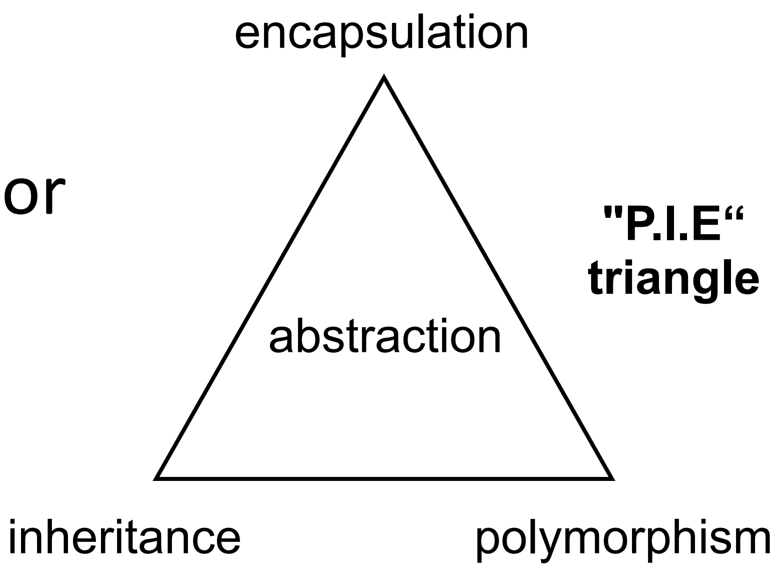
Object-Oriented Programming

Objects and Classes

Contents

Important OO Concepts

- Object & Class
  - Object state and behavior
  - Object identity
  - Messages
- Encapsulation
  - Information hiding
- Inheritance
- Polymorphism
- Abstraction



Java Program

```
public class Greeting {
  public void greet() {
    System.out.print("Hi there!");
  }
}
```

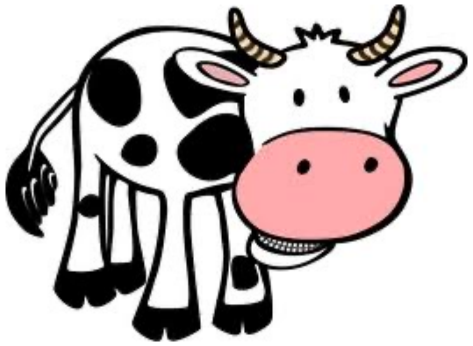
```
public class TestGreeting {
  public static void main(String[] args) {
    Greeting gr = new Greeting();
    gr.greet();
  }
}
```

- A Java program, when we write it, is a collection of classes
- A Java program, when we run it, is a collection of objects. They do things (their methods) and ask other objects to do things (calling methods of others)
- A Java library contains predefined classes that we can use in our programs

Objects

- Object is a "thing" that includes both *data* (*properties/ attributes*) and *functions* (*methods/behaviors*). *In OOP, objects* can either do something or have something done to them

Jenny



I can moo

Ben

I am going for a walk

Objects

- Objects in OOP have 3 essential features:
  - State: what objects have
  - Behavior: what objects do in response to messages
  - Identity: what makes objects unique

