Livret – 10

Construction d'événements

Outil utilisé : C#

RM di scala

Cours informatique programmation

Rm di Scala - http://www.discala.net

10 : Les événements avec

Plan général:

1. Construction de nouveaux événements

Design Pattern observer
Abonné à un événement
Déclaration d'un événement
Invocation d'un événement
Comment s'abonner à un événement
Restrictions et normalisation
Evénement normalisé avec informations
Evénement normalisé sans information

2. Les événements dans les Windows.Forms

Contrôles visuels et événements
Evénement Paint : avec information
Evénement Click : sans information
Code C# généré

rm di Scala - 2006

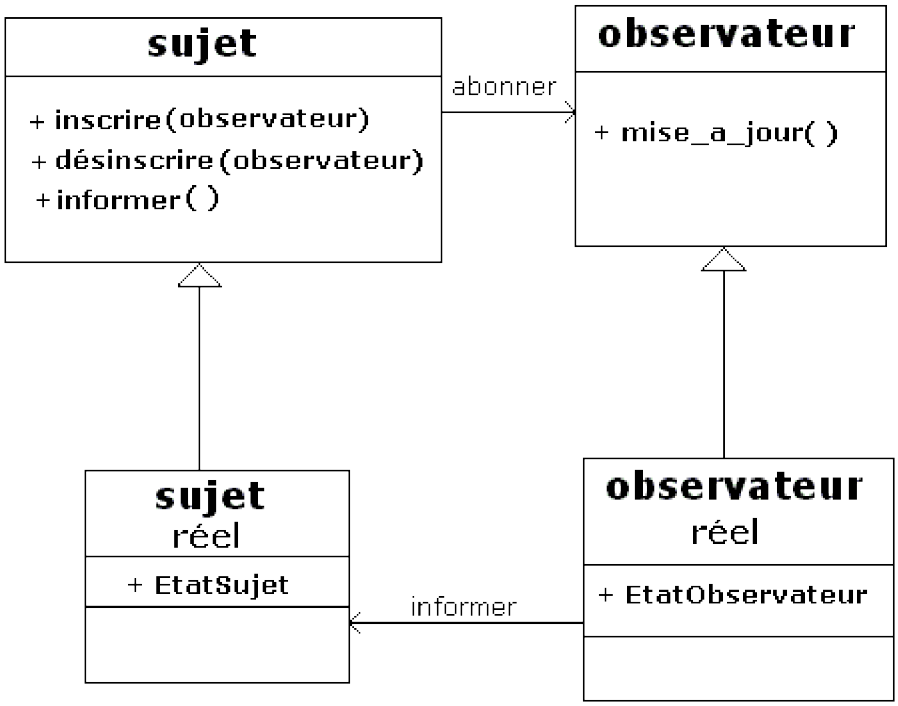
Rappel

Programmation orientée événements

Un programme objet orienté événements est construit avec des objets possédant des propriétés telles que les interventions de l'utilisateur sur les objets du programme et la liaison dynamique du logiciel avec le système déclenchent l'exécution de routines associées. Le traitement en programmation événementielle, consiste à mettre en place un mécanisme d'inteception puis de gestion permettant d'informer un ou plusieurs objets de la survenue d'un événement particulier.

1. Construction de nouveaux événements

Le modèle de conception de l'**observateur** (Design Pattern observer) est utilisé par Java et C# pour gérer un événement. Selon ce modèle, un client s'inscrit sur une liste d'abonnés auprès d'un observateur qui le préviendra lorsqu'un événement aura eu lieu. Les clients délèguent ainsi l'interception d'un événement à une autre entité. Java utilise ce modèle sous forme d'objet écouteur.



Design Pattern observer

Dans l'univers des Design Pattern on utilise essentiellement le modèle observateur dans les cas suivants :

• Quand le changement d'un objet se répercute vers d'autres.
• Quand un objet doit prévenir d'autres objets sans pour autant les connaitre.

C# propose des mécanismes supportant les événements, mais avec une implémentation totalement différente de celle de Java. En observant le fonctionnement du langage nous pouvons dire que C# combine efficacement les fonctionnalités de Java et de Delphi.

Abonné à un événement

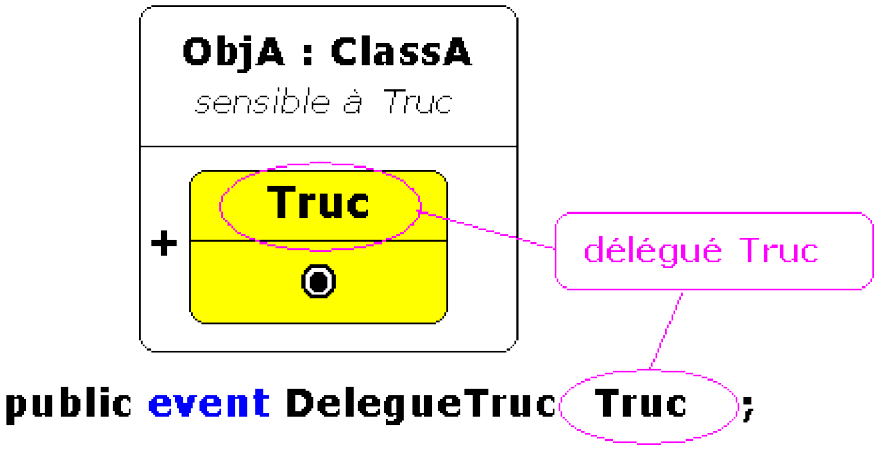
C# utilise les délégués pour fournir un mécanisme explicite permettant de gérer l'abonnement/notification.

En C# la délégation de l'écoute (gestion) d'un événement est confiée à un objet de type délégué : l'abonné est alors une méthode appelée gestionnaire de l'événement (contrairement à Java où l'abonné est une classe) acceptant les mêmes arguments et paramètres de retour que le délégué.

