

Intro to Programming II

ArrayLists

Chris Brooks

Department of Computer Science
University of San Francisco

Department of Computer Science — University of San Francisco — p. 1/77

8-2: The ArrayList class

- Last time, we learned about how to use arrays in Java.
- Java also provides an ArrayList class that can help manage arrays.
- ArrayList is a *generic* container.
 - That means that we can use the same container to store different kinds of elements.

Department of Computer Science — University of San Francisco — p. 2/77

8-3: The ArrayList class

- An example:

```
ArrayList band = new ArrayList(3);
band.add("john");
band.add("paul");
band.add("george");
band.add("ringo");
```

Department of Computer Science — University of San Francisco — p. 3/77

8-4: The ArrayList class

- Notice that we declared an initial array size.
- The array is then able to grow *dynamically* beyond that.
 - It's better to allocate in advance if we can.
- We can also remove elements, and access them.
- We can also add in the middle of a list: `band.add(1, "ringo");`

Department of Computer Science — University of San Francisco — p. 4/77

8-5: Comparison with Arrays

- With arrays, we'd need to allocate a new array and copy everything over. (yuck!)

```
int []array1 = new int[5];
for (i = 0; i < 5; i++) {
    array1 = i;
}
int []array2= new int[10];
for (i = 0; i < 5; i++) {
    array2[i] = array1[i];
}
array1 = array2;
```

Department of Computer Science — University of San Francisco — p. 5/77

8-6: Accessing elements

- `get(index)` lets us access the element at a particular index.
- Elements in an ArrayList are stored as *Objects*.
- This means that we need to cast them back to Strings.
- Can't store primitives.

```
String name = (String)band.get(2);
```

Department of Computer Science — University of San Francisco — p. 6/77

8-7: Exercise 1

- Redo the Student exercise from Monday using an ArrayList.
 - Prompt the user for a number of students.
 - Store the students in an ArrayList.
 - Print them out in order.

8-8: Iterators

- It can be cumbersome to iterate through a list using integers.
- What if the size of the list changes while we're iterating over it?
- Why do we need to know the size of the array to do something to each element?
- Java provides an *abstraction* for us known as an iterator.

8-9: Example

```
List myList = ArrayList(10);
// fill in the list

ListIterator i = myList.ListIterator();
while (i.hasNext())
    System.out.println(i.next());
```

8-10: Exercise 2

- Modify the previous code to use a ListIterator

8-11: Finding elements

- We can use indexOf to find where an element is located.
- remove lets us remove things.

```
int index = band.indexOf('ringo')
band.remove(index)
```

8-12: Exercise 3

- Add a method that allows the user to enter the name of a student.
- Loop through the array to find that student and remove them.
- Now try doing it with indexOf

8-13: Other methods

- also:
 - `size()` - tells us how many elements are in the array.
 - `isEmpty()` - tells us whether there's anything in the list.
 - `contains(Object o)` - tells us whether an object is in the list.
 - `clear()` - removes everything.

8-14: Specifying element types

- If we want to avoid casting elements, we can also define the `ArrayList` to accept particular sorts of objects:
 - Java 1.5 only ...

```
ArrayList<String> al = new ArrayList<String>(3);
al.add("john");
al.add("paul");
al.add("george");
al.add("ringo");

String name = al.get(2);
System.out.println(name);
```

8-15: Exercise 4

- Change your code to use the template notation, so that we don't have to cast everything anymore.

8-16: ArrayList vs arrays

- `ArrayLists` manage inserting, searching, and removing for you.
- Can grow dynamically.
- Only work with objects, not primitives.
- Provides some nice convenience methods.