4장. 객체의 행동

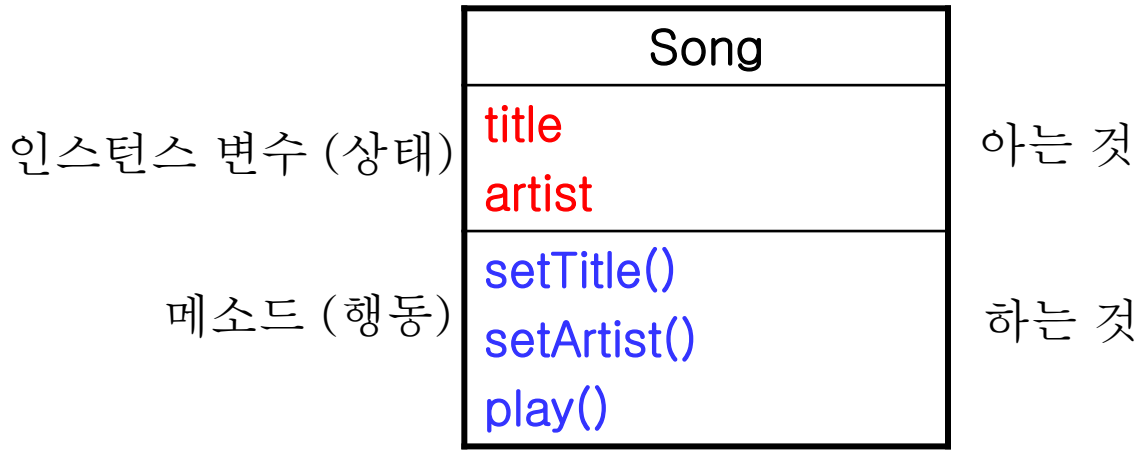
학습목표

1. 인스턴스 변수와 메소드의 상호 관계에 대해 알아봅니다.
2. 매개변수와 리턴값에 대해 알아봅니다.
3. 객체의 동치에 대해 알아봅니다.

객체의 행동

상태 – 인스턴스 변수
행동 – 메소드

객체의 상태와 행동



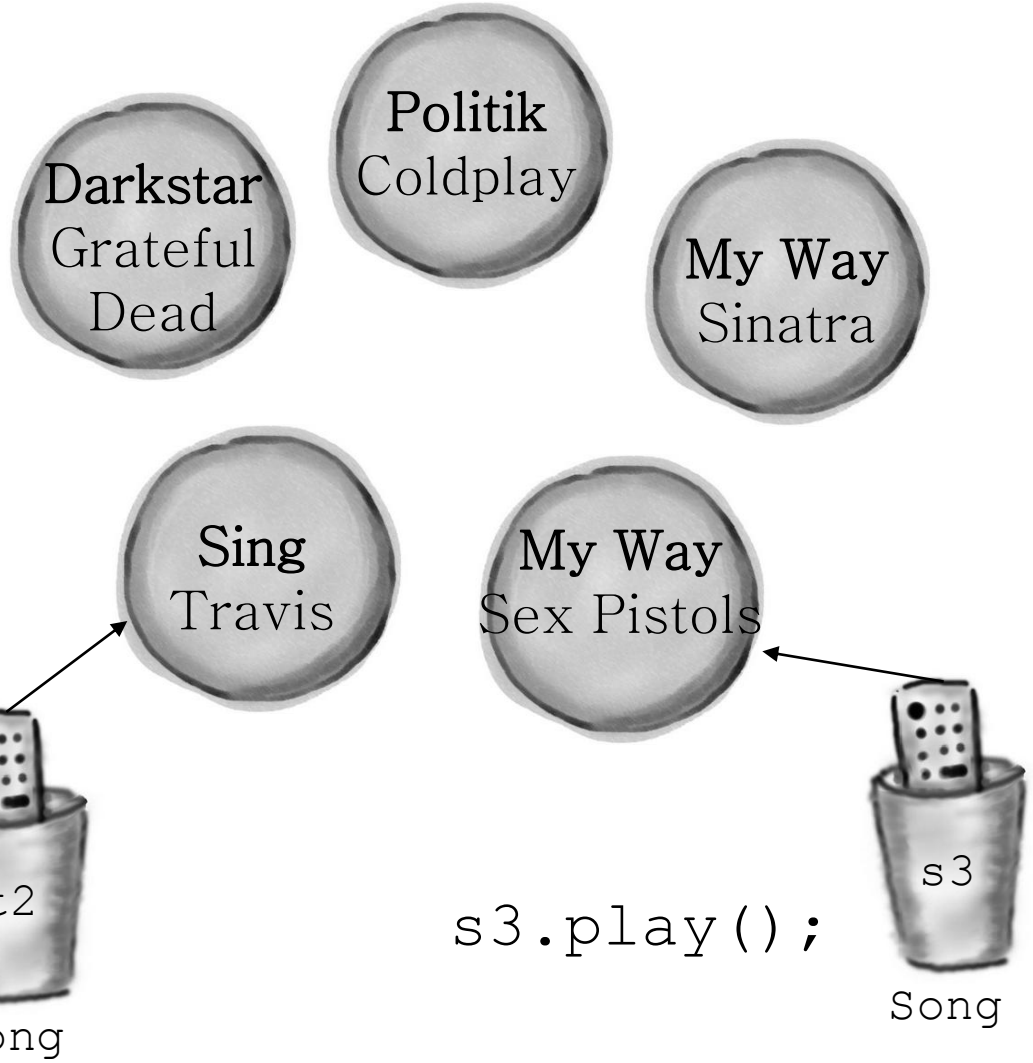
Q: 모든 객체의 인스턴스 변수(상태)는 달라질 수 있습니다. 그런데 메소드(행동)도 달라질 수 있을까요?

A: 메소드 자체는 똑같지만 그 메소드의 실행 결과는 달라질 수 있습니다.

객체의 상태와 행동

```
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");
```



t2.play();

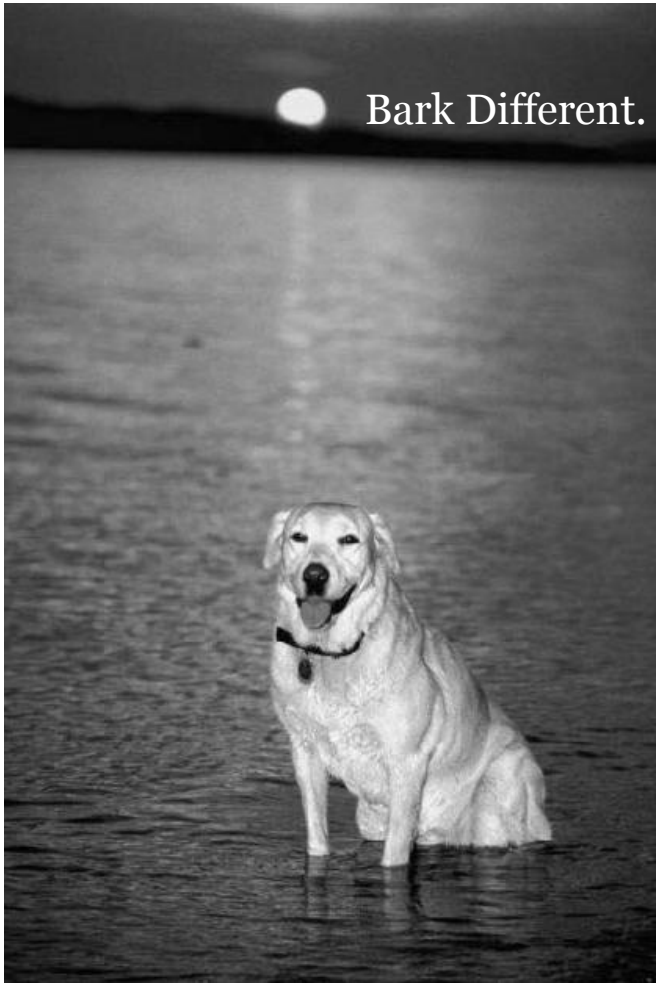
s3.play();

객체의 상태와 행동

```
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!");
      }
   }
}
```

Dog
size name
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();
           }
        }
```

코드를 직접 실행시켜봅시다.

매개변수/인자

- 자바에서도 메소드에 어떤 값을 전달할 수 있습니다.

- 매개변수
  - 메소드에서 사용하는 것
- 인자
  - 호출하는 쪽에서 전달하는 것

- 메소드에서 받은 매개변수는 그 메소드에서 선언한 지역 변수와 똑같이 간주됩니다.
- 메소드에 매개변수가 있으면 반드시 해당 유형의 값을 전달해야만 합니다.

매개변수/인자

1. bark 메소드 호출 (인자로 3을 전달)

```
Dog d = new Dog();
d.bark(3);
```

인자

매개변수

```
void bark(int numOfBarks) {
   while (numOfBarks > 0) {
      System.out.println("ruff");
      numOfBarks = numOfBarks - 1;
   }
}
```

2. 3이라는 값을 나타내는 비트들이 bark 메소드로 전달됨

3. 그 비트들이 numOfBarks 매개변수에 들어감

4. numOfBarks 매개변수를 메소드 코드 내에서 변수로 사용

리턴값

- 메소드로부터 값을 받을 수도 있습니다.

```
void go() {
}
```

```
int giveSecret() {
    return 42;
}
```

- 리턴 유형은 메소드를
  리턴 유형이 정해져 있
  값을 리턴해야만 합니

리턴값

```
int theSecret = life.giveSecret();


int giveSecret() {
    return 42;
}
```

두 개 이상의 매개변수

- 메소드에 두 개 이상의 인자를 전달할 수도 있습니다.
  - 각 인자는 쉼표로 구분합니다.
  - 메소드에 매개변수가 있을 때 반드시 인자를 전달해야 한다는 원칙은 여전히 적용됩니다.

두 개 이상의 매개변수

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

```
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);
}
```

값으로 전달 (pass by value)

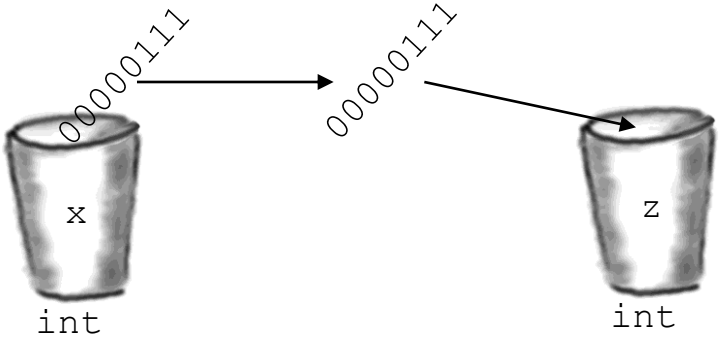
- 자바에서는 값으로 전달하는 방식을 사용합니다.

값으로 전달 (pass by value)

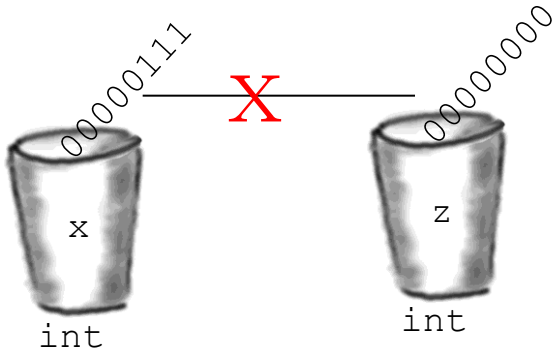
```
int x = 7;
```


