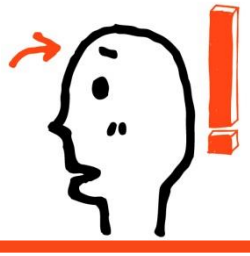
Android Java

8장 접근제어 지시자와 정보은닉

접근제어 지시자와 정보은닉, 그리고 캡슐화



01 정보은닉(Information Hiding)

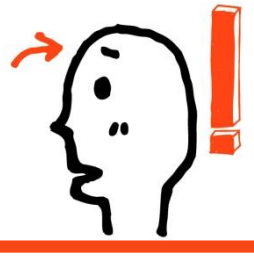
02 접근제어 지시자(Access Control Specifiers)

03 public 클래스와 default 클래스

04 어떤 클래스를 pubilc으로 선언할까요?

05 캡슐화(Encapsulation)

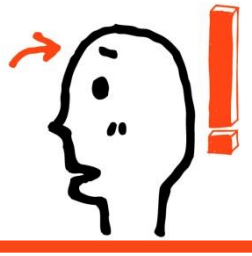
객체지향 관점에서 넌 빵점이야!

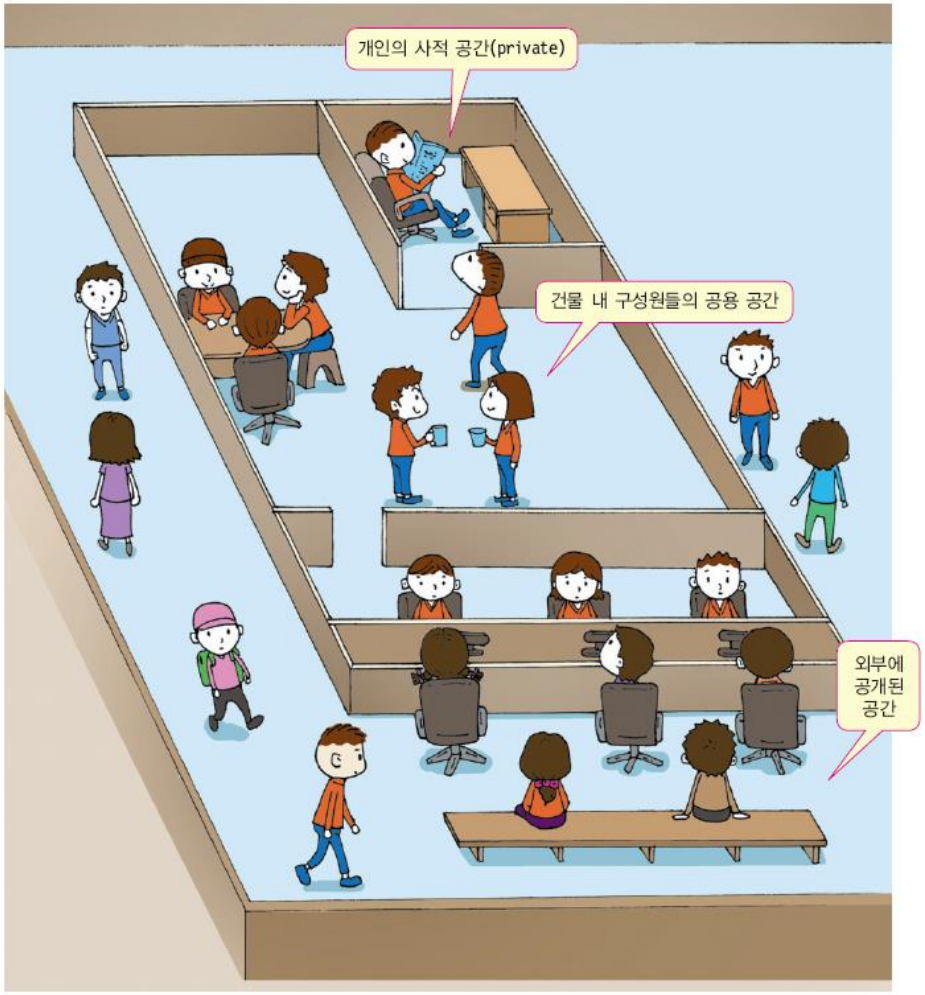


```
class FruitSalesMain4
{
    public static void main(String[] args)
    {
        FruitSeller seller = new FruitSeller(0, 30, 1500);
        FruitBuyer buyer = new FruitBuyer(10000);

        seller.myMoney += 500;
        buyer.myMoney -= 500;

        seller.numOfApple -= 20;
        buyer.numOfApple += 20;

        System.out.println("과일 판매자의 현재 상황");
        seller.showSaleResult();
        System.out.println("과일 구매자의 현재 상황");
        buyer.showBuyResult();
    }
}
```

