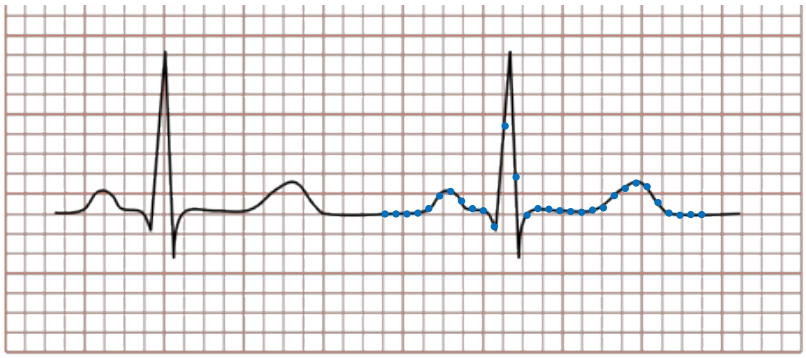
Exercise: Interfaces – Electrocardiography (ECG)

In this exercise, you will work with the collection and processing of *electrocardiography* (ECG) samples. In short, ECG is the process of recording the heart's electrical activity over time.

This exercise focuses on interfaces, which we will use to decouple our software and make it more flexible and "ready for change".

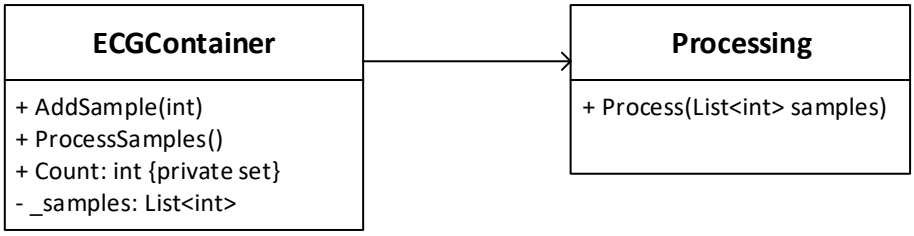
You must finish at least exercises 1-4 incl. of this exercise as they are used in a later exercise.

An *ECG signal* is a continuous signal with – in a healthy person – looks similar to the one below. We will create a small system that can receive and process samples of such a signal periodically.



Our aim in this exercise is to design and implement a system that will allow us to easily change how information about the signal is processed.

A simple design (implemented in the handout) is shown below. The `Process()` method of the `Processing` class simply prints the list of samples to the console. The `Count()` method returns the size of the `_samples` list and therefore has no setter method.



Exercise 1:
Consider the design above. As you can see, the two classes are tightly coupled. How can you use an *interface* to lower the coupling between the two classes?

Exercise 2:
Create a design (a UML class diagram) of the system, which uses an interface of the `Processing` class. Your class diagram should contain all methods for all classes and interfaces.

Exercise 3:
Implement your design. Use the provided `Main()` method in the Program.cs file, which tests your program by adding some samples and prints them afterward. Notice: Did you have to do any changes to the `Main()` method to use your new design? No? That's great! This means that we have implemented the design changes without affecting other parts of your program, namely `Main()`.

Exercise 4:
Now imagine that we must include a new way to process the ECG samples, namely an *extremes processing*: Given a list of samples, we should only print the *minimum* sample value and the *maximum* sample value.

What changes and additions are required to your existing classes and interface to accommodate the new kind of processing? Compare this to the old design - which design is more "flexible", i.e. ready for change? Why?

Add `ExtremesProcessing` to your class diagram from Exercise 2. Add the class to your implementation and test it using the same `Main()` program as before. Again note how the `Main()` program does not need to change to use the new feature.

Exercise 5 (optional):
Change your implementation so that the user can decide which reporting type to use when the program starts. Hint: This requires changes to `SampleContainer`'s constructor and therefore to the setup you do in the beginning of `Main()`.

Exercise 6 (optional):
Add a new way to process ECG samples to your design, `BPMProcessing`. This processing will report the *average pulse* of the ECG signal, defined as the average number of heartbeats per minute (BPM). To do this, you will need some additional info:

- The sampling rate is exactly 1000 Hz.
- Any sample with a value of 20 or below (the blue line on the figure below) can be considered as not being "on the peak" of the heartbeat.

After you have added `BPMProcessing` to your design, implement it and extend your `Main()` to include the selection of this processing.
Note, that you have to change the range of the generated samples to get any samples with a value above 20 and you should generate more than the 100 samples generated in the handout.