Software and Programming II

Arrays and `ArrayLists`

Keith Mannock and Oded Lachish

Department of Computer Science and Information Systems
Birkbeck, University of London

`keith@dcs.bbk.ac.uk` and `oded@dcs.bbk.ac.uk`

September 30, 2014

Overview

Yet again, this is basically revision of SP1 material but just to make sure...

We will discuss how:

- To collect elements using arrays and `ArrayLists`
- To use the enhanced for loop for traversing arrays and `ArrayLists`
- To learn common algorithms for processing arrays and `ArrayLists`
- To work with mutli-dimensional arrays

Birkbeck UNIVERSITY OF LONDON

Arrays — a brief reprise I

- A Computer Program often needs to store a list of values and then process them
- For example, if you had this list of values, how many variables would you need?

```
double input1, input2, input3.
```

Arrays — a brief reprise II

- Declaring an array is a two step process

```
double[] values;   // declare array variable
values = new double[10];  // initialise array
```

- One can also declare and create the array on the same line

```
double[] values = new double[10];
```

- You can also declare and set the initial contents of all elements by:

```
int[] primes = {2, 3, 5, 7};
```

Arrays — a brief reprise III

- Each element of an array is numbered and we call this the *index*; arrays start at the index 0
- One can access an element by using the name of the array and the index number

  `values[i]`
- An array knows how many elements it can hold via the `length` property
- values . length is the size of the array named values
- It is an integer value (index of the last element + 1)
- We can use this to *range check* and prevent *out of bounds* errors

Arrays — a brief reprise IV

- It is key that you know the difference between the:
  - Array variable: The named *handle* to the array
  - Array contents: Memory where the values are stored
- An array variable contains a reference to the array contents.
- The reference is the location of the array contents (in memory).

One can make one array reference refer to the same contents of another array reference — know as an *array alias*

```
int[] scores = { 10, 9, 7, 4, 5 };
Int[] values = scores;  // Copying the array reference
```

In this example values is an *alias* for scores

Arrays — a brief reprise V

An array cannot change size at run time

- The programmer may need to guess at the maximum number of elements required
- It is a good idea to use a constant for the size chosen
- Use a variable to track how many elements are filled

```
final int LENGTH = 100;
double[] values = new double[LENGTH];
int currentSize = 0;
Scanner in = new Scanner(System.in);
while (in.hasNextDouble()) {
  if (currentSize < values.length) {
    values[currentSize] = in.nextDouble();
    currentSize++;
  }
}
```

Birkbeck UNIVERSITY OF LONDON

Arrays — a brief reprise VI

We can then use currentSize, not values.length for the last element to enable us to *walk* a partially filled array.

```
for (int i = 0; i < currentSize; i++)
  System.out.println(values[i]);
```

The Enhanced for Loop I

Using for loops to *walk* arrays is very common

- The enhanced for loop simplifies the process
- Also known as the *for each* loop
- Read the following code as:

  "For each element in the array"

```
double [] values = ... ;
double total = 0;
for (double element : values)
  total += element;
```

The Enhanced for Loop II

As the loop proceeds, it will:

- Access each element in order (0 to length-1)
- Copy it to the *element variable*
- Execute loop body

Please note that it is not possible to:

- Change elements
- Get a bounds error

Use the enhanced `for` loop when:

- You need to access every element in the array
- You do not need to change any elements of the array

Birkbeck UNIVERSITY OF LONDON

Common operations used with arrays

- Filling an Array
- Sum and Average Values
- Find the Maximum or Minimum
- Output Elements with Separators
- Linear Search
- Removing an Element
- Inserting an Element
- Swapping Elements
- Copying Arrays
- Reading Input

Birkbeck
UNIVERSITY OF LONDON

Filling an Array

- Initialise an array to a set of calculated values
- Example: Fill an array with squares of 0 through 10

```
int[] values = new int[11];

for (int i = 0; i < values.length; i++)
  values[i] = i * i;
```

Sum and Average

Use enhanced `for` loop, and make sure not to divide by zero

```
double total = 0, average = 0;

for (double element : values)
  total = total + element;

if (values.length > 0) { average = total / values.length; }
```

Maximum and Minimum I

- Set largest to first element
- Use `for` or enhanced `for` loop
- Use the same logic for minimum

Maximum and Minimum II

Typical for loop to find maximum

```
double largest = values[0];
for (int i = 1; i < values.length; i++)
  if (values[i] > largest)
    largest = values[i];
```

