

Methods Use Instance Variables: How Objects Behave

Samkeun Kim <skim@hknu.ac.kr>

<http://cyber.hankyong.ac.kr>

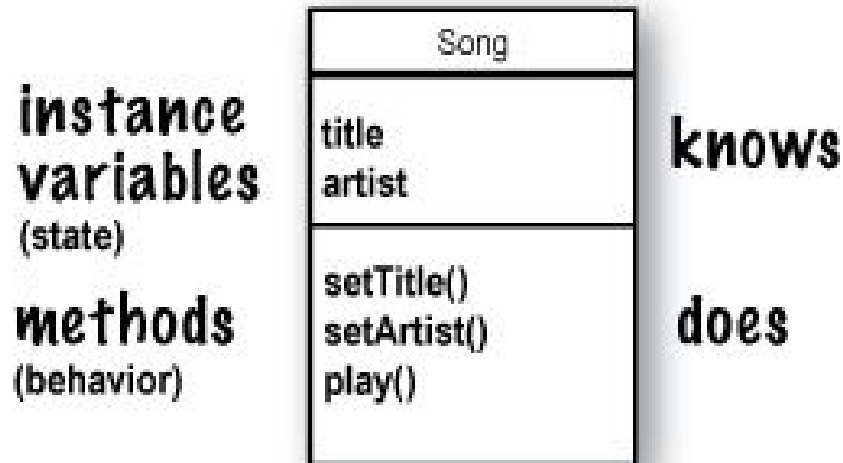
클래스에서는 객체가 아는 것과 수행하는 것을 기술한다

클래스는 객체에 대한 청사진이다.

클래스를 만든다는 것은 "JVM이 그 유형의 객체를 어떻게 만들어야 하는지에 대해 기술해 놓은 것"이라고 말할 수 있다.

그 유형의 모든 객체가 서로 다른 인스턴스 변수 값들을 가질 수 있다는 사실은 이미 알고 있다.

그러나 메소드의 경우는 어떨까?



클래스에서는 객체가 아는 것과 수행하는 것을 기술한다

같은 유형에 속하는 모든 객체들이 서로 다른 행동을 하는 메소드를 가질 수 있을까?

음... 조금은 그렇다고 볼 수 있다.

특정 클래스의 모든 인스턴스가 같은 메소드를 갖지만 그 메소드들은 **인스턴스 변수의 값에 따라** 다르게 동작할 수 있다.

Song이라는 클래스에 **title**과 **artist**라는 두 개의 인스턴스 변수가 있다 하자.

play() 메소드는 곡을 재생한다.

```
void play() {  
    soundPlayer.playSound(title);  
}
```

여기서 **play()**를 호출하는 인스턴스는 그 인스턴스에 대한 **title** 인스턴스 변수의 값이 가리키는 노래를 재생시킬 것이다.

instance variables
(state)

methods
(behavior)

Song	
title	artist
setTitle() setArtist() play()	

knows

does

```
void play() {  
    soundPlayer.playSound(title);  
}
```

```
Song t2 = new Song();  
t2.setArtist("Travis");  
t2.setTitle("Sing");  
Song s3 = new Song();  
s3.setArtist("Sex Pistols");  
s3.setTitle("My Way");
```

Calling play() on this instance
will cause "Sing" to play.

t2.play() ;

메소드는 동일하다!

Calling play() on this instance
will cause "My Way" to play.
(but not the Sinatra one)

five instances
of class Song



Song



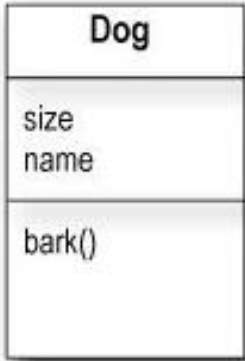
Song

s3.play() ;

실습과제 4-1 The size affects the bark

A small **Dog**'s bark is different from a big **Dog**'s bark. The **Dog** class has an instance variable **size**, that the **bark()** method uses to decide what kind of bark sound to make.

```
class Dog {  
    int size;  
    String name;  
  
    void bark() {  
        if (size > 60) {  
            System.out.println("Woof! Woof!");  
        } else if (size > 14) {  
            System.out.println("Ruff! Ruff!");  
        } else {  
            System.out.println("Yip! Yip!");  
        }  
    }  
}
```



The UML class diagram for the **Dog** class shows a rectangular box with the class name **Dog** in the top section. Below the name, there are two sections for attributes: **size** and **name**. At the bottom of the box is a section for the method **bark()**.

```
class DogTestDrive {  
  
    public static void main (String[] args) {  
        Dog one = new Dog();  
        one.size = 70;  
        Dog two = new Dog();  
        two.size = 8;  
        Dog three = new Dog();  
        three.size = 35;  
  
        one.bark();  
        two.bark();  
        three.bark();  
    }  
}
```



The terminal screenshot shows the execution of the Java program. The command `%java DogTestDrive` is entered at the prompt. The output consists of three lines of bark sounds: `Woof! Woof!`, `Yip! Yip!`, and `Ruff! Ruff!`, corresponding to the three Dog objects created in the main method.

메소드에 뭔가를 보낼 수도 있다

자바에서도 메소드에 어떤 값을 전달할 수 있다.

