Génie Logiciel
La technologie WinForms

BAPTISTE PESQUET

Support de cours

https://bpesquet.developpez.com/tutoriels/csharp/programmation-evenementielle-winforms/

Sommaire

- Le paradigme évènementiel
- La technologie WinForms
- Opérations WinForms courantes
- WinForms et multithreading

Le paradigme évènementiel

Un nouveau paradigme

Paradigme :

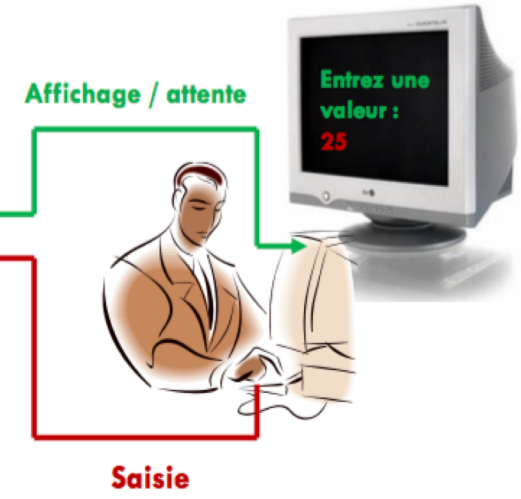
- Ensemble partagé de croyances et de valeurs
- En informatique, style fondamental de programmation

Paradigme **séquentiel**

- Les instructions s'exécutent toujours dans le même ordre
- L'utilisateur fait ce que lui demande le programme
- Exemple : application console C#

Exemple de programme séquentiel

```
// PROGRAMME
Main()
{
    string saisie;
    Console.WriteLine("Entrez une valeur");
    saisie = Console.ReadLine();
    int valeur = Convert.ToInt32(saisie);
    int carre = valeur * valeur;
    ...
}
```



Le paradigme évènementiel

Le programme réagit à des **évènements** externes

- Evènements provenant du réseau ou du système
- Actions de l'utilisateur

L'ordre d'exécution des instructions n'est plus prévu à l'avance

Paradigme utile pour gérer des interactions riches avec l'utilisateur (interfaces graphiques ou **GUI**, *Graphical User Interface*)

Exemple de programme évènementiel

```
// PROGRAMME
Main()
{
   ...
   while(true)  // tantque Mamie s'active
   {
      // récupérer son action (faire une maille ...)
      e = getNextEvent();
      // traiter son action (agrandir le tricot ...)
      processEvent();
   }
   ...
}
```

La technologie WinForms

Introduction

WinForms : plateforme de création d'applications graphiques sous Windows

Adossée au framework .NET

Basée sur le paradigme évènementiel

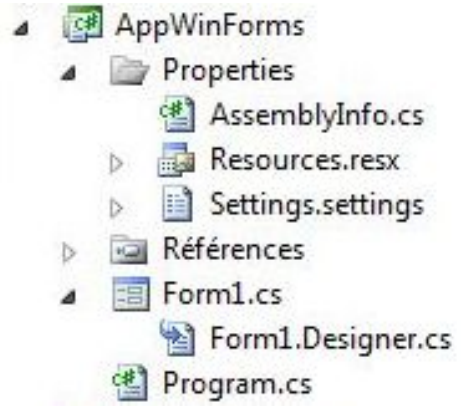
Structurée autour d'un ou plusieurs **formulaires** (*forms*)

Un formulaire = deux fichiers

- MonFormulaire.**Designer.cs** : fichier contenant le code généré automatiquement par l'IDE
- MonFormulaire.**cs** : fichier « code behind » édité par le développeur

