Interfaces

CSE 142, Summer 2002

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/02su/

Readings and References

- Reading
  - » Section 15.1.2, *An Introduction to Programming and Object Oriented Design using Java*, by Niño and Hosch

    skim for reference; it's not the clearest discussion ever written
- Other References
  - » The Java Tutorial on Interfaces

`http://java.sun.com/docs/books/tutorial/java/interpack/interfaces.html`

How can we manage lists of objects?

- We need a class that will let us ...
  - » add things to the list
  - » look at the elements of the list one by one
  - » find out how many things have been put in the list
  - » remove things from the list
  - » … among other things

Recall the discussion of ArrayLists

- ArrayList is a Java class that specializes in storing references to an ordered collection of things
- The ArrayList class is defined in the Java libraries
  - » part of the java.util package
- We can store *any* kind of object in an ArrayList
  - » myList.add(theDog);
- We can retrieve an object from the ArrayList by specifying its index number
  - » myList.get(0)

Casting

- In the previous lecture, we got out of the problem of the compiler not knowing what was in the ArrayList by making a *cast*
  - » We know that we've only placed String objects into the ArrayList. We can promise the compiler that the thing coming back out of the ArrayList is actually a String:

```
public void printFirstNameString(ArrayList names) {
      String name = (String)names.get(0);
      System.out.println("The first name is " + name);
}
```


PetSet example

- In the PetSet homework, we had a group of different animals that we wanted to operate with

```
public void dine() {
      aCat.eat(2*aCat.getMealSize());
      aDog.eat(2*aDog.getMealSize());
      aBird.eat(2*aBird.getMealSize());
}
```

- Can we keep track of the objects in a more general way using ArrayLists?
  - » yes, we can, but there are just a few little details ...

Using ArrayLists : add String

- With Strings, we did this

```
ArrayList names = new ArrayList();
names.add("Billy");
```

- `add(Object o)` method adds an object to the list at the end of the list
- The object can be of any class type
  - » String, Dog, Rectangle, …

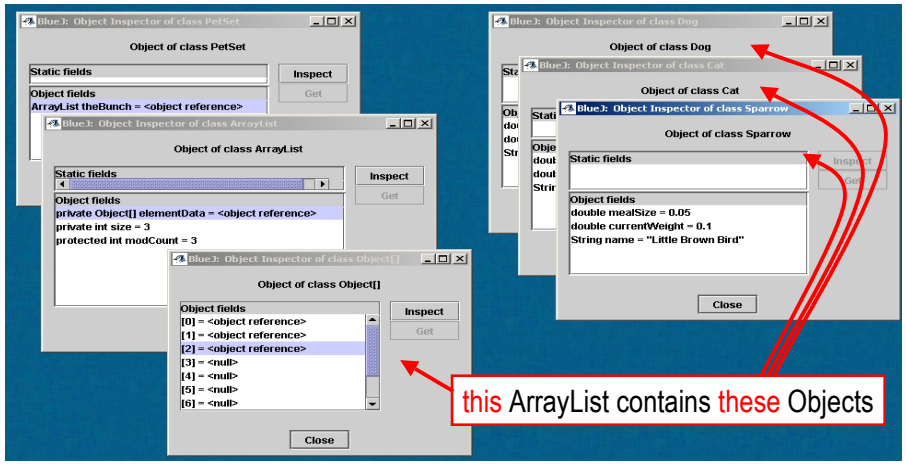
Using ArrayLists : add Dog and Cat

- With Dogs and Cats, we can do the same thing

```
public PetSet() {
      theBunch = new ArrayList();
      theBunch.add(new Cat("Smoky",0.1,5));
      theBunch.add(new Dog("Fred"));
      theBunch.add(new Sparrow("Little Brown Bird"));
   }
```

- `add(Object o)` method adds an object to the list at the end of the list
- So we got them in to the list okay ...

example PetSet ArrayList



Cast to what?

- With Strings, we promised that the Object from the ArrayList was actually a String:

```
public void printFirstNameString(ArrayList names) {
      String name = (String)names.get(0);
      System.out.println("The first name is " + name);
}
```

- But what can we do now?
  - » One Object is a Cat
  - » another Object is a Dog
  - » and another is a Sparrow

Recall the definition of casting

- The pattern is
  - » (<class-name>)<expression>
- For example

  String name = (String)names.get(0);
- Casting an object does ***not*** change the type of the object
- A cast is a promise by the programmer that the object can be used to represent something of the stated type and nothing will go wrong

What can we say about an animal?

