IS-A 기반 상속의 예

```
class Computer{
        String owner;
        public Computer(String name) { }
        public void calculate() { }
}
// 노트북 컴퓨터는 컴퓨터 이다.
class NotebookComp extends Computer{
        int battary;
        public NotebookComp(String name, int initChag) { }
        public void charging() { }
        public void movingCal() { }
}
// 타블렛은 노트북 컴퓨터이다.
class TabletNotebook extends NotebookComp{
        String regstPenModel;

        public TabletNotebook(String name, int initChag, String pen) { }
        public void write(String penInfo){ }
}
```

상속을 위한 기본 조건인 IS-A 관계의 성립

```
class Computer
{
	String owner;

	public Computer(String name){owner=name;}
	public void calculate()
	{
		System.out.println("요청 내용을 계산합니다.");
	}
}
```

상속을 위한 기본 조건인 IS-A 관계의 성립

```
class NotebookComp extends Computer{
        int battary;

        public NotebookComp(String name, int initChag){
                super(name);
                battary=initChag;
        }

        public void charging(){
                battary+=5;
        }

        public void movingCal()    {
                if(battary<1) {
                        System.out.println("충전이 필요합니다.");
                        return;
                }
                System.out.print("이동하면서 ");
                calculate();
                battary-=1;
        }
}
```

상속을 위한 기본 조건인 IS-A 관계의 성립

```
class TabletNotebook extends NotebookComp{

        String regstPenModel;

        public TabletNotebook(String name, int initChag, String pen){
                super(name, initChag);
                regstPenModel=pen;
        }

        public void write(String penInfo){
                if(battary<1){
                        System.out.println("충전이 필요합니다.");
                        return;
                }
                if(regstPenModel.compareTo(penInfo)!=0){
                        System.out.println("등록된 펜이 아닙니다.");
                        return;
                }
                System.out.println("필기 내용을 처리합니다.");
                battary-=1;
        }
}
```

상속을 위한 기본 조건인 IS-A 관계의 성립

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

                NotebookComp nc=new NotebookComp("이수종", 5);
                TabletNotebook tn=new TabletNotebook("정수영", 5,
"ISE-241-242");

                nc.movingCal();
                tn.write("ISE-241-242");
        }
}
```

오버라이딩(overriding)이란?

"조상클래스로부터 상속받은 메서드의 내용을 상속받는 클래스에 맞게 변경하는 것을 오버라이딩이라고 한다."

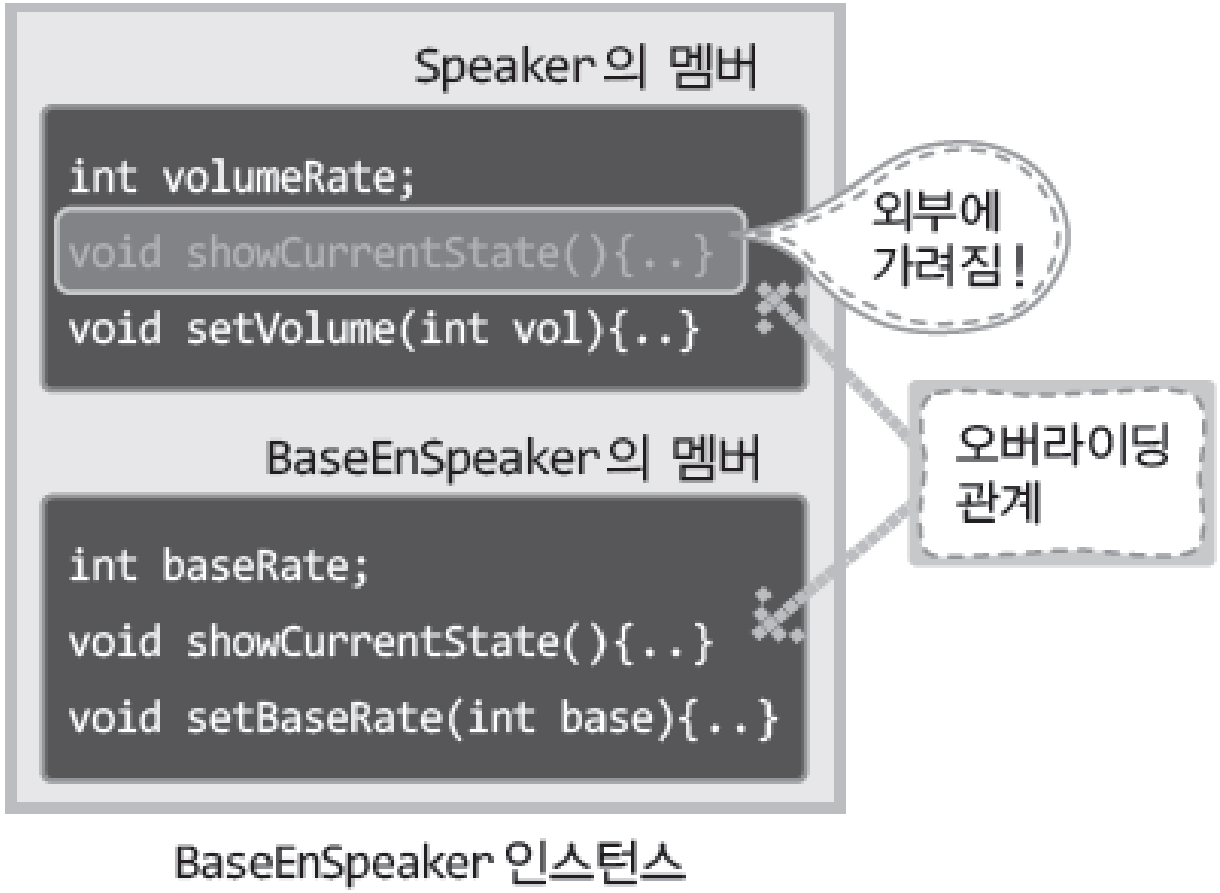
* override - vt. '~위에 덮어쓰다(overwrite).', '~에 우선하다.'

```
class Point {
      int x;
      int y;

      String getLocation() {
            return "x :" + x + ", y :"+ y;
      }
}

class Point3D extends Point {
      int z;
      String getLocation() {        // 오버라이딩
            return "x :" + x + ", y :"+ y + ", z :" + z;
      }
}
```

메소드 오버라이딩

상위 클래스에 정의된 메소드의 이름, 반환형, 매개 변수선언까지 완전히 동일한 메소드를 하위클래스에서 다시 정의하는 것!

하위 클래스에 정의된 메소드에 의해 상위 클래스의 메소드는 가리워진다.



오버라이딩의 조건

1. 선언부가 같아야 한다.(이름, 매개변수, 리턴타입)

2. 접근제어자를 좁은 범위로 변경할 수 없다.

   - 조상의 메서드가 protected라면, 범위가 같거나 넓은 protected나 public으로 만 변경할 수 있다.

3. 조상클래스의 메서드보다 많은 수의 예외를 선언할 수 없다.

