Multi-dimensional Arrays and ArrayLists

Lecture 7 of TDA 540
Object-Oriented
Programming

Jesper Cockx

Fall 2018

Chalmers University of Technology — Gothenburg University

Last week: recap

Last week

- Creating and using arrays
- Value types vs reference types
- Array algorithms: average/standard deviation, removing duplicates, binary search

This week

- Multi-dimensional arrays
- The ArrayList class
- Wrapper classes
- File input and output

Multi-dimensional arrays

Two-dimensional arrays

A two-dimensional array = an array of arrays:

```
int[][] data = {
                  { 16,  3,  2, 13 },
                  {  5, 10, 11,  8 },
                  {  9,  6,  7, 12 },
                  {  4, 15, 14,  1 },
               };
```

Lab assignment 4: image processing

A grey-scale image can be represented as a 2-dimensional array of values 0-255:

- 0 = black pixel
- 255 = white pixel

Create a blank image:

```
int[][] grayImage =
  new int[HEIGHT][WIDTH];
```

Creating an empty two-dimensional array

```
// Create a matrix with
// 5 rows and 7 columns
int[][] matrix = new int[5][7]

// Set element at second row and
// third column to 101
matrix[1][2] = 101;
```

Iterating over a two-dimensional array

```
int[][] matrix = ...;
int rows    = matrix.length;
int columns = matrix[0].length;

for (int row = 0; row < rows; row++) {
  for (int col = 0; col < columns; col++) {
    // do something with matrix[row][col]
  }
}
```

Example: check if a matrix is symmetric

```
boolean isSymmetric(int[][] matrix) {
  int rows    = matrix.length;
  int columns = matrix[0].length;
  if (rows != columns) { return false; }
  for (int row = 0; row < rows; row++) {
    for (int col = 0; col < columns; col++) {
      if (matrix[row][col] != matrix[col][row]) {
        return false;
      }
    }
  }
  return true;
}
```

Question: how to make this more efficient?

Transpose of a matrix

```java
// precondition: the input is a square matrix
void transpose(int[][] matrix) {
  for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix.length; j++) {
      int tmp = matrix[i][j];
      matrix[i][j] = matrix[j][i];
      matrix[j][i] = tmp;
    }
  }
}
```

Question: what is wrong with the above code?

Live coding: matrix operations

Assignment: write methods that compute the sum and product of two matrices.

$$C = A \times B \text{ is defined as}$$

$$C[i][j] = \sum_{k=0 \ldots n} A[i][k] \cdot B[k][j]$$

Ragged arrays

Rows of a multi-dimensional array can have different lengths:

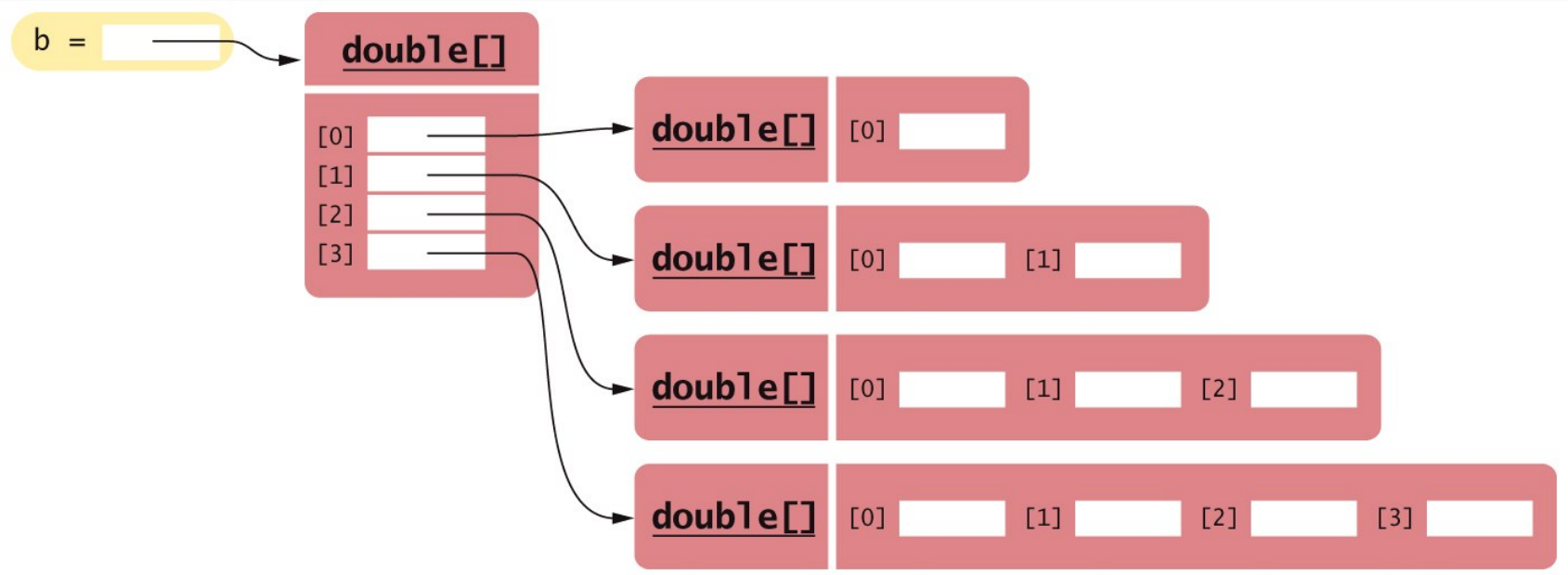

Figure 6.15

Ragged arrays

```
// Create new table with 4 rows
double[][] table = new int[3][];

// Create a new array for each column
for (int i = 0; i < table.length; i++) {
  table[i] = new double[i+1];
}
```

Neighboring elements

Accessing neighboring elements in a matrix:

`x[i-1][j-1]`	`x[i-1][j]`	`x[i-1][j+1]`
`x[i][j-1]`	`x[i][j]`	`x[i][j+1]`
`x[i+1][j-1]`	`x[i+1][j]`	`x[i+1][j+1]`

Warning: some neighbors may not be there at the border of the matrix!

Arrays with dimension > 2

Arrays can have more dimensions than 2:

```
String[][][] rubik =
  new String[6][3][3];
rubik[0][0][0] = "Yellow";
rubik[0][0][1] = "Green";
rubik[0][0][2] = "Blue";
rubik[0][1][0] = "Orange";
rubik[0][1][1] = "Red";
// ...
```

Lab assignment 4: image processing

A color image can be represented as a 3-dimensional array of values 0-255:

- `{0,0,0}` = black pixel
- `{255,255,255}` = white pixel
- `{255,0,0}` = red pixel
- `{0,255,0}` = green pixel
- `{0,0,255}` = blue pixel

Create a blank image:

```
int[][] colorImage =
  new int[HEIGHT][WIDTH][3];
```

15 min. break

Kahoot! Multi-dimensional arrays

The `ArrayList` class

Static vs dynamic data structures

Arrays are a static data structure: the length is fixed upon creation.

To add or remove elements in the middle, we need to:

- keep track of the number of 'used' elements
- shift elements to the right/left when adding/removing an element in the middle
- create a new array when the current one is full

Static vs dynamic data structures

Arrays are a static data structure: the length is fixed upon creation.

To add or remove elements in the middle, we need to:

- keep track of the number of 'used' elements
- shift elements to the right/left when adding/removing an element in the middle
- create a new array when the current one is full

...or just use `ArrayList` from `java.util`!

The `ArrayList` class

```java
// Create a new ArrayList and add some elements:
ArrayList<String> food = new ArrayList<String>();
food.add("Pizza");
food.add("Cheese");

// Get the length of the ArrayList:
int length = food.size();

// Get element at position 0
String someFood = food.get(0);

// Modify element at position 1
food.set(1, "Chocolate");
```

The `ArrayList` class (cont.)

```java
// Add an element at position 1
food.add(1, "Eggs");

// Delete element at position 0
food.remove(0);

// Search list
boolean hasCheese = food.contains("Cheese");

// Find position of first occurrence
int chocolatePosition = food.indexOf("Chocolate");
```

The `ArrayList` class (cont.)

```java
// Check if an ArrayList is empty
boolean noMoreFood = food.isEmpty();

// Prints ["Eggs","Chocolate"]
String foodString = food.toString();

// Remove all elements
food.clear();
```

Example: filtering an `ArrayList`

```
// Remove all the uneven elements from
// the given ArrayList
// Precondition: values != null
// Postcondition: values contains only even elements
static void filterEven(ArrayList<Integer> values) {
    for (int pos = 0; pos < values.size(); pos++) {
        if (values.get(pos) % 2 != 0) {
            values.remove(pos);
        }
    }
}
```

Question: What is wrong with this code?

Generic classes

`ArrayList` is a generic class: for any class `A`, we have a type `ArrayList<A>`.

`A` cannot be a primitive type:

```
ArrayList<int> myArrayList =
  new ArrayList<int>();
```

This gives a compilation error!

Solution: use the wrapper class `Integer` instead.

Wrapper classes

For each primitive type in Java, there is a corresponding wrapper class:

Primitive type	Wrapper class
`int`	`Integer`
`float`	`Float`
`double`	`Double`
`boolean`	`Boolean`
`char`	`Character`
⋮	⋮

Using wrapper classes

```
ArrayList<Integer> list =
  new ArrayList<Integer>();

list.add(new Integer(42));
list.add(new Integer(43));
int x = list.get(0).toValue();
```

Automatic boxing and unboxing

Values of a primitive type and its wrapper class can be used interchangeably:

```
ArrayList<Integer> list =
  new ArrayList<Integer>();

list.add(42);          // auto-boxing
list.add(43);          // auto-boxing
int x = list.get(0); // auto-unboxing
```

Arrays vs `ArrayList`

Array	ArrayList
`int[] list;`	`ArrayList<Integer> alist;`
`list = new int[LENGTH];`	`alist = new ArrayList<Integer>();`
`int length = list.length;`	`int length = alist.size();`
`int x = list[i];`	`int x = alist.get(i);`
`list[i] = x;`	`alist.set(i,x);`
???	`alist.add(x);`
???	`alist.remove(i);`

`list = {1,2,3};`	`alist = new ArrayList<Integer>();`
	`alist.add(1);`
	`alist.add(2);`
	`alist.add(3);`

Live coding: insertion sort

Assignment: implement the following two functions:

- Insert a number into a *sorted* `ArrayList`, making sure the list remains sorted.
- Sort an `ArrayList` by creating a new list and inserting the elements one by one.

File input and output

Ways to communicate input and output

- Command line (`System.in` & `System.out`)
- Graphical user interfaces (`JOptionPane`)
- Reading and writing files
- Transmitting data over the network
- Getting input/output from another program

Ways to communicate input and output

- Command line (`System.in` & `System.out`)
- Graphical user interfaces (`JOptionPane`)
- Reading and writing files
- Transmitting data over the network
- Getting input/output from another program

Reading files

To read a text file, we combine `File` and `Scanner`:

```
public static ArrayList<Integer>
  readNumbers(String fileName)
  throws FileNotFoundException {
    File myFile = new File("data.txt");
    Scanner input = new Scanner(myFile);
    ArrayList<Integer> numbers =
            new ArrayList<Integer>();
    while (input.hasNextInt()) {
      numbers.add(input.nextInt());
    }
    return numbers;
}
```

Writing files

To write to a text file, we use the class `PrintWriter`:

```
public static void
  makeRandomFile(String filename)
  throws FileNotFoundException {
    PrintWriter writer = new PrintWriter(filename);
    for (int i = 0; i < 10000; i++) {
      int x = (int) Math.random() * 100;
      writer.println(x);
    }
    writer.close();
}
```

Putting it all together

Demo code: read out a list of integers from a file and print the sorted values.

What's next?

Next lecture: **Testing and error handling**.

To do:

- Read the book:
  - Today: sections 6.7-6.8
  - Next lecture: chapter 7
- Hand in the third lab assignment (if you haven't already)
- Start on the fourth lab assignment