© Manfred Kerber
Ian Batten
Joseph Gardiner

School of Computer Science
University of Birmingham

Worksheet 3

MSc/ICY Software Workshop

Assessed Exercise: 20% of this term's continuous assessment mark.

Submission: Thursday 13 November 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 30 October. Follow the submission guidelines on `http://www.cs.bham.ac.uk/internal/courses/java/msc/submission.php`.

Exercise 1: (Basic, 10%) Let an array of `int` of length at least `1` be given. Write a method `public static double average(int[] a)` which computes the average of the values stored in the array. (That is, you have to add up the values in the array and divide the result by the length of the array. Note that you should not use integer division.)

Exercise 2: (Basic, 40%)

(a) $e^x =$`exp(x)` can be computed by $\sum_{i=0}^{\infty} \frac{x^i}{i!}$. This value can be approximated by breaking off the sum as soon as the summand $\frac{x^i}{i!}$ falls below a given threshold. Use loops to write static methods `public static int powerIter(double x, int i)` for computing the power x^i, `public static int factorialIter(int n)` for computing the function $n!$, and `public static double expIter(double x, double threshold)` which computes `expIter(x)` until the summand is below the threshold. All three methods should be written in an iterative way.

(b) Write corresponding recursive methods `powerRec`, `factorialRec`, and `expRec`.

Exercise 3: (Medium, 20%) Write an iterative method, `digitTotalIter` (using a `while` loop), and a recursive method, `digitTotalRec`, to compute the sum of the digits in a positive integer. E.g., the digit total of `12345` is `1+2+3+4+5`, i.e., `15`.

Exercise 4: (Advanced, 15%)

`quickSort` is a sorting algorithm. The idea is essentially to reallocate the values into two sub-arrays of the array so that all elements that are smaller than or equal to a given element (the so-called 'pivot') are in the left sub-array and all elements greater than the pivot are in the right sub-array. Then the algorithm is called recursively on the left and on the right sub-arrays. If the pivot is always chosen so that the left and right sub-arrays are of approximately the same length then the algorithm is very efficient. In consequence, the choice of the pivot is important. For this exercise choose the pivot as the middle most element of the array or sub-array, respectively (i.e., for an array `a` of size `n` as `a[n/2]`). Note that the pivot position itself should be neither in the left nor in the right sub-array in order to guarantee termination.

For details see [Deitel & Deitel, Java – How to Program, 8th Ed., end of Chapter 19, Exercise 19.10], or `http://en.wikipedia.org/wiki/Quicksort`.

Implement a static method `quickSort` which sorts an array of strings according to the lengths of the strings.

Exercise 5: (Advanced, 15%) The Tower of Hanoi (see, `http://en.wikipedia.org/wiki/Tower_of_Hanoi`) is a puzzle in which n discs of different size are stacked on top of each other in decreasing size. They have to be moved to another stack, making use of one additional position only. The following rules must be observed: only one disk can be moved at a time; only the top most disk of a stack can be moved to another stack; and no disk must be placed on top of a smaller disk. Let us assume the positions are 1, 2, and 3, and let us assume furthermore that initially all n disks are at position 1.

Represent the n disks by a one-dimensional array `hanoi` of size n with values 1, 2, and 3, depending of whether disk `i` is at position 1, 2, or 3, respectively. Write static methods `init(n)` which puts all disks initially at position 1, and `moveAll(hanoi, i, j, k)` which moves the top most `k` disks at position `i` to position `j`. Furthermore write a static method `trace(n)` which returns all the moves as a string, e.g., `trace(3)` should return the string:

```
"1 1 1
3 1 1
3 2 1
2 2 1
2 2 3
1 2 3
1 3 3
3 3 3"
```