

# Développement en c#.NET

**Ing.Meryem OUARRACHI**

# Plan du module

## Langage C#

- ☐ L'environnement .Net
- ☐ Initiation à la programmation C#
- ☐ Programmation Orienté Objet C#

## Programmation avancée en .Net ,C#

- ☐ Programmation distribuée
- ☐ Gestion de base de donnée
- ☐ Application WPF

## CHAPITRE 3:

# Programmation OO en C#

# Les delegates

- En fait, un type delegate est un concept très simple : c'est un type qui permet de référencer une méthode d'une classe.
- Un type délégué est toujours associé à la signature d'une méthode.
- Seules les méthodes ayant la même signature pourront être utilisées avec ce délégué.

# Les delegates

## -Déclaration de delegate

```
public delegate void calcul1(int x, int y); //delegate vers une méthode  
//qui prend deux paramètres et ne retourne rien  
  
public delegate int calcul2(int x, int y); //delegate vers une méthode  
//qui prend deux paramètres et retourne un entier
```

**Remarque:** La déclaration de delegate se fait à l'intérieur de classe ou namespace.

# Les delegates

## -Utilisation de delegate

```
//Déclaration des méthodes qui prennent la même signature du delegate calcul1  
public static void som(int x, int y)  
    {Console.WriteLine(x+y);}   
  
public static void sout(int x, int y)  
{ Console.WriteLine(x -y); }
```

# Les delegates

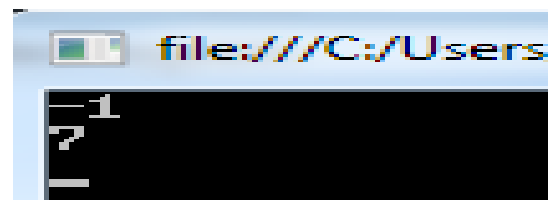
## -Ajouter une méthode au delegate

```
static void Main(string[] args)
{
    calcul1 mydelegate = null;

    //Ajouter des méthodes au delegate
    mydelegate += new calcul1(sout);
    mydelegate += som;

    // passer les valeurs de delegate
    mydelegate(3, 4);
}
```

## -Résultat:



# Les delegates

## -Supprimer une méthode de delegate

```
// Supprimer une méthode de delegate  
mydelegate -= som;
```



# Les delegates

## -Exemple1

```
public static void tri1(int[] t)
{
    for (int i = 0; i < t.Length; i++)
    {
        t[i] = 1;
    }
}
```

```
public static void tri2(int[] t)
{
    for (int i = 0; i < t.Length; i++)
    {
        t[i] = 2;
    }
}
```

# Les delegates

## -Exemple1:

-Ecrire une méthode permettant d'appeler une des méthodes de tri(tri1 ou tri2) .Puis elle affiche le tableau

# Les delegates

## -Exemple1:

```
public static void affiche(int[] t, string choix)
{
    if(choix. Equals("choix 1"))
        tri1(t);
    else
        tri2(t);
    for (int i = 0; i < t.Length; i++)
    { Console.WriteLine(t[i]); }
}
```

# Les delegates

**-Problématique:** supposons qu'on a 20 méthodes de trie alors dans la méthode « **affiche** », on doit passer par 20 tests

**-Solution:** utiliser les delegates parce que grâce aux delegates, on peut transmettre une méthode comme paramètre

# Les delegates

```
public delegate void tri ( int[ ] t);
```

```
public static void affiche(int[] t, tri met)  
{  
    met(t);  
    for (int i = 0; i < t.Length; i++)  
    { Console.WriteLine(t[i]); }  
}
```


# Les delegates

- L'Appel

```
static void Main(string[] args)
{
    int[] t = { 5, 6, 7 };
    //on affecte la bonne méthode selon nos souhaits
    affiche(t, tri1);
    affiche(t, tri2);
    Console.ReadKey();
}
```

# L'utilisation des types délégués

-Les types délégués en C# permettent de passer des méthodes comme argument lors des appels.

 Il est donc possible de créer des comportements différents pour un même code.

 Avec les delegates on peut personnaliser les méthodes que nous les contrôlons pas

# L'utilisation des types délégués

En C#, les délégués sont utilisés dans:

- Des méthodes prédéfinies qui prennent en paramètres des delegates.
- La programmation des threads.
- Responsable de la gestion des événements



# Les méthodes anonymes

- Une méthode anonyme est une méthode sans nom, c'est-à-dire un morceau de code représentant le corps d'une méthode.
- Pour les rendre utilisables, il faut immédiatement les affecter à un delegate qui possède la même définition.
- On utilise les méthodes anonymes pour de petites actions qui ne se reproduisent pas à plus d'un endroit. Leur utilisation permet généralement d'alléger la syntaxe d'une classe

# Les méthodes anonymes

- **Création:**

```
mydelegate += new calcul(delegate(int x, int y)
    { Console.WriteLine(x * y * 10);
    });
```

# Expression Lambda

- On peut créer les méthodes anonymes avec une autre syntaxe qui se base sur les expressions lambda. Une expression Lambda est une façon simplifiée d'écrire les délégués.

```
mydelegate += new calcul((x, y) => Console.WriteLine(x * y * 10));
```

- Le symbole lambda `=>` (se lit « conduit à »). Une expression est toujours constituée de deux parties :
  - Le côté gauche donne les paramètres d'entrées (s'il y en a)
  - Le côté droit donne les instructions de la méthode anonyme.

# Expression Lambda

- Encore une écriture plus simple

```
mydelegate += (x, y) => Console.WriteLine(x * y * 10);
```

# Expression Lambda

- **Autre utilisation:** Les expressions lambda s'utilisent aussi pour donner des équivalents des opérations linq

*Exemple:*

Opérations en Linq	Equivalent en expression lambda
<pre>var x =     from p in personnes     where p.age &gt;= 40     select p;</pre>	<pre>var x = personnes.Where(p =&gt; p.age &gt;= 40);</pre>

**Remarque:** Voir *Equivalent des opérations Linq en expressions Lambda*

[http://www.c-sharpcorner.com/uploadfile/babu\\_2082/linq-operators-and-lambda-expression-syntax-examples/](http://www.c-sharpcorner.com/uploadfile/babu_2082/linq-operators-and-lambda-expression-syntax-examples/)