```
void go(int z) { }
```


```
foo.go(x);
```




```
void go(int z) { }
```




```
void go(int z) {
    z = 0;
}
```

바보 같은 질문은 없습니다.

- 전달하려고 하는 인자가 원시변수가 아니고 객체인 경우에는 어떻게 되나요?
  - 객체 레퍼런스를 인자로 전달하는 경우에도 여전히 값으로만 전달됩니다. 중요한 것은 여기에서 “값”이라는 것이 “객체”가 아니라, 객체를 참조하는 레퍼런스라는 점입니다. 따라서 그 레퍼런스의 복사본이 전달됩니다.

바보 같은 질문은 없습니다.

- 메소드에서 리턴 값을 여러 개 선언할 수 있나요? 값을 두 개 이상 리턴하는 방법이 있나요?
  - 리턴값은 한 가지밖에 선언할 수 없습니다. 하지만 세 개의 int를 리턴하고 싶다면 리턴 유형을 int 배열로 선언하면 됩니다. 여러 유형의 값을 리턴하는 방법은 조금 더 복잡한데, 나중에 배우게 될 ArrayList라는 것을 사용하면 됩니다.

바보 같은 질문은 없습니다.

- 정확하게 처음에 선언한 유형으로만 리턴해야 하나요?
  - 자동으로 해당 유형으로 변환되는 것은 그냥 리턴해도 됩니다.
  - 예: int를 리턴하겠다고 선언한 경우에 byte를 리턴해도 됩니다.
  - 하지만 그렇지 않은 경우에는 강제로 캐스팅을 해야 합니다.
  - 예: return (int) (Math.random() * 10.0);

바보 같은 질문은 없습니다.

- 메소드에서 리턴한 값으로 반드시 뭔가를 해야 하나요? 그냥 무시하면 안 되나요?
  - 자바에서는 리턴값의 사용 여부에는 전혀 신경을 쓰지 않습니다. 따라서 리턴값을 꼭 어떤 변수에 대입한다거나 특정한 용도로 사용하지 않아도 됩니다. 예를 들어 리턴값은 별로 필요 없고 메소드로 어떤 작업을 하기만 되는 경우에는 리턴값은 그냥 무시하고 메소드를 호출하기만 해도 됩니다.

핵심 정리

- 클래스에서는 객체가 하는 것과 객체가 아는 것을 정의합니다.
- 인스턴스 변수(상태)는 객체가 아는 것입니다.
- 메소드(행동)는 객체가 하는 것입니다.
- 메소드에서 인스턴스 변수를 이용하여 같은 형식의 객체가 다른 식으로 행동하도록 할 수 있습니다.
- 메소드에서 매개변수를 사용할 수 있습니다. 즉 메소드에 한 개 이상의 값을 전달할 수 있습니다.

핵심 정리

- 전달하는 값의 개수와 유형은 반드시 메소드를 선언할 때 지정한 것과 같아야 하며 그 순서도 같아야 합니다.
- 메소드 안팎으로 전달되는 값은 상황에 따라 자동으로 더 큰 유형으로 올라갈 수 있습니다. 더 작은 유형으로 바꿀 때는 강제로 캐스팅을 해야 합니다.
- 메소드에 인자를 전달할 때는 리터럴 값(2, 'c' 등)을 사용할 수도 있고 선언된 매개변수 유형의 변수(예를 들어 int 변수 x)를 사용할 수도 있습니다.

핵심 정리

- 메소드를 선언할 때는 반드시 리턴 유형을 지정해야 합니다. 리턴 유형을 void로 지정하면 아무 것도 리턴하지 않아도 됩니다.
- 메소드를 선언할 때 void가 아닌 리턴 유형을 지정했을 때는 반드시 선언된 리턴 유형과 호환 가능한 값을 리턴해야 합니다.

게터(getter)와 세터(setter)

- 게터(getter)
  - 인스턴스 변수의 값을 알아내기 위한 메소드
  - 일반적으로 인스턴스 변수의 값을 리턴함
  - getBrand(), getNumOfPickups()...
- 세터(setter)
  - 인스턴스 변수의 값을 설정하기 위한 메소드
  - 전달된 값을 확인하고 인스턴스 변수의 값을 설정함
  - setBrand(), setNumOfPickups()...

게터와 세터

