WORKSHOP				
Overall Objective	Application to the unit of the techniques and out of context with the problem of the prosit necessary for the resolution of the problematic of the prosit in activity 1 MSCS.			
Technical Objectives	• Implement a delegate • Implement a lambda expression • Implement an anonymous type. • Implement a thread • Implement a parameterized thread • Implement a thread pool • Implement an asynchronous delegate and callback procedure • Implement a vent • Implement a mutex • Implement a monitor • Implement a semaphore • Implement a readerWriterLockSlim			
Duration	6h00	**Version**	1.0	2018

Q1 - Delegate

Let the method 'int method (int v1, int v2)'. This method adds two values and returns the result. Write the delegate who will invoke this method

Q2 - Lambda expression

Construct using a lambda expression a method that will calculate the square of a number

Q3 - Anonymous Type

Implement an anonymous type that has an 'int', a 'string'. Exposing its use

Q4 - Thread & Thread Param

A- In a console type application write a class named "CLpara". In this class write a method named "methode_para". This method receives nothing and returns nothing. It is not "static". This method will be asleep for 1000 ms when running. It will also have the task of displaying a console message 10 times during its execution. From the "main", use a thread that launches the "method_para" method. To use this thread you must use a delegate of your confection.
B- Repeat the question Q4A again by using this time the class "CLpara" as static class.
C- Repeat question Q4B again. Do not use a CLpara class anymore. Instead, use a lambda instruction that does the same job.
D- Repeat the QAC question again. Transform your code so that the thread can take an argument. You will then pass the "Hi" argument to your thread.start () method. Your lambda instruction will receive this argument and display it in the console. You will thus use a thread set with a lambda instruction.

Q5 - Pool Threads

You must create a delegate that invokes using a lambda statement that performs the following code:

A. Retrieve an object of type object from the argument.
B. Unbox the argument in a string variable named msg.
C. The lambda must display 10 times using a while, the message retrieved by the argument, the iteration round. It must be asleep for 1000ms.
D. You must then create three parameterized threads t1, t2, t3.
E. Threads must be placed in a thread pool using an anonymous method and start with the name of its method.

Q6 - Delegate Async

A- You must create a delegate with the same functionality as the previous question.
B- The delegate must be named using the begininvok method.
C- The begininvok method will receive an object to encapsulate the name of the thread (ie "T1"), the object itself (the delegate), and a lambda instruction as a callback.

D- The callback will cast on the argument received via the lambda statement, in order to retrieve the delegate. It will call on this delegate the endinvok method. It will display on the console my mention "callback" and this, using the color yellow.

E- The main thread will wait until the asycResult is completed. As long as this is not the case, it will display every 2000ms message "The main thread is waiting for the call back end". When the asyncresult is completed it will display in red the message "callback end"

Q7 - Evt

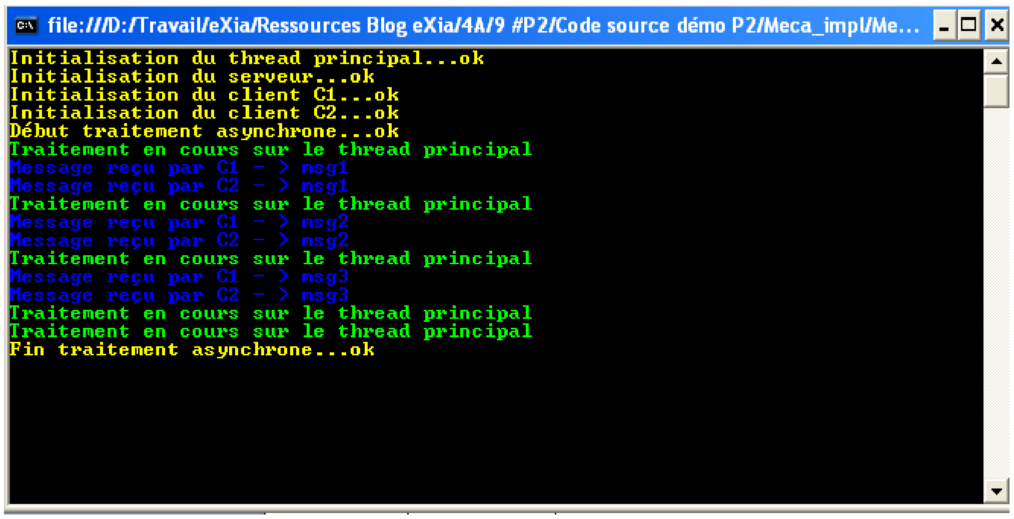
Observe the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace NS_MAIN
{
    class Program
    {
        private delegate void DELG(object o);
        static void Main(string[] args)
        {
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.WriteLine("Initialisation du thread 
principal...ok");
            DELG delg;
            NS_SERVER.CLserver server = new NS_SERVER.CLserver();
            NS_CLIENT.CLclient client1 
= new NS_CLIENT.CLclient(server, "C1");
            NS_CLIENT.CLclient client2 
= new NS_CLIENT.CLclient(server, "C2");
            string[] messages = {"msg1","msg2","msg3"};
            delg = (o) =>
                {
                    for (int i = 0; i < messages.Length; i++)
                    {
                        server.Msg = messages[i];
                        System.Threading.Thread.Sleep(4000);
                    }
                };
            Console.WriteLine("Début traitement asynchrone...ok");
            IAsyncResult asr = 
delg.BeginInvoke(((object)("nostate")),
                (asR)=>
                {
                    delg.EndInvoke(asR);
                    Console.ForegroundColor = ConsoleColor.Yellow;
```

```
                        Console.WriteLine("Fin traitement
asynchrone...ok");
                    },delg);
                while (!asr.IsCompleted)
                {
                    Console.ForegroundColor = ConsoleColor.Green;
                    Console.WriteLine("Traitement en cours sur le thread
principal");
                    System.Threading.Thread.Sleep(3000);
                }
                Console.Read();
            }
        }
    }
```

B- Observe the display.



file:///D:/Travail/eXia/Ressources Blog eXia/4A/9 #P2/Code source démo P2/Meca_impl/Me...

```
Initialisation du thread principal...ok
Initialisation du serveur...ok
Initialisation du client C1...ok
Initialisation du client C2...ok
Début traitement asynchrone...ok
Traitement en cours sur le thread principal
Message reçu par C1 - > msg1
Message reçu par C2 - > msg1
Traitement en cours sur le thread principal
Message reçu par C1 - > msg2
Message reçu par C2 - > msg2
Traitement en cours sur le thread principal
Message reçu par C1 - > msg3
Message reçu par C2 - > msg3
Traitement en cours sur le thread principal
Traitement en cours sur le thread principal
Fin traitement asynchrone...ok
```

C- You must produce the class "CLclient" and the class "CLserver" obtaining the identical the display present in the previous point while respecting the first point.

Q8 — Synchronization

You must set up a mechanism that controls access to objects by providing a single-threaded object's lock. Locks placed on objects allow to restrict access to a block of code commonly called a critical section. As long as a thread has the lock of an object, no other thread can acquire that lock. You can also use this mechanism to ensure that no other thread is allowed to access a section of the application code that is executed by the lock owner unless that other thread executes the code by using a different locked object. The critical section to protect is:

```
string name_thread = (string)state;

++var;

Console.WriteLine("Thread -> {0} -- var -> {1}", name_thread,
var.ToString());

System.Threading.Thread.Sleep(2000);
```

This section is accessible through a lambda instruction. Two "T1" and "T2" threads will have to execute it. You must identify the most appropriate mechanism to address the constraints outlined above and build a code that will expose its use.

Q9 — Synchronization

When two or more threads must access a shared resource at the same time, the system needs a synchronization mechanism to ensure that only one thread at a time uses this resource. This mechanism is a synchronization primitive that grants a single thread exclusive access to the shared resource. If a thread acquires this mechanism, the other thread that wants to acquire this mechanism is interrupted until the first thread releases this mechanism. This mechanism will be of the "Core" type. The code to protect is as follows:

```
for (int i = 0; i < 3; i++)

{

    ++var;

    Console.WriteLine("Thread -> {0} -- var -> {1}",
name_thread,var.ToString());

    System.Threading.Thread.Sleep(2000);

}
```

It is accessible through a lambda instruction. Two "T1" and "T2" threads will have to execute it.
You must identify the most appropriate mechanism to address the constraints outlined above and build a code that will expose its use.

Q10 — Synchronization

Use this mechanism to control access to a resource pool. Threads enter this mechanism by calling the WaitOne method, which is inherited from the WaitHandle class and releases this mechanism by calling the Release method.
The counter of this mechanism is decremented each time a thread enters this mechanism and incremented when a thread releases this mechanism. When the counter is zero, the following requests are blocked until other threads release this mechanism. When all threads have released this mechanism, the counter is at the maximum value specified when this mechanism was created. This mechanism will be of the "Core" type. You must control accessibility to the resource "simu_cnx_db". Two threads must have maximum access to this resource. Three threads are competitors in this application.

```
private static string simu_cnx_db = "Utilisation de la base de données";
```

The code to protect is as follows:

```
string name_thread = (string)state;

Console.WriteLine("Thread -> {0} -- Etat -> {1}", name_thread,
simu_cnx_db.ToString());

System.Threading.Thread.Sleep(2000);
```

Q11 — Synchronization

Use this mechanism to protect a resource read by multiple threads and written by one thread at a time. This mechanism allows multiple threads to be in read mode, a thread to be in write mode with the exclusive property of the lock, and a thread with read access to be in read-to-upgrade mode. from which the thread can upgrade to write mode without having to cancel its read access to the resource. In this program, four threads will compete. Three lambda instructions will be put in place. A lambda instruction will have a critical section accessible for reading and the other two for writing. The instruction in read mode, must read the contents of a variable 'static' type 'int' initialized to 0. At the end of its work, it will mark a pause time of 2 seconds. The first lambda instruction in write mode, will initialize an integer array with 10 odd values between 1 and 19. Returning to the critical section, it will iteratively update the 'static' variable. It will show on the console that a value change has taken place by also indicating, the name thread carrying the operation. The second lambda instruction in write mode, will initialize an integer array with 10 even values between 2 and 20. When returning to the critical section, it will iteratively update the 'static' variable. It will show on the console that a value change has taken place by also indicating the thread name of the operation. Two threads will invoke lambda instructions in write mode. The other two threads will invoke the lambda statement in read mode.

You must identify the most appropriate mechanism to address the constraints outlined above and build a code that will expose its use.