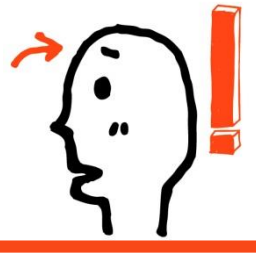
메소드를 통해서 정의한 판매의 규칙이 완전히 무너졌다!

정보은닉이라? 인스턴스 변수의 private





정보은닉이라? 인스턴스 변수의 private



```
class FruitSeller
{
    private int numOfApple;
    private int myMoney;
    private final int APPLE_PRICE;
```

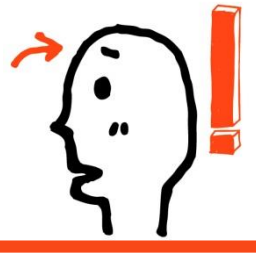
외부에서의 접근 금지!

```
class FruitBuyer
{
    private int myMoney;
    private int numOfApple;
```

외부에서의 접근 금지!

- private과 같은 키워드를 가리켜 접근제어 지시자라 한다.
- 인스턴스 변수의 private 선언으로 인해서 메소드가 유일한 접근수단

private과 public



- private - 클래스 내부(메소드)에서만 접근 가능.
- public - 어디서든 접근 가능(접근을 제한하지 않는다).

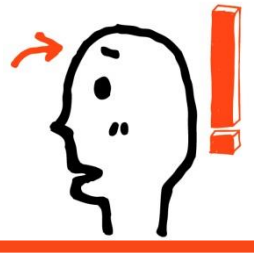
```
class AAA
{
    private int num;
    public void setNum(int n) { num=n; }
    public int getNum() { return num; }
    . . . .
}
```

```
class BBB
{
    public accessAAA(AAA inst)
    {
        inst.num=20;
        inst.setNum(20);
        System.out.println(inst.getNum());
    }
    . . . .
}
```

num은 private 멤버이므로 컴파일 불가!

setNum, getNum은 public이므로 호출 가능!

접근제어 지시자를 선언하지 않는 경우(default)

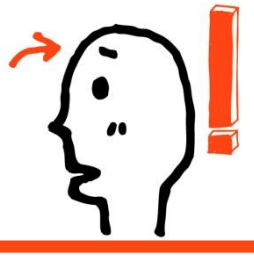


- 접근제어 지시자 선언하지 않는 경우 - default 선언
- default 선언은 동일 패키지 내에서의 접근 허용

```
package orange;

class AAA    // package orange로 묶인다.
{                  default 선언!
     int num;
     . . . .
}

class BBB    // package orange로 묶인다.
{
     public init(AAA a) { a.num=20; }
     . . . .
}
```

BBB는 AAA와 동일 패키지로 선언되었으므로 접근 가능!

protected

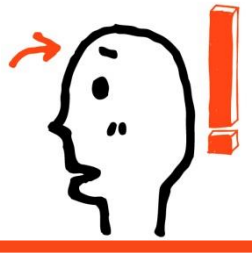


- protected - 상속관계에 놓여 있어도 접근을 허용
- default 선언으로 접근 가능한 영역 접근 가능,
- 그리고 상속관계에서도 접근 가능

```
class AAA
{
    protected int num;
    . . . .
}

class BBB extends AAA
{
    protected int num;        // 상속된 인스턴스 변수

    public init(int n) { num=n; }    // 상속된 변수 num의 접근!
    . . . .
}
```

상속을 의미함.

접근제어 지시자의 관계

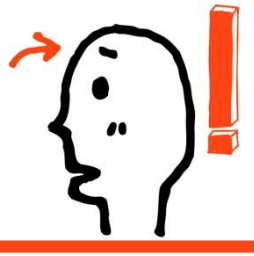


접근 제어자(public, default, protected, private)

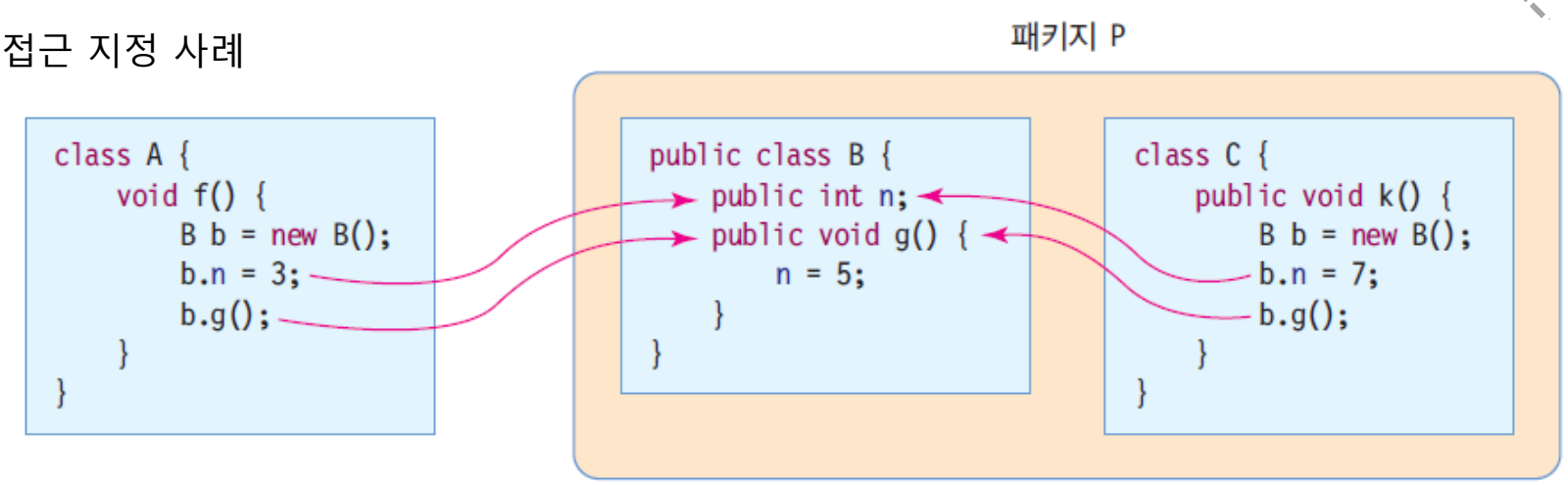
- **접근 제어자 (Access modifier)** : 자원에 대해서 외부에서 접근(사용, 참조 등) 가능한 범위를 제어
- 클래스나 메소드 혹은 변수의 선언부를 정의할 때 가장 처음 등장하는 것이 접근 제어자
- 접근 제어자는 호출하는 클래스의 위치나 관계에 따라서 자원에 대한 접근을 막거나 허락 가능
- 접근 제약이 높은 순으로 **private > default > protected > public**