Déclaration d'un événement

type délégation :

```
public Delegate void DelegueTruc( ... ) ;
```



Code C# :

```
using System;
using System.Collections;

namespace ExempleEvent {
   //--> déclaration du type délégation :(par exemple procédure avec 1 paramètre string )
   public delegate void DelegueTruc (string s);

   public class ClassA   {
      //--> déclaration d'une référence event de type délégué :
      public event DelegueTruc Truc;
   }
}
```

Invocation d'un événement

rm di Scala - 2006

Une fois qu'une classe a déclaré un événement Truc, elle peut traiter cet événement exactement comme un délégué ordinaire. La démarche est très semblable à celle de Delphi, le champ Truc vaudra **null** si le client ObjA de ClassA n'a pas raccordé un délégué à l'événement Truc. En effet être sensible à plusieurs événements n'oblige pas chaque objet à gérer tous les événement, dans le cas où un objet ne veut pas gérer un événement Truc on n'abonne aucune méthode au délégué Truc qui prend alors la valeur **null**.

Dans l'éventualité où un objet doit gérer un événement auquel il est sensible, il doit invoquer l'événement Truc (qui référence une ou plusieurs méthodes). Invoquer un événement Truc consiste généralement à vérifier d'abord si le champ Truc est null, puis à appeler l'événement (le délégué Truc).

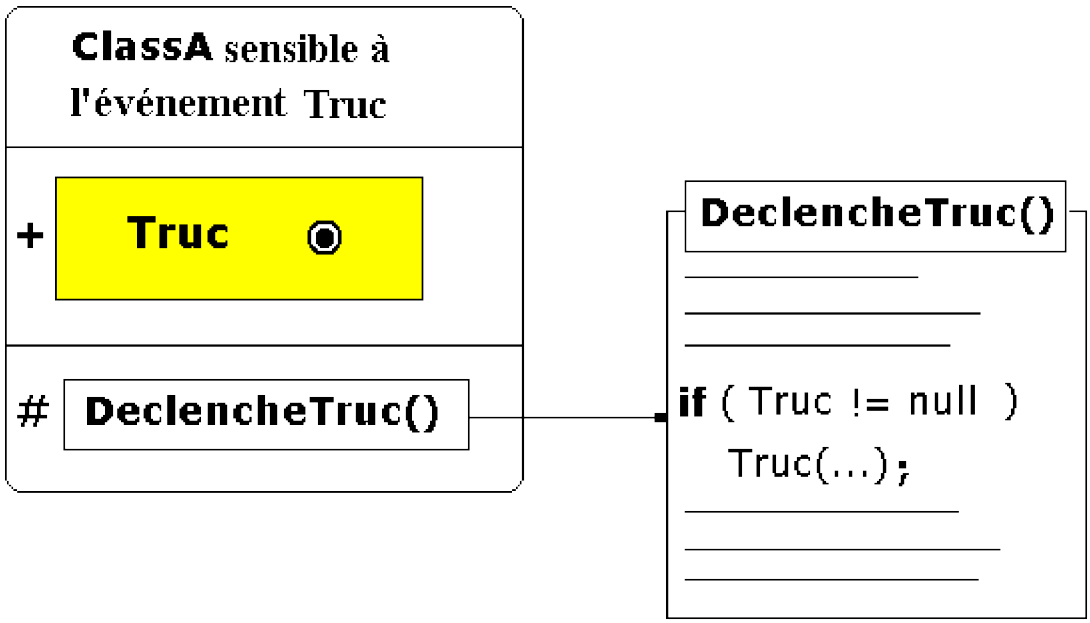
Remarque importante

L'appel d'un événement ne peut être effectué qu'à partir de la classe qui a déclaré cet événement.

Exemple construit pas à pas

Considérons ci-dessous la classe ClassA qui est sensible à un événement que nous nommons Truc (on déclare la référence Truc), dans le corps de la méthode **void** DeclencheTruc( ) on appelle l'événement Truc.

Nous déclarons cette méthode **void** DeclencheTruc( ) comme **virtuelle** et **protégée**, de telle manière qu'elle puisse être redéfinie dans la suite de la hiérarchie; ce qui constitue un gage d'évolutivité des futures classes quant à leur comportement relativement à l'événement Truc :



Il nous faut aussi prévoir une méthode **publique** qui permettra d'invoquer l'événement depuis une autre classe, nous la nommons LancerTruc.

Code C# , construisons progressivement notre exemple, voici les premières lignes du code :

```
using System;
using System.Collections;

namespace ExempleEvent {
   //--> déclaration du type délégation :(par exemple procédure avec 1 paramètre string)
   public delegate void DelegueTruc (string s);

   public class ClassA   {
      //--> déclaration d'une référence event de type délégué :
      public event DelegueTruc Truc;
      protected virtual void DeclencheTruc( ) {
          ....
          if ( Truc != null ) Truc("événement déclenché");
          ....
      }
      public void LancerTruc( ) {
          ....
          DeclencheTruc( ) ;
          ....
      }
   }
}
```

Comment s'abonner (se raccorder, s'inscrire, ...) à un événement

Un événement ressemble à un champ public de la classe qui l'a déclaré. Toutefois l'utilisation de ce champ est très restrictive, c'est pour cela qu'il est déclaré avec le spécificateur **event**. Seulement deux opérations sont possibles sur un champ d'événement qui rapellons-le est un délégué :

- Ajouter une nouvelle méthode (à la liste des méthodes abonnées à l'événement).
- Supprimer une méthode de la liste (désabonner une méthode de l'événement).

