


# .NET

## Semaine 2

### Windows Forms

# Objectifs du cours

- Construction GUI avec Windows Forms
  - Juste un aperçu 
- Modèle de gestion des évènements
  - Delegate
  - Event
- En passant
  - Classe partielle
  - Threading

# Windows Forms

- Un peu équivalent à Swing
- Avec un maqueteur puissant
- « Théoriquement » supplanté par WPF

# Concepts GUI

- Composants GUI typiques  
Label, TextBox, Button, Scrollbar, ...
- Components implémente IComponent
- Control est un Component avec une partie graphique visible
- Certains Components, par ex. MessageQueues ou Timers n'ont pas de représentation graphique
- A Form est un container pour Components

# Concepts GUI (2)

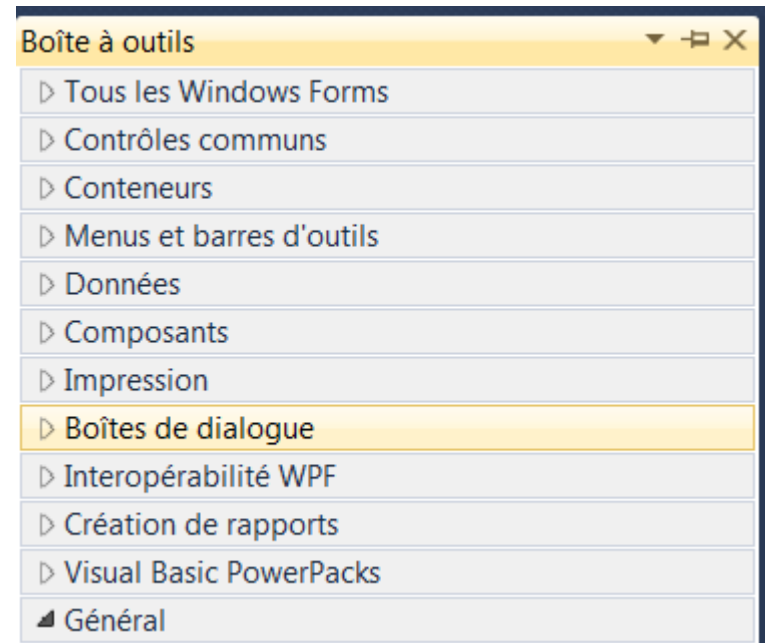
- Un Component peut générer des Events
- Les Event Handlers gèrent ces évènements
- La fenêtre de base est un objet de la classe `System.Windows.Forms.Form`
- Un bouton est un objet de la classe `System.Windows.Forms.Button`

# Concepts GUI (3)

- Le processus de design est le suivant :
  - (1) Créer une Windows Form
  - (2) Fixer ses propriétés
  - (3) Ajouter des contrôles
  - (4) Fixer leurs propriétés
  - (5) Implémenter les Event Handlers

# Types de contrôles

- Comme dans Swing
- Plus des contrôles  
« utilitaires »  
ex: Timer



# Modèle d'évènements

- Les délégués (delegate) sont à l'écoute des évènements et appellent les gestionnaires
- Chaque composant a un délégué pour chaque évènement qu'il peut générer
- On enregistre une méthode auprès du délégué et celui-ci appellera la méthode de façon asynchrone



# Délégués

- Un bouton doit par exemple permettre modifier des objets quand il est cliqué
- On ne veut pas hardcoder (dans le bouton) quel objet appeler
- Un délégué est un type de référence utilisée pour encapsuler une méthode avec certains paramètres

# Délégués

```
using System;

delegate String Foo(String x);           // create a delegate class

class Test {
    public static void Main() {
        Foo f = new Foo(ConvertToUpperCase); // create a delegate object
        String answer = f("abcd");           // call the method in the object

        Console.WriteLine(answer);
    }
    public static String ConvertToUpperCase(String s) {
        return s.ToUpper();
    }
}
```

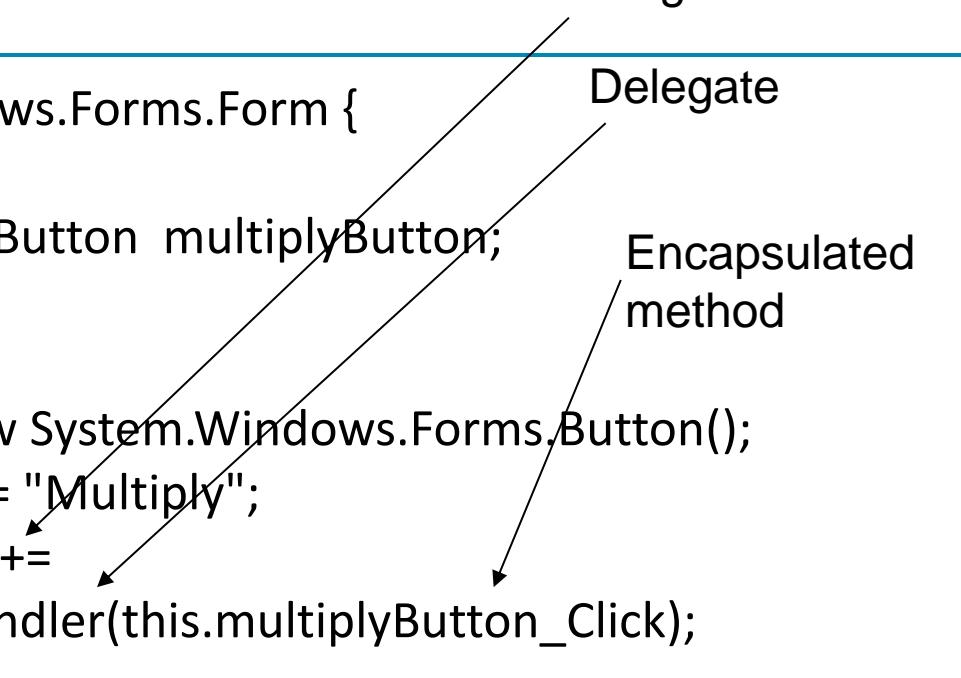
# Délégués

```
public class Form1 : System.Windows.Forms.Form {  
  
    private System.Windows.Forms.Button multiplyButton;  
  
    public void foo() {  
        this.multiplyButton = new System.Windows.Forms.Button();  
        this.multiplyButton.Text = "Multiply";  
        this.multiplyButton.Click +=  
            new System.EventHandler(this.multiplyButton_Click);  
    }  
  
    private void multiplyButton_Click(object sender, System.EventArgs e) {  
        textBox3.Clear();  
        string op1Str = op1.Text;  
        string op2Str = op2.Text;  
        ...  
    }  
}
```

Delegate reference

Delegate

Encapsulated method



# Délégués multicast (1/3)

```
using System;                                // From C# In A Nutshell
delegate void MethodInvoker(); // define delegate class

class Test {

    static void Main() {                      // create a Test object
                                              // and call its constructor
        new Test();
    }
}
```

# Délégués multicast (2/3)

```
Test() {
```

```
    MethodInvocation m = null;  
    m += new MethodInvocation(Foo); // overloaded +=  
    m += new MethodInvocation(Goo); // delegate holds  
    m();                          // pointers to two  
}                                // methods
```

# Délégués multicast (3/3)

```
void Foo() {  
    Console.WriteLine("Foo");  
}  
void Goo() {  
    Console.WriteLine("Goo");  
}  
}
```

Output :

Foo  
Goo