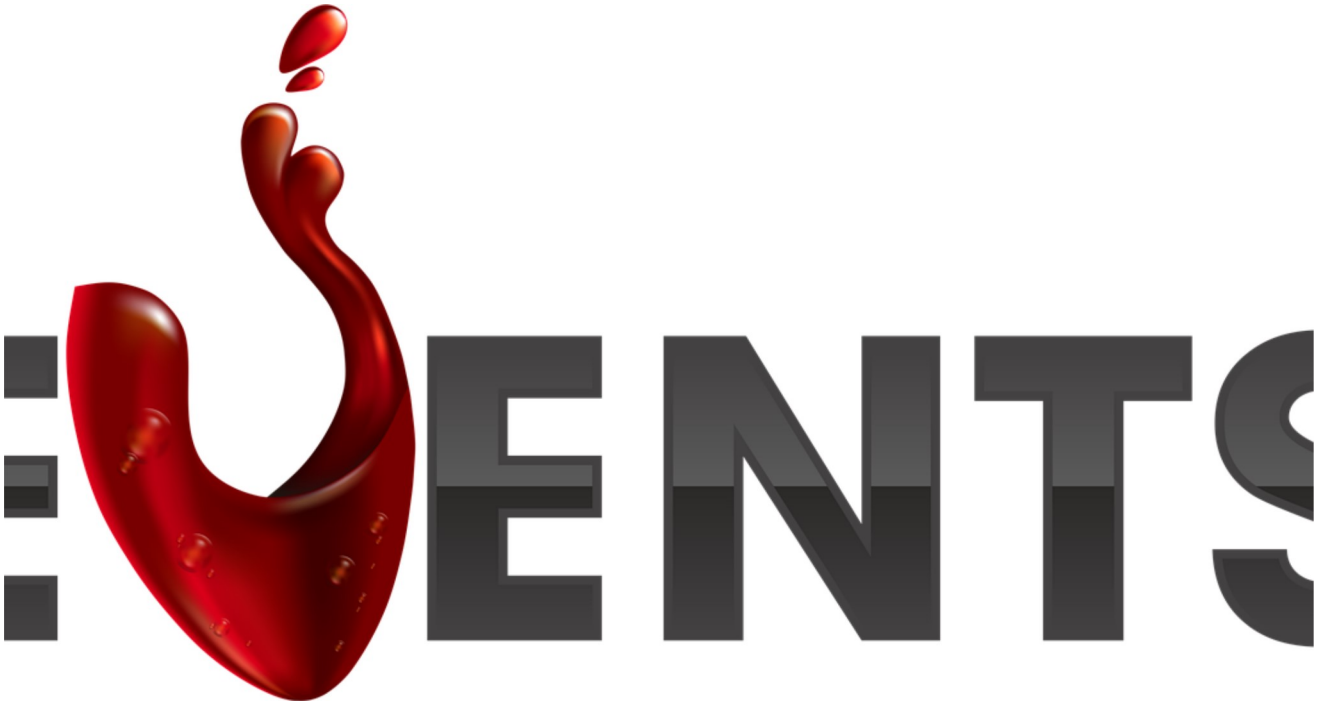
https://dev-dot.net/debuter-avec-les-evenements-en-c/

Débuter avec les évènements en C# - Développeur .NET

par Daniel MINKO FASSINOU



Dans une application, on retrouve des évènements partout. Lorsqu'un traitement en lancé par le clic sur un bouton ou l'appui sur un menu, il y a un évènement derrière. Les évènements dans le Framework .NET suivent le design pattern observateur. Ce design pattern permet à un objet ou plusieurs d'être informés des changements intervenus dans un autre. Voyons comment gérer les évènements en C#.

Généralités sur les évènements

La classe qui contient l'évènement doit d'abord déclarer un délégué, l'évènement sera du même type que ce délégué. Le délégué peut être vu comme une référence vers une ou plusieurs méthodes. Par convention, le nom d'un évènement est de la forme **On<Nom de l'évènement>**, par exemple OnWindowsClose.

```
1  public delegate myDelegate();
```

Ensuite on déclare notre évènement

```
1  event MyDelegate OnEvent;
```

Rien ne vaut un bon exemple.

