合肥學院
HEFEI UNIVERSITY

OOP with Java

9. Arrays

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

website

Arrays

- In the Lesson 8: *Loops*, we learned how to deal with repeated code, by using loops

Arrays

- In the Lesson 8: *Loops*, we learned how to deal with repeated code, by using loops
- What if we have repeated variables?

Arrays

- In the Lesson 8: *Loops*, we learned how to deal with repeated code, by using loops
- What if we have repeated variables?
- We use arrays!

Allocating and Accessing

- An array is basically a (fixed-length) sequence of variables of the same type which can be accessed using an integer index (starting at 0)

Allocating and Accessing

- An array is basically a (fixed-length) sequence of variables of the same type which can be accessed using an integer index (starting at 0)
- Arrays are declared as `type[] variableName`, where `type` is the element type can be any type or class

Allocating and Accessing

- An array is basically a (fixed-length) sequence of variables of the same type which can be accessed using an integer index (starting at 0)
- Arrays are declared as `type[] variableName`, where `type` is the element type can be any type or class
- New arrays can be allocated using `variableName = new type[length]`, which creates an array of length `length`

Allocating and Accessing

- An array is basically a (fixed-length) sequence of variables of the same type which can be accessed using an integer index (starting at 0)
- Arrays are declared as `type[] variableName`, where `type` is the element type can be any type or class
- New arrays can be allocated using `variableName = new type[length]`, which creates an array of length `length`
- The elements of the array can be accessed via `variableName[index]`, where `index` is a 0-based index with a valid range from 0 to `length-1`

Allocating and Accessing

- An array is basically a (fixed-length) sequence of variables of the same type which can be accessed using an integer index (starting at 0)
- Arrays are declared as `type[] variableName`, where `type` is the element type can be any type or class
- New arrays can be allocated using `variableName = new type[length]`, which creates an array of length `length`
- The elements of the array can be accessed via `variableName[index]`, where `index` is a 0-based index with a valid range from 0 to `length-1`
- `variableName[index]` can be treated as a normal variable of type `type` with read and write (assignment) access

Allocating and Accessing

- An array is basically a (fixed-length) sequence of variables of the same type which can be accessed using an integer index (starting at 0)
- Arrays are declared as `type[] variableName`, where `type` is the element type can be any type or class
- New arrays can be allocated using `variableName = new type[length]`, which creates an array of length `length`
- The elements of the array can be accessed via `variableName[index]`, where `index` is a 0-based index with a valid range from 0 to `length-1`
- `variableName[index]` can be treated as a normal variable of type `type` with read and write (assignment) access
- In lesson Lesson 25: *Exceptions*, we will learn what happens if you use a value outside of the $0 \ldots$ `length-1` range as array index.

Listing: An example for a String array

```
/** Example for allocating and using a string arrays and accessing their values */
public class StringArrayNewAccess {

  /** The main routine
   * @param args
   *          we ignore this parameter for now */
  public static final void main(String[] args) {
    String[] array = new String[4]; // create an string array for length 4

    array[0] = "hello ";  //$NON-NLS-1$
    array[1] = "world";  //$NON-NLS-1$
    array[2] = ", it's ";  //$NON-NLS-1$
    array[3] = "me!";  //$NON-NLS-1$

    System.out.print(array[0]);   // prints "hello "
    System.out.print(array[1]);   // prints "world"
    System.out.print(array[2]);   // prints ", it's "
    System.out.println(array[3]); // prints "me!"

    array[3] = "Ni Hao ";  //$NON-NLS-1$

    System.out.print(array[3]); // prints "Ni Hao "
    System.out.print(array[1]); // prints "world"
    System.out.print(array[2]); // prints ", it's "
    System.out.print(array[0]); // prints "hello "
  }
}
```