Enrichissons le code C# précédent avec la classe ClasseUse :

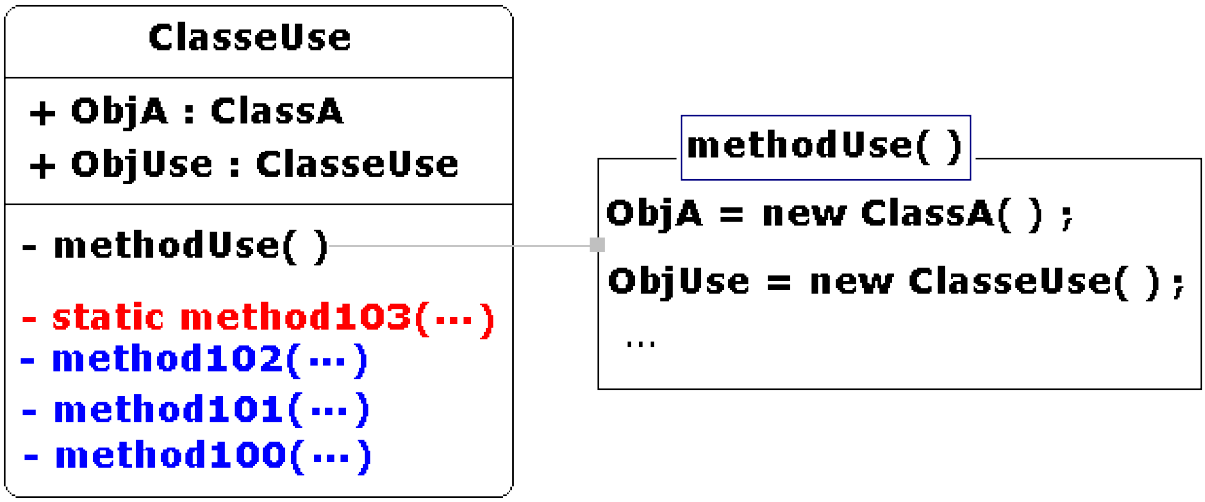
```
using System;
using System.Collections;

namespace ExempleEvent {
   //--> déclaration du type délégation :(par exemple procédure avec 1 paramètre string)
   public delegate void DelegueTruc (string s);

   public class ClassA   {
      //--> déclaration d'une référence event de type délégué :
      public event DelegueTruc Truc;

      protected virtual void DeclencheTruc( ) {
          ....
          if ( Truc != null ) Truc("événement déclenché");
          ....
      }
      public void LancerTruc( ) {
```


rm di Scala - 2006

```
            ....
            DeclencheTruc( ) ;
            ....
        }
    }
    public class ClasseUse {

        static private void methodUse( ) {
            ClassA   ObjA = new ClassA( );
            ClasseUse ObjUse = new ClasseUse ( );
            //....
        }
        static public void Main(string[] x) {
            methodUse( ) ;
            //....
        }
    }
}
```

Il faut maintenant définir des gestionnaires de l'événement Truc (des méthodes ayant la même signature que le type délégation " **public delegate void** DelegueTruc (string s); ". Ensuite nous ajouterons ces méthodes au délégué Truc (nous les abonnerons à l'événement Truc), ces méthodes peuvent être de classe ou d'instance.

Supposons que nous ayons une méthode de classe et trois méthodes d'instances qui vont s'inscrire sur la liste des abonnés à Truc, ce sont quatre gestionnaires de l'événement Truc :

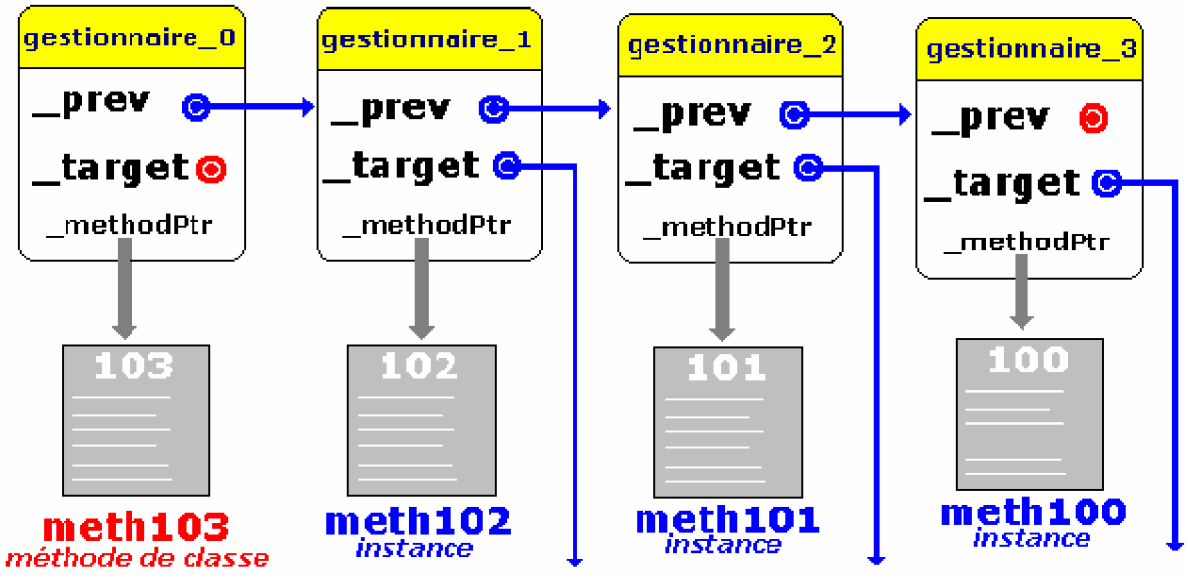


Ajoutons au code C# de la classe ClassA les quatre gestionnaires :

```
using System;
using System.Collections;

namespace ExempleEvent {
    //--> déclaration du type délégation :(par exemple procédure avec 1 paramètre string)
    public delegate void DelegueTruc (string s);

    public class ClassA    {
        //--> déclaration d'une référence event de type délégué :
        public event DelegueTruc Truc;
```