예를 들어,

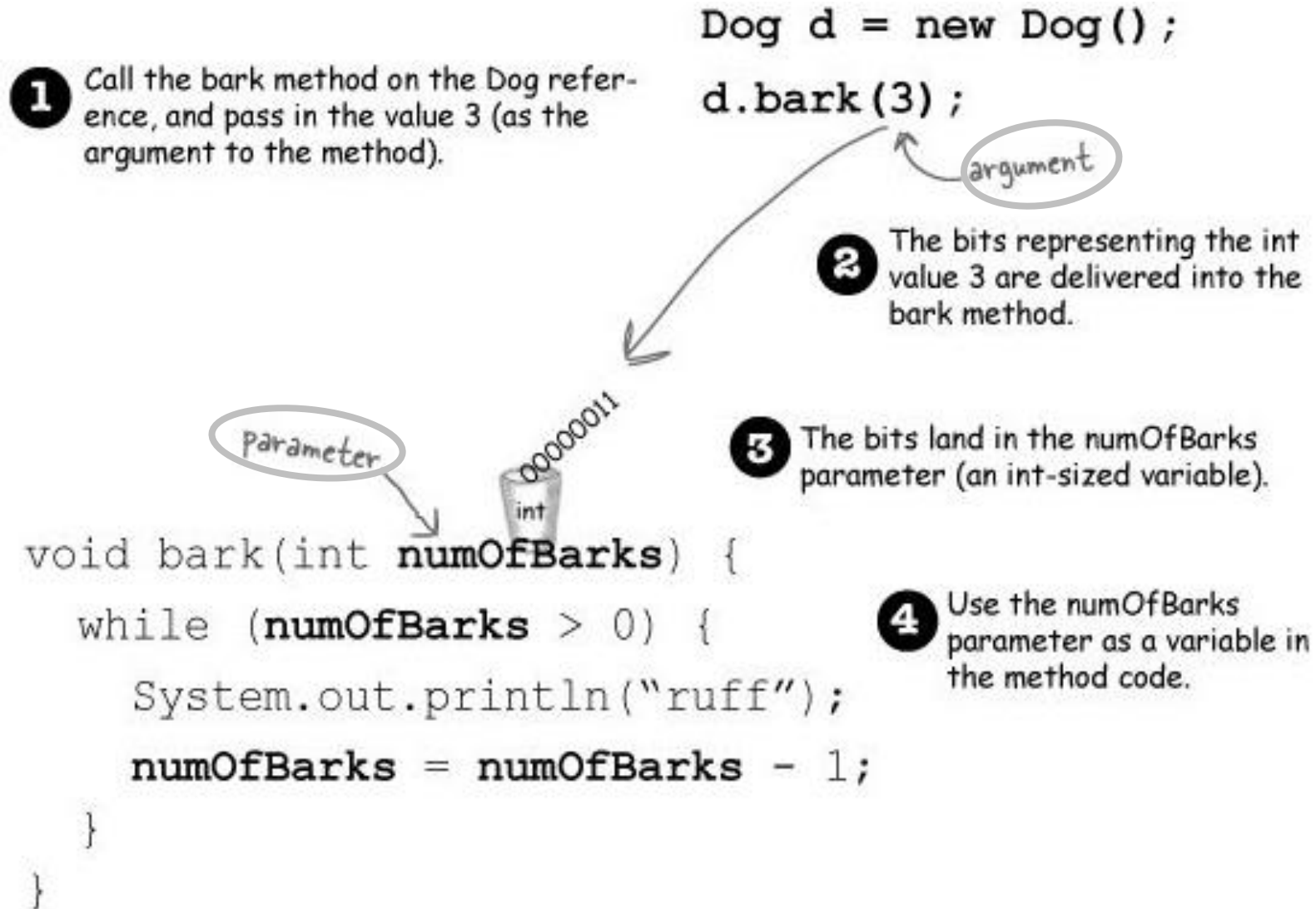
d.bark(3);

와 같은 식으로 **bark**하는 횟수를 **Dog** 객체에 지정해 줄 수 있다.

메소드 정의 => 파라미터(parameter)를 통하여 인자를 받아 들인다:

메소드 호출 => 아규먼트(argument)를 메소드에 전달

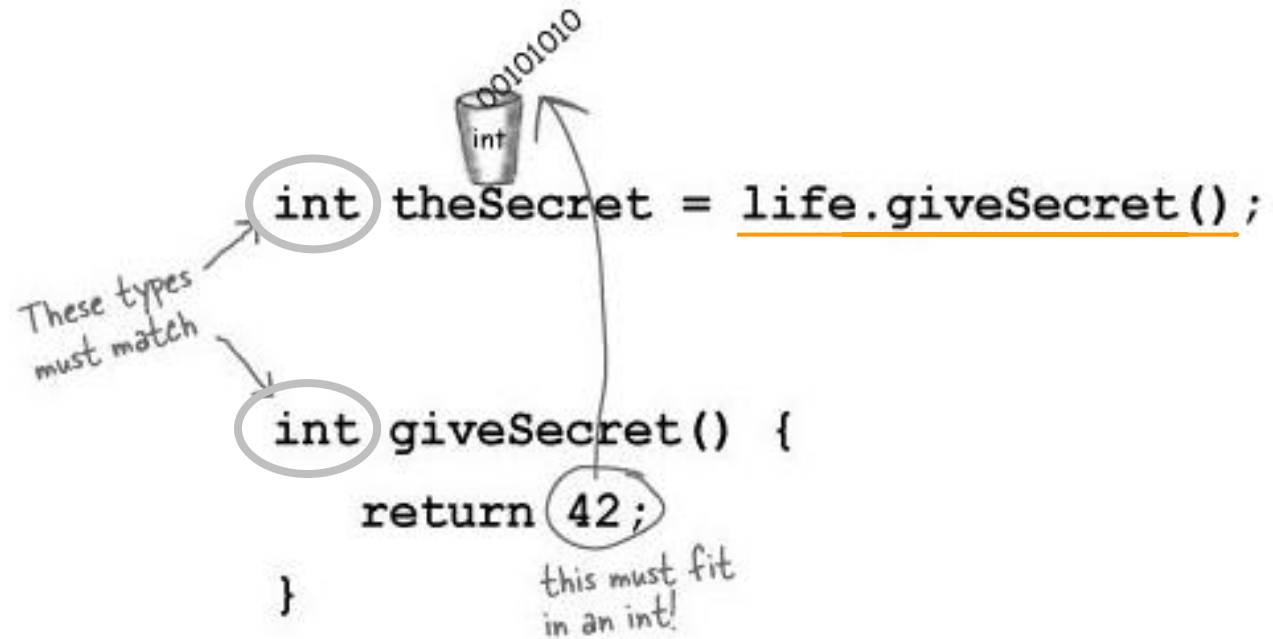
메소드가 파라미터를 가지고 있다면 반드시 뭔가를 거기에 전달해야만 한다!



