
CSE 142

Introduction to Collections – ArrayLists

1/18/2003

(c) 2001–3, University of Washington

L-1

Outline for Today

- Collections of data
- APIs
- ArrayLists
- Technicalities
 - Objects
 - casts
 - reference vs primitive types

1/18/2003

(c) 2001–3, University of Washington

L-2

Collections in the Real World

- Think about:
 - words in a dictionary
 - list of students in a class
 - deck of cards
 - books in a library
 - MP3 files on a computer
- These things are all collections
- They contain multiple instances of like objects
- Some collections are ordered, others are unordered

1/18/2003

(c) 2001–3, University of Washington

L-3

Collections in Some Familiar Models

- Bank, BankAccount
- Student, Registration
- Soap Company
- Cell Phone, Media Player

1/18/2003

(c) 2001–3, University of Washington

L-4

Some Common Types of Collections

- Collections may be ordered or unordered
- Some collections are “sets”
 - no inherent order
 - duplicate elements not allowed
- A very common collection type is a list
 - Elements in a list are in a definite order, one after another

1/10/2003

(c) 2001-3, University of Washington

L-5

Collections and Libraries in Java

- The Java language does not have special keywords or syntax for collections
- Collections and lists are available in Java programs through *class libraries* that are part of every Java implementation
- There are standard Java class libraries for dozens or hundreds of purposes
 - Math
 - Graphics
 - Networking
 - Files
 - Collections
 - etc., etc.

1/10/2003

(c) 2001-3, University of Washington

L-6

More About APIs

- The phrase API (application programming interface) is commonly used to designate a set of classes and methods
- To be an effective Java programmer, you must use APIs!
- Must learn *how* to use them
 - What to expect
 - Requirements and conventions of programming
 - Conventions of documentation
- Must learn *specifics* of particular APIs
 - Which classes and methods are available
 - The internal model of the application
- A long-range goal of 142/143 is to make you confident about using APIs

1/10/2003

(c) 2001-3, University of Washington

L-7

An Ordered Collection in Java: ArrayList

- ArrayList is a Java class whose instances store an ordered collection of things
- ArrayList is one of a number of standard Java library classes for collections
- You can add objects to an ArrayList object and get them back out
- No limit to the length of a list

1/10/2003

(c) 2001-3, University of Washington

L-8

Some ArrayList Methods

- The specification for ArrayList tells us what methods are available. A few of the methods:

```
public class ArrayList
// Create an empty collection
public ArrayList();

// Add the given object to the end of this collection
public void add(Object o);

// Return the size of this collection
public int size();
...
```

- New: **Object** type – means any kind of object at all

1/10/2003

(c) 2001-3, University of Washington

L-9

Using ArrayLists

- Creating a list: ArrayList is a class, and we need an instance of the class (object) to store data:

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

- Adding things:

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

- Getting the size:

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

- If you try typing the above into Dr. Java... it won't work!

1/10/2003

(c) 2001-3, University of Washington

L-10

The *import* Statement

- ArrayList is *not* a keyword of Java
- Any classes not defined in your own program must be *imported*
- The *import* statement tells the compiler which library or external classes you want to use
- ArrayList is in a "package" called *java.util*
- Write *import java.util.*;* to use classes of the *java.util* package
- All *import* statements must be at the beginning of the java file
 - In DrJava's interaction window, you can type them anytime

1/10/2003

(c) 2001-3, University of Washington

L-11

ArrayList Diagram

1/10/2003

(c) 2001-3, University of Washington

L-12

More ArrayList Methods

- Here's more of its interface:

```
public class ArrayList {  
    ...  
    // Return the object at the given index (numbered starting from 0, not 1!).  
    // Raise an exception if index is out-of-bounds.  
    public Object get(int index);  
  
    // Change the object at the given index (starting from 0) to be newElement.  
    // Raise an exception if index isn't in bounds.  
    // Return the element that used to be there.  
    public Object set(int index, Object newElement);  
}
```

1/10/2003

(c) 2001-3, University of Washington

L-13

Using Indexes with ArrayLists

- ArrayLists provide *indexed* access. We can ask for a particular item in the list by its position or *index* number
- 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");
```

- Java expressions:

```
names.get(0)  
names.get(1)
```

1/10/2003

(c) 2001-3, University of Washington

L-14

A Problem

- Let's say we want to get something out of an ArrayList and assign it to a variable

- We might write the following:

```
String name = names.get(0);  
System.out.println("The first name is " + name);
```

- But Java complains about the green line:

"incompatible types: found: Object, required: String"

(DrJava's interactions window allows this without complaining, even though it's not legal in regular Java)

- Why? [Hint: look at the interface of the get method]

1/10/2003

(c) 2001-3, University of Washington

L-15

Problem: *Object*

- The return type of method *get()* is *Object*.
- Think of *Object* as Java's way of saying "any type".
- All classes in Java (including the ones we write) 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
- The reverse is not generally true!
 - every *Object* is not necessarily a String

1/10/2003

(c) 2001-3, University of Washington

L-16

Making False Claims

- Looks weird, but is legal...
`Object someObject = new Soap(. . .);`
... because every Soap is an *Object*.
- In our example:
`String name = names.get(0);`
`System.out.println("The first name is " + name);`
- We are claiming that an Object (the result of get) is a String, which is not necessarily true!
 - What if we passed an ArrayList of Soap to printFirstName?

1/10/2003

(c) 2001-3, University of Washington

L-17

Making Promises: Casting

- It looks like we're stuck. We can add things to the collection, but we can't get them back out!
- The solution is to make a promise
 - Say 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
`String name = (String)names.get(0);`
`System.out.println("The first name is " + name);`
 - This is (another use of) a cast

1/10/2003

(c) 2001-3, University of Washington

L-18

Casting (Review)

- Pattern:
`(<class-name>)<expression>`
- Example:
`String name = (String)names.get(0);`
- Casting does *not* change the type of the object. It is a promise that the object really is of the stated type.
- Casting also used for conversions, as we've seen.
`(int) 3.1415927`

1/10/2003

(c) 2001-3, University of Washington

L-19

Miscasting

- We can abuse casting, but it will be caught at runtime.
`String name = (String)names.get(0);`
`System.out.println("The first name is " + name);`
`Rectangle box = (Soap)names.get(0); // Run time error!!`
`System.out.println("The length is " + box.getLength());`
- An error called a "class cast exception" occurs if a promise is broken.
 - Footnote: "Exceptions" are one way that programs signal that something unexpected or undesirable has occurred (CSE143 topic)

1/10/2003

(c) 2001-3, University of Washington

L-20

Reference vs. Primitive Types

- A few Java types are primitive
 - int, double, char, boolean, and a few other numeric types we normally won't use
- Are atomic chunks, with no parts (i.e., no instance variables)
- Exist without having to be allocated with new
- Cannot receive messages (i.e., do not have methods) but can be arguments of messages and unary and binary operators
- All others are reference types
 - Rectangle, BankAccount, Color, String, etc.
- Instances of some class
- Created by new
- Can have instance variables and methods
- All are special cases of the generic type "Object"

1/10/2003

(c) 2001-3, University of Washington

L-21

When Does the Distinction Matter?

- One place: when putting values in collections

```
ArrayList list = new ArrayList();
list.add(5);
```

// error: int isn't an Object
- Solution (if we really need to do this): create a *wrapper* object containing the primitive value. There is a wrapper class for each primitive type, e.g. Integer, Double.

```
ArrayList list = new ArrayList();
Integer five = new Integer(5); // create an Integer object with a 5 in it
list.add(five);
```

// ok: Integer is an Object
...

```
Integer firstElem = (Integer) list.get(0); // promise that the Object is an Integer
int v = five.intValue();
```

// extract the int value from the Integer object

1/10/2003

(c) 2001-3, University of Washington

L-22

Summary

- Collections: Many kinds
 - Common in computer programs
 - Often correspond to collections of objects in the real world
- A simple collection: ArrayList
 - Sequential, ordered collection
 - Part of the *java.util* package of classes
 - Many methods: add, get, size, isEmpty, ... (see Sun Java Docs)
 - import java.util.*; to access
- Casts
 - Often needed to specify actual type of object retrieved from a collection (since collection can hold any kind of object)
- Primitive vs. reference types: need to place primitive values in wrapper objects if we want to store them in a collection

1/10/2003

(c) 2001-3, University of Washington

L-23