Un formulaire peut (doit) être immédiatement renommé après création

Exemple d'application WinForms



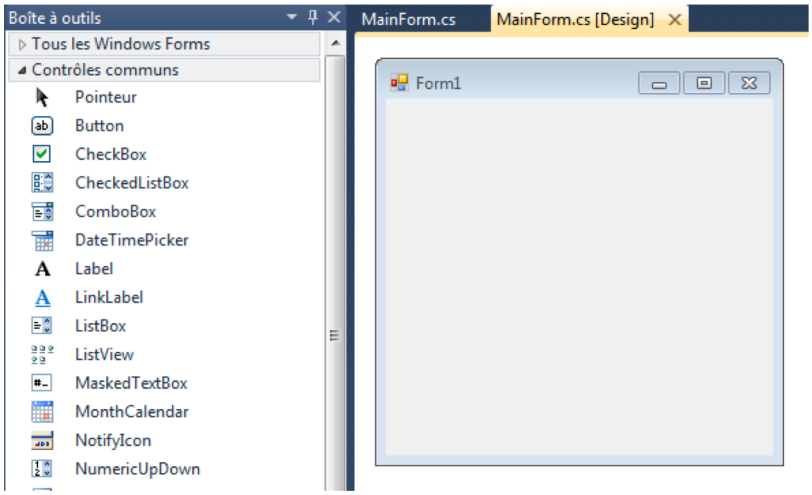

```
// Program.cs
static void Main() {
  Application.EnableVisualStyles();
Application.SetCompatibleTextRendering
Default(false);
  Application.Run(new Form1());
}
```


Ajout de contrôles à un formulaire

Contrôle : élément d'interaction avec l'utilisateur

Nombreux contrôles WinForms prédéfinis

- Zone de saisie
- Bouton
- Case à cocher
- …



Propriétés d'un contrôle

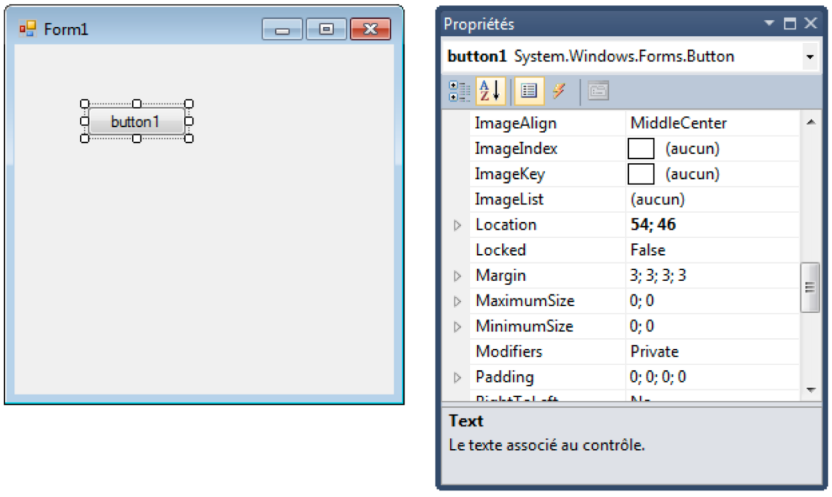
Chaque contrôle possède des **propriétés** qui gouvernent son apparence ou son comportement.

Propriétés communes et essentielles

- **(Name)** : le nom de l'attribut représentant le contrôle dans la classe
- **Dock** : l'ancrage du contrôle dans celui qui le contient
- **Enabled** : indique si le contrôle est actif ou non
- **Text** : le texte affiché par le contrôle
- **Visible** : indique si le contrôle est visible ou non

Un contrôle peut (doit) être immédiatement renommé après création

Edition graphique des propriétés



Gestion des évènements

Chaque contrôle peut réagir à une liste **d'évènements**

Exemple d'événement pour un bouton : le clic

Le formulaire peut aussi réagir à des évènements (ouverture, fermeture, etc)

Gestionnaire d'évènement : code exécuté lorsque l'évènement associé se produit

Tous les gestionnaires pour un formulaire et ses contrôles sont regroupés dans son fichier « code behind »

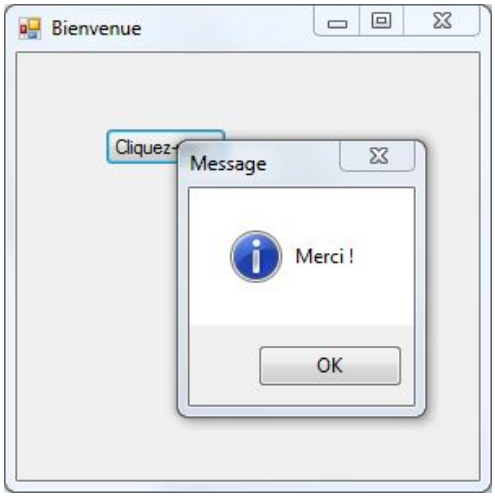
Exemple : fichier .cs « code behind »