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

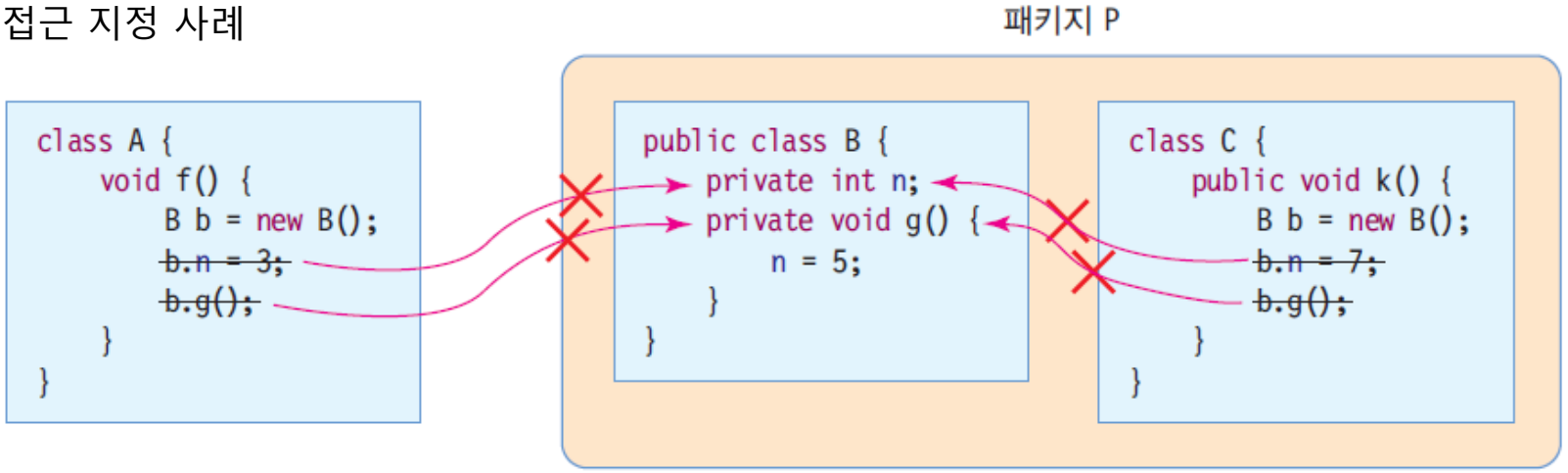
접근제어 지시자의 이해



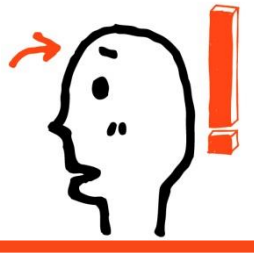
public 접근 지정 사례



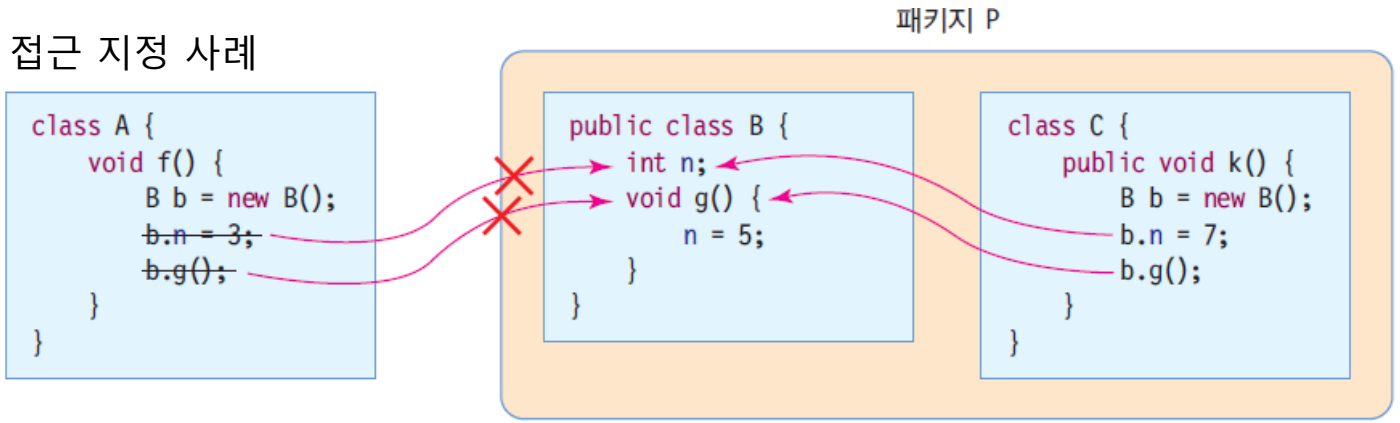
private 접근 지정 사례



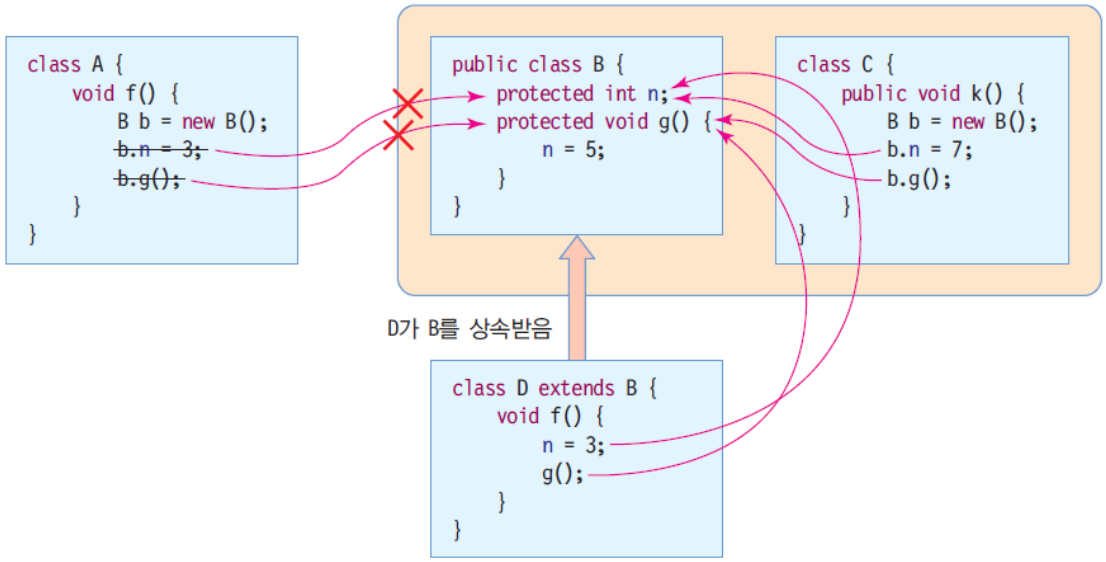
접근제어 지시자의 이해



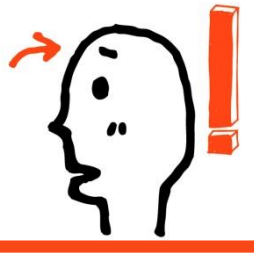
디폴트 접근 지정 사례



protected 접근 지정 사례



default 클래스



- default 클래스
  - 동일한 패키지 내에 정의된 클래스에 의해서만 인스턴스 생성이 가능

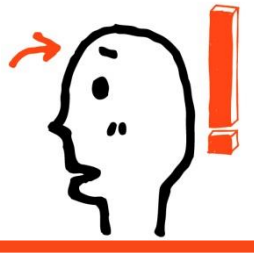
```
package apple;
class AAA    // default 클래스 선언
{
    . . . .
}
```

```
package peal;
class BBB    // default 클래스 선언
{
    public void make()
    {
        apple.AAA inst=new apple.AAA();
        . . . .
    }
    . . . .
}
```

인스턴스 생성 불가!
AAA와 BBB의 패키지가 다르므로!

파일을 대표할 정도로 외부에 의미가 있는 클래스 파일을 public으로 선언한다.

public 클래스



