Introduction to Java™

Module 3: Arrays

Prepared by Costantinos Costa for EPL 233

An Introduction to Arrays

- Arrays in Java are just as they exist on all programming languages
  - Come in handy when you want to organize multiple values of logically related items
  - They are static objects (meaning: Cannot be resized dynamically)
- The syntax used for arrays is: identifier[subscript] (eg. Array1[5])
  - The identifier refers to the array as a whole
  - While the subscript refers to a specific element of the array

Creating Arrays

- To create an array first you must declare it :
  `int numbers[];` or `int[] numbers;`
- Java lets you create arrays only using the new operator, like this:
  ```
  number= new int[x];
  // x is an integer stating the size of the array
  ```
- For primitive types that is enough…
- However for an array of some other class type there is the need to initialize every object of your array manually
- This is necessary because so far you have an array of null objects

Initializing an Array

- To properly initialize an array (not of primitive type) you have to initialize every element of the array like this:

```
MyStrangeObject[] objs;
objs = new MyStrangeObject[20];
for (int I=0;I<20;I++)
     objs[I] = new MyStrangeObject();
```

- NOTE

  `objs[0];` is the first element of the array.

More on Arrays

- CAUTION

  Don’t try to access a nonexistent array element.
  For example:

```
int numbers[];
numbers = new int[10];

n = numbers[15];
```

  would go beyond the boundaries of the array and so Java would generate the exception ArrayIndexOutOfBounds

Number of Elements Vs Array Size

- The member length gives the size of an array.
- An array may have less elements than its size.
- To find the number of elements in an array a counter should be used.
- Alternatively use a for loop to check how many positions of the array are initialized.

```
for (int I=0;I<nums.length;I++)
    if (nums[I]!=null) counter++;
```

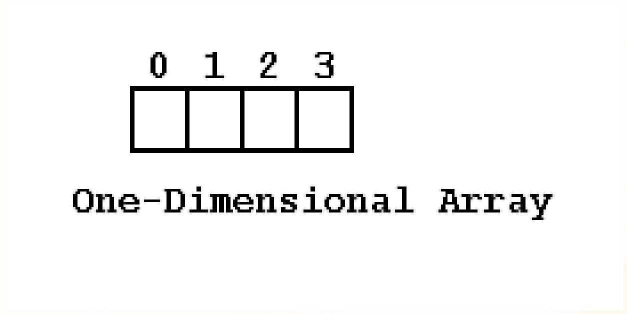
Copying an Array

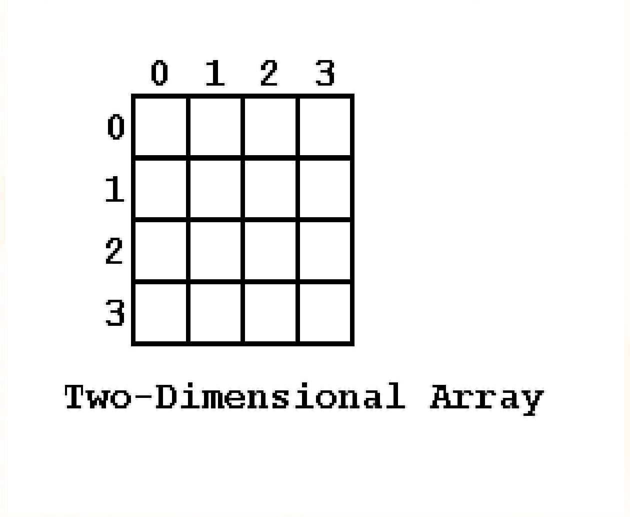
- To copy an array one should
  - Create a new array with the size of the origin array:
    ```
    String[] copy=new String[origin.length()];
    ```
  - Copy each object of the origin array manually:
    ```
    for (int I=0;I<origin.length();I++)
         copy[I]=new String(origin[I]);
    ```
- The following code would result to reference to the same array:
  ```
  String[] origin;
  String[] copy;
  origin = new {“one”, “two”, “three”};
  copy = origin;
  // A common mistake where one would think that
  // he has two different arrays
  ```

