Laboratory Assignment #2 Wrocław, 12.10.2021

Eclipse Environment
The Simple Functions

Task 1

Expand the following code of the functions so they perform operations that match their names.

```
public class Lab2 {
      static void printMaxValue(int [] arr) {
            // ... insert code here
      }
      static void printMaxValue(int [][] arr) {
            // ... insert code here
      }
      static  int getOccurenceNumber (int[] arr, int value2Check) {
            // ... insert code here
      }
      static  void printMostCommonDigit(int[] arr) {
            // ... insert code here
      }
      static  void printMostCommonDigit(int[][] arr) {
            // ... insert code here
      }
      static void printSorted(int [] arr) {
            // ... insert code here
      }

      public static void main(String[] args) {
            int oneDim[]= {1, 10, 23, 87, 14, 1, 6, 177, 71};
            int twoDim[][] = {{1, 2, 12, 2}, {87}, {7, 12,14}, {1,8, 9, 8,}};
            printMaxValue(oneDim);
            printMaxValue(twoDim);
            printMostCommonDigit(oneDim);
            printMostCommonDigit(twoDim);
            printMostCommonNumber(twoDim);
            printSorted(oneDim);
      }
}
```

Remark:
In the implementation of the printSorted function use the bubble sort. It is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted. The algorithm, which is a comparison sort, is named for the way smaller or larger elements "bubble" to the top of the list.

Example [https://en.wikipedia.org/wiki/Bubble_sort]

Take an array of numbers "5 1 4 2 8", and sort the array from lowest number to greatest number using bubble sort. In each step, elements written in **bold** are being compared. Three passes will be required;

First Pass

(**5 1** 4 2 8) → (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.

(1 **5 4** 2 8) → (1 **4 5** 2 8), Swap since 5 > 4

(1 4 **5 2** 8) → (1 4 **2 5** 8), Swap since 5 > 2

(1 4 2 **5 8**) → (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass

(**1 4** 2 5 8) → (**1 4** 2 5 8)

(1 **4 2** 5 8) → (1 **2 4** 5 8), Swap since 4 > 2

(1 2 **4 5** 8) → (1 2 **4 5** 8)

(1 2 4 **5 8**) → (1 2 4 **5 8**)

Now, the array is already sorted, but the algorithm does not know if it is completed. The algorithm needs one additional **whole** pass without **any** swap to know it is sorted.

Third Pass

(**1 2** 4 5 8) → (**1 2** 4 5 8)

(1 **2 4** 5 8) → (1 **2 4** 5 8)

(1 2 **4 5** 8) → (1 2 **4 5** 8)

(1 2 4 **5 8**) → (1 2 4 **5 8**)

Task 2

Using the debugger:

- Demonstrate and describe the operation of the following debugger actions:

DEBUGGING		
`F5`	Step into	
`F6`	Step over	
`F7`	Step return	Also known as step out
`F8`	Resume	
`Ctrl-R`	Run to line	
`F11`	Debug last launched	
`Ctrl-Shift-B`	Toggle line breakpoint	Toggles a breakpoint on the current line

- Set conditional breakpoints
- Evaluate expressions
- Modify value of variables
- Renaming variables or functions

Andrzej Siemiński