# Lab: Asynchronous Programming

This document defines the lab problems for the ["ASP.NET Fundamentals" Course @ SoftUni](https://softuni.bg/trainings/4953/asp-net-fundamentals-may-2025).

## Asynchronous Processing

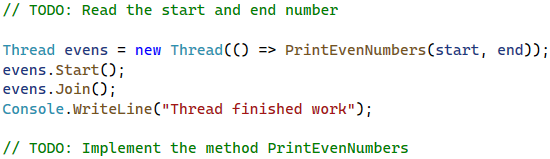
### Even Numbers Thread

Print **all** **even** numbers in a given **range**. Printing should be executed on a **separate** **thread**. After all numbers are printed print "**Thread finished work**".

#### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1  10 | **2**  **4**  **6**  **8**  **10**  **Thread finished work** |

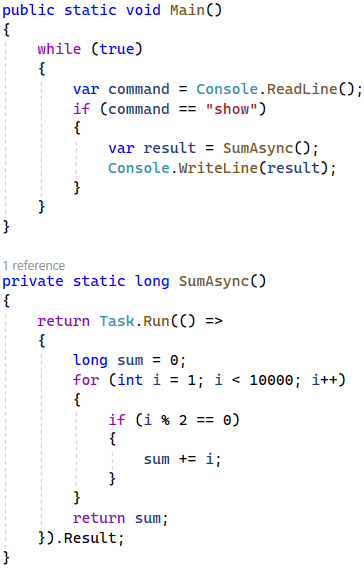
#### Hint



### Sum Evens in Range

Sum all even numbers in given range [**1** to **1000**]. Read commands and print the result only on command "**show**".

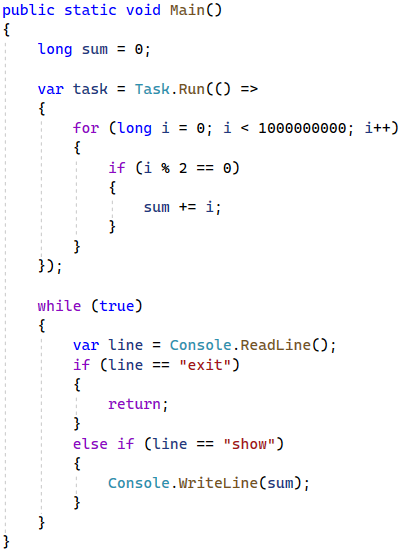
#### Solution



### Sum Evens in Background

This problem is similar to the previous one. You have to sum all even numbers in given range (**1** to **1000000000**), but this time leave the console interface unblocked while calculating the sum. Read commands and print the result only on command "**show**". Stop calculating on command "**exit**".

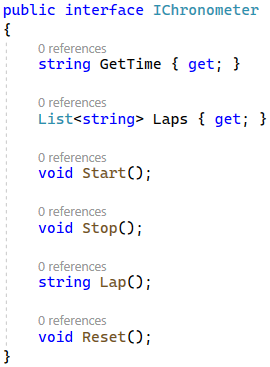
#### Solution



### Chronometer

The Chronometer is one of the easiest examples of an **asynchronous processes**. Let's implement a simple Chronometer.

Create an **interface** IChronometer like this:



Shape, rectangle

Description automatically generated... and implement a class **Chronometer**, that implements it.

Implement a program which provides a **Chronometer functionality**, that responds to several commands from the user input:

* start – starts counting time in milliseconds, seconds and minutes
* stop – stops the process of counting time, but the counted time remains
* lap – creates a lap at the current time
* laps – returns all of the currently recorded laps
* time – returns the currently recorded time
* reset – stops the Chronometer, resets the currently recorded time and deletes all of the currently recoded laps
* exit – stops and exits the program

Here is an example screenshot of the functionality:

The time is outputted in the following format: "{minutes}:{seconds}:{milliseconds}", each of them should be **padded** with **zeros**.

Upon **making** a **lap** you should print the **time** at which it was made.

Requesting **all laps** should print them in the following format:

Laps:  
0. {lap1}  
1. {lap2}  
...

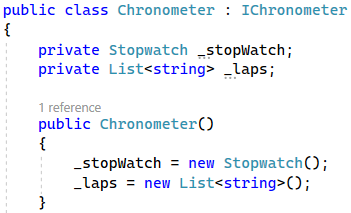
In case there are no laps, you should print "Laps: no laps".

#### Hints

Let's start implementing our **asynchronous chronometer**. First, we need to create the Chronometer **class**, which implements the IChronometer **interface**:



Use the Stopwatch **C#** **class**, which provides a **set of methods and properties** that you can use to accurately measure elapsed **time**. Create a **field** for the stopwatch. Also, create a **collection for the laps**. Initialize the **fields** in the **constructor** like this:



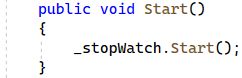
We have the GetTime **property**, which should return the **currently recorded time** since the start of the chronometer counter. Use the Elapsed **property** of the Stopwatch **class** to **get the total elapsed time**. This property returns the time as a TimeSpan, so you should convert it to string in the **correct format**. Do it like this:

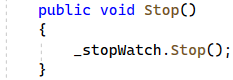


The other property we have is the Laps **property**. It should just **return the current laps collection**:

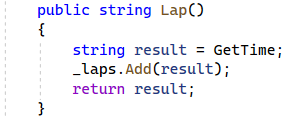


Next, we should **implement** the Start() and Stop() **methods** of the Chronometer **class**. The Stopwatch **class** has its own **methods for starting and stopping** – use them as shown below:

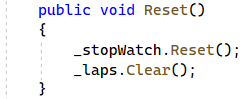




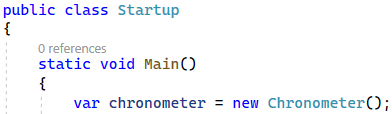
The Lap() **method** returns the **current elapsed time** as a string and **adds it to a collection of laps**. It uses the GetTime **property**:



Finally, we have the Reset() **method**, which should **invoke the** Reset() **method** of the Stopwatch **class** and **clear the laps collection**. Do it like this:



As we already have the Chronometer **class** let's use it and invoke its methods depending on **commands** from the console. **Instantiate the chronometer** in the Main() **method** of a **class** called StartUp:



Then, we will **read a command** from the console, until the "exit" **command**.



In the while **loop**, work with the chronometer depending on the **read command**. Don't forget that the Start() **command** of the Chronometer **class** should be **run as a task to be asynchronous**. After the while **loop** you should **stop the chronometer**. Complete the Startup **class** like this:

