# Seminar 13\_2018-2019. Programare Concurenta in C#

Partea I - probleme simple

1. *Clasa Task. Metode asincrone cu async await*

Consideram urmatoarele doua metode:

private static long Fibonacci(long n)

{

if (n > 50)

{

throw new ArgumentException();

}

if (n == 1 || n == 2)

{

return 1;

}

else

{

return Fibonacci(n - 1) + Fibonacci(n - 2);

}

}

-------------------------------------------------------------------------

private static long Factorial(long n)

{

long p = 1;

for (long i = 1; i <= n; i++)

p = p \* i;

return p;

}

1. Implemetati metoda definita de urmatoarea signatura:

public static Task ComputeFibonaci1(long n),

care returneaza un Task ce calculeaza si afiseaza cel de-al n-lea termen din sirul lui Fibonaci.

1. Implemetati metoda definita de urmatoarea signatura:

public static Task<long> ComputeFibonaci2(long n),

care returneaza un Task ce calculeaza si returneaza cel de-al n-lea termen din sirul lui Fibonaci.

1. Implemetati metoda definita de urmatoarea signatura:

public static Task ComputeFactorial1(long n),

care returneaza un Task ce calculeaza si afiseaza factorialul numarului n.

1. Implemetati metoda definita de urmatoarea signatura:

public static Task<long> ComputeFactorial2(int n),

care returneaza un Task ce calculeaza si returneaza factorialul numarului n.

1. Implementati o metoda *Operations1()* care calculeaza Factorial(n) si Fibonacci(n), pt *n=1,10,* pornind pt fiecare numar *n* doua Task-ri, via metodele ComputeFibonaci1 si ComputeFactorial1. Ce observati?

public static void Operations1()

{

for (int i = 1; i <= 10; i++)

{

Task f = ComputeFactorial1(i);

f.Start();

Task f2 = ComputeFibonaci1(i);

f2.Start();

}

}

1. Implementati o metoda *Operations2()* care calculeaza Factorial(n) si Fibonacci(n), pt n=1,10, pornind pt fiecare n doua Task-ri, via metodele ComputeFibonaci2 si ComputeFactorial2. Ce observati?
2. Implementati o metoda asincrona avand signatura

public async static Task<long> ComputeFibonaciAsync(long n)

care determina al n-lea termen din sirul lui Fibonaci.

1. Implementati o metoda *Operations3()* care calculeaza Factorial(n) si Fibonacci(n), pt n=1,10, apeland metodele ComputeFibonaciAsync si Factorial(n). Ce observati?

1. Implementati o metoda asincrona care returneaza dimensiunea paginii web <http://msdn.microsoft.com>. Metoda TestMsdnLenght va apela metoda asincrona GetMsdnLengthAsync si intre timp va executa alte operatii.

private async static Task<int> GetMsdnLengthAsync()

Partea a II a - Completati aplicatia de la seminarul 12 astfel:

1. *Definiti o clasa ConcurrentTaskRunner care simuleaza un ThreadPool pentru executia concurenta a mai multor Task-ri astfel:*
2. ConcurrentTaskRunner incapsuleaza o coada de actiuni: Queue<Action> itemQ, si a) un vector de workers, Thread[] workers (Action – delegate predefinit in C#, for Java use Runnable), respective b) un ExecutorService (Java). La crearea unui obiect de tipul ConcurrentTaskRunner se vor crea workerCount fire de executie (thread-ri), unde workerCount este specificat ca parametru in constructor, si se vor porni firele de executie create (via metoda start), avand ca metoda delegat metoda Consume, care va executa actiuni (task-ri).
3. Implementati sincronizarea firelor de executie si asigurati accesul exclusiv al unui thread, la un moment dat, la coada itemQ, folosind Monitor (Pulse Wait) – C#, respectiv ReentrantLock, Condition (await si signal) - Java:
   1. Definiti un obiect lacat, readonly object locker = new object(), respectiv
   2. Lock **locker** = **new** ReentrantLock(); pt Java
   3. Simulati problema producatorului – consumatorului, via metodele EnqueueItem si Consume:

/// <summary>

/// Producer: add an action to be added to the Queue

/// </summary>

/// <param name="item">item - delegate method to be consumed</param>

public void EnqueueItem(Action item) //Producer

/// <summary>

/// Consumer: Consume an action from the queue

/// </summary>

void Consume() //Consumer

Cele workerCount fire de executie vor avea fiecare metoda Consume ca metoda delegate care se va executa la pornirea firelor (thread-ri). Daca coada este vida se va astepta, via Monitor.Wait, pana cand un alt fir de executie va adauga un element (Action) in coada.

Important: Un thread ce deţine exclusivitatea asupra unui obiect de tip monitor poate ceda temporar exclusivitatea accesului. Apelul metodei *wait(),* are drept rezultat pentru thread-ul curent oprirea temporară a execuţiei şi trecerea în starea *waiting(),* iar reluarea accesului exclusiv are loc atunci când threadul se trezeşte ca urmare a unui mesaj de anunţ generat de un alt thread activ ce va executa metoda *resume().* În urma acestui eveniment thread-ul recapătă exclusivitatea cedată şi continuă execuţia oprită anterior după apelul *wait().*

1. Implementati o metoda *Shutdown* pentru finalizarea procesului de executie a task-rilor. Metoda va adauga, pentru fiecare fir de executie pornit, elemenul null in coada itemQ si va astepta terminarea executiei tuturor thread-rilor via metoda join()

/// <summary>

/// // Stop the process of task execution and wait till all already existing tasks

/// in the threadpool are done in case of waitForWorkers param is true

/// </summary>

/// <param name="waitForWorkers"></param>

public void Shutdown(bool waitForWorkers)

1. Scrieti o aplicatie de test care creeaza un ConcurrentTaskRunner ce porneste 2 fire de executie (via metoda constructor) si asteapta executia unor MessageTask-ri ce vor fi adaugate ulterior in coada, apoi opreste executia acestor Task-ri, asteptand finalizarea executiei lor. (Metoda *ShutDown*).
2. 11 in Java si C#

CompletableFuture<Integer> AccessTheWebAsync()

{

AsyncHttpClient asyncHttpClient = new DefaultAsyncHttpClient();

return asyncHttpClient

.prepareGet("http://msdn.microsoft.com")

.execute()

.toCompletableFuture()

.thenApply(Response::getResponseBody)

.thenApply(String::length);

}

async static Task<int> GetMsdnLengthAsync()

{

HttpClient client = new HttpClient();

Task<string> getStringTask = client.GetStringAsync("http://msdn.microsoft.com"); //3

string urlContents = await getStringTask;

Console.WriteLine("after await");

return urlContents.Length;

}