```
class Parent {
    void parentMethod() throws IOException, SQLException {
        // ...
    }
}

class Child extends Parent {
    void parentMethod() throws IOException {
        //..
    }
}

class Child2 extends Parent {
    void parentMethod() throws Exception {
        //..
    }
}
```

메소드 오버라이딩

```
class Speaker
{
        private int volumeRate;
        public void showCurrentState()
        {
                System.out.println("볼륨 크기: "+ volumeRate);
        }
        public void setVolume(int vol)
        {
                volumeRate=vol;
        }
}
```

```
class BaseEnSpeaker extends Speaker
{
        private int baseRate;
        public void showCurrentState()
        {
                super.showCurrentState();
                System.out.println("베이스 크기: "+baseRate);
        }
        public void setBaseRate(int base)
        {
                baseRate=base;
        }
}
```

메소드 오버라이딩

```
class Speaker
{
	private int volumeRate;

	public void showCurrentState()
	{
		System.out.println("볼륨 크기: "+ volumeRate);
	}

	public void setVolume(int vol)
	{
		volumeRate=vol;
	}
}
```

메소드 오버라이딩

```
class BaseEnSpeaker extends Speaker
{
        private int baseRate;

        public void showCurrentState()
        {
                super.showCurrentState();
                System.out.println("베이스 크기: "+baseRate);
        }

        public void setBaseRate(int base)
        {
                baseRate=base;
        }
}
```

메소드 오버라이딩

```
class Overriding
{
	public static void main(String[] args)
	{
		BaseEnSpeaker bs=new BaseEnSpeaker();
		bs.setVolume(10);
		bs.setBaseRate(20);
		bs.showCurrentState();
	}
}
```

오버로딩 vs. 오버라이딩

오버로딩(overloading) - 기존에 없는 새로운 메서드를 정의하는 것(new)
오버라이딩(overriding) - 상속받은 메서드의 내용을 변경하는 것(change, modify)

```
class Parent {
    void parentMethod() {}
}

class Child extends Parent {
    void parentMethod() {}          // 오버라이딩
    void parentMethod(int i) {}     // 오버로딩

    void childMethod() {}
    void childMethod(int i) {}      // 오버로딩
    void childMethod() {}           // 에러!!! 중복정의임
}
```

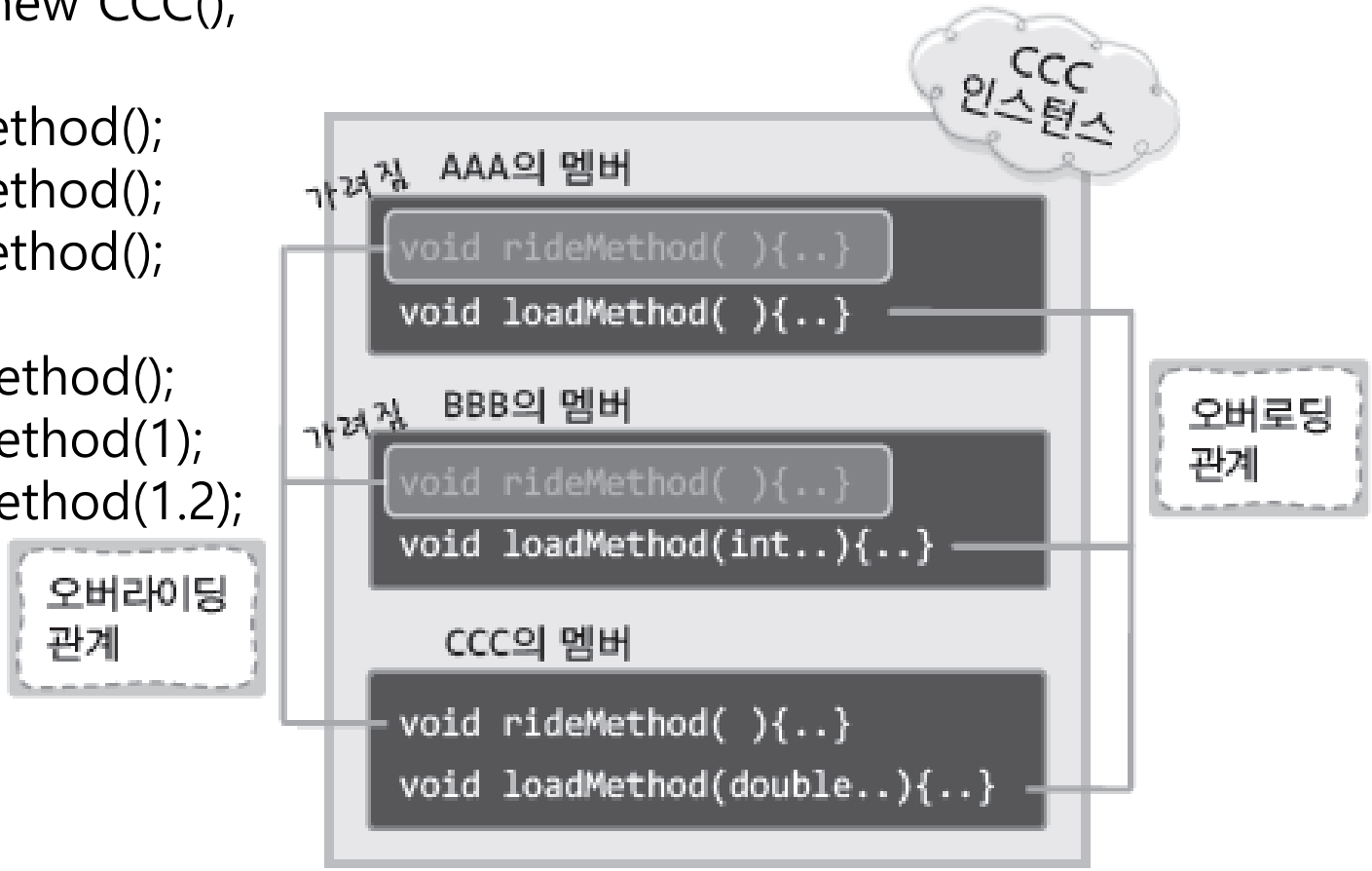
오버라이딩 관계에서의 메소드 호출

