Module 11

Découplage des méthodes et gestion des événements

Sommaire

- Déclarer et Utiliser des Délégués
- Utiliser des Expressions Lambda
- Gérer des Evénements

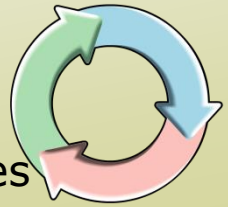
Leçon 1: Déclarer et Utiliser des Délégués

- Pourquoi dissocier une opération d'une méthode?
- Définir un Délégué
- Appeler un Délégué
- Définir des Méthodes Anonymes

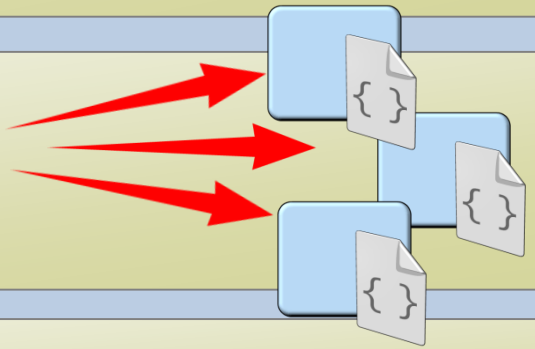
Pourquoi dissocier une opération d'une méthode?

- Méthodes déterminées dynamiquement lors del'exécution
  - Méthode A ou méthode B ou méthode C

- Méthodes de retour (Callback)
  - Vous permet de specifier une ou plusieurs méthodes à exécuter quand des appels à des méthodes asynchrones sont finis, particulièrement lors de l'utilisation d'assemblies tiers



- Opérations de type Multicast
  - Méthode A et méthode B et méthode C



Les délégués représentent essentiellement des pointeurs de méthodes

Définir un Délégué

Définissez un délégué en utilisant le mot clé **delegate**

```
public delegate bool isValidDelegate();
```

Créez une instance du délégué

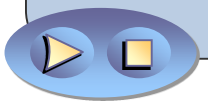
```
public isValidDelegate isValid = null;
```

Ajoutez des références de methode au délégué avec l'opérateur **+=**

```
isValid += CheckStateValid;
isValid += new isValidDelegate(CheckControl);
```

Supprimez des références de methode au délégué avec l'opérateur **-=**

```
isValid -= CheckStateValid;
```



Appeler un Délégué

Toujours vérifié si le délégué n'est pas **null** avant de l'appeler!

Appeler de façon synchrone le délégué en utilisant la même syntaxe que pour une méthode

```
if (isValid != null)
{
    isValid();
}
```

Appeler un délégué de façon asynchrone en utilisant les méthodes **BeginInvoke** et **EndInvoke**

```
if (isValid != null)
{
    isValid.BeginInvoke();
}
```

Définir des Méthodes Anonymes

Ajoutez la méthode anonyme comme un gestionnaire pour un délégué

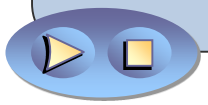
Utiliser le mot clé **delegate**

```
myDelegateInstance += new
myDelegate
(delegate (int parameter)
    {
        // Perform operation.
        return 10;
    });
```

Optionnellement spécifier les noms de paramètres et les types de la méthode anonyme

Ajouter le corps de méthode anonyme

Ne spécifiez de type de retour ; le compilateur calculera le type de retour selon le contexte



Lesson 2: Utiliser des Expresssions Lambda

- Qu’est-ce qu’une Expression Lambda?
- Définir des Expressions Lambda
- Portée d’une Variable dans une Expression Lambda

Qu’est-ce qu’une Expression Lambda?

Une expression lambda est une expression qui retourne une méthode

Les expressions lambda sont définies à l’aide de l’opérateur**=>**

Les expressions lambda utilisent une syntaxe naturelle et concise

```
x => x * x
```

This code example can be read as: Donner x, calculer x * x

Le type du parameter et de la valeur de retour sont déduits

Définir des Expressions Lambda

Une expression simple, où le type du paramètre x est déduit du contexte

```
x => x * x
```

Une expression qui utilise un bloc d'instructions C# au lieu d'une expression simple

```
x => { return x * x ; }
```

Une expression qui appelle une méthode et ne prend aucun paramètre

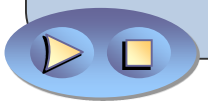
```
() => myObject.MyMethod(0)
```

Une expression où le type de paramètre est déclaré explicitement

```
(int x) => x / 2
```

Une expression avec plusieurs paramètres et un paramètre passé par référence

```
(ref int x,int y) => {x++; return x/y;}
```



Portée d'une Variable dans une Expression Lambda

- Les expressions lambda peuvent utiliser toutes les variables qui sont le scope dans lequel elles sont déclarées
- Lambda expressions can define variables

```
string name; // Class level field.
...

void myMethod() {
    int count = 0;
    del += new MyDelegate(() =>
        {
            int temp = 10;
            CheckName(name);
            count++; // Doe something more interesting.
        });
}
```

L'utilisation d'une variable externe dans une expression lambda peut prolonger son cycle de vie

A utilizer avec précaution!

Leçon 3: Gérer des Evénements

- Qu’est-ce qu’un événement?
- Définir un événement
- Utiliser les événements
- Bonnes pratiques pour l’utilisation des événements

Qu’est-ce qu’un événement?

Un événement fournit un mécanisme pour informer d’autres applications qu’un changement d’état ou autre événement s’est produit dans un type

Les événements sont basés sur les délégués

Contrairement à une instance d’un délégué, un événement peut être déclenché (invoqué) que par la classe conteneur ou un dérivé de celle-ci

Les classes consommatrices peuvent s’abonner à l’événement, ajouter de références aux méthodes qui doivent s’exécuter lorsqu’un événement est déclenché

Événements sont largement utilisées dans le Framework.NET; Ils sont utiliser par presque tous les contrôles de Windows Presentation Foundation (WPF)

Définir un événement

Définissez un délégué sur lequel l'événement se base ; le délégué doit être au moins aussi visible que l'événement

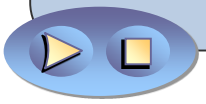
```
public delegate void MyEventDelegate(object sender, EventArgs e);

public event MyEventDelegate MyEvent = null;
```

Utilisez le mot clé **event**

Specifiez le type du délégué

Spécifiez un nom pour l'événement

Vous pouvez définir un événement dans une interface



Utiliser les événements

Pour s'abonner à un événement, utiliser l'opérateur **+=** de la même façon qu vous référencez une méthode au près d'un délégué

```
MyEvent += new MyEventDelegate(myHandlingMethod);
```

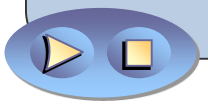
Pour se désabonner d'un événement, utiliser l'opérateur **-=**

```
MyEvent -= myHandlingMethod;
```

Pour déclencher un événement, utilisez le nom de l'événement et spécifiez les paramètres come pour l'appel d'une méthode

```
if (MyEvent != null)
{
    EventArgs args = new EventArgs();
    MyEvent(this, args);
}
```

Toujours verifier si un événement n'est pas null **null** avant de le lever!



Bonnes pratiques pour l’utilisation des événements

- ✓ Utilisez la signature d’événement standard
- ✓ Utilisez une méthode protected virtual pour déclencher un événement
- ✓ Ne passez pas nue valeur null à un événement

Atelier Pratique

- Exercice 1:
- Exercice 2: