ArrayLists

CSE 142, Summer 2003

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/03su/

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
- Simple arrays are one way to do this
- ArrayLists are another way to do this

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);
  - » but not primitive types like int, double, or boolean
- 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 and increases the size of the array if needed
- **`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( );`
- Adding things
  - `names.add("Billy");`
  - `names.add("Susan");`
  - `names.add("Frodo");`
- Getting the size
  - `int numberOfNames = names.size( );`

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 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
  - » Design pattern: 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, File, InputStream, …
  - » can't add "primitive" types like int or double directly
    - Can use the wrapper classes like Integer to store primitives

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 ...

Using ArrayLists: get

- 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");
Object x = names.get(0);
Object y = 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 the compiler complains at the green line:
  - » incompatible types:
  - » found : java.lang.Object
  - » required: java.lang.String

Object (as the name of a class)

- 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”

Object (as the name of a class)

- This is a new usage of the word Object for us
- "Object" refers to a class named Object
  - » "Object" is a class name with a capital "O"
- "object" refers to a little blob of memory, an instance of some class that was created with the new operator
  - » an "object" is a thing that was created at run time
- These are two entirely different meanings

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

The Collections Class

- There is a class called java.util.Collections
  - » utility functions for using classes that implement the Collection interface
  - » This class consists exclusively of static methods that operate on or return collections. It contains polymorphic algorithms that operate on collections, "wrappers", which return a new collection backed by a specified collection, and a few other odds and ends.
  - » These are **static** methods so they exist and can be used without creating an object first

Useful methods in Collections class

- static void sort(List list)
  - » Sorts the specified list into ascending order, according to the natural ordering of its elements.
  - » "natural order" is defined when you implement the interface Comparable
- static void sort(List list, Comparator c)
  - » Sorts the specified list according to the order induced by the specified comparator
  - » Comparator lets you define several different orders