Inheritance(@Override) Ⅱ - Method Overriding

하위 Class에서 Method를 다시 정의한다면? - Method Overriding

- 상위 Class에 정의된 Method의 이름, 반환형, 매개변수 선언까지 완전히 동일한 메소드를 하위 Class에서 다시 정의하는 것

```
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;

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

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

Overriding의 기본원칙 - Method Overriding에서의 super의 의미

- Overriding 대상인 상위 클래스의 Method의 호출을 명령하는 키워드
  (Overriding된 Method에 상위 클래스 Method의 문장들이 담겨진다라고 볼 수 있다.)
- 상위 Class의 메소드에 중요한 정보가 삽입되어 있다면, Overriding을 통해 다시 정의할 때,
  중요한 코드의 실행을 못할 수가 있다.

- Java가 기본제공하는 Class들을 상속받아서 Method Overriding을 통해 사용하는 경우,
  그 메소드에 대한 정보가 부족한 경우, super 키워드를 이용한 상위 클래스의 메소드 호출을 해야한다!
- 또한 상위 클래스 메소드의 기능은 간직한채 작업을 추가하고 싶을 때 유용하다!

```
class AAA {
	public void myMethod(){
		. . . .
		. . . .
	}
}

Class BBB extends AAA {
	@Override
	public void myMethod(){
		super.myMethod();
	}
}
```

상위 Class의 참조변수로 하위 Class의 Instance 참조

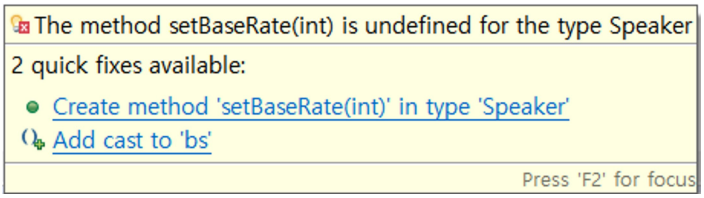
중저음스피커는 스피커의 일종일까? Ok!
스피커는 중저음스피커의 일종일까? No! (이 경우가 성립하려면 모든 스피커는 중저음스피커가 되야한다.

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

    Speaker bs = new BaseEnSpeaker();
    bs.setVolume(10);
    bs.setBaseRate(20); // Compile ERROR!
    bs.showCurrentState();

}
```

다형성

- **`Speaker bs = new BaseEnSpeaker();`**
  Java Compiler는 상속받은 하위 Instance를 상위 Instance로 인정해준다!

- **`bs.setBaseRate(20); // Compile ERROR!`**
  BaseEnSpeaker의 Instance를 참조시켰고, BaseEnSpeaker의 Method를 호출하려는데 ERROR라니?,
  ERROR의 내용은 다음과 같다.



'Speaker Instance에는 setBaseRate()가 존재하지 않는다."

컴파일러가 바보같다. 필자는 분명히 BaseEnSpeaker의 Instance를 참조시켰는데..
Speaker Class가 래핑되어서 Instance를 필터링 하게 된다. Speaker Class에 Member에 없는 것을 걸러낸다.

자식 객체의 멤버들을 사용하고 싶다면?

a. 상위클래스에 setBaseRate를 정의하고 Overriding을 해야한다.
b. 또는 Bs를 (BaseEnSpeaker)로 Casting 하면 된다. **(자식객체로 Casting)**

Overriding 관계에서의 Method 호출

```
class AAA {
    public AAA(){}
    public void rideMethod(){ System.out.println("AAA's Method"); }
    public void loadMethod(){ System.out.println("void Method"); }
}

class BBB extends AAA {
    public BBB(){super();}

    @Override
    public void rideMethod(){ System.out.println("BBB's Method");
    public void loadMethod(int num){ System.out.println("int Method"); }
}

class CCC extends BBB {
```

```
    public CCC(){super();}

    @Override
    public void rideMethod(){ System.out.println("CCC's Method"); }
    public void loadMethod(double num){ System.out.println("double Method"); }
}

class ExtendsMain {
    public static void main(String[] args){
        AAA ref1 = new CCC();
        BBB ref2 = new CCC();
        CCC ref3 = new CCC();

        AAA ref4 = new AAA();
        BBB ref5 = new BBB();
        CCC ref6 = new CCC();

        하위 Class에 정의된 Method로 인해 상위 Class의 Method는 가리워진다.
        외부에서는 어떠한 방법을 쓰더라도 상위 Method에 접근불가!
        ref1.rideMethod();
        ref2.rideMethod();
        ref3.rideMethod();

        // ref4.rideMethod(); → AAA's Method 출력
        // ref5.rideMethod(); → BBB's Method 출력
        // ref6.rideMethod(); → CCC's Method 출력

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

실행결과

```
CCC's Method
CCC's Method
CCC's Method
void Method
int Method
double Method
```