Course: Object Based Modeling
Code: CS-33105
Branch: MCA-3

Lecture -4

Dr. J Sathish Kumar (JSK)
(Faculty & Coordinator)

Department of Computer Science and Engineering

Motilal Nehru National Institute of Technology Allahabad,
Prayagraj-211004

Class Exercise #1

BE the compiler

A

```
class Books {
  String title;
  String author;
}

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

    Books [] myBooks = new Books[3];
    int x = 0;
    myBooks[0].title = "The Grapes of Java";
    myBooks[1].title = "The Java Gatsby";
    myBooks[2].title = "The Java Cookbook";
    myBooks[0].author = "bob";
    myBooks[1].author = "sue";
    myBooks[2].author = "ian";

    while (x < 3) {
      System.out.print(myBooks[x].title);
      System.out.print(" by ");
      System.out.println(myBooks[x].author);
      x = x + 1;
    }
  }
}
```

B

```
class Hobbits {

  String name;

  public static void main(String [] args) {

    Hobbits [] h = new Hobbits[3];
    int z = 0;

    while (z < 4) {
      z = z + 1;
      h[z] = new Hobbits();
      h[z].name = "bilbo";
      if (z == 1) {
        h[z].name = "frodo";
      }
      if (z == 2) {
        h[z].name = "sam";
      }
      System.out.print(h[z].name + " is a ");
      System.out.println("good Hobbit name");
    }
  }
}
```

#Solution

A

```
class Books {
  String title;
  String author;
}
class BooksTestDrive {
  public static void main(String [] args) {
    Books [] myBooks = new Books[3];
    int x = 0;
    myBooks[0] = new Books();
    myBooks[1] = new Books();
    myBooks[2] = new Books();
    myBooks[0].title = "The Grapes of Java";
    myBooks[1].title = "The Java Gatsby";
    myBooks[2].title = "The Java Cookbook";
    myBooks[0].author = "bob";
    myBooks[1].author = "sue";
    myBooks[2].author = "ian";
    while (x < 3) {
      System.out.print(myBooks[x].title);
      System.out.print(" by ");
      System.out.println(myBooks[x].author);
      x = x + 1;
    }
  }
}
```

Remember: We have to actually make the Books objects !

B

```
class Hobbits {
  String name;
  public static void main(String [] args) {
    Hobbits [] h = new Hobbits[3];
    int z = -1;
    while (z < 2) {
      z = z + 1;
      h[z] = new Hobbits();
      h[z].name = "bilbo";
      if (z == 1) {
        h[z].name = "frodo";
      }
      if (z == 2) {
        h[z].name = "sam";
      }
      System.out.print(h[z].name + " is a ");
      System.out.println("good Hobbit name");
    }
  }
}
```

Remember: arrays start with element 0 !

Class Exercise #2

```
File  Edit  Window  Help  Bikini

% java TestArrays
island = Fiji
island = Cozumel
island = Bermuda
island = Azores
```

```
class TestArrays {

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

```
int y = 0;
```

```
ref = index[y];
```

```
index[0] = 1;
index[1] = 3;
index[2] = 0;
index[3] = 2;
```

```
islands[0] = "Bermuda";
islands[1] = "Fiji";
islands[2] = "Azores";
islands[3] = "Cozumel";
```