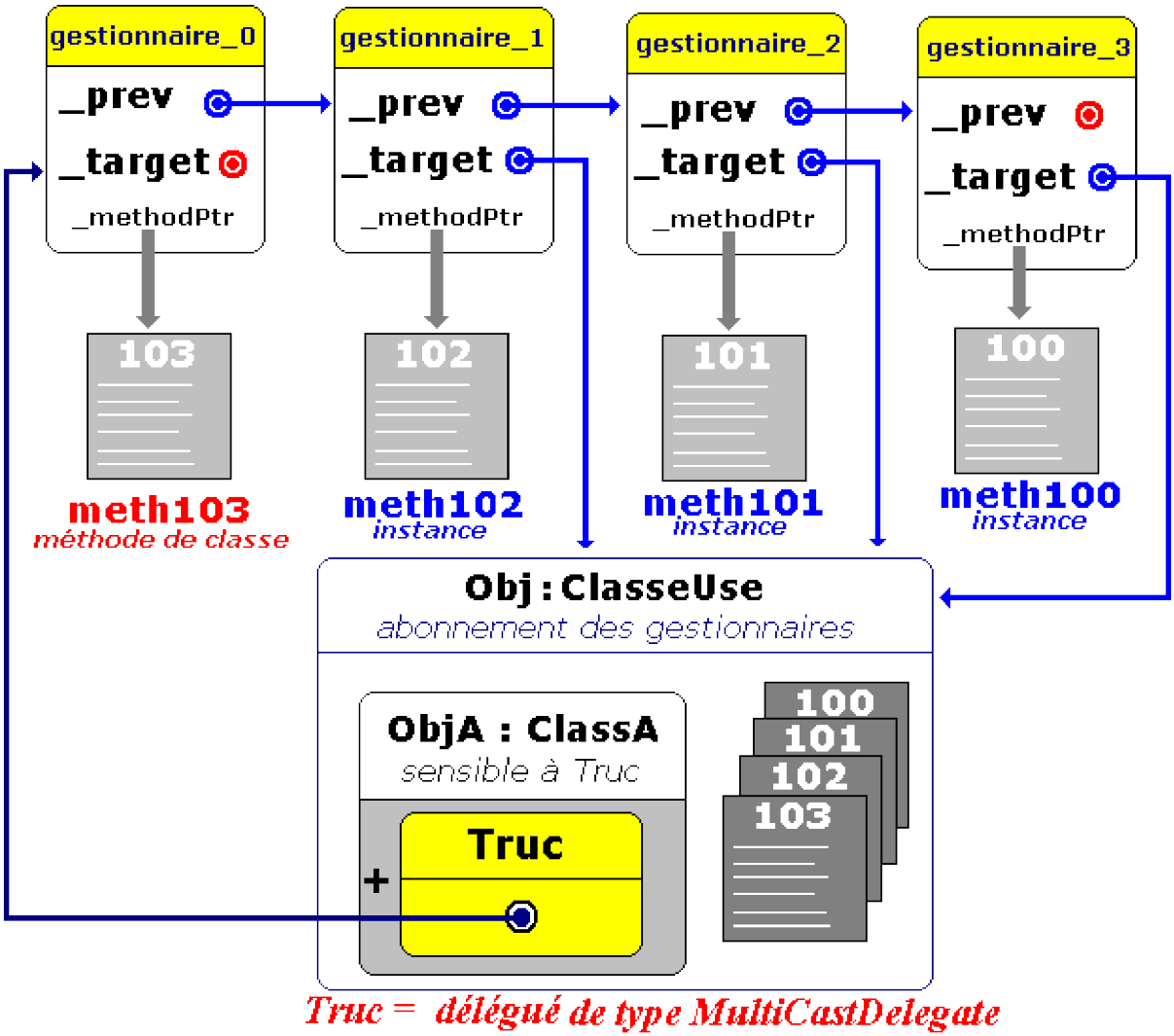
```
        protected virtual void DéclencheTruc( ) {
            ....
            if ( Truc != null ) Truc("événement déclenché")
            ....
        }
        public void LancerTruc( ) {
            ....
            DeclencheTruc( ) ;
            ....
        }
    }
    public class ClasseUse {
        public void method100(string s); {
            //...gestionnaire d'événement Truc: méthode d'instance.
        }
        public void method101(string s); {
            //...gestionnaire d'événement Truc: méthode d'instance.
        }
        public void method102(string s); {
            //...gestionnaire d'événement Truc: méthode d'instance.
        }
        static public void method103(string s); {
            //...gestionnaire d'événement Truc: méthode de classe.
        }
        static private void methodUse( ) {
            ClassA   ObjA = new ClassA( );
            ClasseUse ObjUse = new ClasseUse ( );
            //... il reste à abonner les  gestionnaires de l'événement Truc
        }
        static public void Main(string[ ] x) {
            methodUse( ) ;
        }
    }
}
```

Lorsque nous ajoutons en C# les nouvelles méthodes method100, ... , method103 au délégué Truc, par surcharge de l'opérateur +, nous dirons que les gestionnaires method100,...,method103, s'abonnent à l'événement Truc.



rm di Scala - 2006

Prévenir (informer) un abonné correspond ici à l'action d'appeler l'abonné (appeler la méthode) :



Terminons le code C# associé à la figure précédente :

Nous complétons le corps de la méthode " **static private void** methodUse( ) " par l'abonnement au délégué des quatre gestionnaires.

