Exercise: The 'I' and 'D' in the SOLID acronym

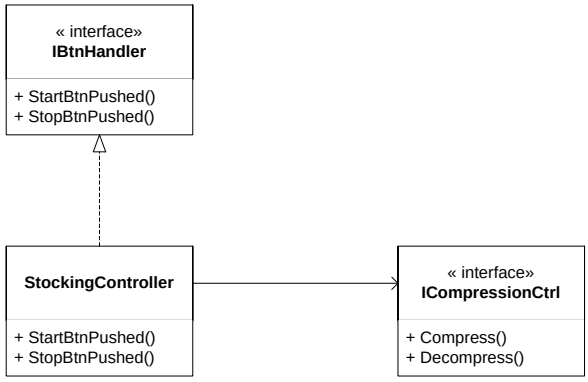
In this exercise, you will make acquaintance with the Interface Segregation Principle and the Dependency Inversion Principle. You should solve this exercise in 2 man teams.

As you go through this exercise, don't forget to stop and reflect on how you are using the design principles, and how they affect your design.

The situation:
You are developing the software for an electronic *compression stocking* (see e.g. http://en.wikipedia.org/wiki/Compression_stockings). Here are the initial requirements for the stocking:

1. When the stocking is relaxed, it shall be possible for the user to start the compression of the stocking via a START button on the stocking.
2. When compression is initiated, the pump shall run for 5 seconds. After this, the pump shall stop and the compression is considered complete.
3. When the stocking is compressed, it shall be possible for the user to decompress the stocking via a STOP button on the stocking.
4. When decompression is initiated, the pump shall run backwards for 2 seconds. After this, the pump shall stop and the decompression is considered complete.

Exercise 1: Design and implementation for initial requirements
An initial, non-complete design for the implementation of requirements 1-4 is given below. The source code for this design is available from BB.



Exercise 1A: Complete the design for the implementation of requirements 1-4 and document it in a class diagram. Use console input for START and STOP button inputs. As you design, remember the Dependency Inversion Principle which will lead you to a nice layered SW structure.

Exercise 1B: Implement your design from Exercise 1A. Be sure to update your class diagrams if you make any changes during implementation

Exercise 2: Additional requirements

The sales guys have been at it again – they've demonstrated the stocking to potential customers, and now they want additional features. This leads to the below new requirements for user information:

5. While compression is running, a green LED shall be lit and a vibrating device shall vibrate.
6. While decompression is running, a red LED shall be lit and a vibrating device shall vibrate.
7. When the stocking is compressed or relaxed, no LED shall be lit an no vibrating device shall vibrate

Exercise 2A: Design the required changes into your design and document it in a class diagram. Here's a hint: The activation/deactivation of the user information when compression is complete may require some callback on the `StockingController`. If so, remember the Interface Segregation Principle. If you did Exercise 1 correctly, there should mainly be design *additions* and only few changes

Exercise 2B: Implement your design from Exercise 2A. Be sure to update your class diagrams if you make any changes during implementation.

Exercise 3:

The customer now requires a second option for compression mechanism: Instead of using air, they require laces which wrap around the stocking and are tightened to compress the leg by means of a "lace tightening device", much like the one found on ski boots. This device (de)compresses the leg by tightening (loosening) the laces 40 "clicks", one click every 100 ms.

Implement the new compression mechanism into the existing design (It's really easy if you have DIP in place!)

Exercise 4 (advanced)

The idea of running the pump for such and such a time is sub-optimal. What is really needed is to implement a pressure sensor which could inform the system of the current pressure. The pump could then be started and kept running until the required pressure was obtained.

Exercise 4A: Design the necessary changes into the system. Note how it may be necessary to use threading to achieve the required functionality.