Allocating and Iterating over a Array

Listing: An example for iterating over a String array

```
/** Example for allocating and using a string arrays and iterating over them */
public class StringArrayNewIterate {

  /** The main routine
   * @param args
   *           we ignore this parameter for now */
  public static final void main(String[] args) {
    String[] array = new String[4]; // create an string array for length 4

    array[0] = "hello␣"; //$NON-NLS-1$
    array[1] = "world"; //$NON-NLS-1$
    array[2] = ",␣it's␣"; //$NON-NLS-1$
    array[3] = "me!"; //$NON-NLS-1$
    for (int i = 0; i < 4; i++) { // if length is 4, indexes from 0 to 3 are allowed
      System.out.print(array[i]); // access an element of array by using "[index]"
    } // prints "hello world, it's me!"
  }
}
```

Allocating and Iterating over a Integer Array

Listing: An example for an integer array

```
/** Example for allocating and using a int arrays and iterating over them */
public class IntArrayNewIterate {

  /** The main routine
   * @param args
   *          we ignore this parameter for now */
  public static final void main(String[] args) {
    int[] array = new int[5]; // create an integer array for length 5

    System.out.print("Initially, all elements are 0:"); //$NON-NLS-1$
    for (int i = 0; i < 5; i++) { // if length is 5, indexes from 0 to 4 are allowed
      System.out.print(' ');
      System.out.print(array[i]); // access an element of array by using "[index]"
    } // prints "0 0 0 0 0"
    System.out.println();

    array[4] = 5; // set fifth element to 5
    array[2] = 3; // set third element to 3
    array[0] = 1; // set first element to 1
    array[1] = 2; // set second element to 2
    System.out.print("Now some elements are set: "); //$NON-NLS-1$
    for (int i = 0; i < 5; i++) { // if length is 5, indexes from 0 to 4 are allowed
      System.out.print(' ');
      System.out.print(array[i]); // access an element of array by using "[index]"
    } // prints "1 2 3 0 5"
  }
}
```

Allocating an Array with Length Stored in Variable

Listing: Allocating an Array with Length Stored in Variable

```
/** Example for allocating an array of a length stored in a variable and
 *  iterating over a character array */
public class CharArrayAllocateWithLengthInVariable {

  /** The main routine
   * @param args
   *          we ignore this parameter for now */
  public static final void main(String[] args) {
    int neededLength = 'Z' - 'A' + 1; // yes, this is allowed, as characters are internally numbers
    char[] AtoZ; // notice how we declare the array here, but actually allocate it later: we can do that!
    AtoZ = new char[neededLength]; // allocate array: length specified in variable
    char letter = 'A'; // set letter to 'A'

    for (int i = 0; i < neededLength; i++) { // iterate from 0 to neededLength-1
      AtoZ[i] = letter++; // store letter in array, then move on to next letter
    }

    for (int i = 0; i < neededLength; i++) { // iterate from 0 to neededLength-1
      System.out.print(AtoZ[i]); // print the array element at index i
    }
  }
}
```

Length of an Array

- We can read the length of an array `array` via `array.length`

Length of an Array

- We can read the length of an array `array` via `array.length`
- Valid indexes to access the elements of an array `array` are from 0 to `array.length-1`

Array Length

Listing: An example for getting the length of an array

```
/** Example for iterating over arrays using their length */
public class IntArrayNewLengthIterate {

  /** The main routine
   * @param args
   *          we ignore this parameter for now */
  public static final void main(String[] args) {
    int[] array = new int[5]; // create an integer array for length 5

    System.out.print("Initially, all elements are 0:"); //$NON-NLS-1$
    for (int i = 0; i < array.length; i++) { // indexes are valid from 0 to array.length-1
      System.out.print(' ');
      System.out.print(array[i]); // access an element of array by using "[index]"
    } // prints "0 0 0 0 0"
    System.out.println();

    array[4] = 5; // set fifth element to 5
    array[2] = 3; // set third element to 3
    array[0] = 1; // set first element to 1
    array[1] = 2; // set second element to 2
    System.out.print("Now some elements are set: "); //$NON-NLS-1$
    for (int i = 0; i < array.length; i++) { //  indexes are valid from 0 to array.length-1
      System.out.print(' ');
      System.out.print(array[i]); // access an element of array by using "[index]"
    } // prints "1 2 3 0 5"
  }
}
```

