CSE 142

Arrays

Introduction

- Quick Review:
  - Collection classes: ArrayList
- Today:
  - A low-level, built-in collection – array
  - Using arrays to implement higher-level collection classes
- Reading
  - Dugan notes: ch. 18-19
  - Niño & Hosch: 22.1 and first part of 22.2

What is an ArrayList?

- ArrayList objects are fairly sophisticated.
  - 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…

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.
- Unlike ArrayLists:
  - The size is fixed when the array is created.
  - No iterator; must use explicit indexes and for/while loops.
  - Can specify the type of the elements of arrays.

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];
}
```


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.!
- ***<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**

Array Element Access

- **Access an array element using the array name and position.**
  <array name> [*<position>*]
- **Details:**
  - ***<position>* is an integer expression.**
  - **Positions count from zero, 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.**

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
    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
    String get(int pos);            // return String at given position
    String put(int pos, String newStr);   // update String at given position,
                                          // return previous String at that position
}
```

- **For simplicity, have a fixed maximum capacity.**

StringList Representation

- **Underlying representation of the collection of strings is an array of Strings.**
- **Need separate variable to keep track of how many Strings have actually been added 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]
    ...
}
```


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) {




    }
}
```


add

```
/**  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

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









}
```

contains

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




}
```


get

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




}
```


put

```
// 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 put(int pos, String newStr) {




}
```


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.**

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 for users.**