```
String [] islands = new String[4];
```

```
int ref;
while (y < 4) {
```

```
System.out.print("island = ");
```

```
int [] index = new int[4];
```

```
System.out.println(islands[ref]);
```

```
y = y + 1;
```

#Solution

```
class TestArrays {
  public static void main(String [] args) {
    int [] index = new int[4];
    index[0] = 1;
    index[1] = 3;
    index[2] = 0;
    index[3] = 2;
    String [] islands = new String[4];
    islands[0] = “Bermuda”;
    islands[1] = “Fiji”;
    islands[2] = “Azores”;
    islands[3] = “Cozumel”;
    int y = 0;
    int ref;
    while (y < 4) {
      ref = index[y];
      System.out.print(“island = “);
      System.out.println(islands[ref]);
      y = y + 1;
    }
  }
}
```

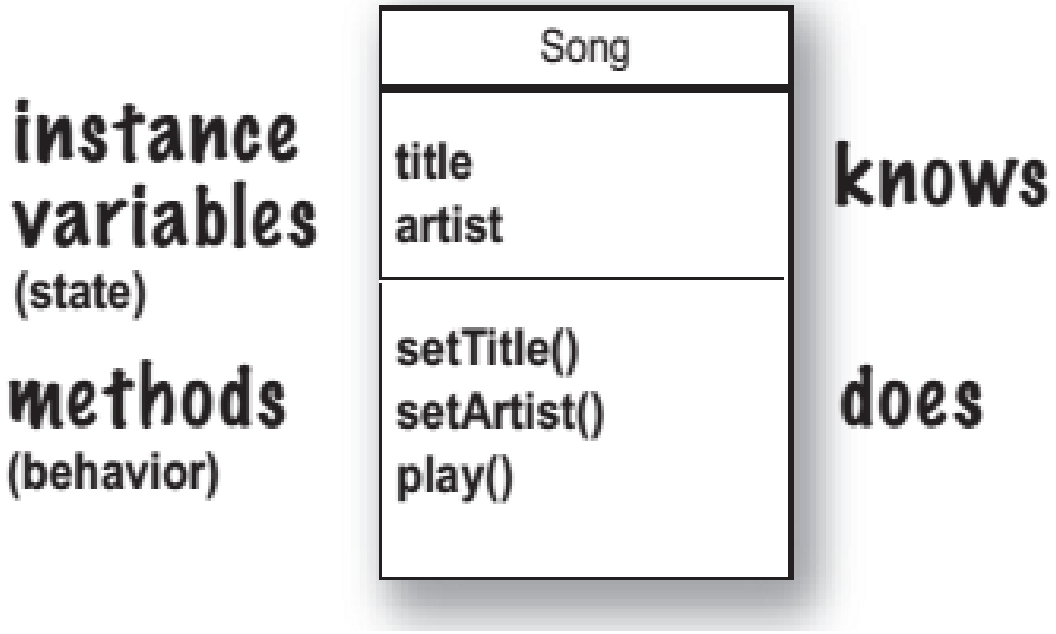
File Edit Window Help Bikini

```
% java TestArrays
island = Fiji
island = Cozumel
island = Bermuda
island = Azores
```

Objects and Methods

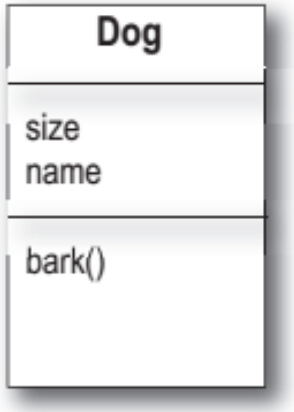
- A class describes what an object knows and what an object does

- But what about the methods?



Can every object of that type have different method behavior?

```
class Dog {
  int size;
  String name;

  void bark() {
    if (size > 60) {
       System.out.println("Wooof! Wooof!");
    } else if (size > 14) {
       System.out.println("Ruff!  Ruff!");
    } else {
       System.out.println("Yip! Yip!");
    }
  }
}
```




```
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();
   }
}
```

```
File Edit Window Help Playdead
%java DogTestDrive
Wooof! Wooof!
Yip! Yip!
Ruff!  Ruff!
```

THE SIZE AFFECTS THE BARK

Method Parameters

- You can send things to a method
- A method uses parameters. A caller passes arguments.

1. Call the bark method on the Dog reference, and pass in the value 3 (as the argument to the method).

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

argument

2. The bits representing the int value 3 are delivered into the bark method.

Parameter

00000011

int

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

3. The bits land in the numOfBarks parameter (an int-sized variable).

4. Use the numOfBarks parameter as a variable in the method code.

Method Parameters

- You can get things back from a method. Methods can return values.
- Every method is declared with a return type, but until now we've made all of our methods with a **void** return type, which means they don't give anything back.
- But we can declare a method to give a specific type of value back to the caller

```
void go() {

}
```

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

Multiple Arguments

- You can send more than one thing to a method
- Methods can have multiple parameters.
- Separate them with commas when you declare them, and separate the arguments with commas when you pass them.
- Most importantly, if a method has parameters, you *must* pass arguments of the right type and order.

```
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.

Multiple Arguments

- You can pass variables into a method, as long as the variable type matches the parameter type.