Pour invoquer l'événement Truc, il faut pouvoir appeler enfin à titre d'exemple une invocation de l'événement :

```
using System;
using System.Collections;

namespace ExempleEvent {
   //--> déclaration du type délégation :(par exemple procédure avec 1 paramètre string)
   public delegate void DelegueTruc (string s);

   public class ClassA    {
      //--> déclaration d'une référence event de type délégué :
      public event DelegueTruc Truc;

      protected virtual void DeclencheTruc( ) {
          //....
          if ( Truc != null ) Truc("événement Truc déclenché");
          //....
      }
      public void LancerTruc( ) {
          //....
          DeclencheTruc( ) ;
          //....
      }
   }
   public class ClasseUse {
      public void method100(string s) {
          //...gestionnaire d'événement Truc: méthode d'instance.
          System.Console.WriteLine("information utilisateur : "+s);
      }
      public void method101(string s) {
          //...gestionnaire d'événement Truc: méthode d'instance.
          System.Console.WriteLine("information utilisateur : "+s);
      }
      public void method102(string s) {
          //...gestionnaire d'événement Truc: méthode d'instance.
          System.Console.WriteLine("information utilisateur : "+s);
      }
      static public void method103(string s) {
          //...gestionnaire d'événement Truc: méthode de classe.
          System.Console.WriteLine("information utilisateur : "+s);
      }
      static private void methodUse( ) {
          ClassA   ObjA = new ClassA( );
          ClasseUse ObjUse = new ClasseUse ( );
          //-- abonnement des gestionnaires:
          ObjA.Truc += new DelegueTruc (  ObjUse.method100 );
          ObjA.Truc += new DelegueTruc (  ObjUse.method101 );
          ObjA.Truc += new DelegueTruc (  ObjUse.method102 );
          ObjA.Truc += new DelegueTruc (  method103 );
          //-- invocation de l'événement:
          ObjA.LancerTruc( ) ;  //...l'appel à cette méthode permet d'invoquer l'événement Truc
      }
      static public void Main(string[] x) {
          methodUse( ) ;
      }
   }
}
```

Restrictions et normalisation .NET Framework

rm di Scala - 2006

Bien que le langage C# autorise les événements à utiliser n'importe quel type délégué, le .NET Framework applique à ce jour à des fins de normalisation, certaines indications plus restrictives quant aux types délégués à utiliser pour les événements.

Les indications .NET Framework spécifient que le type délégué utilisé pour un événement doit disposer de deux paramètres et d'un retour définis comme suit :

- un paramètre de type **Object** qui désigne la source de l'événement,
- un autre paramètre soit de classe **EventArgs ,** soit d'une classe qui **dérive** de **EventArgs**, il encapsule toutes les informations personnelles relatives à l'événement,
- enfin le type du retour du délégué doit être **void**.

Evénement normalisé sans information :

Si vous n'utilisez pas d'informations personnelles pour l'événement, la signature du délégué sera :

public delegate void DelegueTruc (**Object** sender , **EventArgs** e) ;

Evénement normalisé avec informations :

Si vous utilisez des informations personnelles pour l'événement, vous définirez une classe **MonEventArgs** qui hérite de la classe **EventArgs** et qui contiendra ces informations personnelles, dans cette éventualité la signature du délégué sera :

public delegate void DelegueTruc (**Object** sender , **MonEventArgs** e) ;

Il est conseillé d'utiliser la représentation normalisée d'un événement comme les deux exemples ci-dessous le montre, afin d'augmenter la lisibilité et le portage source des programmes événementiels.

Mise en place d'un événement normalisé avec informations

Pour mettre en place un événement Truc normalisé contenant des informations personnalisées vous devrez utiliser les éléments suivants :

1°) une classe d'informations personnalisées sur l'événement
2°) une déclaration du type délégation normalisée (nom terminé par EventHandler)
*3°) une déclaration d'une référence Truc du type délégation normalisée spécifiée **event***
*4.1°) une méthode protégée qui déclenche l'événement Truc (nom commençant par **On**: OnTruc)*
4.2°) une méthode publique qui lance l'événement par appel de la méthode OnTruc
5°) un ou plusieurs gestionnaires de l'événement Truc
6°) abonner ces gestionnaires au délégué Truc
7°) consommer l'événement Truc

Exemple de code C# directement exécutable, associé à cette démarche :

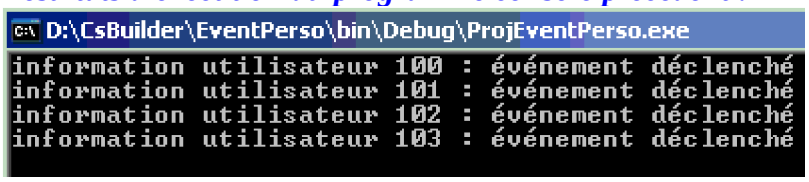
```
using System;
using System.Collections;

namespace ExempleEvent {

//--> 1°) classe d'informations personnalisées sur l'événement

public class TrucEventArgs : EventArgs   {
     public string info ;
     public TrucEventArgs (string s) {
       info = s ;
     }
}

   //--> 2°) déclaration du type délégation normalisé

public delegate void DelegueTrucEventHandler (  object sender,  TrucEventArgs e );

   public class ClassA   {
      //--> 3°) déclaration d'une référence event de type délégué :

       public event DelegueTrucEventHandler Truc;

     //--> 4.1°) méthode protégée qui déclenche l'événement :

      protected virtual void OnTruc(  object sender,  TrucEventArgs e ) {
        //....
        if ( Truc != null ) Truc( sender , e );
        //....
      }

     //--> 4.2°) méthode publique qui lance l'événement :

      public void LancerTruc( ) {
        //....
        TrucEventArgs evt = new TrucEventArgs ("événement déclenché" ) ;
        OnTruc ( this , evt );
        //....
      }

   }

    public class ClasseUse {
          //--> 5°) les gestionnaires d'événement Truc

            public void method100(  object sender,  TrucEventArgs e ){
                  //...gestionnaire d'événement Truc: méthode d'instance.
                  System.Console.WriteLine("information utilisateur 100 : "+e.info);
             }
```


