C# : délégué et évènement

Achref El Mouelhi

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

elmouelhi.achref@gmail.com

Plan

C#

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.

C#

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

C#

Considérons la méthode suivante

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

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 ?

C#

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



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

C#

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

Signature de la méthode déléguée


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

C#

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

Signature de la méthode déléguée

Deux signatures identiques

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

C#

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

Signature de la méthode déléguée

```
PremierDelegate d = DireBonjour;
```

$\Leftarrow$ Deux signatures identiques

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

C#

Créer un délégué (comme attribut)

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

C#

Créer un délégué (comme attribut)

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

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

```
PremierDelegate d;
```

C#

Créer un délégué (comme attribut)

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

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

```
PremierDelegate d;
```

Créer une instance du délégué

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

C#

Créer un délégué (comme attribut)

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

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
```

C#

Créer un délégué (comme attribut)

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

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

C#

On peut faire aussi

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

C#

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);
```

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);
```

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);
```

Trop long ?

C#

Solution : utiliser le multicast

```
Calcul calcul;

calcul = Somme;
calcul += Produit ;
calcul += Division;
calcul += Soustraction;

calcul(7, 5);
```

C#

Ou en plus simple

```
Calcul calcul;

calcul = Somme;
calcul = calcul + Produit + Division + Soustraction;

calcul(7, 5);
```

C#

Pour retirer une méthode

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

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

C#

Pour retirer une méthode

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

calcul(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);
```

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(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;
```

C#

Déclarons une variable de type `Compute`

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

Si un délégué renvoie une valeur, la dernière valeur de la méthode affectée sera renvoyée lorsque le délégué sera appelé.

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

C#

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`).

C#

Pour déclarer un délégué de type `Func`

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

C#

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.

C#

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;
}
```

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 ⇒ un seul type générique à précisé.

C#

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());
}
```

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 ⇒ types des paramètres.

C#

Pour utiliser le délégué

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

C#

Un délégué peut être utilisé pour

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

C#

Méthode anonyme en **C#**

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

C#

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());
};
```

C#

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 évènementielle,
- Un évènement est déclaré avec le mot clé `event`,
- Les applications à interfaces graphiques sont assez associées aux concepts de programmation évènementielle (`click`, `input`, `focus`...)

Event

Event (évènement)

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

Déclarer un évènement

```
public static event Calcul MonEvent;
```

Event

Déclarer un évènement

```
public static event Calcul MonEvent;
```

Déclarer un déclencheur

```
public static void MonTrigger()
{
    MonEvent(7, 5);
}
```

Event

Déclarer un évènement

```
public static event Calcul MonEvent;
```

Déclarer un déclencheur

```
public static void MonTrigger()
{
    MonEvent(7, 5);
}
```


Objectif

Exécuter les méthodes d'un délégué lorsqu'un évènement se déclenche (une méthode ici qui sera appelée).

Event

Abonner des méthodes à cet évènement

```
MonEvent = new Calcul(Somme);
MonEvent += new Calcul(Produit);
```

Event

Abonner des méthodes à cet évènement

```
MonEvent = new Calcul(Somme);
MonEvent += new Calcul(Produit);
```

Ou aussi

```
MonEvent = Somme;
MonEvent += Produit;
```

Event

Déclencher l'évènement

```
MonTrigger();
```

Event

Déclencher l'évènement

```
MonTrigger();
```


Résultat

- Les méthodes abonnées à cet évènement sont exécutées.
- `12 35` sont affichés.
- On n'a pas exécuté les méthodes à travers le délégué.