Les événements

Les événements sont au cœur du développement d'applications avec C#. Ils permettent de baser la logique de l'application sur une série de procédures et de fonctions exécutées lorsque l'un de ses composants demande l'exécution. C'est par exemple le cas avec les composants graphiques : ceux-ci peuvent déclencher des événements lorsque l'utilisateur effectue une action comme la sélection d'un élément dans une liste ou un clic sur un bouton.

1. Déclaration et déclenchement

Les événements de C# sont basés sur l'utilisation de délégués. L'idée générale est que chaque événement peut accepter un ou plusieurs gestionnaires d'événements dont la signature est définie par un type de délégué.

Les événements générés par les classes du framework utilisent fréquemment le type de délégué `EventHandler` pour définir les gestionnaires d'événements qui peuvent leur être associés. Ce délégué est défini de la manière suivante :

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

Le paramètre `sender` correspond à l'objet qui a généré l'événement, tandis que le paramètre de type `EventArgs`, nommé `e`, est utilisé pour fournir des informations aux méthodes qui traitent l'événement. Si aucune valeur ne doit être passée aux gestionnaires d'événements, il est possible d'utiliser une instance de la classe `EventArgs`, mais dans le cas contraire, il est pertinent de passer un objet d'un type hérité de la classe `EventArgs`.

Il existe aussi le type de délégué générique `EventHandler<TEventArgs>`, qui permet le passage d'informations lorsque l'événement est déclenché. Les génériques sont étudiés un peu plus loin dans ce chapitre.

Utiliser le type `EventHandler` ou sa contrepartie générique `EventHandler<TEventArgs>` permet de garder une certaine cohérence entre les événements de l'application et les événements générés par les classes du framework .NET.

La syntaxe générale pour la déclaration d'un événement est la suivante :

```
<modificateur d'accès> event <type de délégué> <nom d'événement>;
```

La déclaration d'un événement déclenché à la fin d'un calcul de factures peut être écrite de la manière suivante :

```
public event EventHandler CalculFacturesTermine;
```

Il est d'usage de placer le déclenchement d'un événement dans une méthode dont l'unique but est de le déclencher si un ou plusieurs gestionnaires existent pour l'événement.

```
protected virtual void DeclencheCalculFacturesTermine(EventArgs e)
{
    EventHandler handler = CalculFacturesTermine;

    //Si un ou plusieurs gestionnaires sont définis,
    //handler ne sera pas null
    if (handler != null)
```

```
    {
            handler(this, e);
    }
}
```

Le déclenchement de l'événement est ensuite effectué à l'endroit où l'on souhaite exécuter le code des gestionnaires liés à l'événement, dans notre cas, à la fin de l'exécution de la méthode `CalculFactures` de la classe `Facturation`.

```
public class Facturation
{
    //Déclaration de l'événement
    public event EventHandler CalculFacturesTermine;

    public void CalculFactures()
    {
        //Code potentiellement long effectuant le calcul
        //Ici, nous simulons ce calcul en
        //incrémentant un compteur de 1 à 1000
        for (int i = 0; i < 1000; i++)
        {
            i++;
        }

        //Déclenchement de l'événement
        EventArgs arguments = new EventArgs();
        DeclencheCalculFacturesTermine(arguments);
    }

    protected virtual void DeclencheCalculFacturesTermine(EventArgs e)
    {
        EventHandler handler = CalculFacturesTermine;

        //Si un ou plusieurs gestionnaires sont définis,
        //handler ne sera pas null
        if (handler != null)
        {
            handler(this, e);
        }
    }
}
```

2. Gestion des événements

Avant de gérer l'événement `CalculFacturesTermines`, exécutons notre méthode de calcul des factures :

```
Facturation fact = new Facturation();
fact.CalculFactures();
```

Ce code fonctionne sans erreurs. Ceci est lié à la vérification effectuée dans le corps de la méthode `DeclencheCalculFacturesTermine`. Aucun gestionnaire n'est associé à l'événement `CalculFacturesTermine`, celui-ci n'est donc pas déclenché. Si cette vérification n'était pas effectuée, une exception de type `NullReferenceException` serait générée et interromprait le cours normal de l'application.

Ajout d'un gestionnaire d'événements

L'ajout d'un gestionnaire pour un événement est effectué à l'aide de l'opérateur `+=` auquel il faut passer un nom de méthode correspondant au type de délégué attendu.

```
<objet>.<nom d'événement> += <nom de la methode gestionnaire>;
```

Dans le cas de l'événement `CalculFacturesTermine`, le type de délégué attendu est `EventHandler`, ce qui implique qu'il faut définir une méthode correspondant à la signature de ce délégué.

```
private void gestionnaire_CalculFacturesTermine(object sender,
EventArgs e)
{
    Console.WriteLine("Le calcul des factures est terminé.");
}
```

Pour que le gestionnaire soit exécuté, il faut maintenant l'affecter à l'événement avant que celui-ci ne soit déclenché :

```
Facturation fact = new Facturation();
fact.CalculFacturesTermine += gestionnaire_CalculFacturesTermine;
fact.CalculFactures();
```

Suppression d'un gestionnaire d'événements

La suppression d'un gestionnaire d'événements est similaire à son ajout, la différence étant dans l'opérateur utilisé. Il faut en effet utiliser l'opérateur `-=` pour déréférencer un gestionnaire.

```
Facturation fact = new Facturation();
fact.CalculFacturesTermine += gestionnaire_CalculFacturesTermine;
fact.CalculFactures();
fact.CalculFacturesTermine -= gestionnaire_CalculFacturesTermine;
fact.CalculFactures();
```

L'exécution de cette portion de code affiche le message *"Le calcul des factures est terminé."* une seule fois, puisque le second calcul est effectué après déréférencement du gestionnaire d'événements.