Arrays of Objects

October 8, 2007

Quick review of last lecture

Arrays

• An array is an ordered list of values

<table>
<thead>
<tr>
<th>scores</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>79</td>
<td>87</td>
<td>94</td>
<td>82</td>
<td>67</td>
<td>98</td>
<td>87</td>
<td>81</td>
<td>74</td>
<td>91</td>
</tr>
</tbody>
</table>

The entire array has a single name
Each value has a numeric index

An array of size N is indexed from zero to N-1
This array holds 10 values that are indexed from 0 to 9

Declaring Arrays

• The scores array could be declared as follows:
  ```java
  int[] scores = new int[10];
  ```
  The type of the variable scores is int[] (an array of integers)
  Note that the array type does not specify its size, but each object of that type has a specific size
  The reference variable scores is set to a new array object that can hold 10 integers

Declaring Arrays

• Some other examples of array declarations:
  ```java
  float[] prices = new float[500];
  boolean[] flags;
  flags = new boolean[20];
  char[] codes = new char[1750];
  ```
The iterator version of the \texttt{for} loop

\begin{verbatim}
for (int score : scores)
    System.out.println(score);
\end{verbatim}

- This is only appropriate when processing all array elements from top (lowest index) to bottom (highest index)

What \texttt{for/\texttt{in}} can't do

\begin{verbatim}
int[] primeNums = {2, 3, 5, 7, 11, 13, 17, 19};
for (int i=0; i<primeNums.length; i++)
    {System.out.print("primeNums[" + i + "]= ");
     System.out.println(primeNums[i]);
    }
\end{verbatim}

What \texttt{for/\texttt{in}} can't do

\begin{verbatim}
String word="Test";
for (int i=0; i<word.length(); i++)
    {
      if(i>0)
        System.out.print(", ");
      System.out.print( word.charAt(i) );
    }
\end{verbatim}

Other Stuff from Sections 7.1 & 7.2

Initializer Lists

- An \textit{initializer list} can be used to instantiate and fill an array in one step
- The values are delimited by braces and separated by commas
- Examples:
  \begin{verbatim}
  int[] units = {147, 323, 89, 933, 540, 269, 97, 114, 298, 476};
  char[] letterGrades = {'A', 'B', 'C', 'D', 'F'};
  \end{verbatim}

Initializer Lists

- Note that when an initializer list is used:
  - the new operator is not used
  - no size value is specified
  - The size of the array is determined by the number of items in the initializer list
  - An initializer list can be used only in the array declaration
  - See \texttt{Primes.java} (page 381)
Example: Primes.java (page 381)

Bounds Checking

• Once an array is created, it has a fixed size
• An index used in an array reference must specify a valid element
• That is, the index value must be in range 0 to N-1
• The Java interpreter throws an ArrayIndexOutOfBoundsException if an array index is out of bounds
• This is called automatic bounds checking

• For example, if the array codes can hold 100 values, it can be indexed using only the numbers 0 to 99
• If the value of count is 100, then the following reference will cause an exception to be thrown:
  
  ```java
  System.out.println (codes[count]);
  ```

  • It’s common to introduce off-by-one errors when using arrays
  ```java
  for (int index=0; index <= 100; index++)
      codes[index] = index*50 + epsilon;
  ```

Example: LetterCount.java (page 376)

Alternate Array Syntax

• The brackets of the array type can be associated with the element type or with the name of the array
• Therefore the following two declarations are equivalent:
  ```java
  float[] prices;
  float prices[];
  ```

  • The first format generally is more readable and should be used

Bounds Checking

• Each array object has a public constant called length that stores the size of the array
• It is referenced using the array name:
  ```java
  scores.length
  ```

  • Note that length holds the number of elements, not the largest index
• See ReverseOrder.java (page 375)
• See LetterCount.java (page 376)
Chapter 7
Section 7.3

Arrays of Objects
- The elements of an array can be object references
- The following declaration reserves space to store 5 references to String objects
  \[\text{String[]} \text{words} = \text{new String[5];}\]
- It does NOT create the String objects themselves
- Initially an array of objects holds null references
- Each object stored in an array must be instantiated separately

Arrays of Objects
- The words array when initially declared:

```
words
```
- At this point, the following reference would throw a NullPointerException:
  \[\text{System.out.println} (\text{words}[0]);\]

Arrays of Objects
- After some String objects are created and stored in the array:

```
words
```

Arrays of Objects
- Keep in mind that String objects can be created using literals
- The following declaration creates an array object called verbs and fills it with four String objects created using string literals

```
String[] verbs = \{\text{"play"}, \text{"work"}, \text{"eat"}, \text{"sleep"}\};
```

Arrays of Objects
- The following example creates an array of Grade objects, each with a string representation and a numeric lower bound
  - See GradeRange.java (page 384)
  - See Grade.java (page 385)
- Now let’s look at an example that manages a collection of CD objects
  - See Tunes.java (page 387)
  - See CDCollection.java (page 388)
  - See CD.java (page 391)
Arrays of Objects

- A UML diagram for the Tunes program:

```
Arrays of Objects

- A UML diagram for the Tunes program:

Tunes
+ main (args : String[]) : void

CDCollection
- collection : CD[]
- count : int
- totalCost : double
+ addCD (title : String, artist : String, cost : double, tracks : int) : void
+ toString() : String
- increaseSize() : void

CD
- title : String
- artist : String
- cost : double
- tracks : int
+ toString() : String
```

THE END