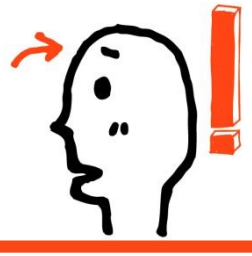
- public 클래스
  - 하나의 소스파일에 하나의 클래스만 public으로 선언 가능
  - public 클래스 이름과 소스파일 이름은 일치해야 한다.

```
package apple;
public class AAA     // public 클래스 선언
{
    . . . .
}
```

```
package peal;
public class BBB     // public 클래스 선언
{
    public void make()
    {
        apple.AAA inst=new apple.AAA();
        . . . .
    }
    . . . .
}
```

AAA는 public 클래스이므로
어디서든 인스턴스 생성 가능!

생성자는 public인데, 클래스는 default?



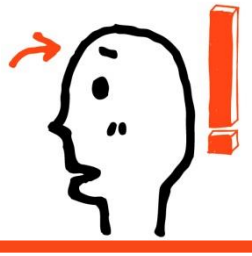
```
public class AAA
{
    AAA(){…}
    . . . .
}
```

클래스는 public으로 선언되어서 파일을 대표하는 상황! 그럼에도 불구하고 생성자가 default로 선언되어서 동일 패키지 내에서만 인스턴스 생성을 허용하는 상황!

```
class BBB
{
    public BBB(){…}
    . . . .
}
```

생성자가 public임에도 클래스가 default로 선언되어서 동일 패키니 내에서만 인스턴스 생성이 허용되는 상황!

디폴트 생성자



디폴트 생성자의 접근제어 지시자는 클래스의 선언형태에 따라서 결정된다.