```
class AAA
{
        public void rideMethod(){System.out.println("AAA's Method");}
        public void loadMethod(){System.out.println("void Method");}
}
class BBB extends AAA
{
        public void rideMethod(){System.out.println("BBB's Method");}
        public void loadMethod(int num){System.out.println("int Method");}
}
class CCC extends BBB
{
        public void rideMethod(){System.out.println("CCC's Method");}
        public void loadMethod(double num){System.out.println("double
Method");}
}
```

참조 변수의 자료형에 상관 없이 오버라이딩된 메소드는 외부로부터 가려지므로, 마지막으로 오버라이딩한 메소드가 호출 된다!

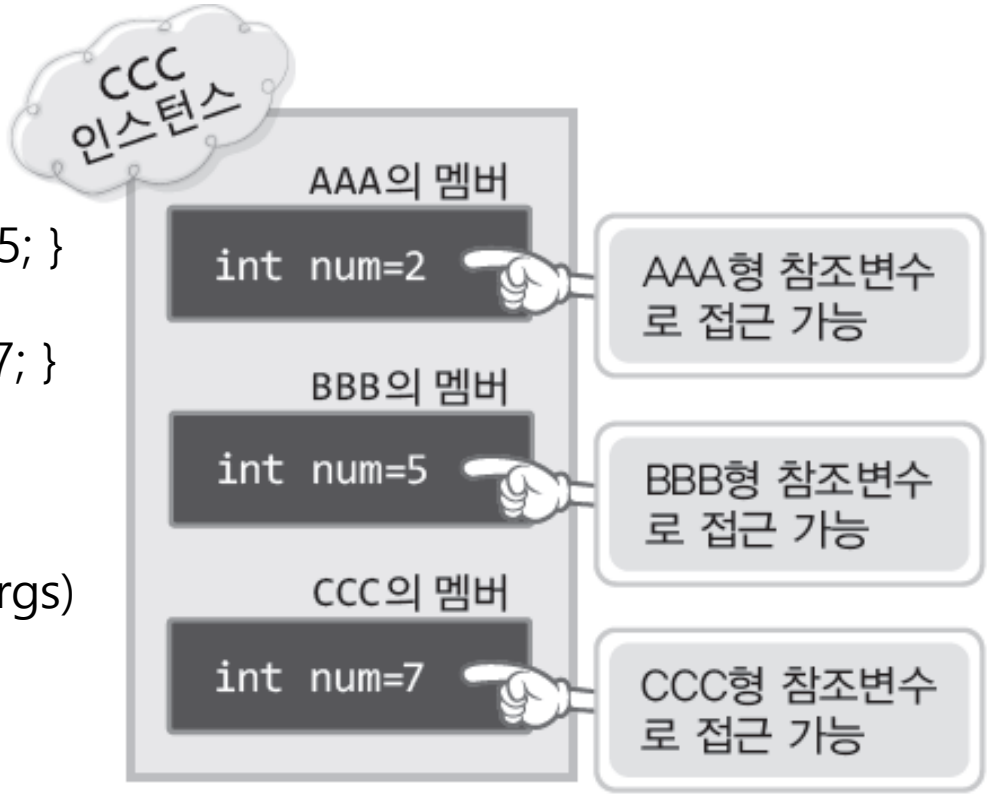
오버라이딩 관계에서의 메소드 호출

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

            AAA ref1=new CCC();
            BBB ref2=new CCC();
            CCC ref3=new CCC();

            ref1.rideMethod();
            ref2.rideMethod();
            ref3.rideMethod();

            ref3.loadMethod();
            ref3.loadMethod(1);
            ref3.loadMethod(1.2);
    }
}
```



인스턴스 변수도 오버라이딩 되나요?

```
class AAA {
        public int num=2;
}

class BBB extends AAA { public int num=5; }

class CCC extends BBB { public int num=7; }

class ValReDecle
{
        public static void main(String[] args)
        {
                CCC ref1=new CCC();
                BBB ref2=ref1;
                AAA ref3=ref2;
                System.out.println("CCC's ref: "+ref1.num);
                System.out.println("BBB's ref: "+ref2.num);
                System.out.println("AAA's ref: "+ref3.num);
        }
}
```



상위 클래스의 참조 변수로 하위 클래스의 인스턴스 참조

중 저음 보강 스피커는 (일종의) 스피커이다. (O)

BaseEnSpeaker is a Speaker. (O)

자바 컴파일러의 실제 관점

스피커는 (일종의) 중 저음 보강 스피커이다. (X)

Speaker is a BaseEnSpeaker. (X)

```
public static void main(String[] args)
{
        Speaker bs= new BaseEnSpeaker();
        bs.setVolume(10);
        bs.setBaseRate(20);          //컴파일 에러
        bs.showCurrentState();
}
```

BaseEnSpeaker도 Speark의 인스턴스 이므로 성립한다.

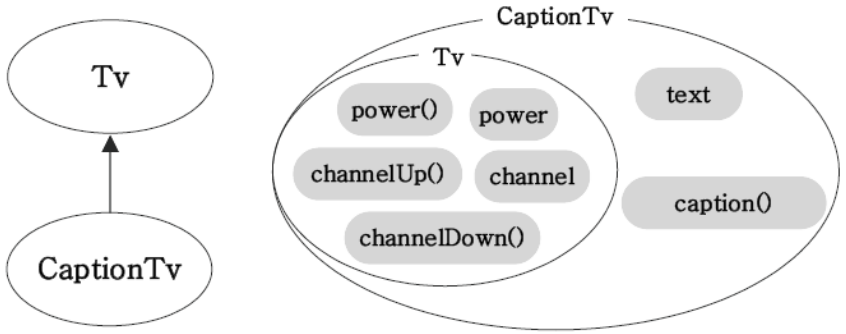
Bs가 참조하는 것은 Speaker의 인스턴스로 인식하기 때문에 BaseEnSpeaker의 멤버에 접근 불가!

다형성(polymorphism)이란?