rm di Scala - 2006

```
        public void method101(  object sender,  TrucEventArgs e ) {
                //...gestionnaire d'événement Truc: méthode d'instance.
                System.Console.WriteLine("information utilisateur 101 : "+e.info);
        }

        public void method102(  object sender,  TrucEventArgs e ) {
                //...gestionnaire d'événement Truc: méthode d'instance.
                System.Console.WriteLine("information utilisateur 102 : "+e.info);
        }

        static public void method103(  object sender,  TrucEventArgs e ) {
                //...gestionnaire d'événement Truc: méthode de classe.
                System.Console.WriteLine("information utilisateur 103 : "+e.info);
        }
```

```
static private void methodUse( ) {
        ClassA   ObjA = new ClassA( );
        ClasseUse ObjUse = new ClasseUse ( );
        //--> 6°) abonnement des gestionnaires:
        ObjA.Truc += new DelegueTrucEventHandler (  ObjUse.method100 );
        ObjA.Truc += new DelegueTrucEventHandler (  ObjUse.method101 );
        ObjA.Truc += new DelegueTrucEventHandler (  ObjUse.method102 );
        ObjA.Truc += new DelegueTrucEventHandler (  method103 );

        //--> 7°) consommation de l'événement:
        ObjA.LancerTruc( ) ;  //...l'appel à cette méthode permet d'invoquer l'événement Truc
    }
    static public void Main(string[] x) {
        methodUse( ) ;
    }
  }
}
```

Résultats d'exécution du programme console précédent :



Mise en place d'un événement normalisé sans information

En fait, pour les événements qui n'utilisent pas d'informations supplémentaires personnalisées, le .NET Framework a déjà défini un type délégué approprié : **System.EventHandler** (équivalent au TNotifyEvent de Delphi):

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

Pour mettre en place un événement Truc normalisé sans information spéciale, vous devrez

utiliser les éléments suivants :

> *1°) la classe System.EventArgs*
> *2°) le type délégation normalisée System.EventHandler*
> *3°) une déclaration d'une référence Truc du type délégation normalisée spécifiée **event***
> *4.1°) une méthode protégée qui déclenche l'événement Truc (nom commençant par **On**: OnTruc)*
> *4.2°) une méthode publique qui lance l'événement par appel de la méthode OnTruc*
> *5°) un ou plusieurs gestionnaires de l'événement Truc*
> *6°) abonner ces gestionnaires au délégué Truc*
> *7°) consommer l'événement Truc*

Remarque, en utilisant la déclaration **public delegate void** EventHandler (**object** sender, **EventArgs** e) contenue dans **System.EventHandler**, l'événement n'ayant aucune donnée personnalisée, le deuxième paramètre n'étant pas utilisé, il est possible de fournir le champ **static** Empty de la classe **EventArgs** .

EventArgs, membres

Constructeurs publics

EventArgs, constructeur
Initialise une nouvelle instance de la classe **EventArgs**.

Champs publics

Empty
Représente un événement sans données d'événement.

Exemple de code C# directement exécutable, associé à un événement sans information personnalisée :

```
using System;
using System.Collections;

namespace ExempleEvent {

//--> 1°) classe d'informations personnalisées sur l'événement

   System.EventArgs est déjà déclarée dans using System;

//--> 2°) déclaration du type délégation normalisé

   System.EventHandler est déjà déclarée dans using System;

    public class ClassA   {
        //--> 3°) déclaration d'une référence event de type délégué :

        public event EventHandler Truc;

        //--> 4.1°) méthode protégée qui déclenche l'événement :
```


rm di Scala - 2006

```
        protected virtual void OnTruc(  object sender,  EventArgs e ) {
          //....
          if ( Truc != null ) Truc( sender , e );
          //....
        }
```

//--> 4.2°) méthode publique qui lance l'événement :

```
        public void LancerTruc( ) {
          //....
          OnTruc( this , EventArgs.Empty );
          //....
        }
    }

    public class ClasseUse {
```

//--> 5°) les gestionnaires d'événement Truc

```
        public void method100(  object sender,  EventArgs e ){
                //...gestionnaire d'événement Truc: méthode d'instance.
                System.Console.WriteLine("information utilisateur 100 : événement déclenché");
        }


        public void method101(  object sender,  EventArgs e ) {
                //...gestionnaire d'événement Truc: méthode d'instance.
                System.Console.WriteLine("information utilisateur 101 : événement déclenché");
        }


        public void method102(  object sender,  EventArgs e ) {
                //...gestionnaire d'événement Truc: méthode d'instance.
                System.Console.WriteLine("information utilisateur 102 : événement déclenché");
        }


        static public void method103(  object sender,  EventArgs e ) {
                //...gestionnaire d'événement Truc: méthode de classe.
                System.Console.WriteLine("information utilisateur 103 : événement déclenché");
        }

          static private void methodUse( ) {
            ClassA   ObjA = new ClassA( );
            ClasseUse ObjUse = new ClasseUse ( );
```

//--> 6°) abonnement des gestionnaires:

```
            ObjA.Truc += new DelegueTruc (  ObjUse.method100 );
            ObjA.Truc += new DelegueTruc (  ObjUse.method101 );
            ObjA.Truc += new DelegueTruc (  ObjUse.method102 );
            ObjA.Truc += new DelegueTruc (  method103 );
```

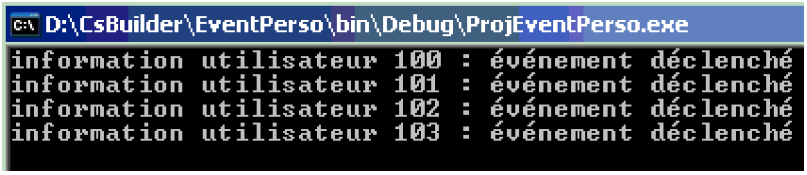
//--> 7°) consommation de l'événement:

```
            ObjA.LancerTruc( ) ;
```

//...l'appel à cette méthode permet d'invoquer l'événement Truc

```
          }
          static public void Main(string[] x) {
            methodUse( ) ;
          }
    }
}
```

Résultats d'exécution de ce programme :



2. Les événements dans les Windows.Forms

Le code et les copies d'écran sont effectuées avec C#Builder 1.0 de Borland version personnelle gratuite.

