© Manfred Kerber
Ian Batten
Joseph Gardiner

School of Computer Science
University of Birmingham

Worksheet 2

MSc/ICY Software Workshop

Assessed Exercise: 10% of this term's continuous assessment mark.

Submission: Thursday 23 October 2014 2pm

5% late submission penalty within the first 24 hours. No submission after 24 hours. JUnit tests and JavaDoc comments are mandatory. All submissions must pass the tests provided on 16 October. Follow the submission guidelines on `http://www.cs.bham.ac.uk/internal/courses/java/msc/submission.php`.

Exercise 1: (Basic, 25%) Write a `static` method that computes whether the results in the Software Workshop module are sufficient to pass or not. The parameters of the method are: `continuousAssessment1`, `continuousAssessment2`, `exam`, `teamWork` (counting 12.5%, 7.5%, 70%, 10%, respectively to the overall mark), and `studentCategory` (which has to be `"MSc"` or `"ICY"`). A student passes the module if their overall mark is greater equal 50 for an MSc student or greater equal 40 for an ICY student. The method should return `true` if the student passes and `false` otherwise.

Exercise 2: (Basic, 25%) Write a `static` method `countOccurences` that counts the occurrences of a string in one-dimensional array of strings and returns the corresponding value. For instance, let us define

```
private String[] stringArray = {"the", "cat", "sat", "on", "the", "mat"};
```

Then your method should return:

```
countOccurences("dog", stringArray)  ⇝  0
countOccurences("cat", stringArray)  ⇝  1
countOccurences("the", stringArray)  ⇝  2
```

Exercise 3: (Medium, 15%) Eratosthenes of Cyrene invented more than 2000 years ago a method to determine all prime numbers up to a maximum `n`. In modern terminology, you first write the numbers between `2` and `n` in a list, then the left most number is a prime number and you delete all its multiples from the list, the next left most number is again a prime number and you delete all its multiples from the list. You continue until you have reached the end of the list. Actually you can stop when the next prime number in the list is bigger than the square root of `n`.

Use an array `prime` of `boolean` values to represent this. Initially the values of `prime[0]` and `prime[1]` are set to false (since `0` and `1` are no prime numbers), and those for `prime[2]` through `prime[n]` are set to true (since all these are candidates). Rather than deleting elements from the list your method should iterate through the left most elements in the array – starting with `2` – which have a value of `true` and set the values of its multiples in the array to `false`. Finally the value of `prime[i]` is `true` if `i` is a prime number and `false` otherwise (for all `i` from `0` through `n`). For instance:

```
   0     1     2     3     4     5     6     7     8     9     10    11    12    13
  [false false true  true  true  true  true  true  true  true  true  true  true  true]
2->[false false true  true  false true  false true  false true  false true  false true]
3->[false false true  true  false true  false true  false false false true  false true]
```

`5` does not need to be processed since `5 > sqrt(13)`.

Exercise 4: (Medium, 15%) Bubble sort is a very simple sorting algorithm which swaps adjacent elements in an array until the array is sorted. Implement bubble sort in order to sort an array of strings with respect to increasing string length. If two elements have the same length they should not be swapped. E.g.:

```
     ["Java", "the", "array", "a", "softwareWorkshop", "to"]
->   ["the", "Java", "array", "a", "softwareWorkshop", "to"]
->   ["the", "Java", "a", "array", "softwareWorkshop", "to"]
->   ["the", "Java", "a", "array", "to", "softwareWorkshop"]
->   ["the", "a", "Java", "array", "to", "softwareWorkshop"]
->   ["the", "a", "Java", "to", "array", "softwareWorkshop"]
->   ["a", "the", "Java", "to", "array", "softwareWorkshop"]
->   ["a", "the", "to", "Java", "array", "softwareWorkshop"]
->   ["a", "to", "the", "Java", "array", "softwareWorkshop"]
```

Write a corresponding `static` method which takes as an argument an array of strings and returns the corresponding sorted array of strings.

Exercise 5: (Advanced, 20%) Let a fully solved Sudoku (of size 9×9) be given as a two-dimensional array of numbers between 1 and 9 so that each number occurs exactly once in each row, each column, and each of the 9 sub-squares.
For instance, the following is a fully solved Sudoku:

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8

Write a `static` method that checks for a 2-dimensional array of type `int` of size 9×9 whether it satisfies the conditions. Concretely, your method should take in as argument a corresponding array and return a two-dimensional `result` array of `boolean` values such that `result[0]` is a one-dimensional array of boolean values indicating for each of the rows 0 through 8 whether it satisfies the condition that each of the values 1 through 9 occurs exactly once in it or not. Likewise `result[1]` states this for the columns and `result[2]` for the nine bigger squares. List the rows top-down, the columns left-right, and the 3×3 squares as

0	1	2
3	4	5
6	7	8

. For the example all 27 values in the `result` array are `true`.
If you changed the 8 in the lower right corner from an 8 to a 9, e.g., the values of `result[0][8]`, `result[1][8]`, and `result[2][8]` would all be false, all the other entries in the `result` array would be true.
If, however you changed in the original Sudoku, the lower left corner from a 9 to an 8, e.g., the values of `result[0][8]`, `result[1][0]`, and `result[2][6]` would all be false, and all the other entries in the `result` array would be true.