TP3 : On s'approche ...

19 novembre 2007

Le contenu de ce TP va être important pour le projet, qui va être annoncé demain sur la page web...

1 Reprise

(1 :00 h)

1.1 Rappel : Programmation par objet

On envisage de créer une liste de primitifs géométrique : Circle, Square. Les deux classes doivent dériver de la classe de base GeoPrimitve. GeoPrimitive impose quelques fonctions de base. Ceci se fait en définissant des fonctions dans GeoPrimitive :

```
public class GeoPrimitive {
    //l'aire
    public abstract float area();
    //perimetre
    public abstract float circumference();
}
```

Créez des fonctions dans les classes dérivées. Rappel :

```
public class Cercle : GeoPrimitive {
    //l'aire
    public override float area(){    ...    }
    //perimetre
    public override float circumference(){    ...    }
}
```

1.2 Rappel : Fonctions Static

Tout ce qui est marqué *static* appartient à une classe. Autrement dit, n'importe quel objet de cette classe va partager ce qui est statique.

Static est utilisé dans deux contextes, premièrement si ca n'a pas vraiment un sens de créer un objet de ce type. Un exemple serait une classe qui trie des

numéros. Il semble naturel de définir une fonction qui prend un tableau à trier. Donc il n'a pas de sens que quelqu'un qui veut appeler cette fonction à besoin de créer un objet pour appeler sa méthode de trie...

L'autre situation est quand on veut avoir une information partagé au lieu d'avoir des valeurs différents par objet. Par exemple si on veut créer une identifiant unique par objet, on peut incrementer un compteur static dans la classe et attribuer à chaque création de le compteur courrant.

On peut faire un appel d'une fonction static dans une fonction membre quelconque. Par contre, un appel d'une fonction non-static à l'intérieur d'une fonction non-static du même objet est impossible.

Testez le programme suivant et comprenez ce qui se passe.

```
using System;
using System.Text;

public class A
{
    public static int a;
    public int b;
    public int t
    {
        set { a = value; }
        get { return a; }
    }
}

class Prog
{
    public static void Main()
    {
        A test = new A();
        A test2 = new A();

        test.t = 1;
        test.b = 2;
        test2.t = 3;
        test2.b = 4;

        Console.WriteLine("test:{0} {1}, test2: {2} {3}",
                          test.t, test.b, test2.t, test2.b);
        Console.ReadLine();
    }
}
```

2 Comment créer une interface

(3 :00 h en total) Maintenant on va commencer à créer notre premier interface. Ceci est très simple :

```
using System;
using System.Windows.Forms; //Attention il faut donc linker avec ces librairies:
using System.Text;//csc test.cs r:System.Windows.Forms r:System.Text

 public class Form1 : Form {
    public Form1()
    {
        InitializeComponent();
    }
    //c'est standard de créer l'interface avec cette fonction
    private void InitializeComponent()
    {
        //on desactive le update de l'interface
        this.SuspendLayout();

        //ici on ajoute les composantes de l'interface
        // par exemple vos buttons.

        //on reactive le update de l'interface
        this.ResumeLayout(false);
    }
    static void Main()
    {
        Application.Run(new Form1());//une boucle infinie qui gère l'interface
    }
}
```

Pouvez vous trouver comment changer la taille de votre interface ? 2 conseils : InitializeComponent() et this.Size ...

Il y a beaucoup de classes qui permettent de créer des composantes d'interface. (regardez sur MSDN)

Par exemple, il y a la classe Button. Pour en ajouter un, il faut le créer et ensuite attacher aux contrôles de l'interface. Ceci se fait dans la fonction InitializeComponent().

```
//création
button=new Button();
//attacher à l'interface
Controls.Add(button);
```

Jouez un peu avec ces propriétés :

```
button.Location = new Point(0, 0); //a field almost all elements have

button.Size = new Size(100, 100); //idem

button.Name ="SuperButton"; //idem

button.Text = "BLABLA";
```

Essayez d'ajouter plusieurs button. Qu'est-ce qui se passe, si on les superposent ?