-"여러 가지 형태를 가질 수 있는 능력"

-"하나의 참조변수로 여러 타입의 객체를 참조할 수 있는 것"
즉, 조상타입의 참조변수로 자손타입의 객체를 다룰 수 있는 것이 다형성.

```
class Tv {
    boolean power;   // 전원상태(on/off)
    int channel;     // 채널

    void power(){    power = !power;}
    void channelUp(){    ++channel; }
    void channelDown(){ --channel; }
}

class CaptionTv extends Tv {
    String text;     // 캡션내용
    void caption() { /* 내용생략 */}
}
```




```
Tv         t = new Tv();
CaptionTv  c = new CaptionTv();
```

```
Tv         t = new CaptionTv();
```

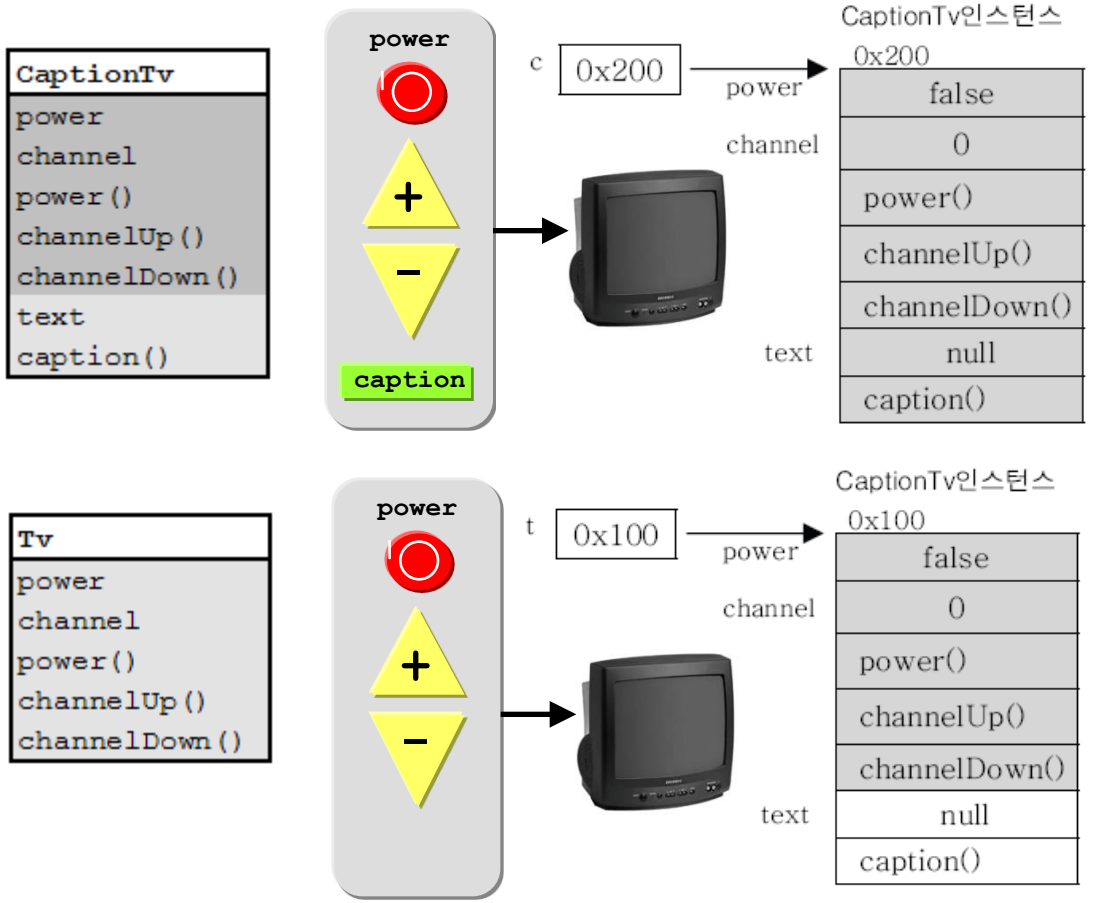
```
CaptionTv c = new CaptionTv();

Tv        t = new CaptionTv();
```

다형성(polyymorphism)

"하나의 참조변수로 여러 타입의 객체를 참조할 수 있는 것"
즉, 조상타입의 참조변수로 자손타입의 객체를 다룰 수 있는 것이 다형성.

```
CaptionTv c = new CaptionTv();

Tv        t = new CaptionTv();
```

```
class Tv {
    boolean power;   // 전원상태(on/off)
    int channel;     // 채널

    void power(){    power = !power;}
    void channelUp(){    ++channel; }
    void channelDown(){ --channel; }
}

class CaptionTv extends Tv {
    String text;     // 캡션내용
    void caption() { /* 내용생략 */}
}
```

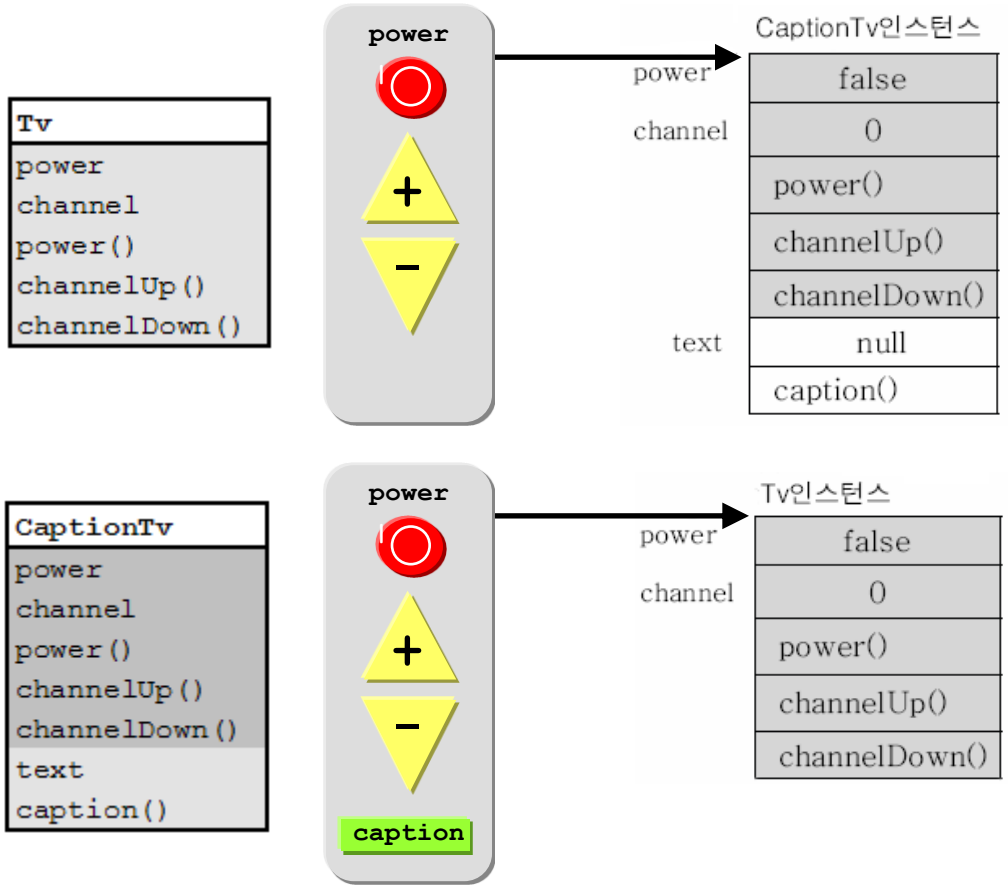


다형성(polyymorphism)