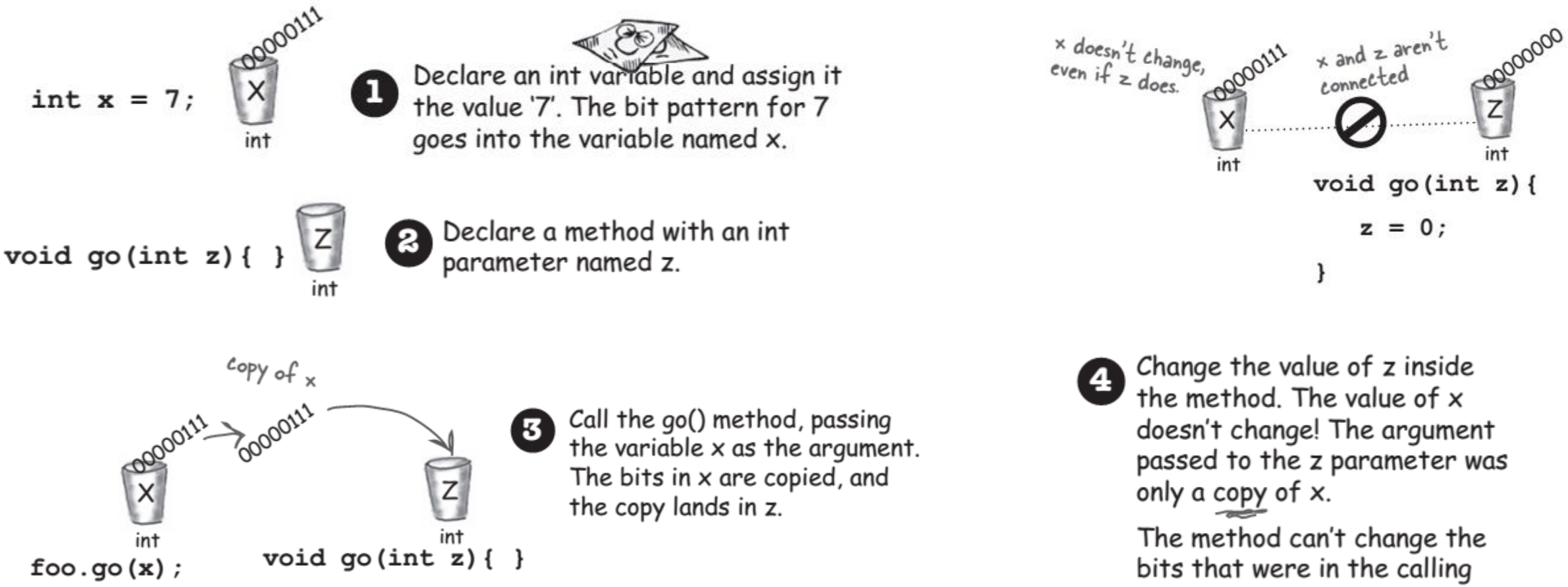
```
void go() {
    int foo = 7;
    int bar = 3;
    t.takeTwo(foo, bar);
}

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

The 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

Methods use instance variables

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



Encapsulation: Hide the data

- Exposed means reachable with the dot operator, as in:
  - **theCat.height = 27;**
- Think about this idea of using our remote control to make a direct change to the Cat object's size instance variable.
- In the hands of the wrong person, a reference variable (remote control) is quite a dangerous weapon. This would be a Bad Thing.
- We need to build setter methods for all the instance variables, and find a way to force other code to call the setters rather than access the data directly.

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

We put in checks to guarantee a minimum cat height.

