

C# : délégué et événement

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

`elmouelhi.achref@gmail.com`



- 1 Delegate
 - Multicast
 - Retrait d'une méthode
 - Délégués génériques
 - Délégués avec valeur de retour
 - Délégués prédéfinis

- 2 Méthode anonyme

- 3 Event

Delegate (délégué)

- Concept inspiré par les pointeurs de fonction en **C** et **C++**,
- Représente des références aux méthodes avec une signature bien définie,
- Déclaré avec le mot clé `delegate`,
- Toute méthode respectant cette signature peut être appelée par le biais du délégué.

Commençons par créer un nouveau projet dans MaSolution

- Dans l'Explorateur de solutions, faire clic droit sur MaSolution
- Aller à Ajouter > Nouveau projet
- Sélectionner Application console
- Cliquer sur Suivant
- Remplir les champs
 - Nom **avec** CoursDelegateLambda
 - Emplacement **avec** MaSolution
- Valider

Considérons la méthode suivante

```
public static void DireBonjour(string nom)
{
    Console.WriteLine($"Bonjour {nom}");
}
```

© Achref EL MOUELHI

C#

Considérons la méthode suivante

```
public static void DireBonjour(string nom)
{
    Console.WriteLine($"Bonjour {nom}");
}
```

Pour exécuter cette méthode

```
static void Main(string[] args)
{
    DireBonjour("Wick"); // affiche Bonjour Wick
    Console.ReadKey();
}
```

C#

Considérons la méthode suivante

```
public static void DireBonjour(string nom)
{
    Console.WriteLine($"Bonjour {nom}");
}
```

Pour exécuter cette méthode

```
static void Main(string[] args)
{
    DireBonjour("Wick"); // affiche Bonjour Wick
    Console.ReadKey();
}
```

Question

Comment confier cette mission à un délégué ?

```
public delegate void PremierDelegate(string s);
```

© Achref EL MOUELHI ©

```
public static void DireBonjour(string nom)
{
    ...
}
```



```
public delegate void PremierDelegate(string s);
```

Signature de la méthode déléguée

```
public static void DireBonjour(string nom)
{
    ...
}
```

```
public delegate void PremierDelegate(string s);
```

Signature de la méthode déléguée

Deux signatures
identiques

```
public static void DireBonjour(string nom)
{
    ...
}
```

```
public delegate void PremierDelegate(string s);
```

Signature de la méthode déléguée

```
PremierDelegate d = DireBonjour;
```

Deux signatures
identiques

```
public static void DireBonjour(string nom)
{
    ...
}
```

Créer un délégué (comme attribut)

```
public delegate void PremierDelegate(string s);
```

© Achref EL MOUELHANI

Créer un délégué (comme attribut)

```
public delegate void PremierDelegate(string s);
```

Explication

- Le delegate définit une signature
- Toute méthode compatible peut y être associée

Déclarer un délégué (dans une méthode, `Main` par exemple)

```
PremierDelegate d;
```

© Achref EL MOUELHI ©

Déclarer un délégué (dans une méthode, `Main` par exemple)

```
PremierDelegate d;
```

Créer une instance du délégué

```
d = new PremierDelegate(DireBonjour);
```

© Achref EL MEHRI

Déclarer un délégué (dans une méthode, `Main` par exemple)

```
PremierDelegate d;
```

Créer une instance du délégué

```
d = new PremierDelegate(DireBonjour);
```

Appeler la méthode `DireBonjour` à travers le délégué

```
d("Bob"); // affiche Bonjour Bob
```


Déclarer un délégué (dans une méthode, `Main` par exemple)

```
PremierDelegate d;
```

Créer une instance du délégué

```
d = new PremierDelegate(DireBonjour);
```

Appeler la méthode `DireBonjour` à travers le délégué

```
d("Bob"); // affiche Bonjour Bob
```

On peut aussi faire la même chose de deux façons différentes

On peut faire aussi

```
PremierDelegate d = DireBonjour;  
d("Bob");
```

Exemple avec plusieurs méthodes

```
public static void Somme (int a, int b)
{
    Console.WriteLine(a + b);
}

public static void Produit(int a, int b)
{
    Console.WriteLine(a * b);
}

public static void Soustraction(int a, int b)
{
    Console.WriteLine(a - b);
}

public static void Division(int a, int b)
{
    Console.WriteLine(a / b);
}
```

Déclarer un délégué

```
public delegate void Calcul(int x, int y);
```

© Achref EL MOUELHI ©

Déclarer un délégué

```
public delegate void Calcul(int x, int y);
```

Utiliser le délégué

```
Calcul calcul;  
  
calcul = Somme;  
calcul(7, 5);  
  
calcul = Produit;  
calcul(7, 5);  
  
calcul = Division;  
calcul(7, 5);  
  
calcul = Soustraction;  
calcul(7, 5);
```

Et si le délégué est nul, on aura une exception

```
Calcul calcul = null;
```

```
calcul(7, 5);
```

© Achref EL

Et si le délégué est nul, on aura une exception

```
Calcul calcul = null;  
  
calcul(7, 5);
```

Trop long ?

