Unzip [01.Multithreading.zip](https://epam.sharepoint.com/:u:/r/sites/NetMentoringprogramA2Belarus/Shared%20Documents/Module%201.%20Multi-threading/Task/01.multithreading.zip?csf=1&e=KMjwAn) archive. Restore packages via “dotnet restore”.

Implement the following tasks using templates from the Multithreading.sln.

## Task 1 (MultiThreading.Task1.100Tasks.csproj)

Write a program, which creates an array of 100 Tasks, runs them and waits all of them are not finished. Each Task should iterate from 1 to 1000 and print into the console the following string:

“Task #0 – {iteration number}”.

## Task 2 (MultiThreading.Task2.Chaining.csproj)

Write a program, which creates a chain of four Tasks.

1. First Task – creates an array of 10 random integer.
2. Second Task – multiplies this array with another random integer.
3. Third Task – sorts this array by ascending.
4. Fourth Task – calculates the average value.

All these tasks should print the values to console.

## Task 3 (MultiThreading.Task3.MatrixMultiplier.csproj, MultiThreading.Task3.MatrixMultiplier.Tests.csproj)

Write a program, which multiplies two matrices and uses class Parallel.

**Details**

Open *MultiThreading.Task3.MatrixMultiplier.csproj*.

a. Implement logic of MatricesMultiplierParallel.cs using Parallel class.

Make sure that all the tests within MultiThreading.Task3.MatrixMultiplier.Tests.csproj run successfully.

b. Create a test inside MultiThreading.Task3.MatrixMultiplier.Tests.csproj to check which multiplier runs faster. Find out the size which makes parallel multiplication more effective than the regular one.

## Task 4 (MultiThreading.Task4.Threads.Join.csproj)

Write a program which recursively creates 10 threads.

Each thread should be with the same body and receive a state with integer number, decrement it,

print and pass as a state into the newly created thread.

Use Thread class for this task and Join for waiting threads.

Implement all the following options:

- a) Use Thread class for this task and Join for waiting threads.

- b) ThreadPool class for this task and Semaphore for waiting threads.

## Task 5 (MultiThreading.Task6.Continuation.csproj)

Write a program which creates two threads and a shared collection:

the first one should add 10 elements into the collection and the second should print all elements in the collection after each adding.

Use Thread, ThreadPool or Task classes for thread creation and any kind of synchronization constructions.

## Task 6

Create a Task and attach continuations to it according to the following criteria:

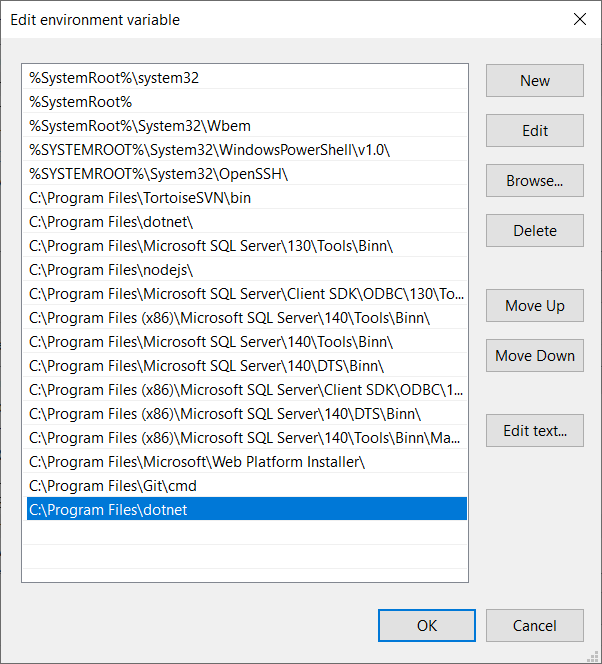
a. Continuation task should be executed regardless of the result of the parent task.

b. Continuation task should be executed when the parent task finished without success.

c. Continuation task should be executed when the parent task would be finished with fail and parent task thread should be reused for continuation

d. Continuation task should be executed outside of the thread pool when the parent task would be cancelled

Demonstrate the work of each case with console utility.



Please make sure that Path variable contains link to dotnet folder.