메소드에서 뭔가 돌려받을 수도 있다



The compiler won't let you return the wrong type of thing.



42를 나타내는 비트 값이 **giveSecret()** 메소드로부터 리턴되며, **theSecret**라는 변수에 들어간다.

메소드에 두 개 이상의 인자를 전달할 수도 있다

Calling a **two-parameter** method, and sending it **two arguments**.

```
void go() {  
    TestStuff t = new TestStuff();  
    t.takeTwo(12, 34);  
}
```

```
void takeTwo(int x, int y) {  
    int z = x + y;  
    System.out.println("Total is " + z);  
}
```

The arguments you pass land in the same order you passed them. First argument lands in the first parameter, second argument in the second parameter, and so on.

메소드에 두 개 이상의 인자를 전달할 수도 있다

변수 타입이 파라미터 타입과 일치하는 한, 변수를 메소드에 전달할 수 있다.

```
void go() {  
    int foo = 7;  
    int bar = 3;  
    t.takeTwo(foo, bar);  
}  
  
void takeTwo(int x, int y) {  
    int z = x + y;  
    System.out.println("Total is " + z);  
}
```

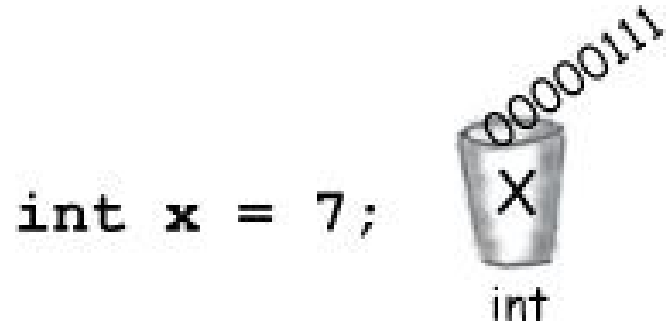
The diagram shows two arrows originating from the arguments 'foo' and 'bar' in the `t.takeTwo(foo, bar);` line of the `go()` method. One arrow points to the parameter 'x' in the `takeTwo(int x, int y)` method signature, and the other points to the parameter 'y'. This illustrates how the values of 'foo' and 'bar' are passed to the parameters 'x' and 'y' respectively.

The values of `foo` and `bar` land in the `x` and `y` parameters. So now the bits in `x` are identical to the bits in `foo` (the bit pattern for the integer '7') and the bits in `y` are identical to the bits in `bar`.

What's the value of `z`? It's the same result you'd get if you added `foo` + `bar` at the time you passed them into the `takeTwo` method

Java is pass-by-value. That means pass-by-copy

1. **int** 변수를 선언하고 그것에 '7'이라는 값을 대입한다. 7에 대한 비트 패턴이 **x** 라는 변수에 들어간다.

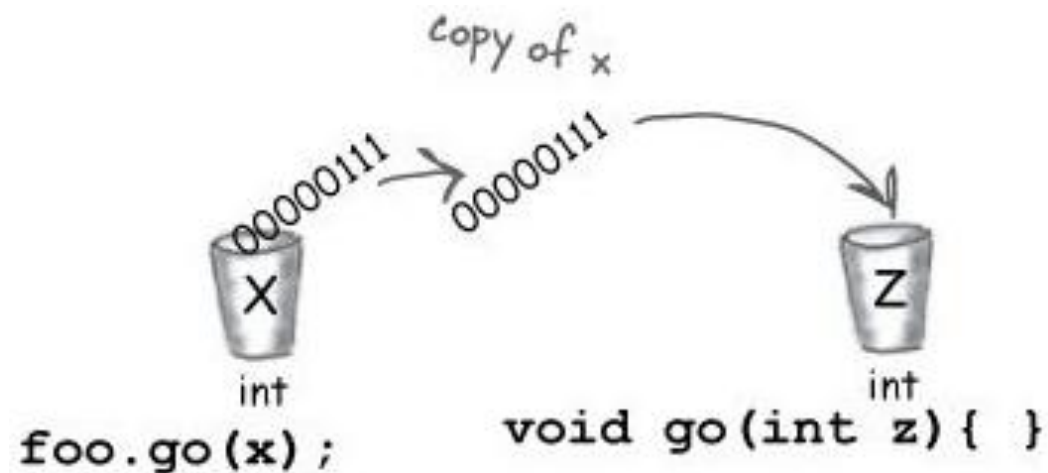


2. **z** 라는 이름의 **int** 파라미터를 가진 메소드를 선언한다.