Contrôles visuels et événements

L'architecture de fenêtres de .Net FrameWork se trouve essentiellement dans l'espace de noms **System.Windows.Forms** qui contient des classes permettant de créer des applications contenant des IHM (interface humain machine) et en particulier d'utiliser les fonctionnalités afférentes aux IHM de Windows.

Plus spécifiquement, la classe System.Windows.Forms.Control est la classe mère de tous les composants visuels. Par exemple, les classes **Form**, **Button**, **TextBox**, etc... sont des descendants de la classe Control qui met à disposition du développeur C# 58 événements auxquels un contrôle est sensible.

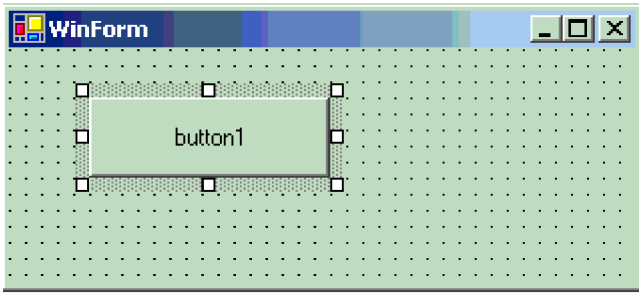
Ces 58 événements sont tous normalisés, certains sont des événements sans information spécifique, d'autres possèdent des informations spécifiques, ci-dessous un extrait de la liste des événements de la classe Control, plus particulièrement les événements traitant des actions de souris :

rm di Scala - 2006

Control : Événements publics	
BackColorChanged	Se produit lorsque la valeur de la propriété BackColor change.
BackgroundImageChanged	Se produit lorsque la valeur de la propriété BackgroundImage change.
.....	.....
Click	Se produit suite à un clic sur le contrôle.
.....	.....
MouseDown	Se produit lorsque le pointeur de la souris se trouve sur le contrôle et qu'un bouton de la souris est enfoncé.
MouseEnter	Se produit lorsque le pointeur de la souris se place dans le contrôle.
MouseHover	Se produit lorsque la souris pointe sur le contrôle.
MouseLeave	Se produit lorsque le pointeur de la souris s'écarte du contrôle.
MouseMove	Se produit lorsque le pointeur de la souris est placé sur le contrôle.
MouseUp	Se produit lorsque le pointeur de la souris se trouve sur le contrôle et qu'un bouton de la souris est relâché.
MouseWheel	Se produit lorsque la roulette de la souris bouge alors que le contrôle a le focus.
Move	Se produit lorsque le contrôle est déplacé.
Paint	Se produit lorsque le contrôle est redessiné.
.....	.....

Nous avons mis en évidence deux événements Click et Paint dont l'un est sans information (Click), l'autre est avec information (Paint). Afin de voir comment nous en servir nous traitons un exemple :

Soit une fiche (classe Form1 héritant de la classe Form) sur laquelle est déposé un bouton poussoir (classe Button) de nom **button1** :



Montrons comment nous programmons la réaction du bouton à un click de souris et à son redessinement. Nous devons faire réagir button1 qui est sensible à au moins 58 événements, aux deux événements Click et Paint. Rappelons la liste méthodologique ayant trait au cycle événementiel, on doit utiliser :

1°) une classe d'informations personnalisées sur l'événement
2°) une déclaration du type délégation normalisée (nom terminé par EventHandler)
*3°) une déclaration d'une référence Truc du type délégation normalisée spécifiée **event***
*4.1°) une méthode protégée qui déclenche l'événement Truc (nom commençant par **On**: OnTruc)*
4.2°) une méthode publique qui lance l'événement par appel de la méthode OnTruc
5°) un ou plusieurs gestionnaires de l'événement Truc
6°) abonner ces gestionnaires au délégué Truc
7°) consommer l'événement Truc

Les étapes 1° à 4° ont été conçues et developpées par les équipes de .Net et ne sont plus à notre charge.

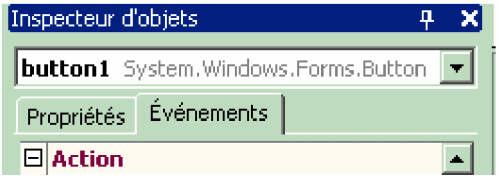
Il nous reste les étapes suivantes :

- 5°) à construire un gestionnaire de réaction de **button1** à l'événement Click et un gestionnaire de réaction de **button1**
- 6°) à abonner chaque gestionnaire au délégué correspondant (Click ou Paint)

L'étape 7° est assurée par le système d'exploitation qui se charge d'envoyer des messages et de lancer les événements.

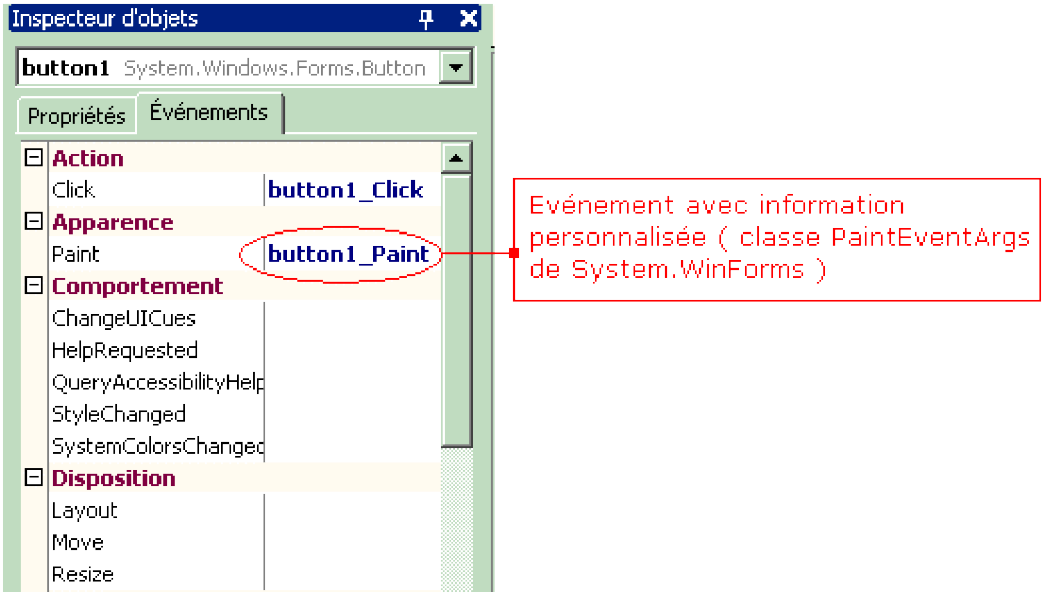
Evénement Paint : normalisé avec informations

