WORKSHOP GESTION DE LA MÉMOIRE

Tout afficher Tout réduire

INTRODUCTION

Ce workshop se concentre uniquement sur la mise en pratique de deux objectifs du prosit Restructuration.

Après avoir réalisé ce workshop, vous serez en mesure :

- Utiliser des delegates et lambda
- Utiliser l'interface iDisposable

Temps de réalisation : 3h/4h

Matériels et logiciels :

- PC
- Internet
- Installation de la séquence 0

Ressources : Moodle

SUJET

Énoncé

EXERCICE

Question

Exercice 1 : Délégué

Soit la méthode ‘int methode(int v1 , int v2)'. Cette méthode additionne deux valeurs et retourne le résultat. Écrire le délégué qui invoquera cette méthode.

Question

Exercice 2 : Expression Lambda

Construire à l'aide d'une expression lambda une méthode qui calculera le carré d'un nombre.

Question

Exercice 3 : Type anonyme

Implémenter un type anonyme qui comporte un ‘int', un ‘string'. Exposer son utilisations.

Question

Exercice 4 : Délégués

Utilisez le code suivant :

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
```

```
using System.Text;
namespace @delegate
{
public delegate void delg();
public class A
{
public void ma()
{
Console.WriteLine("ma");
}
}
public class B
{
public void mb()
{
Console.WriteLine("mb");
}
}
```

Vous devez :

- Déclarer deux objets de type A et B
- Déclarer un type delegate de signature void/void
- Déclarer une instance simple de votre type délégué
- Déclarer un tableau de délégués de votre type délégué
- Déclarer un délégué multicast de votre type délégué
- Allouer de la mémoire pour l'objet de type A
- Allouer de la mémoire pour l'objet de type B
- Allouer de la mémoire pour votre instance simple de délégué et lui assigner l'adresse mémoire de la méthode ma
- Allouer de la mémoire pour votre tableau de délégués qui contiendra deux adresses mémoires
    - La première cellule du tableau de délégués contiendra l'adresse mémoire de ma.
    - La deuxième cellule du tableau de délégués contiendra l'adresse mémoire de mb.
- Allouer de la mémoire pour votre délégué multicast
    - Le premier abonnement sera la référence ma
    - Le deuxième abonnement sera la référence mb
- Appeler ma sur l'objet A
- Appeler mb sur l'objet B
- Invoquer le simple délégué
- Utiliser une boucle for pour invoquer les méthodes du tableau de délégués
- Invoquer le délégué multicast
- Désabonner la référence mb du délégué multicast
- Invoquer le délégué multicast

Exercice 5 :Heap vs Stack

Après avoir complété le code (// to do), faire des comparaisons entre Heap et STack

- la vitesse d'exécution
- Taille maximale (en jouant sur la valeur de largeSize)

```
static void Main()
{
    const int largeSize = 300000;
    AllocateLargeArrayOnHeap(largeSize);
    AllocateLargeArrayOnStack(largeSize);
}

static void AllocateLargeArrayOnStack(int size)
{
```

```
    try // Stack
    {
        Stopwatch stopwatch = Stopwatch.StartNew();
   // to do  créer un tableau  largeArray[[size] utilisant le stack
        for (int j = 0; j < 2000; j++)
        {
            largeArray[0] = 0;
            for (int i = 1; i < largeArray.Length; i++)
            {
                largeArray[i] = largeArray[i - 1];

            }
        }
        stopwatch.Stop();
        Console.WriteLine($"stack : {stopwatch.ElapsedMilliseconds} ms");

    }
    catch (Exception ex)
    {
        Console.WriteLine($"stack allocation not possible: {ex.Message}");
    }

}

static void AllocateLargeArrayOnHeap(int size)
{
    try
    {
        Stopwatch stopwatch = Stopwatch.StartNew();
        // to do // to do  créer un tableau  largeArray[size] utilisant le Heap
        for (int i=0; i < 2000; i++)
```

Question

Exercice 6 : iDisposable

Le code ci-dessous n'utilise pas iDisposable pour gérer la connexion.

Cela nécessite de gérer dans Main les exceptions et la fermeture.

Modifier la classe DataBaseConnection pour que celle-ci implémente iDisposable.

Simplifier dans un 2 ème temps Main avec l'utilisation de Using

```
using System;

public class DatabaseConnection
{
    private bool isOpen = false; // State of the connection

    // Constructor that simulates opening the database connection.
    public DatabaseConnection()
    {
        Console.WriteLine("Database connection opened.");
        isOpen = true;
    }

    // Method to simulate a database operation.
    public void ExecuteQuery(string query)
    {
        // Check if the connection is open.
        if (!isOpen)
            throw new InvalidOperationException("The connection is closed.");

        Console.WriteLine($"Executing query: {query}");
    }

    // Method to close the connection.
```

```
    public void Close()
    {
        // Close the connection if it is open.
        if (isOpen)
        {
            Console.WriteLine("Database connection closed.");
            isOpen = false;
        }
    }
}

// Usage of the DatabaseConnection class
class Program
{
    static void Main()
    {
```