```
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 = yseOrNo;
    }
}
```

캡슐화(encapsulation)

- 데이터 노출!!!
  - 지금까지 우리가 만든 프로그램에는 데이터가 완전히 노출되어있다는 심각한 문제가 있었습니다. 즉 아무나 인스턴스 변수를 마음대로 보고 건드릴 수 있었습니다.
  - `theCat.height = 27;`
  - `theCat.height = 0;`

이렇게 theCat이라는 객체의 height 변수의 값을 마음대로 0으로 바꿀 수 있으면 안 됩니다.

캡슐화

- 이런 문제를 어떻게 해결할 수 있을까요?
  - 세터 메소드를 쓰면 됩니다.

```
theCat.height = 0;
```

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

세터 메소드를 사용하면 이렇게 인스턴스 변수의 값이 합당한지 검사할 수도 있습니다.

캡슐화

- 그렇다면 데이터를 직접 건드릴 수 없도록 하려면 어떻게 해야 할까요?

- 액세스 변경자(access modifier)
  - 인스턴스 변수: private으로 선언
  - 게터 및 세터 메소드: public으로 선언

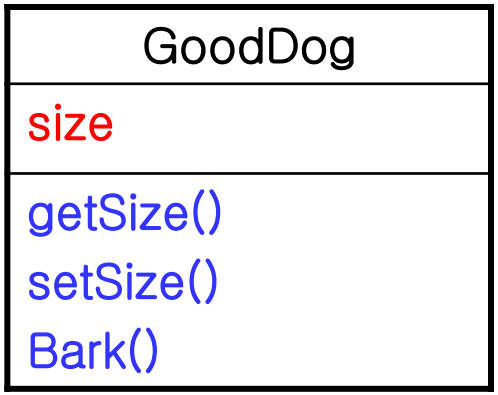
캡슐화 실습

```
class GoodDog {
    private int size;

