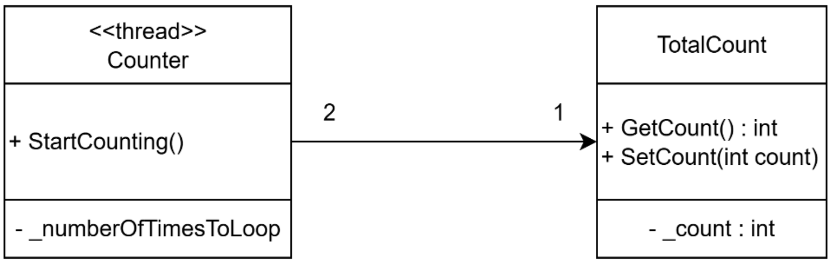
Exercise: Thread synchronization

In these exercises, you will work with C# threads, thread synchronization, multithreaded WPF applications (and the BackgroundWorker in the advanced exercises).

Exercise 1:
Create a console application.
Create a *TotalCount* class. It shall be very simple, with a private integer _*count* member variable and get/set methods, so you can read the _count value and set a new value to _count.
Create a *Counter* class, with a *StartCounting* method, that loops a given number of times and increments the *TotalCount* for each loop.
Create two threads.
One thread shall run a *Counter.StartCounting*, which loops 200000 times.
The other thread shall run a *Counter.StartCounting*, which loops 500000 times.
(hint: pass the number of times to loop as a parameter to the Counter. In the previous lesson, we saw 3 different ways to do that)

The design above can be shown in UML as:



Output the total count, when both threads have finished counting.
(hint: remember that we talked about joining a thread?)

What do you expect the total count to be?
Run your program a couple of times and observe the output.
What is the actual count?

Exercise 2:
Consider: What is the shared resource?
Change the TotalCount class implementation, so the total count becomes correct.

Exercise 3
A .zip file with 3 files containing card tuples are provided on Blackboard.
Each tuple is a line in the file in the form “SPADE, 10” or “HEART, 8”.

Write a console application (without threads), which answers the following questions:

- How many cards are there?
- What is the total sum of all SPADES?
- How many aces (1’s) are there?

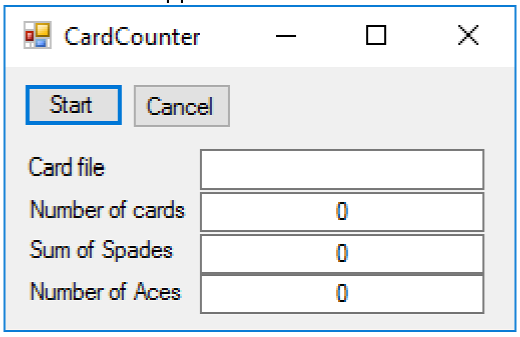
Create three threads and let each thread process one file. You are now counting in parallel!
Verify, that your parallel program gives the same results as you got when you counted without threads.
Hint! Try to design your application in a way, so you can reuse as much code as possible for exercise 4.

Exercise 4:
A .zip file with 3 files containing card tuples are provided on Blackboard.
Each tuple is a line in the file in the form "SPADE, 10" or "HEART, 8".
Pick **one** file to use for this exercise.

Write a Card Counting MAUI application, using a thread to count the cards, which answers the following questions:

- How many cards are there?
- What is the total sum of all SPADES?
- How many aces (1's) are there?

Your finished application should look somewhat like this:



Card file	
Number of cards	0
Sum of Spades	0
Number of Aces	0

Exercise 5:
Let the user select the source file, by adding an OpenFileDialog to the word count application.

Hint: https://www.wpf-tutorial.com/dialogs/the-openfiledialog/

Advanced exercise A1:
What impact do your solution to exercise 3 have on the potential parallelism in the program?
Can you come up with a faster way to get the results and still get the correct results?
(hint: can you minimize locking?)

Advanced exercise A2:
Examine the Task Parallel Library (find the documentation on MSDN). Solve exercise 3 using Tasks instead of Threads.

Advanced exercise A3:
Modify your card counting application, so you can select a folder and process all .txt files in the folder.

Advanced exercise A4:
Modify your card counting application, so you can select all three files.
Use a single BackgroundWorker to process the files one at the time.

Advanced exercise A5:
Modify your card counting application, so all files are processed in parallel, i.e. each file is processed by a separate BackgroundWorker.
Verify that the count is still correct.

Advanced exercise A6:
It has become increasingly difficult to sell the console chat program you wrote in the previous exercise.
Youngsters have stopped using the console… they want nice and shiny WPF programs instead.
And all the old people have either bought your program or are using Linux.

So…
Your mission, *should you choose to accept it*, is to create a WPF chat program.
When your program starts, it shall connect to a chat program on one of your fellow students computers.
The text you enter shall be sent to the other computer and displayed in a nice chat window.
Likewise, text entered on the other computer shall be sent to your computer and displayed in a nice chat window.

Hint: You will probably still need a *TCPListener* and *TCPClient* for your application. And multiple threads of course.