"조상타입의 참조변수로 자손타입의 인스턴스를 참조할 수 있지만,
반대로 자손타입의 참조변수로 조상타입의 인스턴스를 참조할 수는 없다."

```
Tv t = new CaptionTv();

CaptionTv c = new Tv();
```

```
class Tv {
    boolean power;  // 전원상태(on/off)
    int channel;    // 채널

    void power(){   power = !power;}
    void channelUp(){   ++channel; }
    void channelDown(){ --channel; }
}

class CaptionTv extends Tv {
    String text;    // 캡션내용
    void caption() { /* 내용생략 */}
}
```

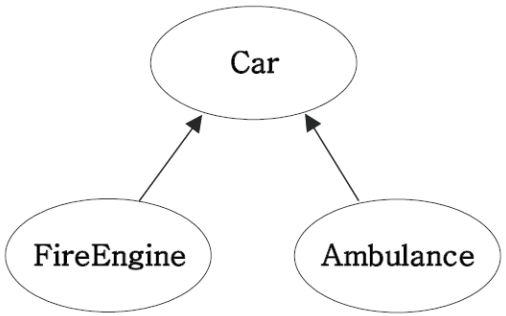


참조변수의 형 변환

- 서로 상속관계에 있는 타입간의 형변환만 가능하다.
- 자손 타입에서 조상타입으로 형변환하는 경우, 형변환 생략가능

자손타입 → 조상타입 (Up-casting) : 형변환 생략가능
자손타입 ← 조상타입 (Down-casting) : 형변환 생략불가

```
class Car {
    String color;
    int door;

    void drive() {  // 운전하는 기능
        System.out.println("drive, Brrrr~");
    }

    void stop() {   // 멈추는 기능
        System.out.println("stop!!!");
    }
}

class FireEngine extends Car {  // 소방차
    void water() {  // 물뿌리는 기능
        System.out.println("water!!!");
    }
}

class Ambulance extends Car {   // 구급차
    void siren() {  // 사이렌을 울리는 기능
        System.out.println("siren~~~");
    }
}
```




```
FireEngine f
Ambulance a;

a = (Ambulance)f;
f = (FireEngine)a;
```

다형성(polyymorphism)

```
class Car {
	String color;
	int door;
	void drive() {			// 운전하는 기능
		System.out.println("drive, Brrrr~");
	}

	void stop() {			// 멈추는 기능
		System.out.println("stop!!!");
	}
}


class FireEngine extends Car {		// 소방차
	void water() {			// 물을 뿌리는 기능
		System.out.println("water!!!");
	}
}
```

다형성(polyymorphism)

```
class CastingTest1 {
        public static void main(String args[]) {
                Car car = null;
                FireEngine fe = new FireEngine();
                FireEngine fe2 = null;

                fe.water();
                car = fe;               // car =(Car)fe;에서 형변환이 생략된 형태다.
//              car.water();        // 에러!!! Car타입의 참조변수로는 water()를 호출
할 수 없다.
                fe2 = (FireEngine)car; // 자손타입 ← 조상타입
                fe2.water();
        }
}
```

참조 변수의 참조 가능성에 대한 일반화 (다형성)

```
class AAA { . . . . . }
class BBB extends AAA { . . . . . }
class CCC extends BBB { . . . . . }
```

아래의 문제 제시를 위한 클래스의 상속관계

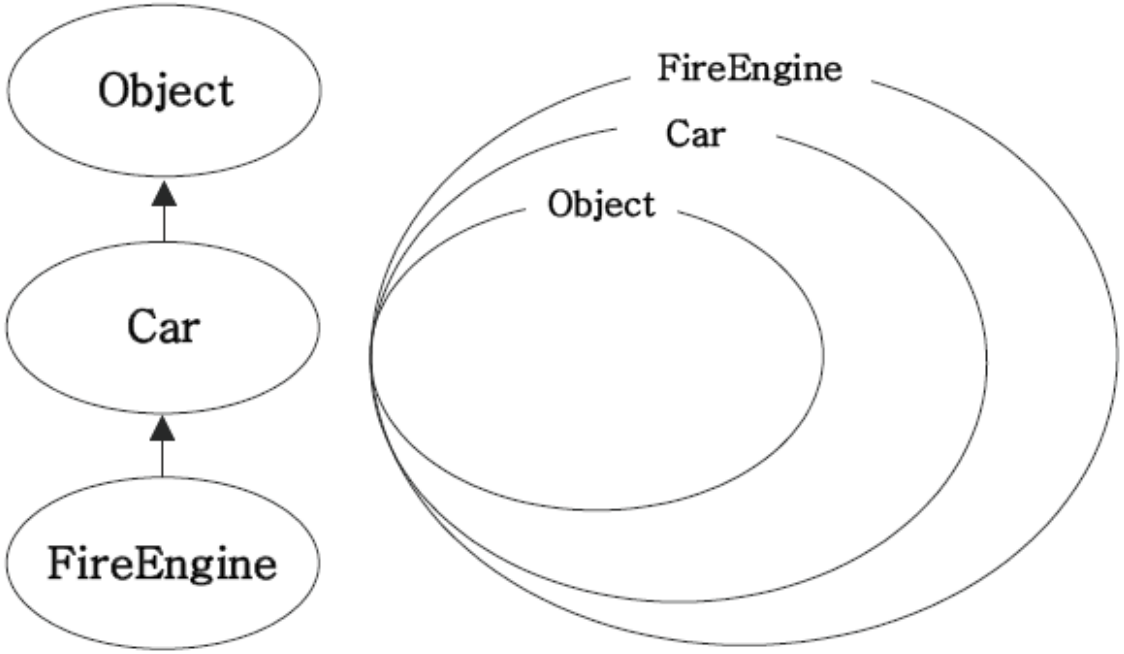
```
AAA ref1 = new BBB();
AAA ref2 = new CCC();     // 가능한가요 ???
BBB ref3 = new CCC();     // 가능한가요 ???
-----------------------------------------------
CCC ref1 = ..............              //컴파일 완료
BBB ref2 = ref1;     // 가능한가요 ???
AAA ref3 = ref1;     // 가능한가요 ???
-----------------------------------------------
AAA ref1 = new CCC();
BBB ref2 = ref1;     // 가능한가요 ???
CCC ref3 = ref1;     // 가능한가요 ???
```

참조변수의 자료형에 따라서 대입연산의 허용 여부가 결정된다.