    public int getSize() {
        return size;
    }

    public void setSize(int s) {
        size = s;
    }

    void bark() {
        ……
    }
}
```

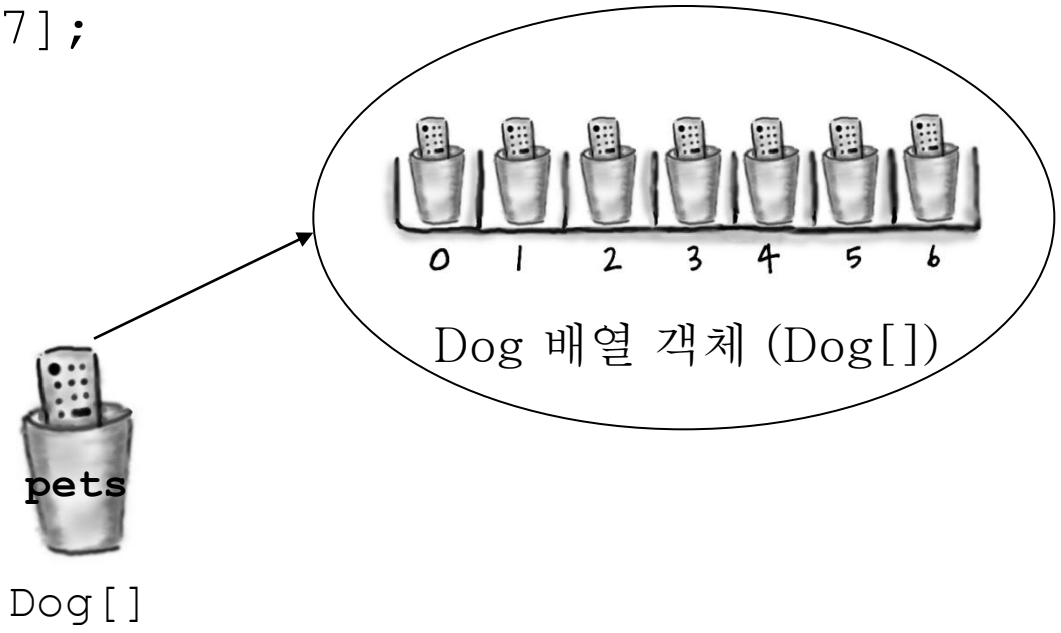


112 페이지에 있는 예제를 직접 실행시켜봅시다.

배열 안에 있는 객체

1. Dog 배열 변수 선언

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



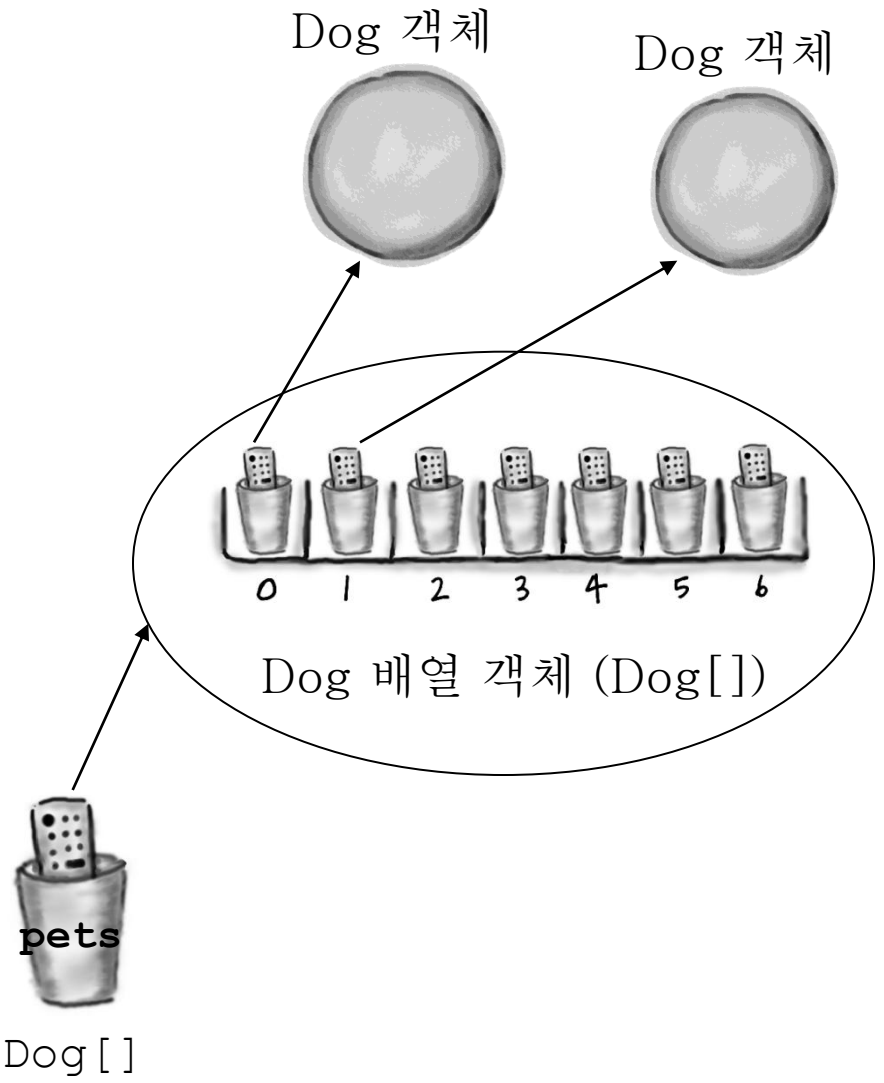
배열 안에 있는 객체

2. 두 개의 새로운 Dog 객체를 만들고 두 개의 배열 원소 대입