Object State

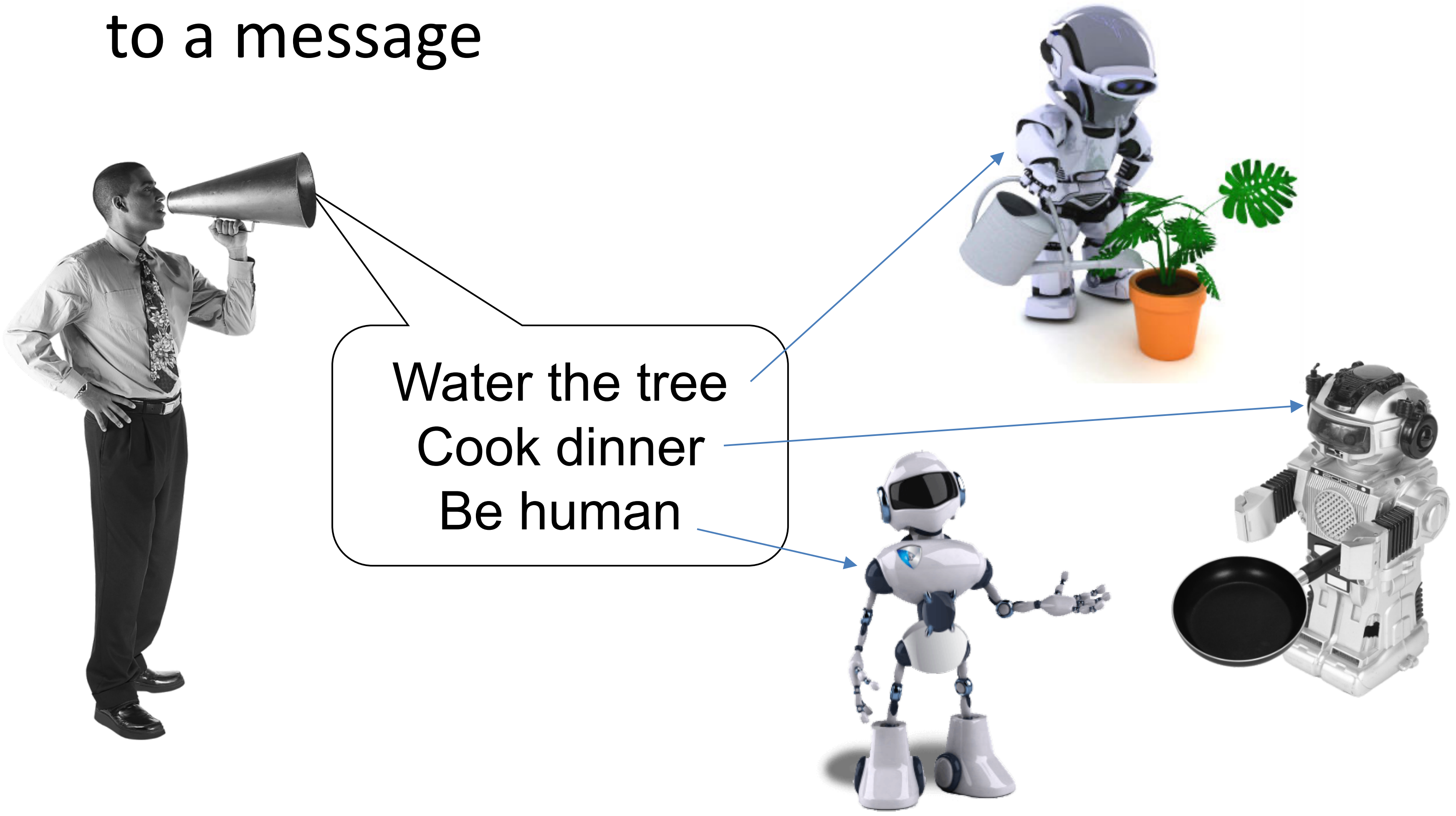
- Defined by the attributes of the object and by the values of these attributes
- Changes over time
  - "Name" attribute does not change over time
  - "Age" attribute changes over time

Dave
Age: 32
Height: 1m80

Peter
Age: 35
Height: 1m75

Object Behavior

- Behavior is what the object do in responding to a message

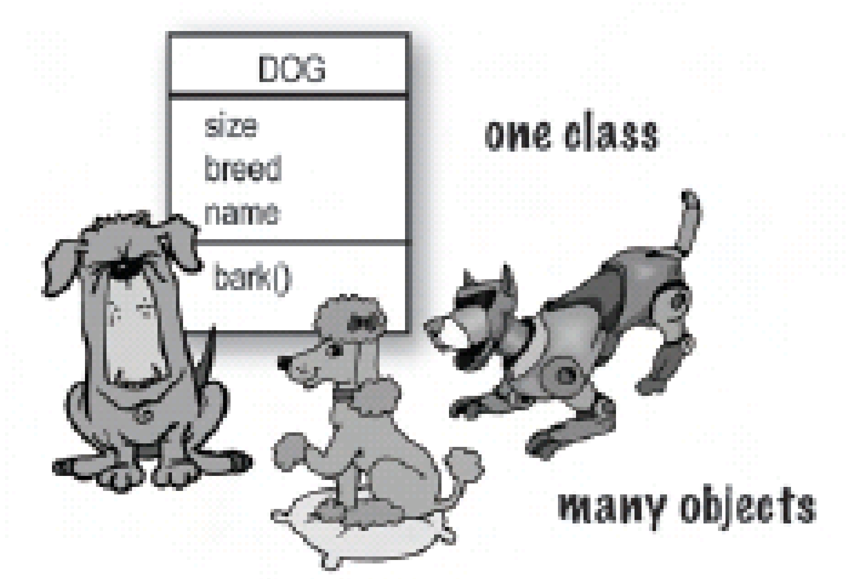


Object Identity

- Identity is what to make the object unique
  - Defined by object address or object ID
- Used to distinguish between objects