Iterating Backwards

Listing: An iterating over an array backwards

```
/** Example for iterating backwards over arrays using their length*/
public class IntArrayIterateBackwards {

  /** The main routine
   * @param args
   *          we ignore this parameter for now */
  public static final void main(String[] args) {
    int[] array = new int[5]; // create an integer array for length 5

    System.out.print("Initially, all elements are 0:"); //$NON-NLS-1$
    for (int i = 0; i < array.length; i++) { // indexes are valid from 0 to array.length-1
      System.out.print(' ');
      System.out.print(array[i]); // access an element of array by using "[index]"
    } // prints "0 0 0 0 0"
    System.out.println();

    array[4] = 5; // set fifth element to 5
    array[2] = 3; // set third element to 3
    array[0] = 1; // set first element to 1
    array[1] = 2; // set second element to 2
    System.out.print("Now some elements are set: "); //$NON-NLS-1$
    for (int i = array.length; (--i) >= 0;) { // iterate backwards is slightly faster
      System.out.print(' ');                  // since we compare i with constance, not variable
      System.out.print(array[i]); // access an element of array by using "[index]"
    } // prints "5 0 3 2 1"
  }
}
```

Iterating over Array Elements

- We already know one way to iterate over the elements of an array `a`:

```
for(int i=0; i<a.length; i++){ System.out.print(a[i]); }
```

Iterating over Array Elements

- We already know one way to iterate over the elements of an array `a`:
  `for(int i=0; i<a.length; i++){ System.out.print(a[i]); }`
- We can also do it the other way around, which is even slightly faster:
  `for(int i=a.length; (--i)>= 0; ){ System.out.print(a[i]); }`, which is slightly faster since `--i` is compared with constant `0` instead of `array.length`

Iterating over Array Elements

- We already know one way to iterate over the elements of an array `a`: `for(int i=0; i<a.length; i++){ System.out.print(a[i]); }`
- We can also do it the other way around, which is even slightly faster: `for(int i=a.length; (--i)>= 0; ){ System.out.print(a[i]); }`, which is slightly faster since `--i` is compared with constant `0` instead of `array.length`
- Since Java 7, there is a very fast way of read-only forward array iteration: `for(int value : a){ System.out.print(value); }`.

Iterating over Array Elements

- We already know one way to iterate over the elements of an array `a`: `for(int i=0; i<a.length; i++){ System.out.print(a[i]); }`
- We can also do it the other way around, which is even slightly faster: `for(int i=a.length; (--i)>= 0; ){ System.out.print(a[i]); }`, which is slightly faster since `--i` is compared with constant `0` instead of `array.length`
- Since Java 7, there is a very fast way of read-only forward array iteration: `for(int value : a){ System.out.print(value); }`.
- This truly is read-only, changing `value` á la `value = ...` has no effect

Fast Read-Only Iteration

Listing: Fast Read-Only Iteration over String Array

```
/** Example for iterating over array elements them in read-only fashion */
public class StringArrayNewFastIterate {

  /** The main routine
   * @param args
   *           we ignore this parameter for now */
  public static final void main(String[] args) {
    String[] array = new String[4]; // create an string array for length 4

    array[0] = "hello␣"; //$NON-NLS-1$
    array[1] = "world"; //$NON-NLS-1$
    array[2] = ",␣it's␣"; //$NON-NLS-1$
    array[3] = "me!"; //$NON-NLS-1$
    for (String str : array) { // str takes on the values of the array elements
      System.out.print(str); // print the value of str during this iteration
    } // prints "hello world, it's me!"
  }
}
```