```
pets[0] = new Dog();
pets[1] = new Dog();
```

3. Dog 객체 두 개에 대해 메소드 호출

```
pets[0].setSize(30);
int x = pets[0].getSize();
pets[1].setSize(8);
```



인스턴스 변수 초기화

- 인스턴스 변수 선언
  - int size;
  - String name;

- 인스턴스 변수 선언 및 초기화
  - int size = 420;
  - String name = “Donny”;

인스턴스 변수의 기본값

```
class PoorDog {
   private int size;
   private String name;

   public int getSize() {
      return size;
   }

   public String getName() {
      return name;
   }
}
```

```
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());
   }
}
```

114 페이지에 있는 예제를 직접 실행시켜봅시다.

인스턴스 변수의 기본값

- 인스턴스 변수에는 항상 기본값이 들어갑니다.

- 각 유형별 기본값
  - 정수 0
  - 부동소수점 수 0.0
  - 부울 false
  - 레퍼런스 null

인스턴스 변수와 지역 변수

- 선언되는 위치
  - 인스턴스 변수 – 클래스 내에서
  - 지역 변수 – 메소드 내에서
- 초기화
  - 인스턴스 변수 – 기본 초기값이 있음
  - 지역 변수 – 기본 초기값이 없음

```
class Horse {
   private double height = 15.2;
   private String breed;
   // 나머지 생략
}
```

```
class AddThing {
  class Foo {
    public void go() {
      int total = a + b;
      int z = x + 3;
      return total;
    }
  }
}
```

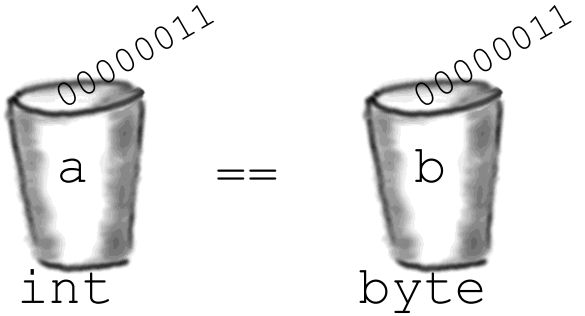
바보 같은 질문은 없습니다.

- 메소드 매개변수는 어떤가요? 지역 변수와 관련된 규칙이 매개변수에는 어떻게 적용되죠?
  - 메소드 매개변수는 지역 변수와 거의 똑같습니다. 메소드 내에서 선언되지요. 다만 메소드를 호출할 때 필요한 인자를 전달하지 않으면 컴파일 과정에서 에러가 나기 때문에 매개변수가 초기화되지 않았다는 에러 메시지가 뜨는 일은 없습니다.

변수 비교

- == 연산자
  - 임의의 유형의 두 변수를 비교하기 위한 연산자
  - 비트 패턴을 비교합니다.

  - 원시값
    - 두 값을 직접 비교합니다.
  - 레퍼런스
    - 두 변수가 같은 객체를 참조하는지를 비교합니다.




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

Foo a = new Foo();
Foo b = new Foo();
Foo c = b;
if (a == b) { // 거짓 }
if (a == c) { // 참 }
if (b == c) { // 거짓 }
```

연필을 깎으며

```
int calcArea(int height, int width) {
   return height * width;
}
```

메소드를 제대로 호출한 부분은?

```
int a = calcArea(7, 12);
short c = 7;
calcArea(c, 15);
int d = calcArea(57);
calcArea(2,3);
long t = 42;
int f = calcArea(t, 17);
int g = calcArea();
calcArea();
byte h = calcArea(4, 20);
int j = calcArea(2, 3, 5);
```