Classes

- A class is a blueprint/template that is used to construct objects



Q - S
N - P
L - M
Name Polly Morfism
Phone 555-0343
eMail pm@wickedlysmart

Classes vs. Objects

- Each object has the same structure and behavior as the class from which it was created

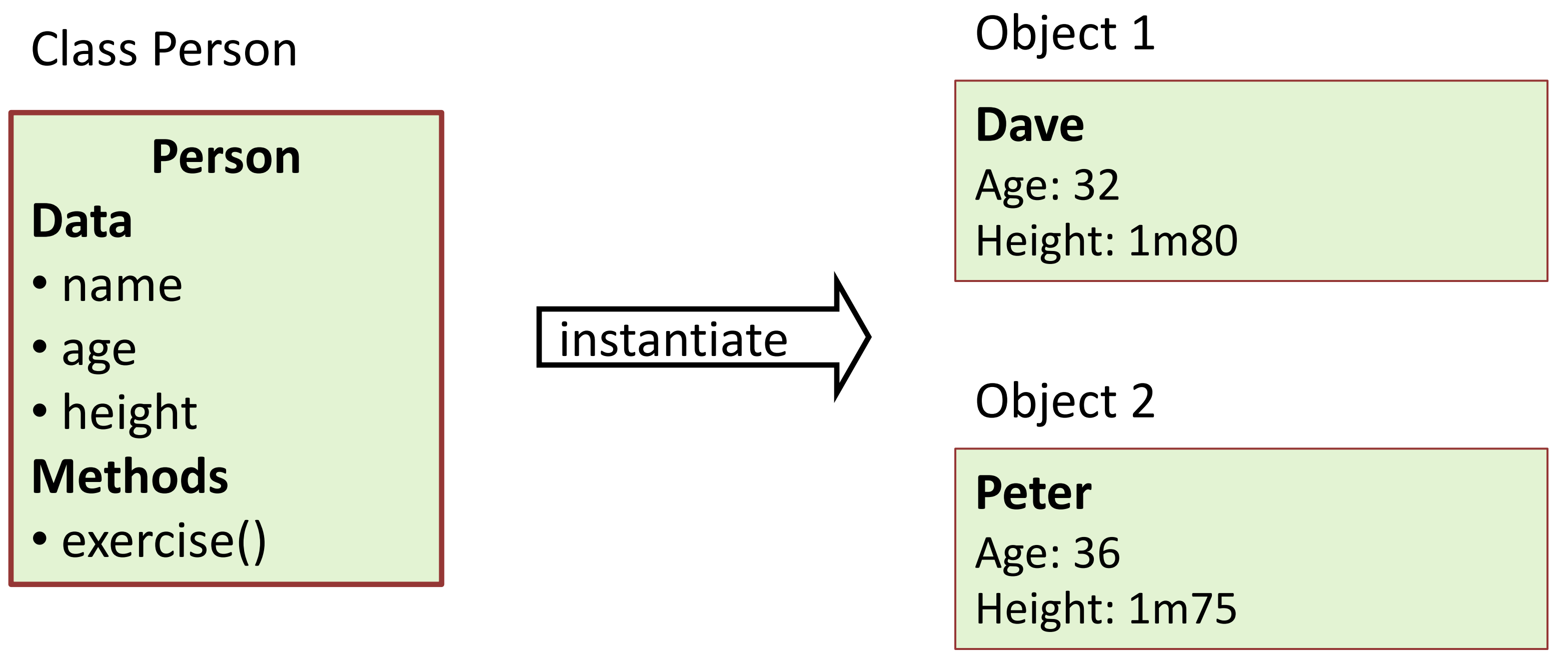
Dave
Age: 32
Height: 1m80

Person
Data
- name
- age
- height

Methods
- exercise()

Classes vs. Objects

- Each object is instantiated from a class. That object is called an instance of the class