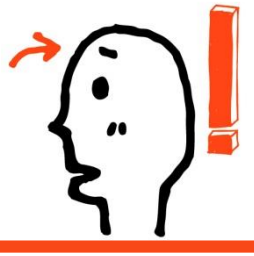
```
public class AAA
{
    public AAA() {…}
    . . . .
}
```

public 클래스에 디폴트로 삽입되는 생성자

```
class BBB
{
    BBB() {…}
    . . . .
}
```

default 클래스에 디폴트로 삽입되는 생성자

제어자로 특별한 의미 부여하기



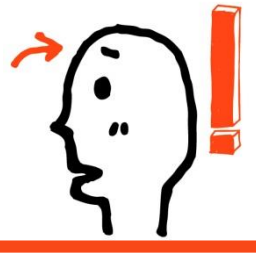
접근 제어자(public, default, protected, private)

코드 6-11 package com.gilbut.java;

```
1   public class ModifierExample
2   {
3       protected int uniqueClassId;
4
5       public ModifierExample()
6       {
7           uniqueClassId = System.identityHashCode(this);
8       }
9
10      protected void printClassId ()
11      {
12          System.out.println("Hashcode of ModifierExample object : " + this.
            uniqueClassId);
13      }
14  }
```

이 두 개의 자원은 com.gilbut.java에 있는 클래스 혹은 ModifierExample 클래스를 상속받는 자식 클래스만 접근 가능합니다.

제어자로 특별한 의미 부여하기



접근 제어자(public, default, protected, private)

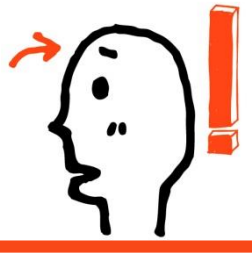
코드 6-12 package com.gilbut.java;

```
1   public class ModifierTester
2   {
3       public static void main(String[] args)
4       {
5           ModifierExample example = new ModifierExample();
6
7           example. printClassId ();
8
9           int hashcode = example.uniqueClassId;
10          System.out.println("Double check. Hashcode : " + hashcode);
11      }
12  }
```

ModifierExample과 같은 com.gilbut.java 패키지에 있으므로 아래의 클래스 변수나 메소드에 접근 가능합니다.

public 클래스와 default 클래스

제어자로 특별한 의미 부여하기



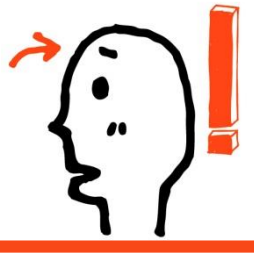
접근 제어자(public, default, protected, private)

실행결과

```
Hashcode of ModifierExample object : 234152390
Double check. Hashcode : 234152390
```

- ModiferTester 클래스는 ModifierExample 클래스와 같은 com.gilbut.java 패키지에 속해 있음
- ModifierTester 클래스는 ModifierExample 클래스의 protected 자원에 대해서 자유로운 접근 가능

제어자로 특별한 의미 부여하기



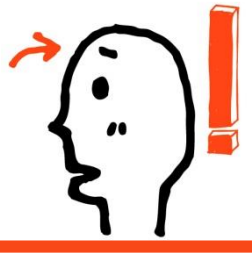
접근 제어자(public, default, protected, private)

코드 6-13 package com.gilbut.chapter6;

```
1   import com.gilbut.java.ModifierExample;
2
3   public class ModifierTester extends ModifierExample
4   {
5      public static void main(String[] args)
6      {
7         ModifierTester tester = new ModifierTester();
8         tester.run();
9      }
10
11     public void run()
12     {
13        this. printClassId ();
14
15        int hashcode = this.uniqueClassId;
16        System.out.println("Double check. Hashcode : " + hashcode);
17     }
18  }
```

같은 패키지가 아닐 경우 상속을 통해서 protected 자원에 접근 가능한 경우입니다. 단, 상속받은 자원이므로 this 키워드를 사용해서 호출해야 한다는 것 잊지 마세요!

제어자로 특별한 의미 부여하기



접근 제어자(public, default, protected, private)

실행결과

```
Hashcode of ModifierExample object : 234152390
Double check. Hashcode : 234152390
```

- ModifierTester 클래스는 com.gilbut.chapter6 패키지에 속해 있기 때문에
  ModifierExample 클래스의 protected 자원에 대한 접근할 수 없음
  하지만 ModifierExample 클래스를 상속받으면 해당 자원에 대해서 접근 가능

- protected 제어자는 상속받은 클래스만이 호출 가능하므로
  extends 키워드로 ModifierExample 클래스를 상속을 받은 다음에
  this 키워드를 사용해서 호출