Freie Universität Berlin, Software Engineering Research Group
Prof. Dr. Lutz Prechelt, Franz Zieris

Practice sheet 12 **Cleanroom SE: Practice** due on 18.01.2016

Please make sure to always prepare your answers and solutions in a way that you are able to present them to your classmates and discuss your solution process effectively. Remember to always list any used material and references.

Task 12-1: Implement Mastermind with Cleanroom

Implement the logical game Mastermind by using the Cleanroom method (but *without* statistical testing).

Program requirements

1. *Rules of the game*: Comply with the rules of the following variant of Mastermind[1].
   - The game is played using *a decoding board*, with a shield at one end covering a row of **four large holes**, and a set of additional rows containing four large holes next to a set of four small holes; *code pegs* of **six different colors**, with round heads, which will be placed in the large holes on the board; and *key pegs*, some colored black, some white, which are flat-headed and smaller than the code pegs; they will be placed in the small holes on the board.
   - In each game, one player becomes the codemaker, the other the codebreaker. The codemaker chooses a pattern of four code pegs. **Duplicates are *not* allowed**. The chosen pattern is placed in the four holes covered by the shield, visible to the codemaker but not to the codebreaker.
   - The codebreaker tries to guess the pattern, in both order and color. Each guess is made by placing a row of code pegs on the decoding board. Once placed, the codemaker provides feedback by placing up to four key pegs in the small holes of the row with the guess. A colored or black key peg is placed for each code peg from the guess which is correct in both color and position. A white key peg indicates the existence of a correct color code peg placed in the wrong position. The order of the key pegs does not correspond to the order of the code pegs.
   - Once feedback is provided, another guess is made; guesses and feedback continue to alternate until the codebreaker guesses correctly.
2. *Play modes*: On startup, your program should offer two play modes:
   (a) Play mode 1: The program is the codemaker, i.e. it devises of a combination of colors the user has to break. After each guess the program judges the moves.
   (b) Play mode 2: The program is the codebreaker and tries to break the combination the user made up. After each guess by the program the user judges the moves.

   In the tutorial we will compare the strength of the programs. Think about a reasonable strategy yourselves[2].
3. *Programming language*: Write your program in Java.
4. *User interface*: Implement a *character-based* user interface for the program. Use the following descriptions on the console:
   (a) Use the numbers **`1`**, **`2`**, **`3`**, **`4`**, **`5`**, **`6`** to represent the colored code pegs.
   (b) Use the letters **`b`** and **`w`** for the black and white key pegs, and a dot **`.`** for an empty small hole.

[1] adapted from https://en.wikipedia.org/wiki/Mastermind_(board_game)
[2] You can definitely find various hints (or even implementations) on the web. However, don't be *that* guy: Do not spoil the party and learning effect by looking at these solutions.

(c) It is up to you whether you display the board entirely after each move or the moves are displayed one-by-one.

Example for the board's output after the third move:

```
1.   1234   www.
2.   3521   wbb.
3.   5316   www.
```

Procedure requirements

- *Cleanroom method*: Implement the program by using the Cleanroom method. Have another good look at the corresponding slides of the lecture first and use the half-formal verification presented there.

  In case the slides' information does not suffice, refer to the web for further sources or to the original Cleanroom Software Engineering Reference Model (http://www.sei.cmu.edu/reports/96tr022.pdf), in particular the sections *"Function Specification Process"* (pp. 50–55), *"Increment Design Process"* (pp. 79–85), and *"Correctness Verification Process"* (pp. 86–93).

- *No tests*: Do not execute the program during development! You should, however, develop your program in an IDE and therefore use a syntax checker. **The first program execution will take place during the tutorial!**

- *Teamwork*: Work in pairs. Your half-formal description should suffice to convince each other and later your fellow students in the tutorial of each step's correctness.

What to bring to the tutorial

Bring the following to the tutorial:

- The finished program: Not tested before, but executable on your laptop.
- The entire stepwise decomposition ("boxes") with the corresponding verifications as text document on your laptop. Be prepared to present it in the tutorial.