Maximum and Minimum III

Enhanced `for` to find maximum

```
double largest = values[0];
for (double element : values)
  if element > largest)
    largest = element;
```

Maximum and Minimum IV

Enhanced `for` to find minimum

```
double smallest = values[0];
for (double element : values)
  if element < smallest)
    smallest = element;
```

Output Elements with Separators

- Output all elements with separators between them
- No separator before the first or after the last element

```
for (int i=0; i < values.length; i++) {
  if (i > 0)
    System.out.print(" | ");
  System.out.print(values[i]);
}
```

- Useful `Array` method: `Arrays.toString()`
- Useful for debugging!

```
import java.util.Arrays;
System.out.println(Arrays.toString(values));
```

Birkbeck UNIVERSITY OF LONDON

Linear Search

- Search for a specific value in an array
- Start from the beginning (left), stop if/when it is found
- Uses a boolean found flag to stop loop if found

```
int searchedValue = 100;  int pos = 0;
boolean found = false;
while (pos < values.length && !found) {
  if (values[pos] == searchedValue)
    found = true;
  else
    pos++;
}

...

if (found)
    System.out.println(Found at position:  + pos);
  else
    System.out.println(Not found);
```

Removing an element (at a given position)

- Requires tracking the *current size* (number of valid elements)
- But dont leave a *hole* in the array!
- Solution depends on if you have to maintain *order*
  If not, find the last valid element, copy over position, update size

```
values[pos] = values[currentSize  1];
currentSize--;
```

Removing an element and maintaining order

- Requires tracking the *current size* (number of valid elements)
- But dont leave a *hole* in the array!
- Solution depends on if you have to maintain *order*
  If so, move all of the valid elements after `pos` up one spot, update size

```
for (int i = pos; i < currentSize - 1; i++)
  values[i] = values[i + 1];

currentSize--;
```

Inserting an Element

- Solution depends on if you have to maintain *order*
- If not, just add it to the end and update the size
- If so, find the right spot for the new element, move all of the valid elements after pos down one spot, insert the new element, and update size

```
if (currentSize < values.length){
  currentSize++;

  for (int i = currentSize - 1; i > pos; i--)
    values[i] = values[i - 1];  // move down

  values[pos] = newElement;     // fill hole
}
```

Swapping Elements

Three steps using a temporary variable

```
double temp = values[i];
values[i] = values[j];
values[j] = temp;
```

Copying Arrays

- Not the same as copying only the reference
- Copying creates two set of contents!
- Use the `Arrays.copyOf` method

```
double[] values = new double[6];
. . . // Fill array
double[] prices = values;   // Only a reference so far
double[] prices = Arrays.copyOf(values, values.length);
// copyOf creates the new copy, returns a reference
```

Growing an array

- Copy the contents of one array to a larger one
- Change the reference of the original to the larger one
- Example: Double the size of an existing array
  - Use the `Arrays.copyOf` method
  - Use `2 *` in the second parameter
  - `Arrays.copyOf` second parameter is the length of the new array

```
double[] values = new double[6];
. . . // Fill array
double[] newValues = Arrays.copyOf(values, 2 * values.length);
values = newValues;
```

Reading Input

- Known number of values to expect
  Make an array that size and fill it one-by-one

```
double[] inputs = new double[NUMBER_OF_INPUTS];
for (i = 0; i < values.length; i++)
  inputs[i] = in.nextDouble();
```

- Unknown number of values
  Make maximum sized array, maintain as partially filled array

```
double[] inputs = new double[MAX_INPUTS];
int currentSize = 0;
while (in.hasNextDouble() && currentSize < inputs.length) {
  inputs[currentSize] = in.nextDouble();
  currentSize++;
}
```

Example: largest element in an array

```
import java.util.ArrayList;
import java.util.Scanner;

public class LargestInArray {
    public static void main(String[] args) {
        final int LENGTH = 100;
        double[] data = new double[LENGTH];
        int currentSize = 0;

        // read inputs

        System.out.println("Please enter values, Q to quit:");
        Scanner in = new Scanner(System.in);
        while (in.hasNextDouble() && currentSize < data.length) {
            data[currentSize] = in.nextDouble();
            currentSize++;
        }

        // Find the largest value

        double largest = data[0];
        for (int i = 1; i < currentSize; i++)
            if (data[i] > largest)
                largest = data[i];

        // print all values, marking the largest

        for (int i = 0; i < currentSize; i++) {
            System.out.print(data[i]);
            if (data[i] == largest)
                System.out.print(" <= largest value");
            System.out.println();
        }
    }
}
```

Using Arrays with Methods I

- Methods can be declared to receive references as parameter variables
- What if we wanted to write a method to sum all of the elements in an array?
- Answer: pass the array reference as an argument!

```
priceTotal = sum(prices);

...
public static double sum(double[] values){
  double total = 0;
  for (double element : values)
    total = total + element;
  return total;
}
```

- Arrays can be used as method arguments and method return values.