```
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("Wooof! Wooof!");
    } else if (size > 14) {
      System.out.println("Ruff!  Ruff!");
    } else {
      System.out.println("Yip! Yip!");
    }
  }
}
```

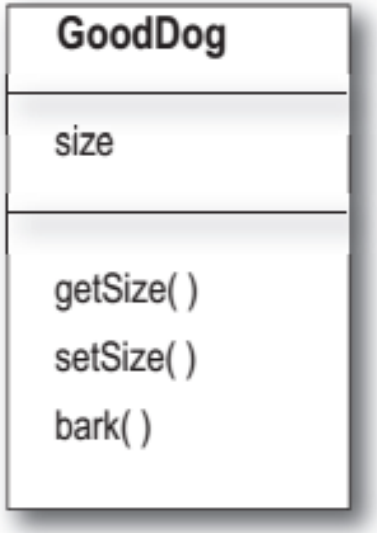
Make the instance variable private.

Make the getter and setter methods public.

Even though the methods don't really add new functionality, the cool thing is that you can change your mind later. you can come back and make a method safer, faster, better.

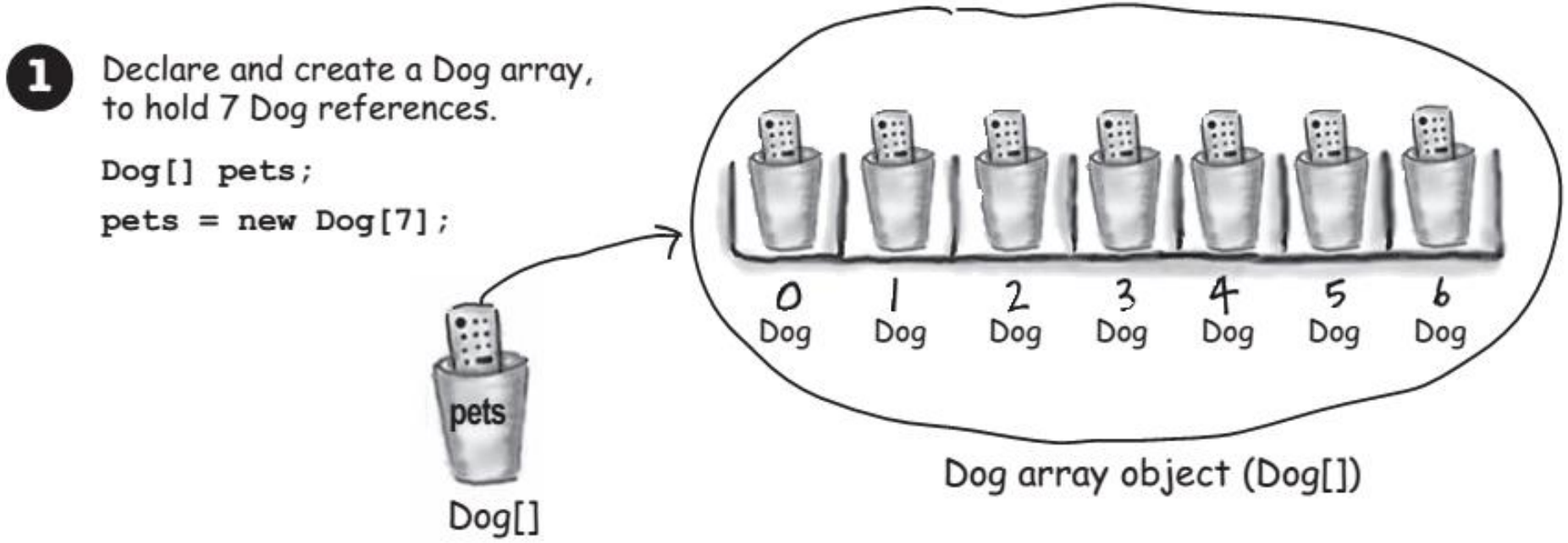
```
class GoodDogTestDrive {