Java is pass-by-value. That means pass-by-copy

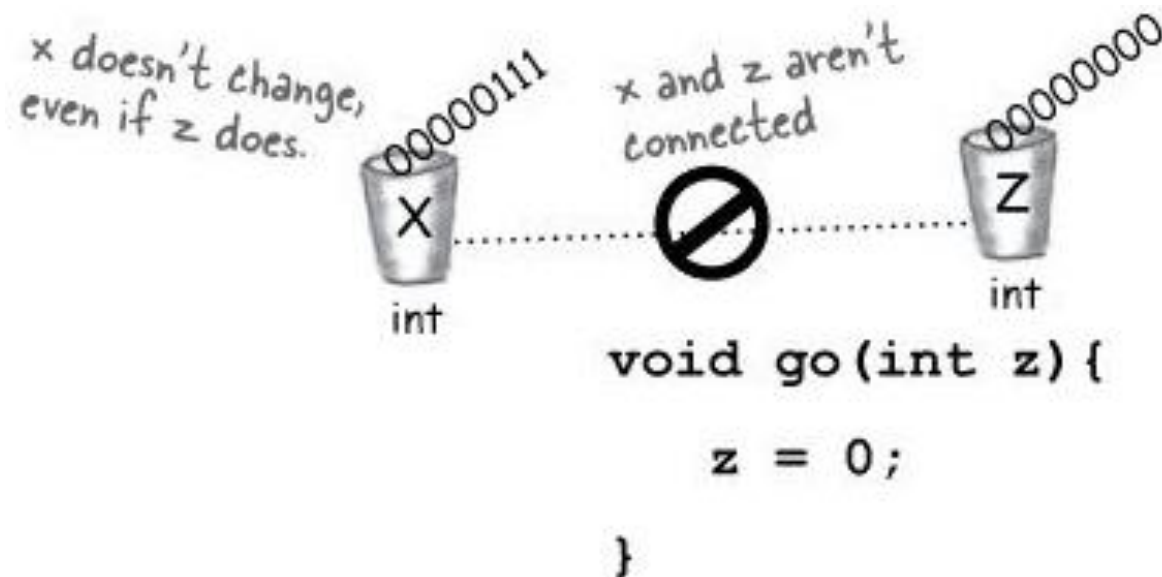
3. **go()** 메소드를 호출하여 변수 **x** 를 아규먼트로 전달한다. **x** 의 비트 패턴이 복사되어 그 복사본이 **z**에 들어가게 된다.



4. 메소드 내에서 **z** 의 값을 변경한다
=> **x** 의 값은 변경되지 않는다.

z 파라미터에 전달된 아규먼트는 **x** 의 복사본이다.

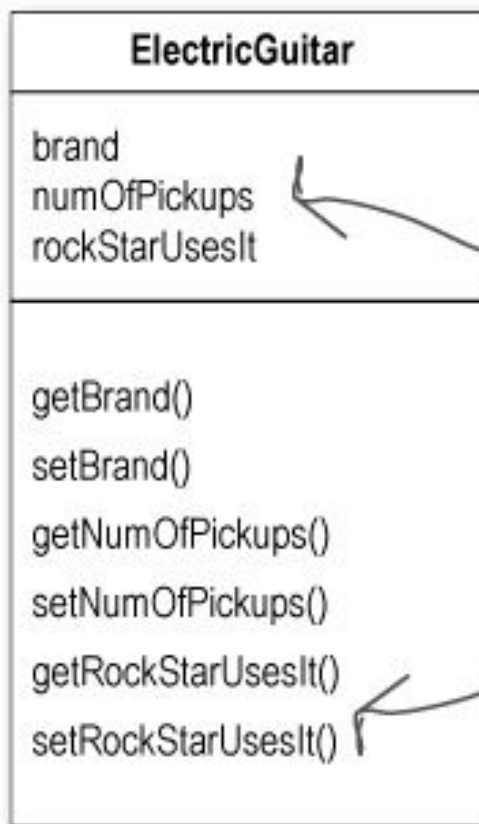
5. 이 메소드는 호출 변수 **x** 의 비트 패턴을 변경시킬 수 없다.



파라미터와 리턴 타입 활용 방법

파라미터와 리턴 타입의 가장 대표적인 예

⇒ 게터(getter) 와 세터(setter)



Note: Using these naming conventions means you'll be following an important Java standard!



```
class ElectricGuitar {
```

```
    String brand;  
    int numOfPickups;  
    boolean rockStarUsesIt;
```

```
    String getBrand() {  
        return brand;  
    }
```

```
    void setBrand(String aBrand) {  
        brand = aBrand;  
    }
```

```
    int getNumOfPickups() {  
        return numOfPickups;  
    }
```

```
    void setNumOfPickups(int num) {  
        numOfPickups = num;  
    }
```

```
    boolean getRockStarUsesIt() {  
        return rockStarUsesIt;  
    }
```

```
    void setRockStarUsesIt(boolean yesOrNo) {  
        rockStarUsesIt = yesOrNo;  
    }
```

```
}
```

캡슐화 (encapsulation)

캡슐화하지 않으면 웃음거리가 될 수 있다.

지금까지 아무나 데이터를 볼 수 있도록, 심지어는 아무나 건드릴 수 있도록 방치시켜 놓았다.