Pour éviter l'exception précédente, on peut utiliser `Invoke`

```
Calcul calcul = null;  
  
// calcul = Somme;  
calcul?.Invoke(7, 5);  
  
calcul = Produit;  
calcul?.Invoke(7, 5);  
  
calcul = Division;  
calcul?.Invoke(7, 5);  
  
calcul = Soustraction;  
calcul?.Invoke(7, 5);
```


Pour éviter l'exception précédente, on peut utiliser `Invoke`

```
Calcul calcul = null;  
  
// calcul = Somme;  
calcul?.Invoke(7, 5);  
  
calcul = Produit;  
calcul?.Invoke(7, 5);  
  
calcul = Division;  
calcul?.Invoke(7, 5);  
  
calcul = Soustraction;  
calcul?.Invoke(7, 5);
```

Trop long ?

Delegate multicast

- Un delegate peut référencer plusieurs méthodes
- Elles sont exécutées dans l'ordre d'ajout

Solution : utiliser le multicast

```
Calcul? calcul = null;  
  
calcul = Somme;  
calcul += Produit;  
calcul += Division;  
calcul += Soustraction;  
  
calcul?.Invoke(7, 5);
```

Ou en plus simple

```
Calcul? calcul = null;  
  
calcul = Somme;  
calcul = calcul + Produit + Division + Soustraction;  
  
calcul?.Invoke(7, 5);
```

Pour retirer une méthode

```
calcul = calcul - Produit;  
calcul -= Division;
```

```
calcul?.Invoke(2, 3);  
// affiche 5  
// -1
```

© Achre

Pour retirer une méthode

```
calcul = calcul - Produit;  
calcul -= Division;  
  
calcul?.Invoke(2, 3);  
// affiche 5  
// -1
```

Pour connaître le nombre de méthodes abonnées à notre délégué

```
Console.WriteLine(calcul?.GetInvocationList().Length);
```

C#

On peut aussi créer un délégué générique

```
public delegate void Calcul<T>(T x, T y);
```

© Achref EL MOUELHI ©

C#

On peut aussi créer un délégué générique

```
public delegate void Calcul<T>(T x, T y);
```

La déclaration

```
Calcul<int> calcul;
```


C#

On peut aussi créer un délégué générique

```
public delegate void Calcul<T>(T x, T y);
```

La déclaration

```
Calcul<int> calcul;
```

Le reste ne change pas

```
Calcul calcul;
```

```
calcul = Somme;
```

```
calcul = calcul + Produit + Division + Soustraction;
```

```
calcul?.Invoke(7, 5);
```

C#

Considérons les deux méthodes suivantes

```
public static int Sum(int a, int b)
{
    return a + b;
}
```

```
public static int Product(int a, int b)
{
    return a * b;
}
```

C#

Considérons les deux méthodes suivantes

```
public static int Sum(int a, int b)
{
    return a + b;
}
```

```
public static int Product(int a, int b)
{
    return a * b;
}
```

Créons un délégué avec une valeur de retour

```
public delegate int Compute(int x, int y);
```

C#

Déclarons une variable de type `Compute`

```
Compute compute = Sum;  
compute += Product;
```

© Achref EL MOUELHI ©

C#

Déclarons une variable de type `Compute`

```
Compute compute = Sum;  
compute += Product;
```

Et pour tester

```
Console.WriteLine(compute?.Invoke(2, 5));  
// affiche 10
```

C#

Déclarons une variable de type `Compute`

```
Compute compute = Sum;  
compute += Product;
```

Et pour tester

```
Console.WriteLine(compute?.Invoke(2, 5));  
// affiche 10
```

Explication

- Toutes les méthodes sont exécutées
- Seule la dernière valeur est retournée

Trois délégués prédéfinis

- `Func` : délégué des méthodes acceptant 0 ou plusieurs paramètres et retournant au moins une valeur.
- `Predicate` : délégué des méthodes acceptant un seul paramètre et retournant un booléen.
- `Action` : délégué des méthodes ne retournant aucune valeur (`void`).

Pour déclarer un délégué de type `Func`

```
Func<int, int, int> somme = Sum;
```

© Achref EL MOUELHANI

Pour déclarer un délégué de type `Func`

```
Func<int, int, int> somme = Sum;
```

Explication

- Le dernier type générique correspond au type de la valeur de retour.
- Les autres correspondent à ceux des paramètres.

Pour utiliser le délégué

```
Console.WriteLine(somme(2, 3));  
// affiche 5
```

C#

Considérons la méthode `EstPair`

```
public static bool EstPair(int a)
{
    return a % 2 == 0;
}
```

© Achref EL MOUELHI

C#

Considérons la méthode `EstPair`

```
public static bool EstPair(int a)
{
    return a % 2 == 0;
}
```

Pour déclarer un délégué de type `Predicate`

```
Predicate<int> parity = EstPair;
```

C#

Considérons la méthode `EstPair`

```
public static bool EstPair(int a)
{
    return a % 2 == 0;
}
```

Pour déclarer un délégué de type `Predicate`

```
Predicate<int> parity = EstPair;
```

Explication

- Pas besoin d'indiquer le type de la valeur de retour.
- Un seul paramètre accepté \Rightarrow un seul type générique à préciser.