이 사실을 바탕으로 왼쪽 문장 중에서 컴파일 에러가 발생하는 문장들을 모두 고르면?

instanceof연산자

- 참조변수가 참조하는 인스턴스의 실제 타입을 체크하는데 사용.
- 이항연산자이며 피연산자는 참조형 변수와 타입. 연산결과는 true, false.
- instanceof의 연산결과가 true이면, 해당 타입으로 형변환이 가능하다.



다형성(polyymorphism)

```
class InstanceofTest {
        public static void main(String args[]) {
                FireEngine fe = new FireEngine();

                if(fe instanceof FireEngine) {
                        System.out.println("This is a FireEngine instance.");
                }

                if(fe instanceof Car) {
                        System.out.println("This is a Car instance.");
                }

                if(fe instanceof Object) {
                        System.out.println("This is an Object instance.");
                }
        }
} // class

class Car {}
class FireEngine extends Car {}
```

매개변수의 다형성

- 참조형 매개변수는 메서드 호출시, 자신과 같은 타입 또는 자손타입의 인스턴스를 넘겨줄 수 있다.

```
class Product {
    int price;       // 제품가격
    int bonusPoint; // 보너스점수
}

class Tv extends Product {}
class Computer extends Product {}
class Audio extends Product {}

class Buyer { // 물건사는 사람
    int money = 1000;    // 소유금액
    int bonusPoint = 0; // 보너스점수
}
```

```
Buyer b = new Buyer();

Tv tv = new Tv();
Computer com = new Computer();

b.buy(tv);
b.buy(com);
```

```
Product p1 = new Tv();
Product p2 = new Computer();
Product p3 = new Audio();
```

```
void buy(Tv t) {
    money -= t.price;
    bonusPoint += t.bonusPoint;
}
```

```
void buy(Product p) {
    money -= p.price;
    bonusPoint += p.bonusPoint;
}
```

다형성(polyymorphism)

```
class Product
{
	int price;				// 제품의 가격
	int bonusPoint;			// 제품구매 시 제공하는 보너스점수

	Product(int price) {
		this.price = price;
		bonusPoint =(int)(price/10.0); // 보너스점수는 제품가격의 10%
	}
}

class Tv extends Product {
	Tv() {
		// 조상클래스의 생성자 Product(int price)를 호출한다.
		super(100);				// Tv의 가격을 100만원으로 한다.
	}

	public String toString() {	// Object클래스의 toString()을 오버라이딩한다.
		return "Tv";
	}
}
```

다형성(polyymorphism)

```
class Computer extends Product {
	Computer() {
		super(200);
	}

	public String toString() {
		return "Computer";
	}
}
class Buyer {				// 고객, 물건을 사는 사람
	int money = 1000;		// 소유금액
	int bonusPoint = 0;// 보너스점수

	void buy(Product p) {
		if(money < p.price) {
			System.out.println("잔액이 부족하여 물건을 살수 없습니다.");
			return;
		}

		money -= p.price;		// 가진 돈에서 구입한 제품의 가격을 뺀다.
		bonusPoint += p.bonusPoint;// 제품의 보너스 점수를 추가한다.
		System.out.println(p + "을/를 구입하셨습니다.");
	}
}
```

다형성(polyymorphism)

```
class PolyArgumentTest {
	public static void main(String args[]) {
		Buyer b = new Buyer();
		Tv tv = new Tv();
		Computer com = new Computer();

		b.buy(tv);
		b.buy(com);

		System.out.println("현재 남은 돈은 " + b.money + "만원입니다.");
		System.out.println("현재 보너스점수는 " + b.bonusPoint + "점입니다.");
	}
}
```

다형성(polyymorphism)

여러 종류의 객체를 하나의 배열로 다루기 ★★★★★

- 조상타입의 배열에 자손들의 객체를 담을 수 있다.

```
Product p1 = new Tv();
Product p2 = new Computer();
Product p3 = new Audio();
```

```
Product p[] = new Product[3];
p[0] = new Tv();
p[1] = new Computer();
p[2] = new Audio();
```

```
class Buyer { // 물건사는 사람
    int money = 1000;   // 소유금액
    int bonusPoint = 0; // 보너스점수

    Product[] cart = new Product[10]; // 구입한 물건을 담을 배열

    int i=0;

    void buy(Product p) {
        if(money < p.price) {
            System.out.println("잔액부족");
            return;
        }

        money -= p.price;
        bonusPoint += p.bonusPoint;
        cart[i++] = p;
    }
}
```

다형성(polyymorphism)

```
class Product {
        int price;                      // 제품의 가격
        int bonusPoint;                 // 제품구매 시 제공하는 보너스점수

        Product(int price) {
                this.price = price;
                bonusPoint =(int)(price/10.0);
        }

        Product() {
                price = 0;
                bonusPoint = 0;
        }
}

class Tv extends Product {
        Tv() {
                super(100);
        }

        public String toString() {
                return "Tv";
        }
}
```

다형성(polyymorphism)

```
class Computer extends Product {
        Computer() {        super(200);         }
        public String toString() {      return "Computer";          }
}
class Audio extends Product {
        Audio() {           super(50);          }
        public String toString() {      return "Audio";     }
}
class Buyer {                               // 고객, 물건을 사는 사람
        int money = 1000;  // 소유금액
        int bonusPoint = 0;// 보너스점수
        Product[] item = new Product[10];       // 구입한 제품을 저장하기 위한 배열
        int i =0;                           // Product배열에 사용될 카운터
        void buy(Product p) {
                if(money < p.price) {
                        System.out.println("잔액이 부족하여 물건을 살수 없습니다.");
                        return;
                }

                money -= p.price;           // 가진 돈에서 구입한 제품의 가격을 뺀다.
                bonusPoint += p.bonusPoint;// 제품의 보너스 점수를 추가한다.
                item[i++] = p;                          // 제품을 Product[] item에 저장한다.
                System.out.println(p + "을/를 구입하셨습니다.");
        }
```

다형성(polyymorphism)

```
    void summary() {                    // 구매한 물품에 대한 정보를 요약해서 보여 준다.
        int sum = 0;                                // 구입한 물품의 가격합계
        String itemList =""; // 구입한 물품목록

        // 반복문을 이용해서 구입한 물품의 총 가격과 목록을 만든다.
        for(int i=0; i<item.length;i++) {
            if(item[i]==null) break;
            sum += item[i].price;
            itemList += item[i] + ", ";
        }

        System.out.println("구입하신 물품의 총금액은 " + sum + "만원입니다.");
        System.out.println("구입하신 제품은 " + itemList + "입니다.");
    }
}
```

다형성(polyymorphism)

```
class PolyArgumentTest2 {
        public static void main(String args[]) {
                Buyer b = new Buyer();
                Tv tv = new Tv();
                Computer com = new Computer();
                Audio audio = new Audio();
                b.buy(tv);
                b.buy(com);
                b.buy(audio);
                b.summary();
        }
}
```

instanceof 연산자

형 변환이 가능한지를 묻는 연산자이다.