Classes vs. Objects

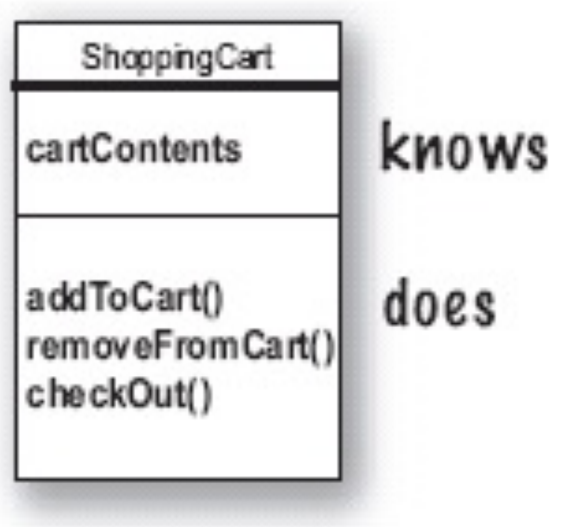
- In programming, relation between "Class and Object" is similar to relation between "Data Type and Variable"

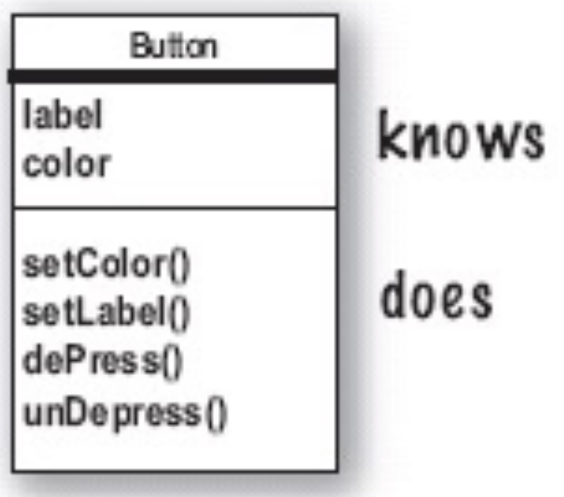
```
class Dog {

  int size;
  String breed;
  String name;

  void bark() {
   System.out.println("Ruff!");
  }
}
```

```
class Person {
  String name;
  Date birthday;
  String address;

  Dog petDog;
}
```

Designing a Class

- When you design a class, think about the objects that will be created from that class
  - things the object knows about itself
  - actions the object does





Designing a Class

- Things the object knows about itself

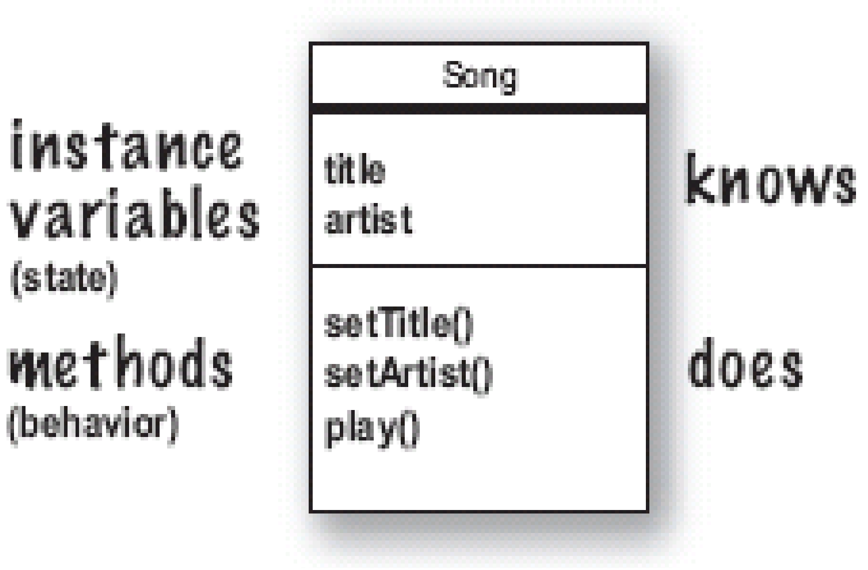
  → **instance variables**

  →represent object *state*

- Actions the object does

  → **methods**

  → represent object *behavior*



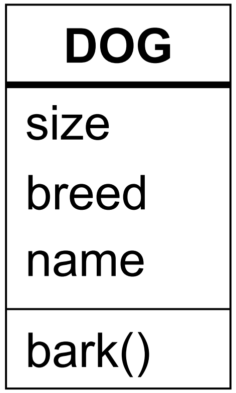
Writing a Class

1. Write the class

```
class Dog {

   int size;
   String breed;
   String name;

   void bark() {
      System.out.println("Ruff! Ruff!");
   }
}
```

instance variables

a method



Writing a Class

2. Write a tester class

```
public class DogTestDrive {
   public static void main(String [] args) {
      Dog d = new Dog();
      d.name = "Bruno";
      d.bark();
   }
}
```

dot notation (.) gives access to instance variables and methods of the object

make a Dog object

set the name of the Dog

call its bark() method

Writing a Class

- Instance variables/methods belong to an object. Thus, when accessing them, you MUST specify which object they belong to

```
public class DogTestDrive {
  public static void main(String [] args) {
    Dog d = new Dog();
    d.name = "Bruno";
    d.bark();
  }
}
```

dot notation (.) and the object reference

*access 'name' of **the Dog***

*call **its** bark() method*

Object Reference

$$\overbrace{\texttt{Dog myDog}}^{1}\ \overset{3}{=}\ \overbrace{\texttt{new Dog()}}^{2}\ ;$$

- 3 steps to declare, create & assign an object:

1. Declare a reference variable

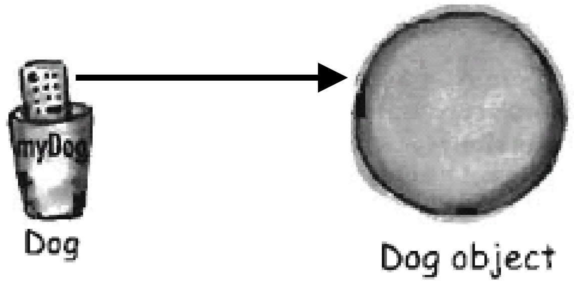
   Dog myDog = new Dog();

2. Create an object

   Dog myDog = **new Dog();**

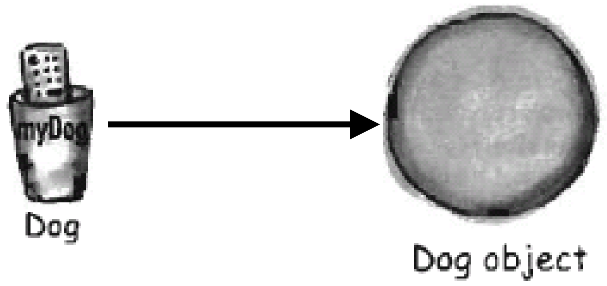
3. Link the object and the reference

   Dog myDog **=** new Dog();



Object Reference

```
Dog myDog = new Dog();
```



Note: **Reference is not object!**

Messaging between Objects

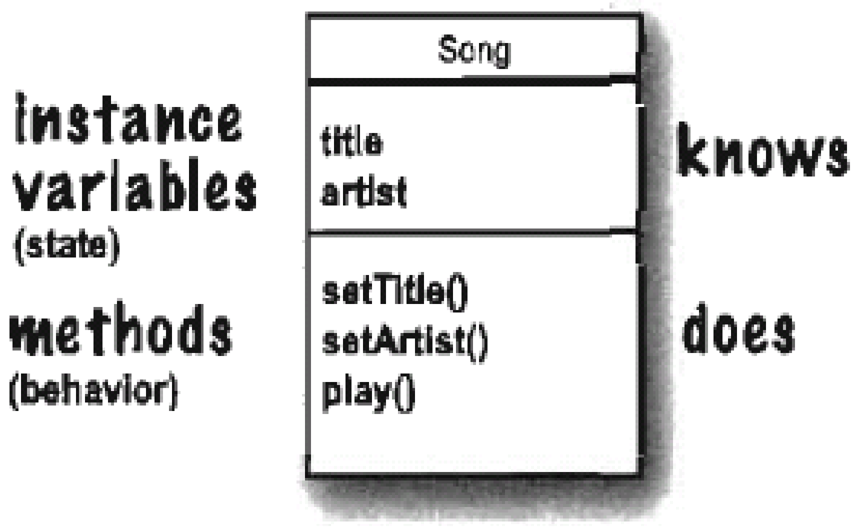
- Sending a message to an object is actually calling a method of the object

  d.bark()