Pour utiliser le délégué

```
Console.WriteLine($"5 est pair : { parity(5) }");  
// affiche 5 est pair : False
```

C#

Considérons la méthode `AfficherEnMajuscule`

```
public static void AfficherEnMajuscule(string word)
{
    Console.WriteLine(word.ToUpper());
}
```

© Achref EL MOUELHI

C#

Considérons la méthode `AfficherEnMajuscule`

```
public static void AfficherEnMajuscule(string word)
{
    Console.WriteLine(word.ToUpper());
}
```

Pour déclarer un délégué de type `Action`

```
Action<string> print = AfficherEnMajuscule;
```


C#

Considérons la méthode `AfficherEnMajuscule`

```
public static void AfficherEnMajuscule(string word)
{
    Console.WriteLine(word.ToUpper());
}
```

Pour déclarer un délégué de type `Action`

```
Action<string> print = AfficherEnMajuscule;
```

Explication

- Pas besoin d'indiquer le type de la valeur de retour (qui est `void`).
- Types génériques précisés \Rightarrow types des paramètres.

Pour utiliser le délégué

```
print("wick");  
// affiche WICK
```

Un délégué peut être utilisé pour

- référencer une ou plusieurs méthodes,
- déclarer une méthode anonyme (section suivante),
- déclarer une expression Lambda (chapitre suivant).

Méthode anonyme en C#

- Méthode sans nom,
- Définie avec le mot-clé `delegate`,
- Peut être affectée à une variable de type délégué.

En utilisant les méthodes anonymes, le code suivant

```
public static void AfficherEnMajuscule(string word)
{
    Console.WriteLine(word.ToUpper());
}
Action<string> print = AfficherEnMajuscule;
```

Peut être remplacé par

```
Action<string> print = delegate (string word)
{
    Console.WriteLine(word.ToUpper());
};
```

Rien à changer pour l'utilisation

```
print("wick");  
// affiche WICK
```

Event

Event (évènement)

- L'une des utilisations la plus importante des délégués est la programmation événementielle,
- Un évènement est déclaré avec le mot clé `event`,
- Les applications à interfaces graphiques sont assez associées aux concepts de programmation événementielle (`click`, `input`, `focus...`)

© Achren

Event

Event (évènement)

- L'une des utilisations la plus importante des délégués est la programmation événementielle,
- Un évènement est déclaré avec le mot clé `event`,
- Les applications à interfaces graphiques sont assez associées aux concepts de programmation événementielle (`click`, `input`, `focus`...)

Nomenclature

- L'objet qui déclenche l'évènement est appelé éditeur.
- Celui qui capture l'évènement et y répond est appelé abonné.

Event

Exemple UI : bouton

- Un bouton déclenche un événement (**Éditeur**)
- Plusieurs composants peuvent réagir (**Abonnée**)
- Le bouton ne connaît pas les réactions

Considérons la classe `Button`

```
namespace CoursDelegate
{
    internal class Button
    {
        public event Action Click;

        public void SimulerClic()
        {
            Console.WriteLine("Bouton cliqué");
            Click?.Invoke();
        }
    }
}
```

Considérons la classe `Button`

```
namespace CoursDelegate
{
    internal class Button
    {
        public event Action Click;

        public void SimulerClic()
        {
            Console.WriteLine("Bouton cliqué");
            Click?.Invoke();
        }
    }
}
```

Explication

- Le bouton (**Éditeur**) expose un événement
- Il ne sait pas ce qui va se passer après le clic

Event

Déclarer un abonné 1 : afficher un message (dans Program)

```
static void AfficherMessage()  
{  
    Console.WriteLine("Affichage d'un message");  
}
```

© Achref EL MOU

Event

Déclarer un abonné 1 : afficher un message (dans Program)

```
static void AfficherMessage()  
{  
    Console.WriteLine("Affichage d'un message");  
}
```

Déclarer un abonné 2 : enregistrer un log (dans Program)

```
static void LoggerClic()  
{  
    Console.WriteLine("Clic enregistré dans le journal");  
}
```

Event

Abonnement au clic

```
Button bouton = new Button();  
  
bouton.Click += AfficherMessage;  
bouton.Click += LoggerClic;
```

© Achref EL MOUËL

Event

Abonnement au clic

```
Button bouton = new Button();  
  
bouton.Click += AfficherMessage;  
bouton.Click += LoggerClic;
```

Déclenchement

```
bouton.SimulerClic();
```

Event

Abonnement au clic

```
Button bouton = new Button();  
  
bouton.Click += AfficherMessage;  
bouton.Click += LoggerClic;
```

Déclenchement

```
bouton.SimulerClic();
```

Résultat

```
bouton.SimulerClic();
```


Event

Remarques

- Sans `event`, n'importe qui pourrait faire `Click()` depuis l'extérieur.
- Avec `event`, seul le bouton peut déclencher le clic.
- **WinForms**, **WPF**, **Blazor**, **MAUI** utilisent exactement ce mécanisme.