Using Arrays with Methods II

- Passing a reference give the called method access to all of the data elements
- It CAN change the values!
- Example: Multiply each element in the passed array by the value passed in the second parameter

```
multiply(values,    10);

...
public static void multiply(double[] data, double factor){
  for (int i = 0; i < data.length; i++)
    data[i] = data[i] * factor;
}
```

Method returning an array

- Methods can be declared to return an array

```
public static int[] squares(int n)
```

- To Call: Create a compatible array reference

```
int[] numbers = squares(10);
```

- Call the method

```
public static int[] squares(int n) {
  int[] result = new int[n];
  for (int i = 0; i < n; i++)
    result[i] = i * i;
  return result;
}
```

n-dimensional arrays

Arrays can be used to store data in n dimensions; the 2D version is like a spreadsheet
We refer to rows and columns and the data structure is also known as a *matrix*

- Use two pairs of square braces

```
const int COUNTRIES = 7;
const int MEDALS = 3;
int[][] counts = new int[COUNTRIES][MEDALS];
```

- You can also initialize the array

```
const int COUNTRIES = 7;
const int MEDALS = 3;
int[][] counts =
{
  { 1, 0, 1 },
  { 1, 1, 0 },
  { 0, 0, 1 },
  { 1, 0, 0 },
  { 0, 1, 1 },
  { 0, 1, 1 },
  { 1, 1, 0 }
};
```

- Note the use of two levels of curly braces. Each row has braces with commas separating them.

ArrayLists

- When you write a program that collects values, you dont always know how many values you will have.
- In such a situation, a Java `ArrayList` offers two significant advantages:
  1. `ArrayLists` can grow and shrink as needed.
  2. The `ArrayList` class supplies methods for common tasks, such as inserting and removing elements.
- An `ArrayList` expands to hold as many elements as needed

Birkbeck UNIVERSITY OF LONDON

Declaring and Using `ArrayLists`

The `ArrayList` class is part of the `java.util` package

- It is a generic class designed to hold many types of objects (but typesafe at compile time)
- Provide the type of element during declaration
  - Inside < > as the *type parameter*
  - The type must be a *Class*
    Cannot be used for primitive types (e.g., `int`, `double`, ...)

```
ArrayList<String> names = new ArrayList<String>();
```

or in the new Java 7 syntax

```
ArrayList<String> names = new ArrayList<>();
```

ArrayList API

...

Wrappers and Auto-boxing I

Structures such as `ArrayLists` can only contain references to objects Java provides wrapper classes for primitive types and the conversions are automatic using auto-boxing

- Primitive to wrapper Class

```
double x = 29.95;
Double wrapper;
wrapper = x;  // boxing
```

- Wrapper Class to primitive

```
double x;
Double wrapper = 29.95;
x = wrapper;  // unboxing
```

Wrappers and Auto-boxing II

Primitive Type	Wrapper Class
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Wrappers and Auto-boxing III

- You cannot use primitive types in an `ArrayList`, but you can use their wrapper classes
- This depends on auto-boxing for conversion
- Declare the `ArrayList` with wrapper classes for primitive types

  For example,
  - Use `ArrayList<Double>`
  - Add primitive double variables
  - Or double values

```
double x = 19.95;
ArrayList<Double> values = new ArrayList<Double>();
values.add(29.95); // boxing
values.add(x); // boxing
double x = values.get(0);    // unboxing
```

Birkbeck UNIVERSITY OF LONDON

Preferring Arrays over `ArrayLists`

Use an Array if:

- The size of the array never changes
- You have a long list of primitive values
- For efficiency reasons

Use an Array List:

- For just about all other cases
- Especially if you have an unknown number of input values

Birkbeck UNIVERSITY OF LONDON

Questions