  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();
  }
}
```

Encapsulating the GoodDog class



Methods use instance variables

- How do objects in an array behave?



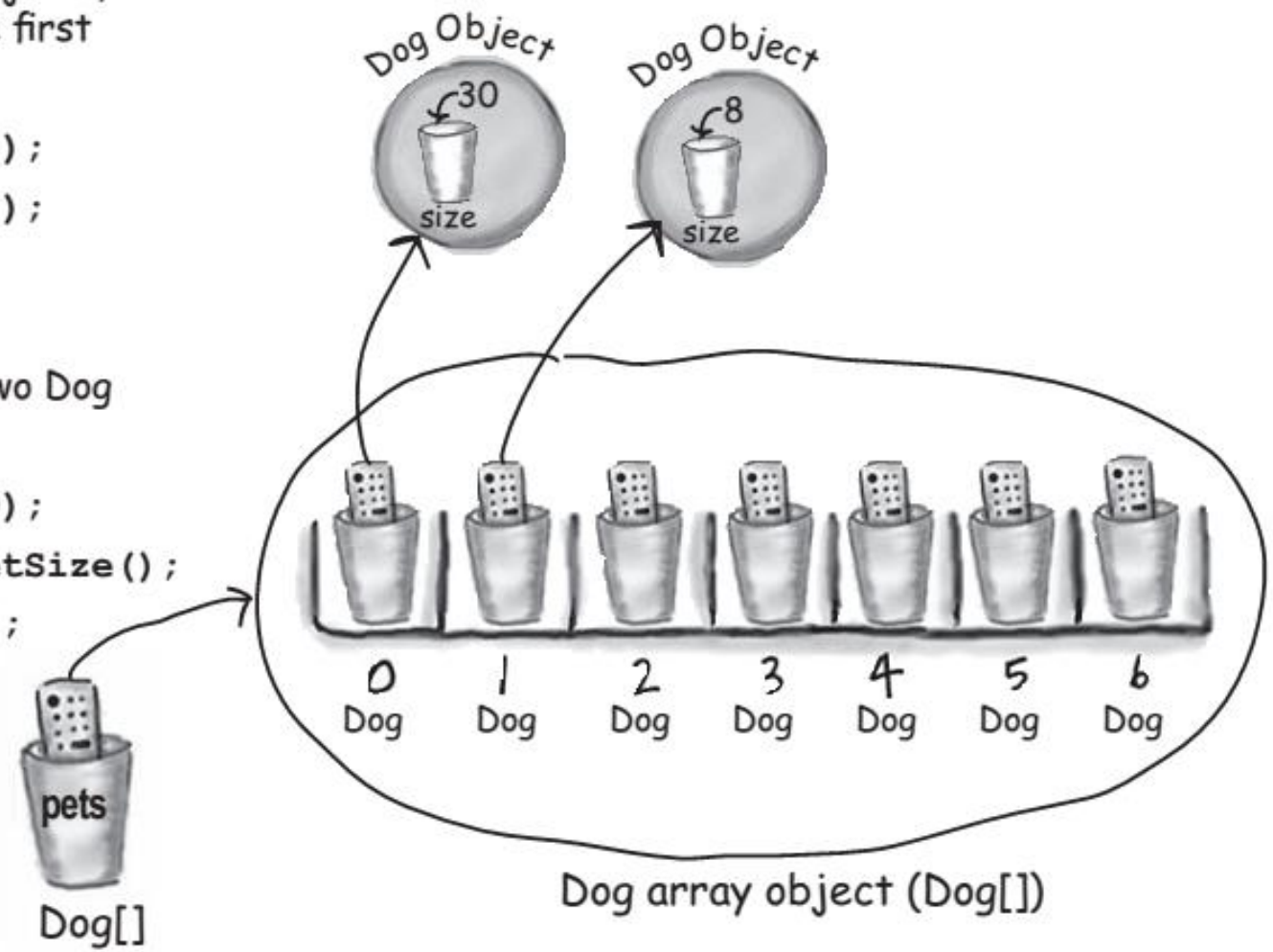
Methods use instance variables

2 Create two new Dog objects, and assign them to the first two array elements.

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

3 Call methods on the two Dog objects.

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



Initializing instance variables

- Declaring and initializing instance variables
  - We already know that a variable declaration needs at least a name and a type:
    - **int size;**
      String name;
  - And we know that you can initialize (assign a value) to the variable at the same time:
    - **int size = 420;**
      String name = “Donny”;
- But when you don’t initialize an instance variable, what happens when you call a getter method? In other words, what is the *value* of an instance variable *before* you initialize it?

Initializing instance variables

```
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??

```
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
```

You don't have to initialize instance variables, because they always have a default value. Number primitives (including char) get 0, booleans get false, and object reference variables get null.

(Remember, null just means a remote control that isn't controlling / programmed to anything. A reference, but no actual object.

Initializing instance variables

- Instance variables always get a default value.
- If you don’t explicitly assign a value to an instance variable, or you don’t call a setter method, the instance variable still has a value!

integers	0
floating points	0.0
booleans	false
references	null

Difference between instance and local variables

1 **Instance** variables are declared inside a class but not within a method.

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

2 **Local** variables are declared within a method.

```
class AddThing {
   int a;
   int b = 12;

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

3 **Local** variables MUST be initialized before use!

```
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

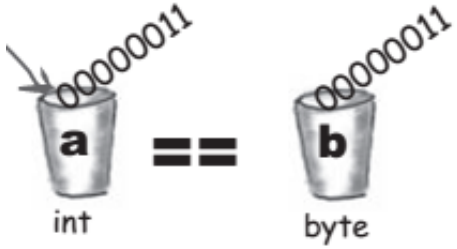
```
% javac Foo.java

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

      int z = x + 3;
1 error        ^
```

Local variables do NOT get a default value! The compiler complains if you try to use a local variable before the variable is initialized.

Object Equality

- Comparing variables (primitives or references)
- Use == to compare two primitives, or to see if two references refer to the same object.
  - The == operator can be used to compare two variables of any kind, and it simply compares the bits.
- Use the equals() method to see if two different objects are equal.

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




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