

---

## CSE 142

### Arrays Implementing Lists with Arrays

5/11/2003

(c) 2001-3, University of Washington

P-1

---

## Outline for Today

- Quick Review – collection classes: ArrayList
- Arrays – a low-level collection-like data structure
- Using arrays to implement higher-level collection classes

5/11/2003

(c) 2001-3, University of Washington

P-2

---

## What is an ArrayList?

- ArrayLists are fairly sophisticated objects
  - Contain 0 or more objects
  - Can add new objects to the collection
  - Can delete objects from the collection
  - Can find objects in the collection
  - Can iterate through the collection
- How is this implemented?
  - We've already gotten some idea from drawing the pictures...

5/11/2003

(c) 2001-3, University of Washington

P-3

---

## Arrays

- Java (and many other languages) include arrays as the most basic kind of collection
  - Simple, ordered collections, similar to ArrayLists
  - Special syntax for declaring values of array type
  - Special syntax for accessing elements by position

5/11/2003

(c) 2001-3, University of Washington

P-4

## Java Arrays vs. Everyday Arrays

- In everyday use, an array is a regular arrangement, usually rectangular



- In programming terms, these are two-dimensional arrays
- One-dimensional arrays are very common in programming
- and that's all we consider for now



5/11/2003

(c) 2001-3, University of Washington

P-5

## Arrays vs ArrayLists

- Array: The size is fixed when the array is created
- ArrayList: size is open-ended
  - "Always room for one more"
- Arrays: no iterator; must use explicit indexes and for/while loops
- Arrays: all elements are of same type
- ArrayLists: elements can be of different types
- Which is which:



5/11/2003

(c) 2001-3, University of Washington

P-6

## Array Example

```
String[] pets = new String[3];
```

```
pets[0] = "Sally";  
pets[1] = "Puff";  
pets[2] = "Spot";
```

```
pets[1] = "Rex";
```

```
String allMyPets = "";  
for (int i = 0; i < pets.length; i++) {  
    allMyPets = allMyPets + " " + pets[i];  
}
```

5/11/2003

(c) 2001-3, University of Washington

P-7

## Array Declaration and Creation

- Array have special type and new expression syntax:  
`<element type>[] <array name> = new <element type> [ <length> ];`
- Arrays can only hold elements of the specified type.
  - No clunky casts needed!
  - Unlike ArrayList etc., element type can be int, double, boolean, etc., as well as any kind of object
- `<length>` is any integer expression.
  - Doesn't need to be a constant
  - Value should be greater than 0
- Elements of newly created arrays initialized to null (zero, false).
- Arrays have an instance variable, `length`, that stores their length.  
`<array name> . length`

5/11/2003

(c) 2001-3, University of Washington

P-8

## Array Initialization

- Variable initialization review:
  - Instance variables are initialized automatically
    - Objects: null
    - Numbers: 0
    - Booleans: false
  - Local variables are *not* initialized automatically
- An array variable per se follows the rules above
- For array *elements* the rules have a slight twist:
  - Elements of newly created arrays initialized to null, zero, or false --- regardless of whether the array variable is instance or local

5/11/2003

(c) 2001-3, University of Washington

P-9

## Array Element Access

- Access an array element using the array name and desired position  
`<array name> [ <position> ]`
- Details:
  - *<position>* is an integer expression
  - Positions count from 0, as with ArrayLists
  - Type of result is the element type of the array (not necessarily Object)
- Can update an array element by assigning to it  
`<array name> [ <position> ] = <new element value> ;`
- Like ArrayList's set method

5/11/2003

(c) 2001-3, University of Washington

P-10

## Extended Example: Class *StringList*

- Arrays can be used to implement an ArrayList-like class that holds a list of Strings.

- Specification:

```
class StringList {           // a list of strings
    StringList(int capacity); // create new StringList with given capacity
    int size();              // return # of Strings in this StringList
    boolean add(String str); // add str to this StringList, result true if success
    boolean contains(String str); // return whether this StringList contains str
    void clear();            // remove all strings from this StringList
    String get(int pos);     // return String at given position
    String set(int pos, String newStr); // update String at given position, and
                                     // return previous String at that position
}
```

- For simplicity, we'll use a fixed maximum capacity.

5/11/2003

(c) 2001-3, University of Washington

P-11

## StringList Representation

- Underlying representation is an array of String objects
- Need separate variable to keep track of how many Strings have actually been added to the collection so far (Why?)

```
public class StringList {           // a list of strings
    // instance variables:
    private String[] strings;        // Strings in this StringList are stored in
    private int numStrings;          // strings[0] through strings[numStrings-1]
    ...
}
```

- Array length gives the *capacity* of the StringList container; `numStrings` is the container's current *size*
- Class invariant relates instance variables – essential comment

5/11/2003

(c) 2001-3, University of Washington

P-12

## StringList Constructor

- Need to allocate the actual array and initialize the StringList to "empty"

```
public class StringList {           // a list of strings
    private String[] strings;      // Strings in this StringList are stored in
    private int    numStrings;     // strings[0] through strings[numStrings-1]
    /** Construct new empty StringList with given maximum capacity */
    public StringList(int capacity) {

    }
}
```

5/11/2003

(c) 2001-3, University of Washington

P-13

## add

```
/** Add str to this StringList if there is room. Return true if added successfully,
 * otherwise return false. */
public boolean add(String str) {

}
```

5/11/2003

(c) 2001-3, University of Washington

P-14

## size

```
/** Return number of elements currently in this StringList */
public int size() {

}
```

5/11/2003

(c) 2001-3, University of Washington

P-15

## contains

```
/** Return whether this StringList contains str (testing using equals) */
public boolean contains(String str) {

}
```

5/11/2003

(c) 2001-3, University of Washington

P-16

## get

```
/** Return the string at position pos in this StringList, or null if pos is out of bounds */  
public String get(int pos) {
```

```
}
```

5/11/2003

(c) 2001-3, University of Washington

P-17

## set

```
/** update String at given position to be newStr, and return previous String at that  
 * position, or return null if pos is out of bounds */  
public String set(int pos, String newStr) {
```

```
}
```

5/11/2003

(c) 2001-3, University of Washington

P-18

## clear

```
/** Remove all strings from this StringList */  
public boolean clear() {
```

```
}
```

5/11/2003

(c) 2001-3, University of Washington

P-19

## Challenges

### • Remove and insert operations

```
// Remove and return string at given position (shifting all later strings up)
```

```
String remove(int pos) { ... }
```

```
// Insert the given string at the given position (shifting all later strings down)
```

```
void add(int pos, String str) { ... }
```

### • Requires shifting elements after pos up or down one position

### • Remove the maximum capacity limitation

- When out of space, allocate a bigger array, copy the current elements over, then replace the old array with the bigger array

5/11/2003

(c) 2001-3, University of Washington

P-20

## Array Summary

---

- **Arrays are the fundamental low-level collection type built in to the Java language**
  - Also found in essentially all interesting programming languages
- **Size fixed when created**
- **Indexed access to elements**
- **Used to implement higher-level, richer container types**
  - More convenient, less error-prone, closer to what users normally want