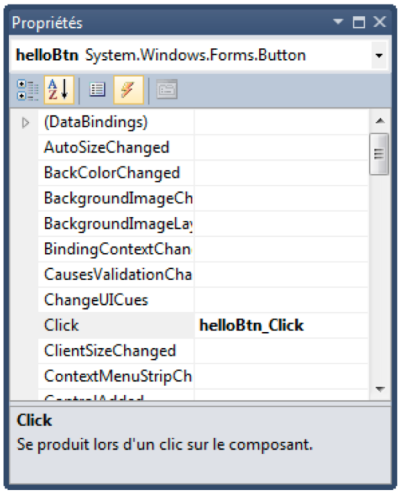
```
public partial class MainForm : Form {
  // …
  // Gère le clic sur le bouton helloBtn
  // sender : contrôle origine de l'événement
  // e : informations sur l'évènement
  private void helloBtn_Click(object sender, EventArgs e) {
    MessageBox.Show("Merci !", "Message", MessageBoxButtons.OK,
MessageBoxIcon.Information);
  }
}
```

Exemple : fichier .Designer.cs

```
partial class MainForm {
  private System.Windows.Forms.Button helloBtn;
  // …
  private void InitializeComponent() {
    this.helloBtn = new System.Windows.Forms.Button();
    // ...
    // Ajout d'un gestionnaire pour l'événement « click »
    this.helloBtn.Click += new System.EventHandler(this.helloBtn_Click);
    // ...
  }
}
```


Résultat obtenu





Opérations WinForms courantes

Redimensionnement et positionnement

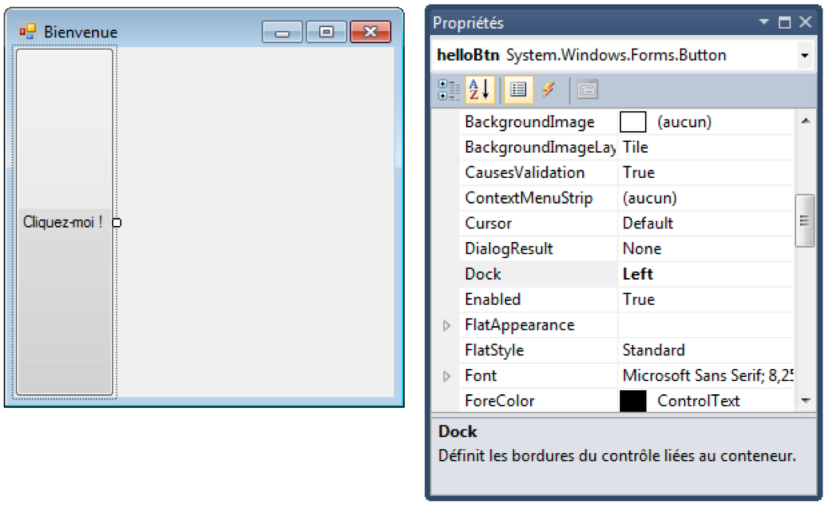
Interdire le redimensionnement d'un formulaire

- **FormBorderStyle** : *Fixed3D* (ou une autre valeur *FixedXX*)
- **MaximizeBox** et **MinimizeBox** : *false*

Positionner les contrôles par rapport au formulaire

- **Anchor** : ancrage du contrôle par rapport aux bordures de son conteneur (distance constante). Par défaut : *Top* | *Left*
- **Dock** : bordure(s) du contrôle directement attachée(s) au conteneur parent. Le contrôle prendra toute la place disponible sur la ou les bordure(s) en question. Par défaut : *None*

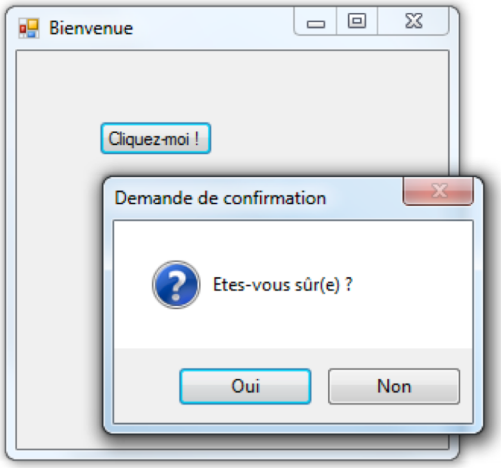
Exemple de docking à gauche



Fermeture de l'application

Déclencher l'arrêt de l'application : `Application.Exit();`

Afficher une demande de confirmation : ajouter un gestionnaire pour l'évènement `FormClosing` du formulaire