Fast Read-Only Iteration with Break

Listing: Fast Read-Only Iteration over int Array with Break

```
/** Example for allocating and using a int arrays and iterating over them */
public class IntArrayNewIterateBreak {

  /** The main routine
   * @param args
   *           we ignore this parameter for now */
  public static final void main(String[] args) {
    int[] array = new int[5]; // create an integer array for length 5

    array[4] = 5; // set fifth element to 5
    array[2] = 3; // set third element to 3
    array[0] = 1; // set first element to 1
    array[1] = 2; // set second element to 2

    for (int in : array) { // iterate over array items
      if (in == 3) { // exit loop here if in == 3
        break;
      }
      System.out.print(in);
    } // prints "12"
  }
}
```

Initializing During Construction

- If we create an array, its elements are set to `0` (or `null` in object arrays such as `String[]`, but we haven't learned yet what that is...)

Initializing During Construction

- If we create an array, its elements are set to `0` (or `null` in object arrays such as `String[]`, but we haven't learned yet what that is...)
- We can also specify all the values of the array directly.

Initializing During Construction

- If we create an array, its elements are set to `0` (or `null` in object arrays such as `String[]`, but we haven't learned yet what that is...)
- We can also specify all the values of the array directly.
- The syntax is to use a comma-separated value list in curly braces

Initializing During Construction

- If we create an array, its elements are set to `0` (or `null` in object arrays such as `String[]`, but we haven't learned yet what that is...)
- We can also specify all the values of the array directly.
- The syntax is to use a comma-separated value list in curly braces
- `int[] array = {1, 2, 3, }` creates an `int` array with the three elements 1, 2, and 3

Initialization of Array

Listing: A double array with specified initial values

```java
/** Example for initializing arrays during construction */
public class DoubleArrayInitialization {

  /** The main routine
   * @param args
   *             we ignore this parameter for now */
  public static final void main(String[] args) {
    double[] array = { 0.3d, 0.7d, 1.0d, 0.2d, 0.8d, 0.6d, 0.4d }; // initialize array

    double sum = 0d; // set sum to 0

    for (double value : array) { // iterate over array
      sum += value; // add each element
    }

    System.out.println(sum);// prints 4, the sum of all 7 array elements
  }
}
```

Multi-Dimensional Arrays

- An array can have elements which are arrays too

Multi-Dimensional Arrays

- An array can have elements which are arrays too
- This can be used to represent 2-D matrices or data of a pixel graphic, for instance

Multi-Dimensional Arrays

- An array can have elements which are arrays too
- This can be used to represent 2-D matrices or data of a pixel graphic, for instance
- All the things discussed so far then apply fully to the multi-dimensional arrays as well

Multi-Dimensional Arrays

- An array can have elements which are arrays too
- This can be used to represent 2-D matrices or data of a pixel graphic, for instance
- All the things discussed so far then apply fully to the multi-dimensional arrays as well
- Creating a 3-D integer array with $3 \times 4 \times 5$ elements is done via

```
int[][][] a = new int[3][4][5]
```

Multi-Dimensional Arrays

- An array can have elements which are arrays too
- This can be used to represent 2-D matrices or data of a pixel graphic, for instance
- All the things discussed so far then apply fully to the multi-dimensional arrays as well
- Creating a 3-D integer array with $3 \times 4 \times 5$ elements is done via

```
int[][][] a = new int[3][4][5]
```

- `a[1][2][4]` accesses the 5th element of the third one-dimensional array in the second two-dimensional array stored in the three-dimensional array

Multi-Dimensional Arrays

- An array can have elements which are arrays too
- This can be used to represent 2-D matrices or data of a pixel graphic, for instance
- All the things discussed so far then apply fully to the multi-dimensional arrays as well
- Creating a 3-D integer array with $3 \times 4 \times 5$ elements is done via

```
int[][][] a = new int[3][4][5]
```

- `a[1][2][4]` accesses the 5th element of the third one-dimensional array in the second two-dimensional array stored in the three-dimensional array
- Initialization can be done via nested braces, e.g.,

```
char[][] z = { {'a', 'b'}, {'c', 'd'}}
```

