Deadline for evaluation: 21-10-2016

These exercises must be solved with a **test driven approach**: **one single** solution must contain one subproject for each of the exercises plus one unit testing project.

Exercise 01

Write two methods that, given two numbers find the **greater** and the **smaller** of them **without using conditional** statements. Call this method from a Main that reads two numbers from the console and prints the result. While reading from the Console you will have to manage possible wrong format inputs by the user.

Exercise 02

Write a method that calculates the first 100 numbers in the **Fibonacci sequence**: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, … More about the **Fibonacci sequence** can be found in Wikipedia at:http://en.wikipedia.org/wiki/Fibonacci_sequence. For the solution of the problem use 2 temporary variables in which store the last 2 calculated values and with a loop calculate the rest (each subsequent number in the sequence is a sum of the last two). Use a **`for-loop`** to implement the repeating logic.

Exercise 03

Write a method that gets the coefficients ***a***, ***b*** and ***c*** of a quadratic equation: $ax^2 + bx + c$, calculates and finds its real roots (if they exist). Quadratic equations may have 0, 1 or 2 real roots. Call this method from a Main that reads the coefficients from the console and prints the results. While reading from the Console you will have to manage possible wrong format inputs by the user.

Exercise 04

Write a method that converts a number in the range [0…999] to words, corresponding to the English pronunciation. Examples:

- 0 --> "Zero"
- 12 --> "Twelve"
- 98 --> "Ninety eight"
- 273 --> "Two hundred seventy three"
- 400 --> "Four hundred"
- 501 --> "Five hundred and one"
- 711 --> "Seven hundred and eleven".

Exercise 05

Write a method that for a given integers **N** and **X**, **calculates the sum**: $S = 1 + 1!/X + 2!/X^2 + \ldots + N!/X^N$.

Exercise 06

Write four methods performing the following conversion of a given number: 1) **from decimal to binary notation, 2) from binary to decimal notation, 3) from decimal to hexadecimal notation, 4) from hexadecimal to decimal notation**. Read in Wikipedia what **numeral systems** are: http://en.wikipedia.org/wiki/Numeral_system. After that consider how you can **convert numbers from decimal numeral system to another**.

Deadline for evaluation: 21-10-2016

Exercise 07

Write a method that by a given integer **N** prints the numbers from 1 to N in **random order**. Search in the Internet for information about **the class** `System.Random`. Create an **array with N elements** and write in it the numbers from 1 to N. The idea is to start from the last element, swap it with a randomly selected element from the whole array (including last). Now consider the array from 0 to n-2 (size reduced by 1), and repeat the process till we hit the first element.

Exercise 08

Write two methods that given two numbers find their **greatest common divisor (GCD)** and their **least common multiple (LCM)** respectively. You may use the formula **LCM(a, b) = |a*b| / GCD(a, b)**.

Exercise 09

Write a method, which finds the **maximal sequence of increasing elements** in an array **arr[n]**. It is not necessary the elements to be consecutively placed. E.g.: {9, 6, **2**, 7, **4**, 7, 6, **5**, **8**, 4} -> {2, 4, 5, 8}.
We can solve the problem with **two nested loops** and one more array **len[0…n-1]**. In the array **len[i]** we can keep the length of the longest consecutively increasing sequence, which starts somewhere in the array (it does not matter where exactly) and ends with the element **arr[i]**. Therefore **len[0]=1**, **len[x]** is the maximal sum **max(1 + len[prev])**, where **prev < x** and **arr[prev] < arr[x]**. Following the definition, we can calculate **len[0…n-1]** with two nested loops: the **outer loop** will iterate through the array **from left to right** with the loop variable **x**. The **inner loop** will iterate through the array from the start to position **x-1** and searches for the element **prev** with maximal value of **len[prev]**, where **arr[prev] < arr[x]**. After the search, we initialize **len[x]** with 1 + the biggest found value of **len[prev]** or with **1**, if such a value is not found.
The described algorithm **finds the lengths of all maximal ascending sequences**, which end at each of the elements. The biggest one of these values is the length of the **longest increasing sequence**. If we need to find **the elements themselves**, which compose that longest sequence, we can start from the element, where the sequence ends (at index **x**), we can print it and we can search for a previous element (**prev**). By definition **prev < x** and **len[x] = 1 + len[prev]** so we can find **prev** with a **for**-loop from **1** to **x-1**. After that we can repeat the same for **x=prev**. By finding and printing the previous element (**prev**) many times until it exists, we can find **the elements, which compose the longest sequence** in reversed order (from the last to the first).

Exercise 10

Write four methods for creating **square matrices** like those in the **figures below** and a Main that prints them formatted to the console asking to the user the size of the square. The size of the matrices will be read from the console. E.g. matrices with size of 4 x 4:

Deadline for evaluation: 21-10-2016

a)

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

b)

1	8	9	16
2	7	10	15
3	6	11	14
4	5	12	13

c)

7	11	14	16
4	8	12	15
2	5	9	13
1	3	6	10

d)*

1	12	11	10
2	13	16	9
3	14	15	8
4	5	6	7

a), b), c) Think about appropriate **ways for iterating through the matrices** with **two nested loops**. d) We can start from (0, 0) and go **down N times**. Therefore, go to the **right N-1 times**, after that **up N-1 times**, after that **left N-2 times**, after that **down N-2** times and etc. At each iteration we place the next number in a sequence 1, 2, 3, …, N in the cell, which we are leaving.