- Syntax:

$$\underbrace{\text{<object\_reference>}}_{\text{recipient}}.\underbrace{\text{<method\_name>}}_{\text{message content}}(\underbrace{\text{<arguments>}}_{\text{extra information}})$$

Methods – How objects behave



- Objects have
  - state (instance variables)
  - behavior (methods)
- A method can use/change value of instance variables
  → state of the object can be changed

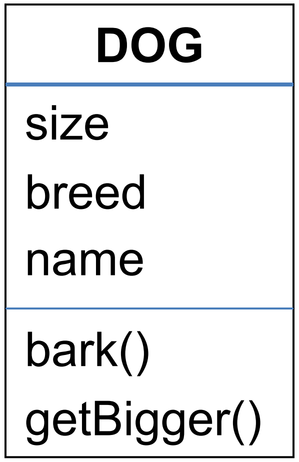
State affects behavior and vice versa

```
class Dog {

  int size;
  String breed;
  String name;

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

  void getBigger() {
    size += 5;
  }
}
```

State affects behavior:

Dogs of different sizes behave differently

method changes state

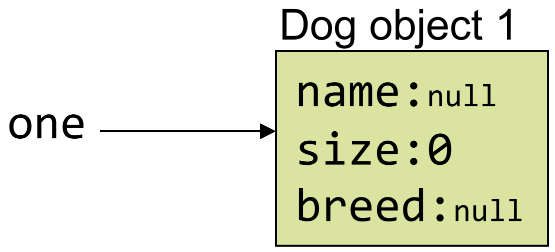


State affects behavior and vice versa

```
class DogTestDrive {

  public static void main (String[] args) {

    Dog one = new Dog();
    one.size = 7;
    Dog two = new Dog();
    two.size = 13;

    two.bark();
    two.getBigger();
    two.bark () ;

    one.bark();
  }
}
```

State affects behavior and vice versa

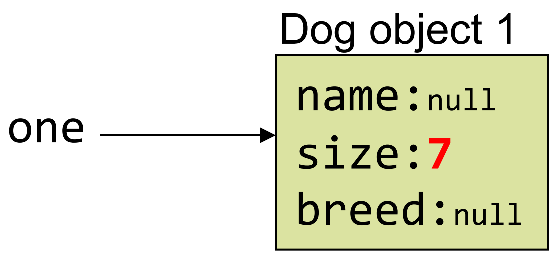
```
class DogTestDrive {

  public static void main (String[] args) {

    Dog one = new Dog();
    one.size = 7;
    Dog two = new Dog();
    two.size = 13;

    two.bark();
    two.getBigger();
    two.bark () ;

    one.bark();
  }
}
```



```
%> java DogTestDrive
```

State affects behavior and vice versa

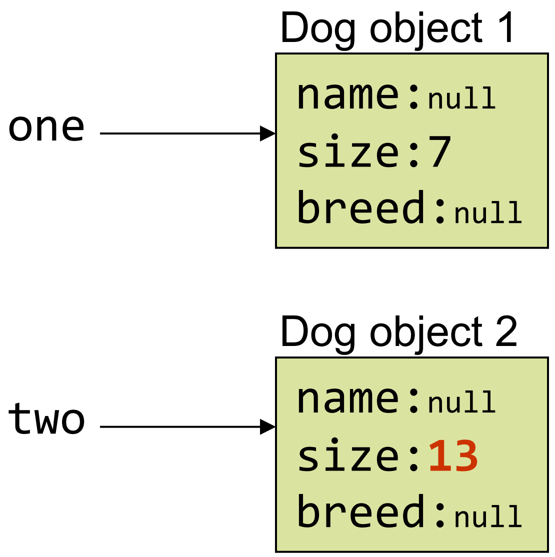
```
class DogTestDrive {

  public static void main (String[] args) {

    Dog one = new Dog();
    one.size = 7;
    Dog two = new Dog();
    two.size = 13;

    two.bark();
    two.getBigger();
    two.bark () ;

    one.bark();
  }
}
```



```
%> java DogTestDrive
```

State affects behavior and vice versa

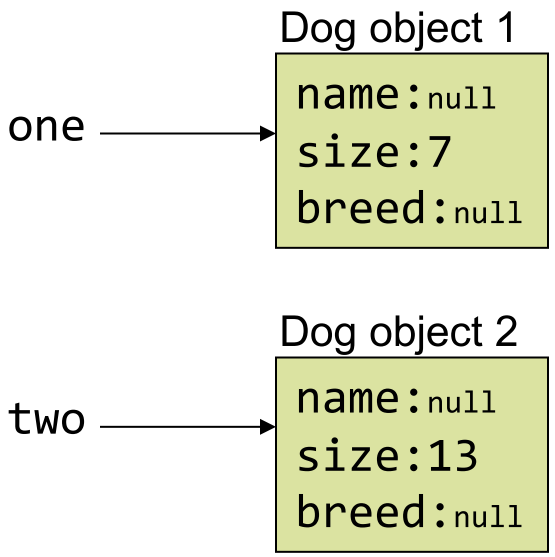
```java
class DogTestDrive {

  public static void main (String[] args) {

    Dog one = new Dog();
    one.size = 7;
    Dog two = new Dog();
    two.size = 13;

    two.bark();
    two.getBigger();
    two.bark () ;

    one.bark();
  }
}
```



```
%> java DogTestDrive
```

State affects behavior and vice versa

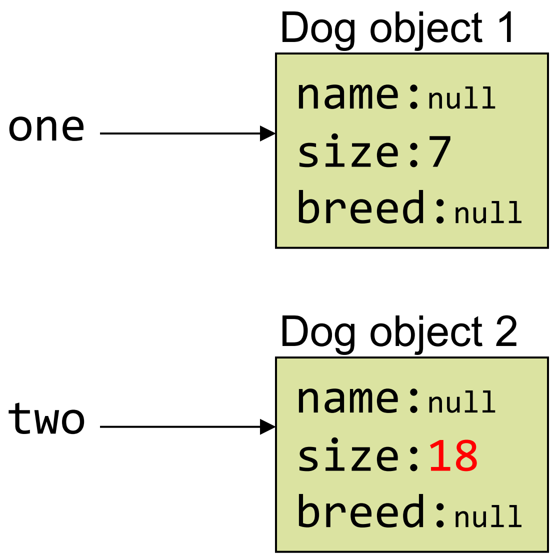
```java
class DogTestDrive {

  public static void main (String[] args) {

    Dog one = new Dog();
    one.size = 7;
    Dog two = new Dog();
    two.size = 13;

    two.bark();
    two.getBigger();
    two.bark () ;

    one.bark();
  }
}
```



```
%> java DogTestDrive

Yip! Yip!
```

State affects behavior and vice versa

```
class DogTestDrive {

  public static void main (String[] args) {

    Dog one = new Dog();
    one.size = 7;
    Dog two = new Dog();
    two.size = 13;

    two.bark();
    two.getBigger();
    two.bark () ;

    one.bark();
  }
}
```

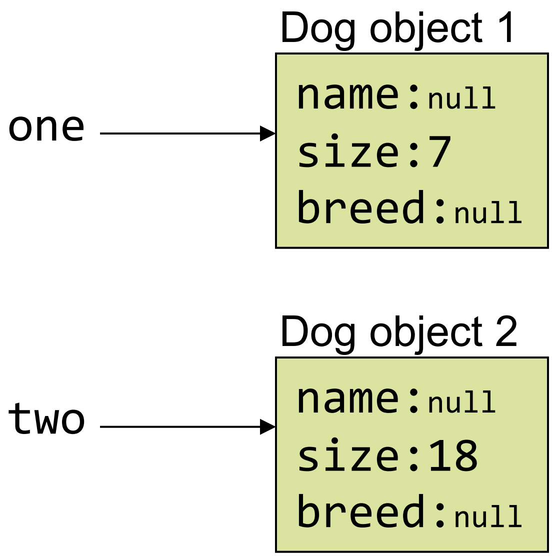


```
%> java DogTestDrive

Yip! Yip!
```