노출시켜 놓았다는 의미는 점 연산자로 접근할 수 있다는 것을 의미:

theCat.height = 27;

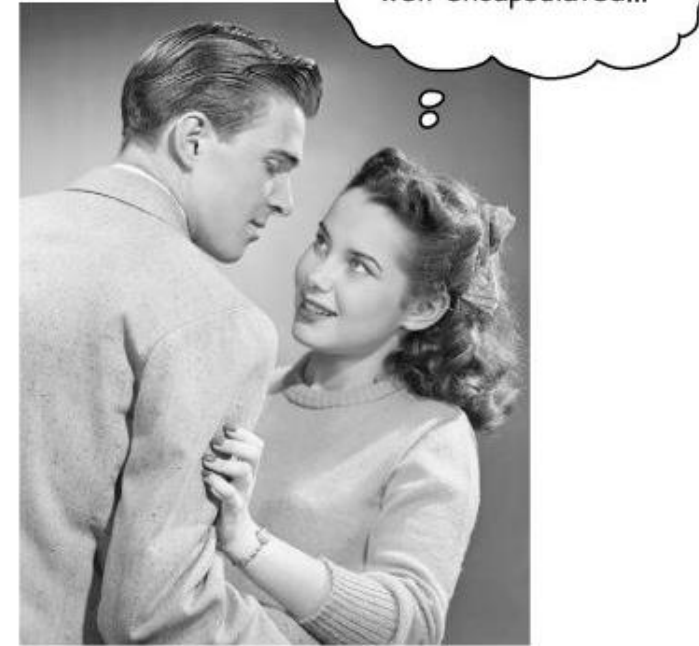
이렇게 리모컨을 이용하여 **theCat** 이라는 객체의 **height** 인스턴스 변수(고양이의 키)를 직접 변경하는 것을 생각해보자.

`theCat.height = 0;` ← yikes! We can't let this happen!

반드시 setter 메소드를 사용해야 한다!

```
public void setHeight(int ht) {  
    if (ht > 9) {  
        height = ht;  
    }  
}
```

← We put in checks to guarantee a minimum cat height.



데이터를 숨겨보자!

정확하게 어떻게 해야 데이터를 숨길 수 있을까?

⇒ **public**과 **private**라는 접근 한정자 사용

public => 모든 **main** 메소드 앞에 붙어 있었다!

캡슐화를 하려면

⇒ 인스턴스 변수를 **private**로 지정하고

⇒ 접근 제어를 위해 **public**으로 지정된 게터와 세터를 만들면 된다

인스턴스 변수는 **private**로!

게터와 세터는 **public**으로!

Encapsulating the GoodDog class

```
class GoodDog {  
    private int size;  
    public int getSize() {  
        return size;  
    }  
    public void setSize(int s) {  
        size = s;  
    }  
  
    void bark() {  
        if (size > 60) {  
            System.out.println("Woof! Woof!");  
        } else if (size > 14) {  
            System.out.println("Ruff! Ruff!");  
        } else {  
            System.out.println("Yip! Yip!");  
        }  
    }  
}
```

Make the instance variable private.

Make the getter and setter methods public.

GoodDog
size
getSize() setSize() bark()


```

class GoodDogTestDrive {

    public static void main (String[] args) {
        GoodDog one = new GoodDog();
        one.setSize(70);
        GoodDog two = new GoodDog();
        two.setSize(8);
        System.out.println("Dog one: " + one.getSize());
        System.out.println("Dog two: " + two.getSize());
        one.bark();
        two.bark();
    }
}

```

특정한 값이 들어갈 수 있는 자리에 같은 타입을 리턴하는 메소드 콜이 사용될 수 있다:

int x = 3 + 24;

대신에

int x = 3 + one.getSize();

와 같이 쓸 수 있다.

Access Modifiers

public, protected, 그리고 private은 액세스 한정자(access modifier)이다.

public은 Subject가 임의의 모든 클래스에 의해 접근될 수 있고,

protected는 Subject가 서브 클래스에 의해서 접근될 수 있고,

private은 Subject가 클래스 자체에서만 접근될 수 있다.

Modifier가 지정되지 않은 경우는 "**package protected**"를 의미하며, 동일한 패키지의 클래스에 의해서만 액세스될 수 있도록 한다.