Prenons la classe Account qui représente un compte en banque. L'évènement OnDebitBalance est levé lorsque le solde est négatif.

```
1
2  class Account
3  {
4
5      double credit;
6
7      public delegate void DebitBalanceDelegate();
8
9      public event DebitBalanceDelegate OnDebitBalance;
10
11     void withdraw(double amount)
12     {
13         if(credit - amount < 0 && OnDebitBalance != null) OnDebitBalance();
14     }
15 }
```

On vérifie que OnDebitBalance n'est pas null, car s'il n' a pas eu d'abonnement à notre évènement, on risque d'avoir NullReferenceException.

Abonnement aux évènements

Une fois notre évènement créé, des objets peuvent s'y abonner. Ces objets définissent l'action à exécuter lorsque l'évènement est levé. Dans notre exemple ci-dessous, on créé les méthodes **Sms_DebitBalanceReached** et **Email_DebitBalanceReached** qui seront exécutées lorsque l'évènement sera levé. La classe Main est notifiée de l'évènement et exécute la réponse définie.

```
static void Main(string[] args)
{
    Account acc = new Account();
    acc.OnDebitBalance += Sms_DebitBalanceReached;
    acc.OnDebitBalance += Email_DebitBalanceReached;
    acc.withdraw(100);
}
static void Sms_DebitBalanceReached()
{
    Console.WriteLine("Sms debit balance sent");
}
static void Email_DebitBalanceReached()
{
    Console.WriteLine("Email debit balance sent");
}
```

L'exécution du code ci-dessus, affichera les messages **Sms debit balance sent** et **Email debit balance sent**. Lorsqu'on appelle la méthode withdraw, le solde du compte est négatif et donc l'évènement OnDebitBalance est levé. La classe main étant abonnée à cet évènement par les méthodes Sms_DebitBalanceReached et Email_DebitBalanceReached, le code de ces deux méthodes est exécuté

Passage de paramètre

En pratique, pas besoin de créer des délégués, le Framework .NET propose les délégués **EventHandler** et **EventHandler<TEventArgs>** pour gérer les évènements.

On utilise EventHandler<TEventArgs> lorsqu'on veut passer des données à la méthode qui traitera l'évènement et EventHandler lorsqu'on ne passe aucune donnée. En utilisant EventHandler, notre classe Account devient.

```
class Account
{
    double credit;
    public event EventHandler OnDebitBalance;
    public void withdraw(double amount)
    {
        if (credit - amount < 0 && OnDebitBalance != null) OnDebitBalance(this, EventArgs.Empty);
    }
}
```

Il faudra également modifier les méthodes qui géreront l'évènement pour qu'elles aient la même signature que EventHandler.

```
static void Sms_DebitBalanceReached(object sender, EventArgs e)
{
    Console.WriteLine("Sms debit balance sent");
}
static void Email_DebitBalanceReached(object sender, EventArgs e)
{
```

```
    Console.WriteLine("Email debit balance sent");
}
```

Si dans notre exemple précédent, on veut passer le solde du compte en paramètre de notre message. On doit utiliser EventHandler<TEventArgs>. Pour cela on crée, une classe qui hérite de EventArgs.

```
class DebitBalanceEventArgs:EventArgs
{
    public double Balance { get; set; }
    public DebitBalanceEventArgs(double balance)
    {
        Balance = balance;
    }
}
```

Notre classe Account devient

```
class Account
{
    double credit;
    public event EventHandler<DebitBalanceEventArgs> OnDebitBalance;
    public void withdraw(double amount)
    {
        if (credit - amount < 0 && OnDebitBalance != null) OnDebitBalance(this, new DebitBalanceEventArgs(credit - amount));
    }
}
```

On modifie nos méthodes gestionnaires d'évènement en conséquence.

```
static void Sms_DebitBalanceReached(object sender, DebitBalanceEventArgs e)
{
    Console.WriteLine($"Sms debit balance sent. Value = {e.Balance}");
}
static void Email_DebitBalanceReached(object sender, DebitBalanceEventArgs e)
{
    Console.WriteLine($"Email debit balance sent. Value = {e.Balance}");
}
```

Voilà! Vous en savez assez pour créer vos propres évènements. J'espère que cet article vous aura aidé à mieux comprendre leur fonctionnement. Vous pourrez trouver plus d'informations dans la documentation officielle de Microsoft.

Auteur : Daniel MINKO FASSINOU