형 변환이 가능하면 true를 가능하지 않으면 false를 반환.

```
class Box
{
        public void simpleWrap() { . . . . }
}

class PaperBox extends Box
{
        public void paperWrap() { . . . . }
}

class GoldPaperBox extends PaperBox
{
        public void goldWrap() { . . . . }
}
```

instanceof 연산자

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

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

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

instanceof 연산자

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

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

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

instanceof 연산자

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

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

예제

문제2.

예제 InstanceOf.java를 instanceof 연산자를 사용하지 않는 형태로 변경하고자 한다. 즉 클래스의 상속관계를 그대로 유지하면서, instanceof 연산자를 사용하지 않고도 동일한 실행결과를 보일 수 있어야 한다.변경되어야 할 wrapBox 메소드를 아래에 제시.

```
Public static void wrapBox(Boxbox)
{
        box.wrap();
}
```

HINT. 메소드 오버라이딩

모든 클래스가 상속하는 Object 클래스

모든 클래스는 Object 클래스를 상속한다.

```
class MyClass { . . . }
```

```
class MyClass extends
Object{ . . . }
```

```
Object obj1 = new MyClass();
Object obj2 = new int[5];
```

자바 클래스가 아무것도 상속하지 않으면 java.lang 패키지의 Object 클래스를 자동으로 상속한다.

때문에 모든 자바 클래스는 Object 클래스를 직접적 혹은 간접적으로 상속한다.

모든 클래스가 Object 클래스를 직접 혹은 간접적으로 상속하므로, 다음 두 가지가 가능하다.

* **자바의 모든 인스턴스는 Object 클래스의 참조변수로 참조가능**
* **자바의 모든 인스턴스를 대상으로 Object 클래스에 정의된 메소드 호출 가능**

모든 클래스는 Object 클래스를 상속한다.

Object클래스 – 모든 클래스의 최고조상

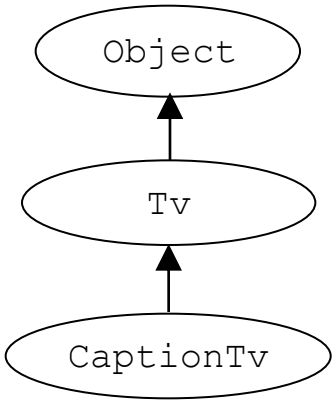
- 조상이 없는 클래스는 자동적으로 Object클래스를 상속받게 된다.
- 상속계층도의 최상위에는 Object클래스가 위치한다.
- 모든 클래스는 Object클래스에 정의된 11개의 메서드를 상속받는다.

  toString(), equals(Object obj), hashCode(), ...

```
class Tv {
    // ...
}

class CaptionTv extends Tv {
    // ...
}
```



```
class Tv extends Object {
    // ...
}

class CaptionTv extends Tv {
    // ...
}
```



String 클래스와 Object 클래스

모든 클래스가 Object 클래스를 상속하는 것과 관련해서 기억할 것

- Object 클래스에는 **toString 메소드**가 다음의 형태로 정의되어 있다.
  public String toString( ) { . . . }
- 그리고 우리가 흔히 호출하는 println 메소드는 다음과 같이 정의되어 있다.
  public void println(Object x) { . . . }
- 때문에 모든 인스턴스는 println 메소드의 인자로 전달될 수 있다.
- 인자로 전달되면, toString 메소드가 호출되고, 이때 반환되는 문자열이 출력된다.
- 때문에 toString 메소드는 적절한 문자열 정보를 반환 하도록 오버라이딩 하는 것이 좋다!

String 클래스와 Object 클래스

```
class Friend
{
	String myName;
	public Friend(String name)
	{
		myName=name;
	}

	public String toString()
	{
		return "제 이름은 "+myName+"입니다.";
	}
}

public static void main(String[] args)
{
	Friend fnd1=new Friend("이종수");
	Friend fnd2=new Friend("현주은");

	System.out.println(fnd1);
	System.out.println(fnd2);
}
```

클래스의 상속
: 상속의 목적

개인정보 관리 프로그램

개인 정보화 프로그램

```
import java.util.Scanner;

class Friend{
	String name;
	String phoneNum;
	String addr;

	public Friend(String name, String phone, String addr){
		this.name=name;
		this.phoneNum=phone;
		this.addr=addr;
	}
	public void showData(){
		System.out.println("이름 : "+name);
		System.out.println("전화 : "+phoneNum);
		System.out.println("주소 : "+addr);
	}
	public void showBasicInfo(){}
}
```

개인 정보화 프로그램

```
class HighFriend extends Friend                    // 고교동창
{
        String work;

        public HighFriend(String name, String phone, String addr, String job)
        {
                super(name, phone, addr);
                work=job;
        }
        public void showData()
        {
                super.showData();
                System.out.println("직업 : "+work);
        }
        public void showBasicInfo()
        {
                System.out.println("이름 : "+name);
                System.out.println("전화 : "+phoneNum);
        }
}
```

개인 정보화 프로그램

```
class UnivFriend extends Friend      // 대학동기
{
        String major;                // 전공학과
        public UnivFriend(String name, String phone, String addr, String major)
        {
                super(name, phone, addr);
                this.major=major;
        }
        public void showData()
        {
                super.showData();
                System.out.println("전공 : "+major);
        }
        public void showBasicInfo()
        {
                System.out.println("이름 : "+name);
                System.out.println("전화 : "+phoneNum);
                System.out.println("전공 : "+major);
        }
}
```