Par défaut le layout se fait en coordonnées absolues (il y a quand même des gens qui ont créé des layouts standards). Dans l'exemple qui sera sur le web (demain), on peut voir comment créer une interface qui s'adapte à la taille de la fenêtre. Pour la suite (aussi pour le projet) on va créer une fenêtre de taille fixe. En pratique, on utilisera la résolution du PDA Dell Axim (480×640)[1]. On va probablement faire tourner le projet final sur cette machine...

2.1 Interaction

C'est joli d'avoir une interface, mais maintenant on veut de l'interactivité.
Créez une fonction

```
public void button_pressed(object sender, EventArgs e) {
    try{
        Button b= (Button)sender;
        b.name="TEST";
    }catch (Exception e)
    {
        //quelle exception pourrait on rencontrer ici???
    }
}
```

Ajoutez cette fonction à votre button dans la fonction *initializeComponent()*.

```
button.Click+= new EventHandler(button_pressed);
```

Testez-le.

2.2 Complétez le TicTacToe.

Astuce : Utilisez le nom du button pour savoir qui a été cliqué ?

3 Les évenements

```
button.Click+= new EventHandler(button_pressed);
```

[1] Oui, c'est plus haut que large

Qu'est-ce que ca veut dire ?

Alors, en CSharp il y a ce qu'on appelle events (évènements). Un event peut être déclanché comme une exception. La différence est qu'une exception est quelque chose qui ne devrait pas se passer. De l'autre côté un event est qqc qui pourrait se produire et ce qui est prévu. Par exemple, si un utilisateur clique sur un button, ce n'est pas du tout "exceptionnelle"... c'est plutôt un évènement (peut-être pas nécessairement majeure ;)) ... Il est impossible de prédire ce que l'utilisateur fait du coup on veut seulement le traiter au moment que ca arrive. Un autre exemple est un timer, imaginez qu'à midi on veut être notifié pour aller manger. Imaginez que vous créez du coup une boîte de dialogue, qui va être affiché quand l'horloge arrive à midi. Maintenant vous pouvez dire pourquoi je mettrai donc pas d'horloge dans l'objet boîte de dialogue ? Imaginez que pas seulement l'ordinateur va afficher "temps à manger" mais en plus il va commencer à lancer des applications de maintenance (défragmenter, diminuer la consommation d'énergie etc.). Au lieu d'avoir une horloge par tache, il est plus facile de créer un évènement "pause" qui va être connecté à toutes ces tâches. Si maintenant la pause est décalée, seulement l'horloge a besoin de changer le moment qu'elle envoie l'event "pause". Ce concept d'events est très puissant et il est utilisé dans beaucoup de librairies d'interface (e.g. Qt).

Un autre concept extraordinaire de CSharp sont les *delegates*[2]. Un delegate définit l'apparence d'une fonction (ces paramètres ainsi que son type de retour). Maintenant on peut créer un représentant d'un delegate en passant une fonction du bonne type. Pourquoi aurait-on besoin de ca ?Pour beaucoup de raisons ! Un delegate va nous permettre de passer une fonction en paramètre d'une fonction. **Relisez la phrase au pour bien la comprendre ! ! !** Ceci permet par exemple de réaliser une fonction de trie générale. Elle accepte en paramètre une fonction de comparaison. Cette fonction pourrait assurer un ordre croissant ou décroissant. Voilà un exemple :

```
using System;
using System.Text;

public class Sorting
{
    public delegate bool Compare(int a, int b);

    //en CSharp on peut definir un constructeur static!
    //il peut être défini en plus d'un constructeur normal.
    //la difference est qu'il n'est appelé qu'une seule fois
    static Sorting()
    {
        Croissant = new Compare(Crois);
        Decroissant=new Compare(Decrois);
    }
```