Multidimensional Arrays

- Java doesn't support multidimensional arrays in the conventional sense
- Possible to create arrays of arrays
- To create a two-dimensional array of integers you would write something like this:

```
int table[][] = new int[4][4];
```





Multidimensional Arrays

- You refer to a value stored in a n-dimensional array by using subscripts for all the dimensions like this:
```
int value = table[3]…[2];
```
- A quick way to initialize an n-dimensional array is to use nested for loops

A Bit About Collections

- Collections vs Arrays
  - Dynamic sizing
  - Any type of object can be put in a collection
  - No support for primitive types
- Vector: Like a dynamic array of objects
- BitSet: A vector of bits (minimum size 64 bits)
- Hashtable: An *associative* dynamic array (links a key object with a value object)
- Stack: A last-in, first-out (LIFO) collection with push and pop methods

ArrayList: Another collection

- Resizable-array implementation of the List interface
- Implements all optional list operations, and permits all elements, including null
  - In addition it provides methods to manipulate the size of the array that is used internally to store the list
- Roughly equivalent to Vector, except that it is unsynchronized

More on ArrayList

- Each ArrayList instance has a *capacity*
  - The capacity is the size of the array used to store the elements in the list
  - It is always at least as large as the list size
  - As elements are added to an ArrayList, its capacity grows automatically
- The details of the growth policy are not specified
  - Adding an element has constant amortized time cost

More on ArrayList

- **Note that this implementation is not synchronized**
- If multiple threads access an ArrayList instance concurrently, and at least one of the threads modifies the list structurally, it *must* be synchronized externally.
    - A structural modification is any operation that adds or deletes one or more elements, or explicitly resizes the backing array
        - setting the value of an element is not a structural modification
- Typically accomplished by synchronizing on some object that naturally encapsulates the list
    - If no such object exists, the list should be "wrapped" using the Collections.synchronizedList method
    - This is best done at creation time, to prevent accidental unsynchronized access to the list:
      List list = Collections.synchronizedList(new ArrayList(...));

ArrayList Usage

```
a =  new ArrayList(cap)      // Creates ArrayList with initial int capacity cap

// Adding elements
a.add(e)                  // adds e to end of ArrayList a
a.add(i, e)               // Inserts e at index i, shifting elements up as necessary.

// Replacing an element
a.set(i,e)                // Sets the element at index i to e.

// Getting the elements
e = (E) a.get(i)             // Returns the object at index i.
oarray = a.toArray()         //Returns values in array of objects.
earray = a.toArray(E[])      // The array parameter should be of the E class.
```

ArrayList Usage

```
// Searching
b = a.contains(e)      // Returns true if ArrayList a contains e
i = a.indexOf(e)       // Returns index of first occurrence of e, or -1 if not there.
i = a.lastIndexOf(e)   // Returns index of last occurrence of e, or -1 if not there.

// Removing elements
a.clear()              // Removes all elements from ArrayList a
a.remove(i)            // Removes the element at position i.
a.removeRange(i, j)    // Removes the elements from positions i thru j.

// Other
i = a.size()           // Returns the number of elements in ArrayList a.
a.trimToSize()         // Trims the capacity of this ArrayList instance to be the
                       // list's current size.
a.isEmpty()            // Tests if this list has no elements.
```

Example-Exercise

- A anagram is a word that can be created by rearranging the letters of another given word. We ignore white spaces and letter case. The all letters of “Fillers" can be rearranged to the phrase "refills".
- Implement a Java program that checks to given Strings whether one is an anagram of the other.
- **Hint**: Objects s of type String can be converted to lower case with s.toLowerCase(). s.toCharArray() returns the content of the String as an char-array. Note that char values can be used in Java everywhere where an int value is allowed.
- Here some anagrams
  - Refills→ fillers
  - Relayed → layered
  - Rentals → antlers
  - Rebuild →builder