CSE 142 Summer 2001

Arrays

Introduction

- Quick Review:
  - Collection classes: ArrayList
- Today:
  - Primitive collections – arrays
  - Using arrays to implement higher-level collection classes

What is an ArrayList?

- ArrayList objects are fairly sophisticated
  - Contain 0 or more objects
  - Can retrieve or delete any object in the collection
  - Can report how many objects there are, what's in the collection
- How is this implemented?
  - We've already gotten some idea from drawing the pictures…

Java Arrays

- The Java language provides *arrays*
  - Simple, ordered collections
  - Elements of a particular array all have the same type
  - Indexed (direct) access to elements
  - Size fixed when array is created

Java Arrays

- Example

```
String[ ] friends = new String[3];
friends[0] = "Sally";
friends[1] = "Puff";
friends[2] = "Spot"
if (friends[2].equals("cat") {
    System.out.println("Something's broken!!!")
}
```

  - Draw the picture!

Array Declaration and Creation

- Array declarations have additional syntax to indicate arrays are involved, but are otherwise like before

  <element type>[ ] <array name> = new <element type> [<capacity>];
- Details
  - Arrays can only hold elements of the specified type
    (can create arrays of generic Objects if you want, but normally you don't…)
  - <capacity> is any integer expression – value should be > 0
    (doesn't need to be a constant)
  - As with other variables, can separate declaration from initialization
    (fairly common with arrays that are object instance variables)

Array Element Access

- Access an array element using the array name and position
  <array name> [<position>]
  - <position> is any integer-valued expression, but MUST be in the range 0 to array capacity minus 1
    - If not, KABOOM!!!!
  - <position> is often called an index or subscript
  - An array element is a variable and can be used anywhere a name of that type may appear

```
if (friends[2].equals("Ichiro")) {
    System.out.println("Go " + friends[2] + "!!!");
}
```


Array Length

- Arrays are a (somewhat peculiar) kind of object
  - Allocated like other objects (new)
  - Special syntax
- If arr is an array, the member arr.length is it's capacity (allocated size)

Arrays and Iteration

- Arrays are very low level – don't have iterators
- Sequential access usually involves for loops
- Example: Assume that vector is an array of doubles. Print the sum of the values in vector.
- Solution

Exercise

- Search array for a string and return its position

```
// return position of aString in names if found, otherwise return –1
int indexOf(String[ ] names, String aString) {




}
```


Implementing Containers

- Example: Implement an ArrayList-like class that contains a list of Strings. Specification:

```
class StringList {                  // a list of strings
    StringList(int capacity);       // create new StringList with given capacity
    boolean isEmpty( );             // = "this StringList is empty"
    boolean isFull( );              // = "this StringList is full"
    int size( );                    // = # of Strings in this StringList
    boolean add(String str);        // add str to this StringList, result true if success
    boolean contains(String str);   // = "this StringList contains str"
    String get(int pos);            // return String at given position
    String remove(int pos);         // return String at given position and remove
                                    //   it from this StringList
```


StringList Representation

- Underlying Representation is an array of Strings
  - Size is fixed when it is created
- Need additional variable to keep track of how many Strings have actually been added so far

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

StringList Constructor

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

```
class StringList {                      // a list of strings
    private String[ ] strings;          // Strings in this StringList are stored in
    private int       size;             //    strings[0] through strings[size-1]

    // Construct new empty StringList with given capacity
    public StringList(int capacity) {




    }
}
```


Method add

- Gotcha – what do we do if there's no more room?

```
/**  Add new String to this StringList if there is room.
  * @param str String to be added
  * @return True if str was added successfully, otherwise false  */
 public boolean add(String str) {




}
```


size, isEmpty, isFull

- These are pretty simple…

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


}
/** = "this StringList is empty" */
public boolean isEmpty() {


}
 /** = "this StringList is full" */
public boolean isFull() {


}
```


contains

```
/** = "This StringList contains str" */
public boolean contains(String str) {




}
```


get

- Must *absolutely guarantee* that this doesn't crash when called – regardless of any bogus position given

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




}
```


remove

- Like get, must not blow up if given pos is bad
- Also, need to fill in the "hole" created if something is removed

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




}
```

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
- Normally used to implement higher-level, richer container types
  - More convenient for users
  - Less error prone than raw arrays