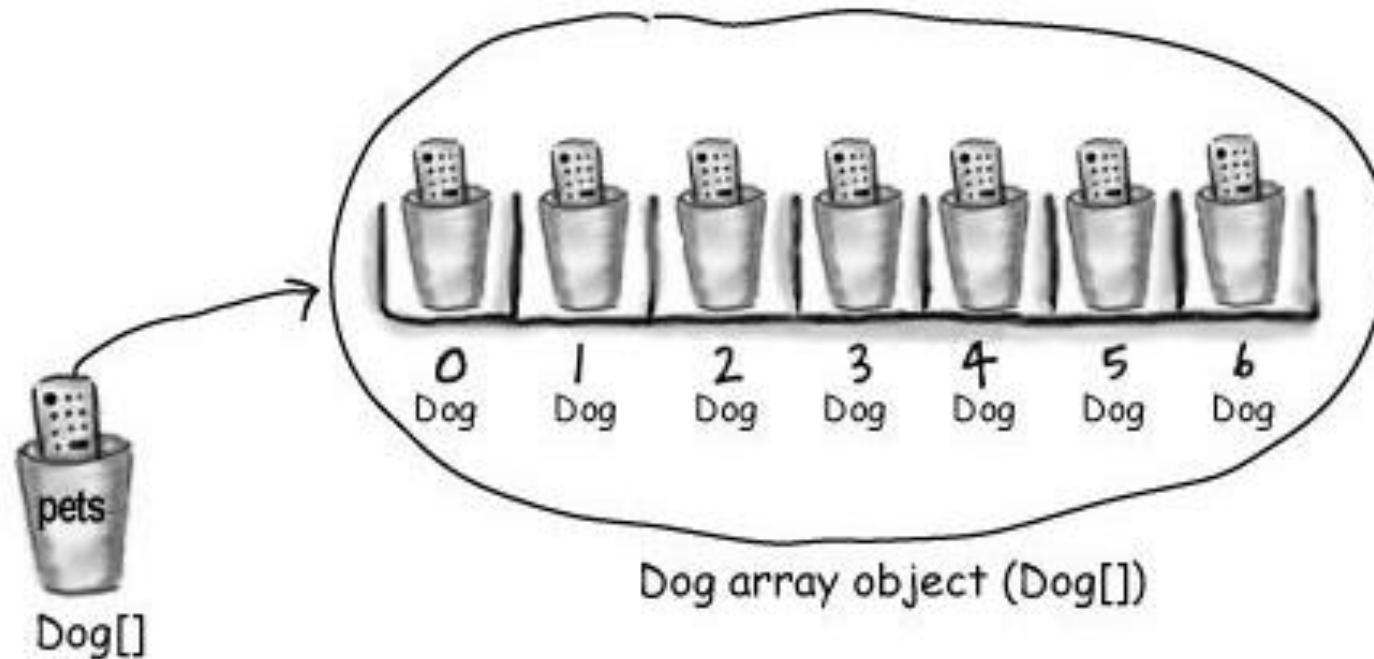
Subject는 클래스, 메소드, 멤버 변수이다.

배열에 있는 객체는 어떻게 행동할까?

1. **Dog** 레퍼런스 7개를 담을 수 있는 **Dog** 배열을 선언하고 생성하자.

```
Dog[] pets;
```

```
pets = new Dog[7];
```



배열에 있는 객체는 어떻게 행동할까?

2. **Dog** 객체 2개를 새로 만들고 첫 번째와 두 번째 배열 원소에 대입한다.

```
pets[0] = new Dog();
```

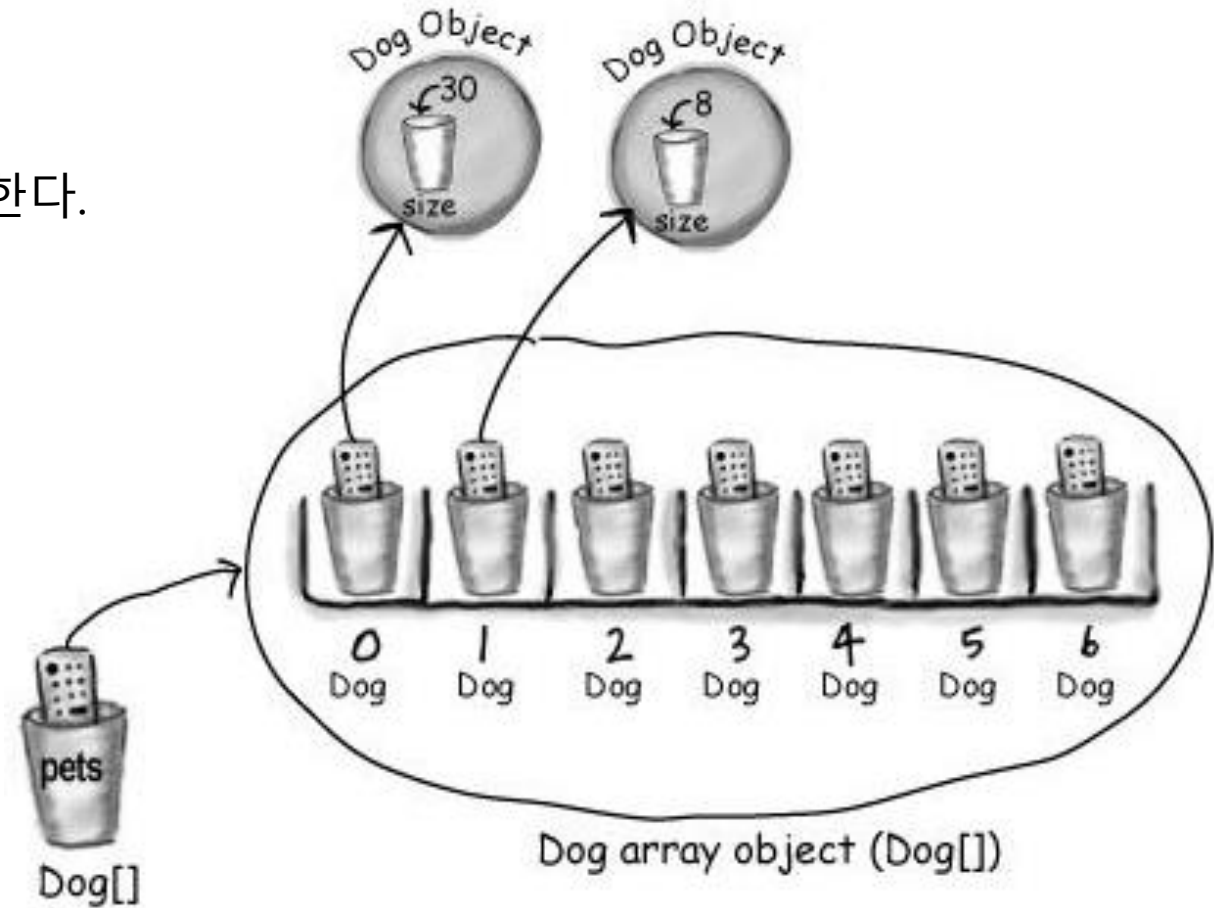
```
pets[1] = new Dog();
```

3. **Dog** 객체 2개에 대해 메소드를 호출한다.

```
pets[0].setSize(30);
```

```
int x = pets[0].getSize();
```

```
pets[1].setSize(8);
```