[2]dans d'autres langages l'équivalent est souvent appelé functor

```
    public static void Sort(ref int a, ref int b, Compare c)
    {
        if (c(a,b))
        {
            int temp=a;
            a=b;
            b=temp;
        }
    }

    public static Compare Croissant;
    public static Compare Decroissant;

    private static bool Decrois(int a, int b)
    {
        return a<b;
    }

    private static bool Crois(int a, int b)
    {
        return a>b;
    }
}

class Program
{
    static void Main(string[] args)
    {
        int a=1, b=2;
        Console.WriteLine("Initial: {0} - {1}", a, b);
        Sorting.Sort(ref a, ref b, Sorting.Decroissant);
        Console.WriteLine("Decroissant: {0} - {1}", a,b);
        Sorting.Sort(ref a, ref b, Sorting.Croissant);
        Console.WriteLine("Croissant: {0} - {1}", a, b);
        Console.ReadLine();
    }
}
```

On va maintenant retrouver ce qu'on vient de dire dans le code ci-dessus.

```
button.Click+= new EventHandler(button_pressed);
```

L'event est *button.Click* et on y associe un delegate *EventHandler* qui prend la fonction qui gère le fait que l'utilisateur à appuyé le button.

Si vous voulez définir vos propres Events la syntaxe est la suivante :

Definissez une class qui représente les paramètres que vous voulez passer avec l'évenement qui dérive de EventArgs.

```
public class MyEventArgs: EventArgs
{
    public int myInfo;
    public MyEventArgs(int t)
    {
        myInfo=t;
    }
}
```

Maintenant le delegate du handler :

```
public delegate void MyHandler(object source, MyEventArgs e);
```

Encore une fois, le handler va être fournie de l'extérieure (par exemple par l'utilisateur de votre classe) ! C'est l'handler qui gère lévènement.

Pour définir un event pour la classe ci-dessus on écrit :

```
public event MyEventArgs MyEvent;
```

La syntaxe est un peu bizarre, il faut lire : *l'evenement MyEvent qui a comme argument MyEventArgs*. Example :

```
using System;
using System.Text;

public class MyNumber
{
    public MyNumber(int i)
    {
        _n = i;
    }

    public int n
    {
        set
        {
            _n = value;
            ValueChangedEventArgs args = new ValueChangedEventArgs(_n);
            ValueChanged(this, args);
        }
        get
        {
            return _n;
        }
    }
    private int _n;

    public class ValueChangedEventArgs: EventArgs
```

```
    {
        public int myInfo;
        public ValueChangedEventArgs(int t)
        {
            myInfo=t;
        }
    }
    public delegate void ValueChangedHandler(object source, ValueChangedEventArgs e);
    public event ValueChangedHandler ValueChanged;
}

class Prog
{
    static void noOnePlease(object source, MyNumber.ValueChangedEventArgs e)
    {
        if (e.myInfo==1)
            ((MyNumber) source).n=2;
    }

    static void Main()
    {
        MyNumber n = new MyNumber(9);
        n.ValueChanged += new MyNumber.ValueChangedHandler(noOnePlease); ;

        n.n = 1;
        Console.WriteLine("{0}", n.n);

        n.n = 3;
        Console.WriteLine("{0}", n.n);

        Console.ReadLine();
    }
}
```

Testez le, on peut aussi enlever un eventhandler avec-=. Écrivez votre propre eventhandler qui change un 3 en 2. Essayez de remplacer l'eventhandler après la première attribution.

Qu'est-ce qui se passe si on utilise aucun eventhandler ? Il y a d'autre events : MouseDown (appuyé le button de la souris), MouseUp (rélaché), MouseMove (movement de la souris). Ils passent MouseEventArgs et peut être connecté à un MouseEventHandler delegate. Regardez ce qui est disponible dans MouseEventArgs. Réfléchissez comment on pourrait faire suivre un *Label*.