개인 정보화 프로그램

```
class FriendInfoHandler
{
	private Friend[] myFriends;
	private int numOfFriends;
	public FriendInfoHandler(int num)  {
		myFriends=new Friend[num];
		numOfFriends=0;
	}
	private void addFriendInfo(Friend fren){
		myFriends[numOfFriends++]=fren;
	}
	public void addFriend(int choice){
		String name, phoneNum, addr, job, major;
		Scanner sc=new Scanner(System.in);
		System.out.print("이름 : "); name=sc.nextLine();
		System.out.print("전화 : "); phoneNum=sc.nextLine();
		System.out.print("주소 : "); addr=sc.nextLine();
		if(choice==1){
			System.out.print("직업 : "); job=sc.nextLine();
			addFriendInfo(new HighFriend(name, phoneNum, addr,
job));
		}
```

```
                                        else        // if(choice==2)
                {
                        System.out.print("학과 : "); major=sc.nextLine();
                        addFriendInfo(new UnivFriend(name, phoneNum, addr,
major));
                }
                System.out.println("입력 완료! ₩n");
        }

        public void showAllData(){
                for(int i=0; i<numOfFriends; i++){
                        myFriends[i].showData();
                        System.out.println("");
                }
        }
        public void showAllSimpleData(){
                for(int i=0; i<numOfFriends; i++){
                        myFriends[i].showBasicInfo();
                        System.out.println("");
                }
        }
}
```

개인 정보화 프로그램

```
class MyFriendInfoBook
{
	public static void main(String[] args)
	{
		FriendInfoHandler handler=new FriendInfoHandler(10);

		while(true)
		{
			System.out.println("*** 메뉴 선택 ***");
			System.out.println("1. 고교 정보 저장");
			System.out.println("2. 대학 친구 저장");
			System.out.println("3. 전체 정보 출력");
			System.out.println("4. 기본 정보 출력");

			System.out.println("5. 프로그램 종료");
			System.out.print("선택>> ");

			Scanner sc=new Scanner(System.in);
			int choice=sc.nextInt();
```

개인 정보화 프로그램

```
                switch(choice)
                {
                case 1: case 2:
                        handler.addFriend(choice);
                        break;
                case 3:
                        handler.showAllData();
                        break;
                case 4:
                        handler.showAllSimpleData();
                        break;
                case 5:
                        System.out.println("프로그램을 종료합니다.");
                        return;
                }
            }
        }
}
```

다음 클래스 정의에서 상속의 이유를 찾아보자.

Object Oriented Programming

각 객체 생성의 목적과

각 개체의 역할을

잘 생각해 봅시다.

Friend 클래스는 인스턴스화 되지 않는다.

다만 HighFriend 클래스와 UnivFriend 클래스의 상위 클래스로만 의미를 지닌다.

Friend 클래스의 showBasicInfo 메소드를 하위클래스에서 각각 오버라이딩 하고 있다.

Friend 클래스의 showBasicInfo 메소드는 비어있다!

FriendInfoHandler 클래스의 관찰

Friend 클래스를 상속했기 때문에 Friend의 하위클래스의 인스턴스가 저장 가능하다!

```
class FriendInfoHandler
{
        private Friend[] myFriends;
        private int numOfFriends;
        public FriendInfoHandler(int num){
                myFriends=new Friend[num];
                numOfFriends=0;
        }
        private void addFriendInfo(Friend fren){
                myFriends[numOfFriends++]=fren;
        }
        ……
        public void showAllSimpleData(){
                for(int i=0; i<numOfFriends; i++)
                {
                        myFriends[i].showBasicInfo();
                        System.out.println("");
                }
        }
}
```

FriendInfoHandler 클래스의 관찰

Friend 클래스를 상속했기 때문에 Friend의 하위클래스의 인스턴스가 저장 가능하다!

showBasicInfo 메소드를 오버라이딩 했기때문에 Friend 클래스의 참조변수를 통해서도 하위클래스의 showBasicInfo 메소드를 호출할 수 있다! 이것이 바로showBasicInfo 메소드를 오버라이딩의 관계에둔 이유이다!

FriendInfoHandler 클래스는 Friend의 하위클래스의 인스턴스를 저장 및 관리한다.
FriendInfoHander 클래스 입장에서는 HighFriend 클래스의 인스턴스도, UnivFriend 클래스의 인스턴스도 모두 Friend 클래스의 인스턴스로 간주한다.

showBasicInfo 메소드의 오버라이딩 이유?

showBasicInfo 메소드를 오버라이딩 관계에 두지 않는다면 FriendInfoHandler 클래스의 showAllSimpleData 메소드는 다음과 같이 변경 되어야한다.

```
public void showAllSimpleData()
 {
      for(int i=0; i<numOfFriends; i++)
      {
         myFriends[i].showBasicInfo();
         System.out.println("");
      }
 }
```

```
public void showAllSimpleData()
 {
      if(myFriends[i] instanceof HighFriend)
         ((HighFriend)myFriends[i]).showBaicInfor();
         else
         ((UnivFriend)myFriends[i]).showBaicInfor();

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

그러나 이것이 전부가 아니다. Friend 클래스를 상속하는 하위클래스가 하나 더 등장할 때마다 위의 메소드는 엄청나게 복잡해진다. 특히 UnivFriend 클래스와 HighFriend 클래스를 상속의 관계로 묶지않았다면, FriendInfoHandler 클래스는 지금보다 훨씬 더 복잡해져야 하며, Friend 클래스를 상속하는 하위클래스의 수가 증가 할 때마다 엄청난 코드의 확장이 필요해진다

상속과 오버라이딩이 가져다 주는 이점

"상속을 통해 연관된 일련의 클래스에 대한 공통적인 규약을 정의할 수 있습니다."

FriendInfoHandler 클래스는, 상속을 통해 연관된 HighFriend, UnivFriend 클래스에 대해(일련의 클래스에 대해) 동일한 방식으로 배열에 저장 및 메소드 호출을 할 수(공통적인 규약을 정의할 수) 있습니다.

정리합시다

- 상속의 목적
- 상속의 구조
- 상속관계에서의 인스턴스 생성 : 상위 클래스의 생성자 호출
- 다형성
- 다형성을 이용한 배열 처리

Project

Project : ver 0.40

다음 두 클래스를 추가로 삽입. 상속 구조가 가능하다면 상속 구조로 구성 해보세요.
PhoneUnivInfor, PhoneCompaanyInfor, 개인적인 클래스 추가

각 클래스에 정의되어야 하는 인스턴스 변수.

PhoneUnivInfor		
이름	name	String
전화번호	phoneNumber	String
주소	address	String
이메일	email	String
전공	major	String
학년	year	String

PhoneCompaanyInfor		
이름	name	String
전화번호	phoneNumber	String
주소	address	String
이메일	email	String
회사	company	String

Ex) PhoneCafeInfor 또는 PhoneFamilyInfor 등을 추가해 보자