ArrayLists

CSE 142, Summer 2002

Computer Programming 1

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

Readings and References

- Reading
  - » Chapter 14 and 17, *Introduction to Programming in Java*, Dugan
- Other References
  - » The Java Tutorial on Collections, by Joshua Block
    `http://java.sun.com/docs/books/tutorial/collections/`
  - » Josh Block is also the author of the Java code that implements Collections in the Java libraries

Collections in the Real World

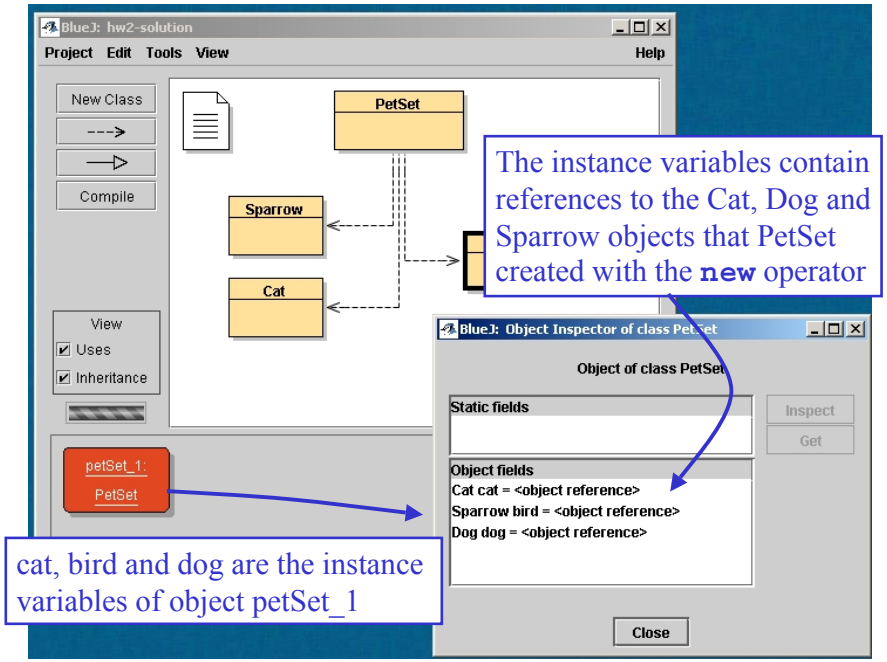
- Think about:
  - » words in a dictionary
  - » list of pets in your household
  - » deck of cards
  - » books in a library
  - » songs on a CD
- These things are all *collections*.
- Some collections are *ordered*, others are *unordered*

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

PetSet example

- Think about PetSet in homework 2
  - » There were two animal objects in the distributed version of PetSet
  - » You designed a new type of animal, and then created at least one new object of this new type
  - » In order to manage the activities of the new animal you had to change the source code in PetSet
- Changing source code in order to implement variations in the data set is costly and inflexible



PetSet example

- Changing source code in order to implement variations in the data set is costly and inflexible

```
public void dine() {
        cat.eat(cat.getMealSize()*2);
        bird.eat(bird.getMealSize()*2);
        dog.eat(dog.getMealSize()*2);
    }
```

- It would be nice if we could somehow keep track of the objects in a more general way

An Ordered Collection: ArrayList

- ArrayList is a Java class that specializes in representing 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)

ArrayList

- **`ArrayList()`**
  - » This constructor builds an empty list with an initial capacity of 10
- **`int size()`**
  - » This method returns the number of elements in this list
- **`boolean add(Object o)`**
  - » This method appends the specified element to the end of this list
- **`Object get(int index)`**
  - » This method returns the element at the specified position

Using ArrayLists

- ArrayList is part of the java.util package
  - » `import java.util.*;` to use ArrayList
- Creating a list

```
ArrayList names = new ArrayList ( );
```

- Getting the size

```
int numberOfNames = names.size( );
```

- Adding things

```
names.add("Billy");
names.add("Susan");
names.add("Frodo");
```

NameList.java

Using ArrayLists : import

- ArrayList is part of the java.util package
  - » `import java.util.ArrayList;` to use ArrayList
- The import statement tells the Java compiler where to look when it can't find a class definition in the local directory
  - » We defined Cat, Dog, Sparrow but not ArrayList
  - » We tell the compiler to look in package java.util for the definition of ArrayList by putting an import statement at the top of the source code file
  - » Java always looks in package java.lang on its own

Using ArrayLists : constructor

- Creating a new ArrayList object

```
ArrayList names = new ArrayList ( );
```

- There are several constructors available
  - » **`ArrayList()`**
    Construct an empty list with an initial capacity of 10
  - » **`ArrayList(int initialCapacity)`**
    Construct an empty list with the specified initial capacity
  - » **`ArrayList(Collection c)`**
    Construct a list containing elements from another collection

Using ArrayLists : size

- Getting the size

```
int numberOfNames = names.size( );
```

- `size()` method returns integer value that caller can use to control looping, check for limits, etc
  - » This is similar to the getMealSize() method that you had in your animal object
  - » The object keeps track of relevant information, and can tell the caller when there is a need to know

Using ArrayLists : add

- Adding things

```
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, …
  - » can't add "primitive" types like int or double

So now what?

- We can create a list, and we can add items to it.
- But we need to get them out, too!
- Use the `get(int index)` method to retrieve references to objects in the ArrayList

```
String tag = (String)names.get(0);
```

- But there are just a few little details to be worked out ...

indexed access to elements

- ArrayLists provide *indexed* access
  - » We can ask for the i^{th} item of the list, where the first item is at index 0, the second at index 1, and the last item is at index n-1 (where n is the size of the collection).

```
ArrayList names = new ArrayList ( );
names.add("Billy");
names.add("Susan");
names.get(0)
names.get(1)
```

A Problem

- We want to get things out of an ArrayList
- We might write the following:

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

- But BlueJ complains at the green line:
  - » incompatible types:
  - » found: Object
  - » required: String

Object

- The return type of the method get() is Object.
- Think of Object as Java's way of saying "any type of class"
- *All* classes in Java have an "is-a" relationship to Object. In other words:
  - » every String is an Object
  - » every Rectangle is an Object
  - » every ArrayList is an Object
- Object is the "mother of all classes"

Making Promises: Casting

- The solution to our get() problem is to make a promise
  - » 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);
}
```

- This promise is called a *cast*.

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

Miscasting

- We can lie about casting, but it will be caught at runtime

```
public void printFirstNameString(ArrayList names) {
    String name = (String)names.get(0);
    System.out.println("The first name is " + name);
    Oval ovoid = (Oval)names.get(1);
}
```

this will fail when you run the program

Reference vs. Primitive Types

- A few Java types are *primitive*:
  int, double, boolean, and a few other numeric types we haven't seen
  - » Are atomic chunks with no parts (no instance variables)
  - » Exist without having to be allocated with new
  - » Cannot be message receivers, but can be arguments of messages and unary and binary operators
- All others are *reference types*:
  Rectangle, BankAccount, Color, String, etc.
  - » Instances of the class are created using "new"
  - » Can have instance variables and methods
  - » All are special cases of the generic type "Object"