- What we need is something that we can say that is true of all the various animals that we have created
- They all have eat(), sleep(), getMealSize(), and a voice of some sort
- So we *can* promise that:
  - » We don't know exactly what kind of an animal it is, but we do know that it can eat, sleep, make a noise, and tell you its meal size

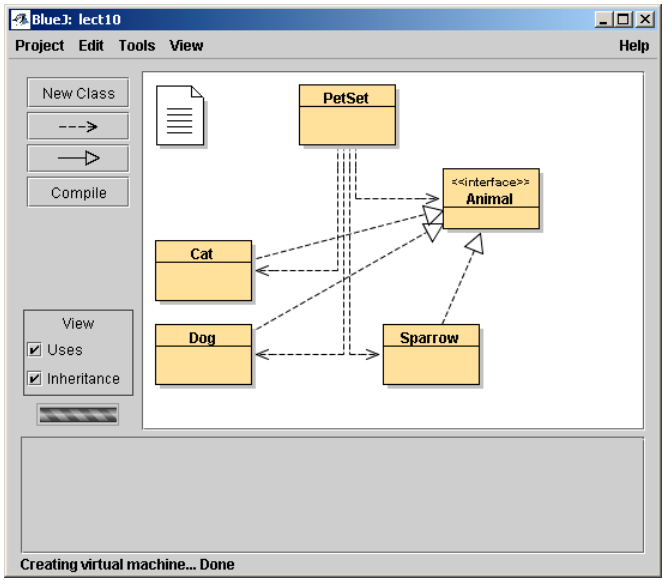
an Interface

- Java has a very cool mechanism for this
  - » an interface
- You can say that any class that claims to be an Animal will guarantee that it has methods for all the things that any Animal must do
- The definition of the interface shows exactly what the methods must look like to the public
  - » the actual implementation is not in the interface

public interface Animal

```
/**
 * This interface specifies the behavior that a class
 * must implement in order to be considered a real Animal.
 *
 */
public interface Animal {
   /**
    * Provide this animal with a way to rest when weary.
    */
    public void sleep();
    /**
    * Eat some goodies.  There is some weight gain after eating.
    * @param pounds the number of pounds of food provided.
    */
    public void eat(double pounds);
    /**
     * get the meal size defined for this animal.
     * @return meal size in pounds
     */
    public double getMealSize();
    /**
    * Provide this animal with a voice.
    */
    public void noise();
}
```


The Animal interface



using an interface in a class definition

- Each of the classes that wants to be considered an Animal must say so at the very beginning of the class definition

```
public class Dog implements Animal {...
public class Cat implements Animal {...
public class Sparrow implements Animal {...
```

- You are telling the compiler that this class guarantees that it will implement all the methods that are required in the interface

what is the guarantee?

```
/**
 * This interface specifies the behavior that a class
 * must implement in order to be considered a real Animal.
 *
 */
public interface Animal {
    /**
     * Provide this animal with a wa
     */
    public void sleep();
    /**
     * Eat some goodies.  There is some weight gain after eating.
     * @param pounds the number of pounds of food provided.
     */
    public void eat(double pounds);
    /**
      * get the meal size defined for this animal.
      * @return meal size in pounds
      */
    public double getMealSize();
    /**
     * Provide this animal with a voice.
     */
    public void noise();
}
```

No problem. We already have these in every animal we've written so far.

Small problem. Each animal so far has had a different voice.

conform to expectations ...

- Rewrite each of the animal classes to use the same method name when they make their noise

```
public class Dog implements Animal {
...
  /**
   * Provide this animal with a voice.
   */
   public void noise() {
       System.out.println(name+" : Woof! Woof!");
   }


public class Cat implements Animal {
...
  /**
   * Provide this animal with a voice.
   */
   public void noise() {
       System.out.println(name+" : Meow! Meow!");
   }
```


using the Animal interface in PetSet

- Now we know that all of the animals will satisfy the Animal interface, no matter what kind of object they are
- So PetSet can guarantee that they are Animals being retrieved from the ArrayList, no matter what else they might be

Cast to Animal

- Tell the compiler that the ArrayList contains objects that are Animals

```
public void dine() {
    for (int i=0; i<theBunch.size(); i++) {
        Animal pet = (Animal)theBunch.get(i);
        double s = pet.getMealSize();
        pet.eat(2*pet.getMealSize());
    }
}
```

Cast to Animal and use the result

- The cast can be buried right in the usage
  - » you don't have to declare a local variable first
  - » but many times it is clearer to use local variables instead of trying to do everything at once

```
public void sleep() {
    for (int i=0; i<theBunch.size(); i++) {
        ((Animal)theBunch.get(i)).sleep();
    }
}
```


The Actor interface

- We are already using interfaces in our code

```
public class OvalSlider implements Actor {...
```

- OvalSlider guarantees that it has implemented the methods in the Actor interface

`void addTo(uwcse.graphics.GWindow w)`
Every Actor must be able to draw itself on a GWindow.

`void doAction(Stage stage)`
Every Actor must implement some fundamental action.

`void removeFromWindow()`
Every Actor must be able to remove itself from its GWindow.

The Shape interface

- From OvalSlider.java

```
/** the Shape that we are moving around on the screen */
private Shape theShape;
```

- OvalSlider doesn't care about the special characteristics of an Oval, it only cares that an Oval can do the things that a good Shape should be able to do

```
void addTo(GWindow gw)
Rectangle getBoundingBox()
int getCenterX()
int getCenterY()
java.awt.Color getColor()
int getHeight()
int getWidth()
int getX()
int getY()
uwcse.graphics.InternalGWindow currentWindow()
boolean intersects(Shape other)
void moveBy(int deltaX, int deltaY)
void moveTo(int x, int y)
void paint(java.awt.Graphics g)
void recordWindow(uwcse.graphics.InternalGWindow gw)
void removeFromWindow()
void rotateAround(int pivotX, int pivotY, double degrees)
void setColor(java.awt.Color c)
```