실습과제 4-2 인스턴스 변수 선언과 초기화

인스턴스 변수를 초기화하지 않으면 **getter** 메소드를 호출할 때 어떤 일이 발생할까?
즉, 인스턴스 변수를 초기화하기 전의 값은 무엇일까?

```
class PoorDog {  
    private int size;  
    private String name;  
  
    public int getSize() {  
        return size;  
    }  
    public String getName() {  
        return name;  
    }  
}
```

declare two instance variables,
but don't assign a value

What will these return??

인스턴스 변수는 항상 기본값을 갖는다.
명시적으로 인스턴스 변수에 값을 지정하지
않거나, setter 메소드를 호출하지 않아도 인
스턴스 변수에는 여전히 값이 있다!

integers	0
floating points	0.0
booleans	false
references	null

```
public class PoorDogTestDrive {  
    public static void main (String[] args) {  
        PoorDog one = new PoorDog();  
        System.out.println("Dog size is " + one.getSize());  
        System.out.println("Dog name is " + one.getName());  
    }  
}
```

What do you think? Will
this even compile?

```
File Edit Window Help CallVet  
% java PoorDogTestDrive  
Dog size is 0  
Dog name is null
```

인스턴스 변수와 지역 변수의 차이점

1. 인스턴스 변수는 클래스 내에서 선언된다. 메소드 내에서 선언되는 것이 아니다.

```
class Horse {  
    private double height = 15.2;  
    private String breed;  
    // more code...  
}
```

2. 지역 변수는 메소드 내에서 선언된다.

```
class AddThing {  
    int a;  
    int b = 12;  
  
    public int add() {  
        int total = a + b;  
        return total;  
    }  
}
```

3. 지역 변수는 사용하기 전에 반드시 초기화해야 한다.

```
class Foo {  
    public void go() {  
        int x;  
        int z = x + 3;  
    }  
}
```

Won't compile!! You can declare x without a value, but as soon as you try to USE it, the compiler freaks out.

File Edit Window Help Yikes

```
% javac Foo.java
```

```
Foo.java:4: variable x might  
not have been initialized
```

```
        int z = x + 3;
```

```
1 error
```

```
^
```

원시 변수와 레퍼런스 변수 비교

원시값 2개가 같은지 알아보려면 그냥 `==` 를 사용하면 된다.

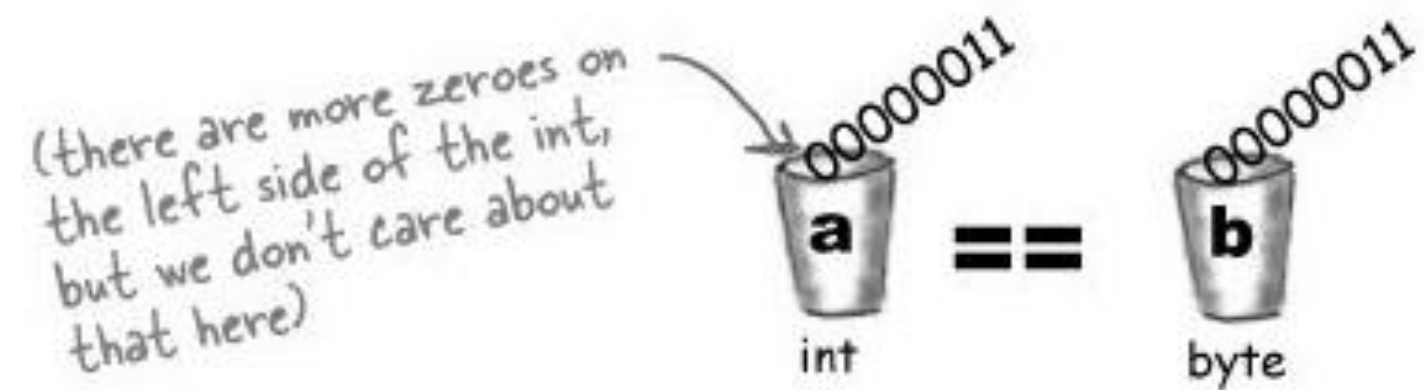
또는 두 레퍼런스가 같은 객체를 참조하는지 알아보는 것도 그냥 `==` 를 사용하면 된다.

두 개의 객체가 같은지를 알아보려면 **`equals()`** 메소드를 이용해야 한다.