Voici dans l'inspecteur d'objets de C#Builder l'onglet Evénements qui permet de visualiser le délégué à utiliser ainsi que le gestionnaire à abonner à ce délégué.



Dans le cas de l'événement Paint, le délégué est du type **PaintEventArgs** situé dans System.WinForms :

rm di Scala - 2006



signature du délégué Paint dans System.Windows.Forms :

public delegate void PaintEventHandler (**object** sender, **PaintEventArgs** e);

La classe **PaintEventArgs** :

PaintEventArgs

Constructeur public

PaintEventArgs, constructeur

Initialise une nouvelle instance de la classe **PaintEventArgs** avec les graphiques et le rectangle de découpage spécifiés.

Propriétés publiques

ClipRectangle

Obtient le rectangle dans lequel peindre.

Graphics

Obtient le graphique utilisé pour peindre.

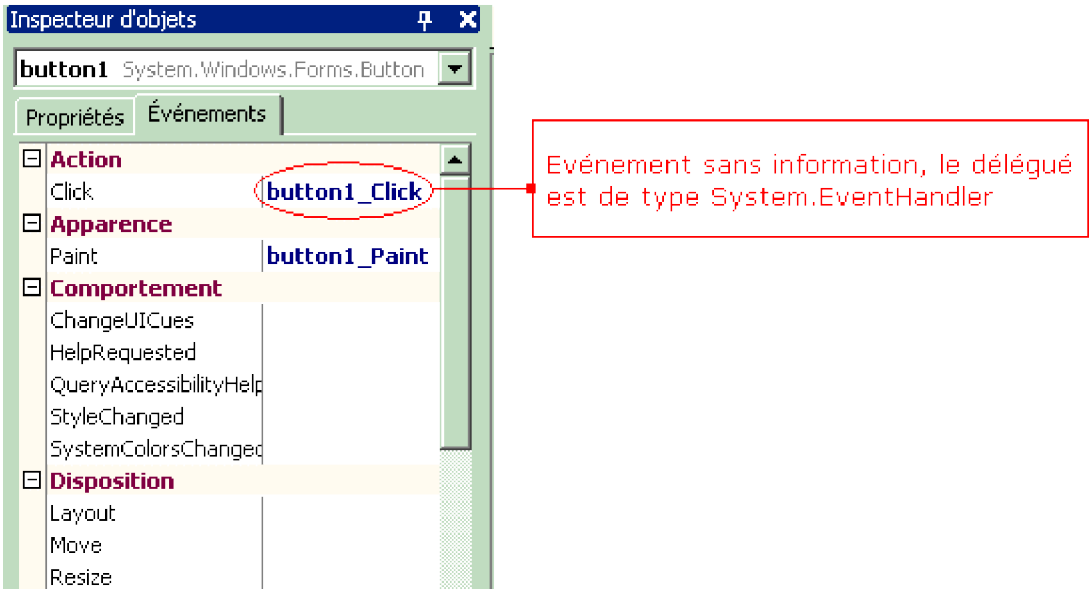
Méthodes publiques

Dispose

Surchargé. Libère les ressources utilisées par **PaintEventArgs**.

Evénement Click normalisé sans information

Dans le cas de l'événement Click, le délégué est de type **Event Handler** situé dans System :



signature du délégué Click dans System :

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

En fait l'inspecteur d'objet de C#Builder permet de réaliser en mode visuel la machinerie des étapes qui sont à notre charge :

- le délégué de l'événement, [**public event** EventHandler Click;]
- le squelette du gestionnaire de l'événement, [**private void** button1_Click(object sender, System.EventArgs e){ }]
- l'abonnement de ce gestionnaire au délégué. [**this.**button1.Click += **new** System.EventHandler (**this.**button1_Click);]

Code C# généré

rm di Scala - 2006

Voici le code généré par C#Builder utilisé en conception visuelle pour faire réagir button1 aux deux événements Click et Paint :

```
public class WinForm : System.Windows.Forms.Form    {
      private System.ComponentModel.Container components = null ;
      private System.Windows.Forms.Button button1;

      public WinForm( ) {
         InitializeComponent( );
      }

      protected override void Dispose (bool disposing) {
         if (disposing) {
            if (components != null ) {
               components.Dispose( ) ;
            }
         }
         base.Dispose(disposing) ;
      }
      private void InitializeComponent( ) {
         this.button1 = new System.Windows.Forms.Button( );
         ......

         this.button1.Click += new System.EventHandler( this.button1_Click );


         this.button1.Paint += new System.Windows.Forms.PaintEventHandler( this.button1_Paint );


            .....
         }
         static void Main( ) {
            Application.Run( new WinForm( ) );
         }

      private void button1_Click(object sender, System.EventArgs e)       {
         //..... gestionnaire de l'événement OnClick
       }


       private void button1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)       {
         //..... gestionnaire de l'événement OnPaint
       }

}
```

La machinerie événementielle est automatiquement générée par C#Builder comme dans Delphi, ce qui épargne de nombreuses lignes de code au développeur et le laisse libre de penser au code spécifique de réaction à l'événement.