CMSC 131

Object-Oriented Programming I

Introduction to Arrays

Dept of Computer Science
University of Maryland College Park

This material is based on material provided by Ben Bederson, Bonnie Dorr, Fawzi Emad, David Mount, Jan Plane

Overview

- Arrays

Arrays

- Data structure → mechanism for storing data in a structured way
- We have seen simple data structures implemented as classes:
  - Rational.java
- **Problem** → You need to keep track of the scores of students in a class
  - Declaring and handling 50 variables is not an easy task
  - Arrays come to the rescue
- **Array** → Collection of values that can be treated as a unit or individually.

```
a = new int[5];
```

- You can visualize an array as a set of variables one after another

Arrays

- Array →a very useful data structure provided by Java and other programming languages
- Array → sequence of variables of the same type
  - Homogeneous data structure
  - Size (quantity) fixed when space is allocated
  - Ordered
- Individual elements of sequence can be referenced, updated, etc.
- Arrays are objects (hence allocated on heap) with a reference on the stack
- Like other objects, “instance variables” of array = cells in array are assigned default values (0 / null / etc.) when array is created
- Let’s draw a memory map representing an array
- Variable that represents the array can be seen as storing the address of the first element of the array

Array Indexing

- Java provides a special syntax for uniformly accessing cells in an array
  - Declaration of a:
    int[] a;
  - Allocation of space for array named a:
    a = new int[5]; // or combined: int[] a = new int[5];
    - This creates five int variables "named"::a[0],a[1],a[2],a[3],a[4]
  - To modify contents of cell #2 to 6 and cell #1 to 74:
    a[2] = 6;
    a[1] = 74;
  - To use the contents of cell #2 and cell #1 :
    System.out.println("value = " + (a[1]-a[2]));
- This access mechanism to the individual elements is called array indexing
  - In Java / C / C++, array cells are indexed beginning at 0 and going up to n-1 (where n is number of cells)
  - Beware: start at 0! and end at one less than the size!!
  - If we assume the array name represents the address of the first array entry, then the index value represents how many units to move forward
- **Example:** ReadValues.java

Square Brackets: [ ] and length

- Three uses in Java:
  - Array variable declaration → **int[] a**
  - Array object creation → **new int[10]**
  - Array indexing → **a[0]**
- Arrays also have a.length value that holds the amount of space currently allocated for that array

Alternate Declaration Syntax

- To maintain consistency with C / C++, following declaration of array variables also possible

  int grade[];

  Compare to Java standard:

  int[] grade;
- Java standard generally preferred
  - "type[]" emphasizes array status
- Alternative syntax sometimes handy:

  int grade[], size, gpa[];
  - Declares two arrays of base type int: grade, gpa
  - Declares a single int variable: size

Summary of Arrays

- Arrays are:
  - Sequences of cells holding values of the same type ("base type")
  - Objects (hence created using new)
- To define an array variable:
  int[] a; // an array with base type int
- To create an array object:
  a = new int[10];
  - Creates an array of 10 cells on the heap
  - The base type is int
- To access individual array cells → use indexing
  - a[0], a[1], …, a[9]
  - Cells are just like variables:
    - They may be read → x = a[3];
    - They may be written →a[2] = 7;
- **Example:** GeneratingRandomValues.java

A Common Programming Idiom

- To process all elements in array …
- Do the following:

```
for (int i = 0; i < a.length; i++){
   …process the one element at a[i]…
}
```

  - Use fresh loop counter to avoid overwriting another variable of same name elsewhere
  - Remember:
    - Use 0 as one end of the array, not 1
    - Use i **<** a.length as the other end, not i **<=** a.length

Copying Arrays

- Does the following copy a into b?

```
int[] a = new int[5];
int[] b = a;
```

No!: **a, b are aliases!**

- How to make a copy?

```
int[] a = new int[5];
int[] b = new int[a.length];
for (int i = 0; i < a.length; i++){
    b[i] = a[i];
}
```

Making Arrays Bigger

- Suppose we want to make an array bigger by adding an element
- Does the following work?

```
int[] a = new int[5];
a.length++;
```

- **No!**
  - We get the following:

```
Exception in thread "main" java.lang.Error: Unresolved
  compilation problem:
The final field array.length cannot be assigned
at Sample.main(Sample.java:15)
```

  - a.length is immutable
  - No assignment to it is allowed

To Make an Array Bigger…

- Create a new larger array object
- Copy old array contents into new object
- Assign address of new object to variable

```
int[] a = new int[5];
int[] temp = new int[a.length + 1];
for (int i = 0; i < a.length; i++){
    temp[i] = a[i];
 }
a = temp;
```

  - New variable **temp** created to hold copy
  - Previous contents of a become garbage
- Usually when you grow an array you allocate an array that is twice the size of the original array. This approach is more efficient that growing the array one element at a time
  - How we know? Amortized analysis.