두 원시값을 비교하기 위해서는 == 를 사용하라.

```
int a = 3;  
byte b = 3;  
if (a == b) { // true }
```

the bit patterns are the same, so these two are equal using ==

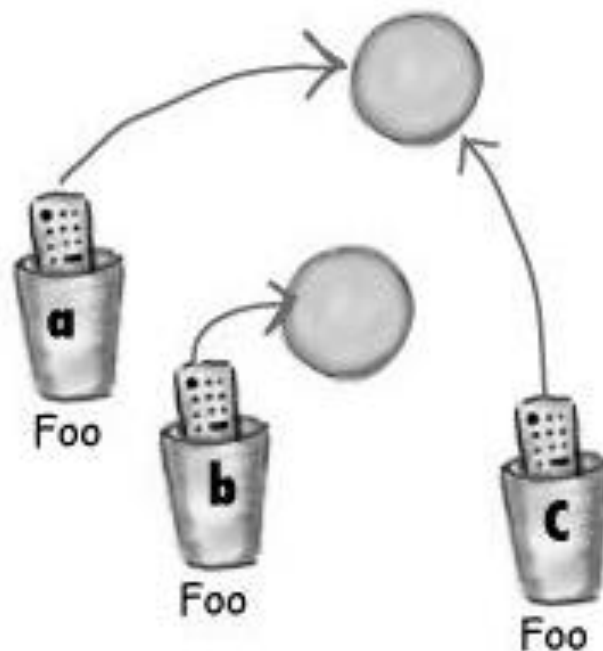


두 레퍼런스가 같은지를 알아보기 위해서는 `==` 를 사용하라.

```
Foo a = new Foo();  
Foo b = new Foo();  
Foo c = a;  
if (a == b) { // false }  
if (a == c) { // true }  
if (b == c) { // false }
```

the bit patterns are the same for a and c, so they are equal using ==

*a == c is true
a == b is false*



실습과제 4-3 BE the compiler

이 페이지의 각 Java 파일은 완전한 소스 파일을 나타낸다. 임무는 컴파일러를 실행해서, 이들 파일이 컴파일되는지 여부를 결정하는 것이다. 만약 컴파일되지 않는다면, 어떻게 수정해야 하는가? 컴파일된다면 출력은 어떻게 되는가?

A

```
class XCopy {  
  
    public static void main(String [] args) {  
  
        int orig = 42;  
  
        XCopy x = new XCopy();  
  
        int y = x.go(orig);  
  
        System.out.println(orig + " " + y);  
    }  
  
    int go(int arg) {  
  
        arg = arg * 2;  
  
        return arg;  
    }  
}
```

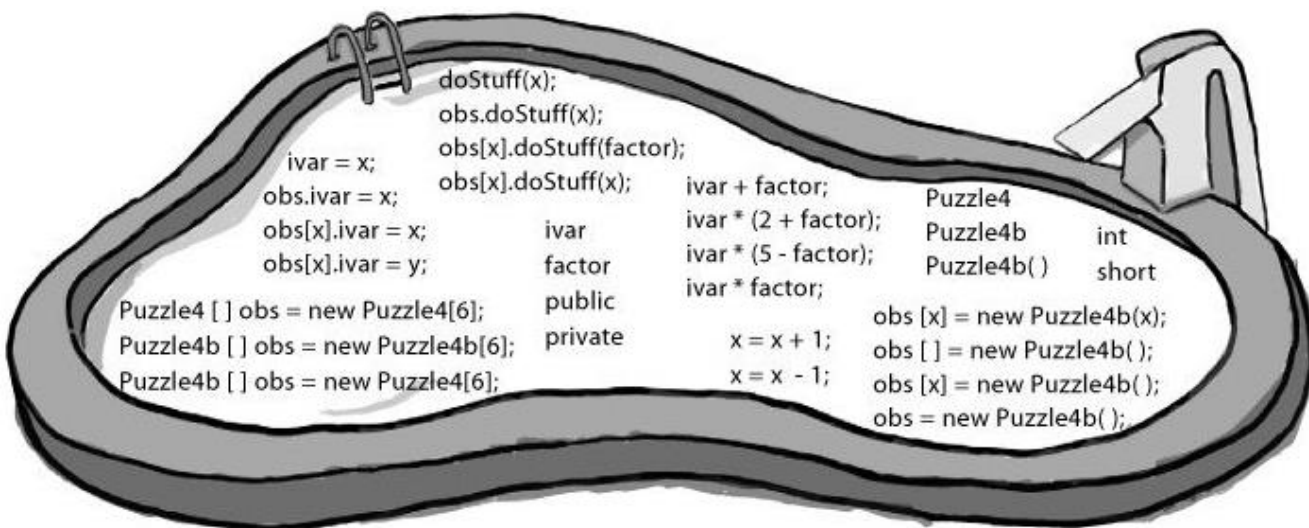
B

```
class Clock {  
    String time;  
  
    void setTime(String t) {  
        time = t;  
    }  
  
    void getTime() {  
        return time;  
    }  
}  
  
class ClockTestDrive {  
    public static void main(String [] args) {  
  
        Clock c = new Clock();  
  
        c.setTime("1245");  
        String tod = c.getTime();  
        System.out.println("time: " + tod);  
    }  
}
```

실습과제 4-4 Pool Puzzle

Output

```
File Edit Window Help BellyFlop
%java Puzzle4
result 543345
```



Note: Each snippet from the pool can be used only once!

```
public class Puzzle4 {
    public static void main(String [] args) {

        _____

        int y = 1;
        int x = 0;
        int result = 0;
        while (x < 6) {

            _____

            _____

            y = y * 10;

        }
        x = 6;
        while (x > 0) {

            _____

            result = result + _____

        }
        System.out.println("result " + result);
    }
}

class _____ {
    int ivar;

    _____ doStuff(int _____) {
        if (ivar > 100) {
            return _____
        } else {
            return _____
        }
    }
}

}
```