Multi-Dimensional Arrays

- An array can have elements which are arrays too
- This can be used to represent 2-D matrices or data of a pixel graphic, for instance
- All the things discussed so far then apply fully to the multi-dimensional arrays as well
- Creating a 3-D integer array with $3 \times 4 \times 5$ elements is done via
  ```
  int[][][] a = new int[3][4][5]
  ```
- `a[1][2][4]` accesses the 5th element of the third one-dimensional array in the second two-dimensional array stored in the three-dimensional array
- Initialization can be done via nested braces, e.g.,
  ```
  char[][] z = { {'a', 'b'}, {'c', 'd'}}
  ```
- We can allocate and initialize each element/dimension of the array separately as well

Allocation and Access of 2-D Array

Listing: Allocation and Access of 2-D Array

```java
/** Example for allocating and using a 2d-int arrays and iterating over them */
public class IntArray2DNewIterate {

  /** The main routine
   * @param args
   *          we ignore this parameter for now */
  public static final void main(String[] args) {
    int[][] array = new int[5][3]; // create a 2-d integer array for length 5*3

    for (int i = 0; i < array.length; i++) { // iterating over first dimension
      for (int j = 0; j < array[i].length; j++) { // iterating over second dimension
        array[i][j] = (i * j); // setting element
      }
    }

    // now we want to print the array
    for (int[] row : array) { // fast read-only iteration over "rows", i.e., first dim of 2d
      for (int value : row) { // fast read-only iteration over elements in selected row
        System.out.print('␣');
        System.out.print(value); // print the element value
      }
      System.out.println();
    }
  }
}
```

2-D Array with Different Row-Lengths

Listing: Allocation and Access of 2-D Array with Different Row-Lengths

```java
/** Example for allocating and using 2d-int arrays of different row size and iterating over them */
public class IntArray2DwithDifferentLengthRowsNewIterate {

  /** The main routine
   * @param args
   *          we ignore this parameter for now */
  public static final void main(String[] args) {
    int[][] array;// delare array variable

    array = new int[5][]; // allocate only first dimension!
    for (int i = 0; i < array.length; i++) { // iterating over first dimension
      array[i] = new int[(i + 1) * 2]; // allocate 2d dimensions (rows) of different length
      for (int j = 0; j < array[i].length; j++) { // iterating over second dimension
        array[i][j] = (i * j); // setting element
      }
    }

    // now we want to print the array
    for (int[] row : array) { // fast read-only iteration over "rows", i.e., first dim of 2d
      for (int value : row) { // fast read-only iteration over elements in selected row
        System.out.print('␣');
        System.out.print(value); // print the element value
      }
      System.out.println();
    }
  }
}
```

Initializing Multi-Dimensional Arrays During Creation

Listing: Initializing Multi-Dimensional Arrays During Creation

```
/** Example for initializing multi-dimensional double arrays during construction */
public class DoubleArray3DInitialization {

  /** The main routine
   * @param args
   *           we ignore this parameter for now */
  public static final void main(String[] args) {
    double[][][] array = { { { 0d, 3d }, { 1d, 2d } },
                           { { 0.3d, 0.7d }, { 1.0d, 0.2d, 0.8d }, { 0.6d, 0.4d } } };

    for (double[][] matrix : array) { // iterate first dimension: 2d-arrays
      System.out.println("---␣strange␣matrix␣^_^␣---"); //$NON-NLS-1$
      for (double[] row : matrix) { // iterate over second dimension: 1d-arrays, rows
        for (double value : row) { // iterate over values inside row
          System.out.print('␣');
          System.out.print(value);
        }
        System.out.println();
      }
    }
  }
}
```

Summary

- We have learned how to create "repetitive data structures": arrays.
- We have learned how to allocate arrays of fixed sizes or with sizes provided in variables/expressions.
- We have learned how to access (read, write) array elements
- We have learned how to loop over array elements
- We have learned how to initialize arrays upon creation
- We have learned how to deal with multi-dimensional, i.e., nested arrays

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog