Compte rendu (TP 3 & 4) Grpe B

Génie.Info 2

ADOGNIBO Hortice

TP3

Exo1:

```
using System;

namespace OperationApp
{
    class Program
    {

        public delegate double Operation(double a, double b);



        public class Calculatrice
        {
            public double Addition(double a, double b)
            {
                return a + b;
            }

            public double Soustraction(double a, double b)
            {
                return a - b;
            }
```

```
    public double Multiplication(double a, double b)
    {
        return a * b;
    }

    public double Division(double a, double b)
    {
        if (b == 0)
        {
            Console.WriteLine("Erreur : division par zéro !");
            return double.NaN;
        }
        return a / b;
    }
}


static void Main(string[] args)
{
    Console.WriteLine("=== Mini Calculatrice avec Délégué ===\n");


    Console.Write("Saisissez le premier nombre : ");
    double a = double.Parse(Console.ReadLine());

    Console.Write("Saisissez le deuxième nombre : ");
    double b = double.Parse(Console.ReadLine());
```

```
Console.WriteLine("\nChoisissez une opération :");
Console.WriteLine("1. Addition");
Console.WriteLine("2. Soustraction");
Console.WriteLine("3. Multiplication");
Console.WriteLine("4. Division");
Console.Write("Entrez votre choix : ");
int choice = int.Parse(Console.ReadLine());

Calculatrice calc = new Calculatrice();


Operation op = null;


switch (choice)
{
    case 1:
        op = calc.Addition;
        break;
    case 2:
        op = calc.Soustraction;
        break;
    case 3:
        op = calc.Multiplication;
        break;
    case 4:
        op = calc.Division;
        break;
    default:
```

```
                    Console.WriteLine("Aucune opération ne correspond.");
                    return;
                }



                double resultat = op(a, b);
                Console.WriteLine($"\nRésultat = {resultat}");
            }
        }
}
```

Exécution:

```
=== Mini Calculatrice avec Délégué ===

Saisissez le premier nombre : 5
Saisissez le deuxième nombre : 6

Choisissez une opération :
1. Addition
2. Soustraction
3. Multiplication
4. Division
Entrez votre choix : 3

Résultat = 30
```

Explication:

Ce code définit une mini-calculatrice en C# utilisant un **délégué** (Operation) qui permet de stocker dynamiquement une fonction correspondant à une opération mathématique (addition, soustraction, multiplication ou division) définie dans la classe Calculatrice. Dans le Main, le programme demande à l'utilisateur deux nombres puis un choix d'opération, et affecte à la variable op la méthode correspondante de l'objet Calculatrice grâce à un switch. Enfin, il exécute l'opération sélectionnée via le délégué (op(a, b)) et affiche le résultat, en gérant notamment le cas particulier de la division par zéro.

Exo 2:

```
using System;


namespace SimulationAlarme
{
```

```
public delegate void TemperatureDepasseeEventHandler(double temperature);



public class Capteur
{
    public double Temperature { get; private set; }



    public event TemperatureDepasseeEventHandler TemperatureDepassee;

    public void Mesurer(double valeur)
    {
        Temperature = valeur;
        Console.WriteLine($"Température mesurée : {Temperature} °C");

        if (Temperature > 30)
        {
            TemperatureDepassee?.Invoke(Temperature);
        }
    }
}

public class Alerte
{

    public void AfficherAlerte(double t)
    {
        Console.WriteLine($"  Alerte : température élevée ({t} °C) !");
    }
```

```
    }



    class Program
    {
        static void Main(string[] args)
        {
            Capteur capteur = new Capteur();
            Alerte alerte = new Alerte();


            capteur.TemperatureDepassee += alerte.AfficherAlerte;


            /
            capteur.Mesurer(28);
            capteur.Mesurer(32);
            capteur.Mesurer(35);
    }
}
```

Exécution:

```
Température mesurée : 28 °C
Température mesurée : 32 °C
?? Alerte : température élevée (32 °C) !
Température mesurée : 35 °C
?? Alerte : température élevée (35 °C) !
```

Explication:

Ce code simule un système d'alarme où un capteur mesure une température et déclenche un **événement** lorsqu'elle dépasse 30 °C. L'événement TemperatureDepassee est relié à la méthode AfficherAlerte de la classe Alerte, qui affiche un message d'avertissement lorsque l'événement est lancé. Ainsi, chaque appel à Mesurer() vérifie la valeur mesurée et déclenche automatiquement l'alerte si la température est trop élevée.

TP 4:

Exo 1 :

```
using System;

namespace ExerciceGeneriques
{

    public class Boite<T>
    {
        private T contenu;

        public void Ajouter(T element)
        {
            contenu = element;
        }

        public T Lire()
        {
            return contenu;
        }
    }

    public class Personne
    {
```

```
        public string Nom { get; set; }
        public int Age { get; set; }

        public Personne(string nom, int age)
        {
            Nom = nom;
            Age = age;
        }

        public override string ToString()
        {
            return $"Nom : {Nom}, Âge : {Age}";
        }
    }

    class Program
    {
        static void Main(string[] args)
        {

            Boite<int> boiteInt = new Boite<int>();
            boiteInt.Ajouter(123);
            Console.WriteLine($"Boîte d'entier contient : {boiteInt.Lire()}");

            Boite<string> boiteString = new Boite<string>();
            boiteString.Ajouter("Bonjour ");
            Console.WriteLine($"Boîte de chaîne contient :
{boiteString.Lire()}");
```

```
            Boite<Personne> boitePersonne = new Boite<Personne>();

            boitePersonne.Ajouter(new Personne("Ali", 25));

            Console.WriteLine($"Boîte de personne contient :
{boitePersonne.Lire()}");

        }

    }

}
```

Exécution:

```
Boîte d'entier contient : 123
Boîte de chaîne contient : Bonjour
Boîte de personne contient : Nom : Ali, Âge : 25
```

Explication:

Ce code définit une classe générique Boite<T> capable de stocker n'importe quel type d'objet grâce au paramètre **T**, avec des méthodes pour ajouter un élément et le lire. La classe Personne représente un objet personnalisé avec un nom et un âge, et redéfinit ToString() pour afficher ses informations. Dans le Main, plusieurs boîtes sont créées pour stocker un entier, une chaîne et une personne, montrant que la même classe générique peut gérer différents types sans être réécrite.

Exo 2 :

```
using System;

using System.Threading.Tasks;


namespace ExerciceAsyncAwait

{

    class Program

    {


        static async Task<string> TéléchargerDonnéesAsync()

        {
```

```
        Console.WriteLine("Téléchargement en cours...");
        await Task.Delay(3000);
        return "Données téléchargées !";
    }

    static async Task Main(string[] args)
    {
        Console.WriteLine("=== Test sans await ===");
        Console.WriteLine("Début du programme");

        // Appel SANS await
        Task<string> tache = TéléchargerDonnéesAsync();

        Console.WriteLine("Fin du programme\n");

        await Task.Delay(3000);

        Console.WriteLine("=== Test avec await ===");
        Console.WriteLine("Début du programme");

        // Appel AVEC await
        string resultat = await TéléchargerDonnéesAsync();
        Console.WriteLine(resultat);
        Console.WriteLine("Fin du programme");
    }
  }
}
```

Exécution:

```
=== Test sans await ===
Début du programme
Téléchargement en cours...
Fin du programme

=== Test avec await ===
Début du programme
Téléchargement en cours...
Données téléchargées !
Fin du programme
```

Explication:

Ce code montre la différence entre appeler une méthode asynchrone **avec** et **sans** await : sans await, le programme continue immédiatement son exécution sans attendre la fin du téléchargement simulé. La méthode TéléchargerDonnéesAsync() utilise Task.Delay pour imiter une opération longue, puis renvoie un texte une fois terminée. Dans la seconde partie, l'utilisation de await force le programme à attendre la fin de la tâche avant d'afficher le résultat, démontrant le fonctionnement du modèle async/await.

Exo 3:

```
using System;


namespace NotificationApp
{
    public interface INotificationService
    {
        void Envoyer(string message);
    }
    public class EmailNotificationService : INotificationService
    {
        public void Envoyer(string message)
        {
            Console.WriteLine("Email envoyé : " + message);
```

```
    }
}

public class SMSNotificationService : INotificationService
{
    public void Envoyer(string message)
    {
        Console.WriteLine("SMS envoyé : " + message);
    }
}

public class Utilisateur
{
    private readonly INotificationService _notificationService;

    public Utilisateur(INotificationService notificationService)
    {
        _notificationService = notificationService;
    }

    public void EnvoyerNotification(string message)
    {
        _notificationService.Envoyer(message);
    }
}

class Program
{
    static void Main(string[] args)
```

```
        {
            Console.WriteLine("Notification App");
            INotificationService emailService = new
EmailNotificationService();
            Utilisateur userEmail = new Utilisateur(emailService);
            Console.WriteLine(" Envoi via Email");
            userEmail.EnvoyerNotification("Bienvenue !");
            INotificationService smsService = new SMSNotificationService();
            Utilisateur userSMS = new Utilisateur(smsService);
            Console.WriteLine(" Envoi via SMS ");
            userSMS.EnvoyerNotification("Bienvenue !");
        }
    }
}
```

Exécution:

```
Notification App
 Envoi via Email
Email envoyé : Bienvenue !
 Envoi via SMS
SMS envoyé : Bienvenue !
```

Explication:

La classe **Utilisateur** dépend d'une **interface** (INotificationService) et non d'un type concret, ce qui permet de changer facilement le type de notification. Les classes **EmailNotificationService** et **SMSNotificationService** implémentent cette interface et définissent chacune leur façon d'envoyer un message. Dans **Main()**, on change simplement l'implémentation injectée dans Utilisateur pour passer de l'email au SMS sans modifier son code.