Affichage modal ou non modal

```
SubForm subForm = new SubForm();


// Affiche le formulaire subForm de manière non modale
subForm.Show();
```

```
SubForm subForm = new SubForm();


// Affiche le formulaire subForm de manière modale
// (il doit être fermé pour pouvoir accéder de nouveau au reste de l'application)
if (subForm.ShowDialog() == DialogResult.OK) {
   // L'utilisateur a cliqué sur OK
   // …
}
```


Echange de données entre formulaires

```
public partial class SubForm : Form {
  public SubForm(string message) {
    InitializeComponent();
    inputLbl.Text = message;
  }
  public string Input {
    get { return inputBox.Text; } }
  // ...
```

```
SubForm subForm = new SubForm(
   "Entrez votre login");
if (subForm.ShowDialog() == DialogResult.OK) {
   string login = subForm.Input;
   // …
}
```

Gestion des erreurs

```
static void Main() {
  Application.EnableVisualStyles();
  Application.SetCompatibleTextRenderingDefault(false);
  try {
    Application.Run(new MainForm());
  }
  catch (Exception ex) {
    MessageBox.Show(ex.Message, "Erreur", MessageBoxButtons.OK,
MessageBoxIcon.Error);
  }
}
```


WinForms et multithreading

La notion de thread

Thread : contexte dans lequel s'exécute du code

Un programme en cours d'exécution = au minimum un thread

- Un seul thread => application **monothread**
- Plusieurs threads => application **multithread**

Une application peut décider de créer de nouveaux threads

Avantage : possibilité de réaliser de plusieurs tâches en parallèle

Inconvénient : complexité du partage de ressources entre les threads

Les limites du monothread

Une application WinForms reçoit et traite en permanence des **messages** provenant du système d'exploitation ; appui sur une touche du clavier, déplacement de la souris, ordre de rafraîchir l'affichage d'une fenêtre…

Elle s'exécute dans un thread unique (**UI thread**) qui gère le traitement de ces messages ET le code des gestionnaires d'évènement

Si un gestionnaire déclenche un traitement long, la gestion des messages système ralentira => application bloquée

Types de threads utilisables

Classe .NET **Thread** : ne permet pas les interactions avec les formulaires

Classe **BackgroundWorker** : réalisation d'un traitement dans un thread séparé + interactions possibles avec les formulaires

- Méthode **RunWorkerAsync()** : démarre un nouveau thread
- Evènement **DoWork** : définir le traitement à exécuter dans le thread
- Evènement **ProgressChanged** : notifier le formulaire de l'avancement du traitement
- Evènement **RunWorkerCompleted** : signaler au formulaire la fin du traitement

29 sur 31

Utilisation d'un BackgroundWorker

```
private void startMultiBtn_Click(object sender, EventArgs e) {
  infoLbl.Text = "Opération en cours...";
  worker.RunWorkerAsync(); // Démarre un thread
}
private void worker_DoWork(object sender, DoWorkEventArgs e) {
  Thread.Sleep(5000); // Arrête le thread pendant 5 secondes
}
private void worker_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e) {
  infoLbl.Text = "Opération terminée";
}
```

30 sur 31

Exécution de code à intervalles réguliers

```
private void countdownBtn_Click(object sender, EventArgs e) {
  Timer timer = new Timer();  // Création d'un contrôle WinForms Timer
  timer.Tick += new EventHandler(timer_Tick); // timer_Tick est appelé à chaque déclenchement
  timer.Interval = 1000; // Le déclenchement a lieu toutes les secondes
  timer.Enabled = true; // Démarre la minuterie
}
// Code exécuté à chaque déclenchement du timer
void timer_Tick(object sender, EventArgs e) {
  // Pas de traitement long ici, sinon blocage de l'UI !
}
```