State affects behavior and vice versa

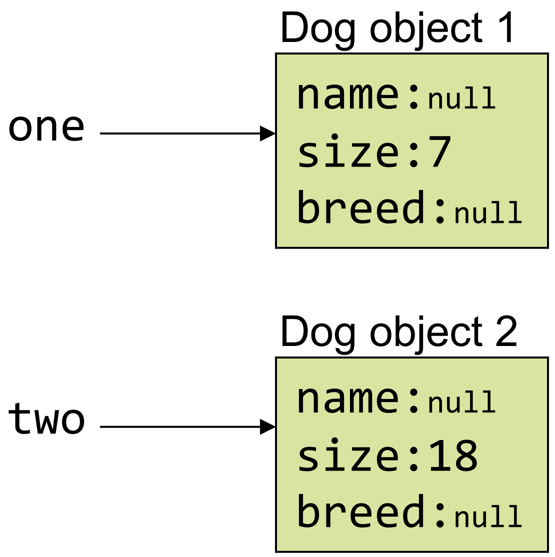
```java
class DogTestDrive {

  public static void main (String[] args) {

    Dog one = new Dog();
    one.size = 7;
    Dog two = new Dog();
    two.size = 13;

    two.bark();
    two.getBigger();
    two.bark () ;

    one.bark();
  }
}
```

```
%> java DogTestDrive
Yip! Yip!
Ruff! Ruff!
```



State affects behavior and vice versa

```
class DogTestDrive {

  public static void main (String[] args) {

    Dog one = new Dog();
    one.size = 7;
    Dog two = new Dog();
    two.size = 13;

    two.bark();
    two.getBigger();
    two.bark () ;

    one.bark();
  }
}
```



```
%> java DogTestDrive
Yip! Yip!
Ruff! Ruff!
Yip! Yip!
%>
```

Instance Variables vs. Local Variables

Instance variables

- belong to an object
- declared inside a class but NOT within a method
- have default values (0, 0.0, false, null, etc.)

```
class Dog {
  int size;
  String name;
  …
  void getBigger() {
    size += 5;
  }
}
```

Local variables

- belong to an method
- declared within a method
- MUST be initialized before use

```
public class DogTestDrive {
  public static void main(String
   [] args) {
    Dog d= new Dog();
    d.name = "Bruno";
    …
    int size = d.size;
  }
}
```

Encapsulation

- Group related things together
  - Functions encapsulate instructions
  - Objects encapsulate data and functions

❑ Bad

```
class Person {
  String name;
  Date birthday;
  String address;

  // about his/her dog
  String dogName;
  String dogBreed;
  int dogSize;
}
```

❑ Better

```
class Dog {
  int size;
  String breed;
  String name;
  ...
}
```

```
class Person {
  String name;
  Date birthday;
  String address;

  Dog petDog;
}
```

Information hiding

- Encapsulate to hide internal implementation details from outsiders:
  - Outsiders see only interfaces
  - Programmers implement details of the system

Information hiding

```
class Dog {
   int size;
   String breed;
   String name;
   ...
}
```

```
Dog d = new Dog();
d.size = -1;
```

- What’s wrong with this code?
  - It allows for a supernatural dog
    - → no verification of size
  - Object's data is exposed
    - → size is accessed directly from outsider
- Exposed instance variables can lead to invalid states of object
- What to do about it?
  - Write set methods (*setters*) for instance variables
  - Force other codes to use the set methods instead of accessing them directly

Information hiding: Rule of thumb

- Mark instance variables private
- Make getters and setters and mark them public
- Don’t forget to check data validity in setters

```
class Dog {
  private int size;

  public void setSize(int s) {
    if (s > 0) size = s;
  }

  public int getSize() {
    return size;
  }
  ...
```

Example of Encapsulation

```
public class Person {
   private String name;
   private int age;

   public int getAge() {
        return age;
   }
   public String getName() {
        return name;
   }

   public void setAge( int newAge) {
        age = newAge;
   }
   public void setName(String newName) {
        name = newName;
   }
}
```

mark instance variables private

make getters and mark them public

make setters and mark them public

Example of Encapsulation

```
public class PersonTest {
   public static void main(String args[]) {
        Person p = new Person();

        p.setName("James");
        p.setAge(20);

        System.out.println("Name: " + p.getName());
        System.out.println("Age: " + p.getAge());
   }
}
```

Set attribute values from outsider

Retrieve attribute values from outsider

Class Access Control

Access modifiers:

- public: accessible anywhere by anyone
- private: only accessible within the current class
- protected: accessible only to the class itself and to its subclasses or other classes in the same package
- default (no keyword): accessible within the current package

Implementation vs. Interface

- DogTestDrive: a "client" of Dog class
- Implementation
  - Data structures and code that implement object features
  - Usually have complex inner workings
  - Clients don't need to know
- Interface
  - The controls exposed to the "client"

- "Don't expose internal data structure to end users or client modules"