CM3

- La programmation Windows
  - Introduction, Structure d'un programme, Architecture
- Création d'une application Windows
- Le squelette de programme généré
- Les fenêtres
- 1ère application
- Composants et hiérarchie de classes
  - La classe « Control »
  - L'organisation des composants
  - Les menus

La programmation Windows Introduction

- L' "engouement" pour la programmation Windows s'explique par plusieurs raisons :
  - L'importance de développer des interfaces utilisateur graphiques conviviales.
  - Windows fournit un "quasi-standard" d'interface graphique et permet donc à un utilisateur de "prendre en main" très rapidement une application et son environnement.
- Notons l'importance mais aussi la complexité du développement d'interfaces graphiques.
  - Windows est une interface complexe et touffue qui propose selon le mode de comptage entre 750 et 1250 appels de fonctions possibles (API natif de Windows (API = Application Programming Interface));

La programmation Windows Introduction

- *La programmation pilotée par message*
  - Programmation évènementielle
- Un évènement est une action réalisée par l'utilisateur sur l'interface
  - Windows doit être capable de récupérer les informations liées à l'évènement
  - Transmettre cette information à l'application concernée pour déclencher le traitement de cet évènement

La programmation Windows Introduction

- *Les sorties graphiques*
- Tous les objets créés par Windows sont graphiques
  - Les droites, les cercles, les boîtes sont traités comme des objets graphiques et sont créés à l'aide de l'interface graphique (GDI : Graphic Device Interface).
  - → indépendance des traitements vis-à-vis des périphériques
- GDI reconnaît quatre types de systèmes d'affichage : l'écran, les périphériques de recopie d'écran (imprimantes, traceurs), les bitmaps et les métafichiers. *Ces 2 derniers sont vus comme des pseudo-périphériques (stockage d'images en mémoire ou sur disque, partage d'images).*

→ l'homogénéité des sorties graphiques est un point fort de Windows

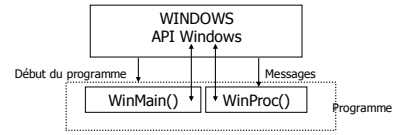
La programmation Windows Introduction

- *Les objets graphiques*
- Windows supporte un certain nombre d'objets consacrés au support de l'interface utilisateur :
  - *fenêtres, icônes, boîtes de dialogue, ...*
- Possibilité pour une application d'utiliser ces objets et de les personnaliser

→ diminution du coût de développement

Principe de programmation Windows Structure d'un programme Windows

- Sous sa forme la plus réduite un programme Windows, écrit uniquement avec l'API Windows comporte, 2 fonctions :
  - La fonction WinMain() qui est l'équivalent de la fonction main() d'un programme console.
  - La fonction WinProc() que Windows appelle pour traiter les messages destinés à l'application
- Ces deux fonctions composent un programme complet mais elles ne sont pas directement liées. C'est Windows qui gère les appels à ces deux fonctions.



Principe de programmation Windows
Structure d'un programme Windows

- La fonction WinMain()
  - La boucle de messages

    while (Getmessage(&msg, NULL, 0,0))
    { TanslateMessage(&msg);
    DispatchMessage(&msg);
    }
  - Cette boucle fonctionne tant que les messages reçus ne sont pas un message de terminaison de l'application.
  - GetMessage extrait un message de la file d'attente,
  - TranslateMessage convertit le message extrait, si nécessaire
  - DispatchMessage demande à Windows d'appeler la fonction WinProc() dans l'application pour traiter le message

Principe de programmation Windows
Structure d'un programme Windows

- **Taxonomie des messages**
- Il existe environ 250 types de messages prédéfinis dans Windows. On peut distinguer 8 catégories principales de messages dont
- les messages hardware : entrées d'informations avec le clavier ou la souris
- *messages de la souris*
  - WM_LBUTTONDOWN, WM_LBUTTONUP, …
- *messages du clavier*
  - WM_CHAR, WM_KEYDOWN
- messages de maintenance des fenêtres :
  - notification, demande d'action (WM_CLOSE, WM_PAINT, …)
- les messages de maintenance de l'interface utilisateur concernant les menus
  - WM_COMMAND,WM_INITMENU, …
- les messages de terminaison (WM_QUIT, …)
- les messages privés (pour fenêtres spécifiques : boîtes à liste, bouton, …)
- les messages de notification concernant les ressources système (WM_TIMECHANGE,…)

Principe de programmation Windows
Structure d'un programme Windows

- Fonctions de traitement des messages
  - Le décodage des messages est effectué à l'aide d'un instruction conditionnelle multiple dans la fonction WinProc()
  - *Exemple*
    - switch (message)
    - { case WM_LBUTTONUP : …….; break;
    - case WM_PAINT : ……..; break;
    - ……..
    - default : …….
    - }

Principe de programmation Windows
Structure d'un programme Windows

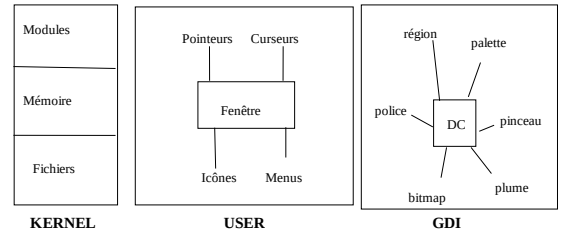
- Bilan
  - Principe de fonctionnement d'un programme Windows uniquement avec l'API natif
    - Tâche ardue et complexe
    - API Windows comporte plus d'un millier de fonctions qui permettent d'assurer les services et la communication nécessaires à la création d'une application Windows.
  - l'API Windows et Visual C++, C#
    - L'API n'est pas spécialement conçue pour le C++ ou C# (antérieure à C++ et l'approche objet)
  - L'environnement .NET propose une ensemble de classes qui "englobent" l'API native.
    - Windows Forms = bibliothèque de classes pour la gestion des éléments graphiques d'une interface

Principe de programmation Windows
Architecture de Windows

- Windows repose sur trois bibliothèques :
  - **KERNEL, USER, GDI**
- **KERNEL gère le support à bas niveau**
  - des fichiers d'entrées/sorties,
  - de la mémoire,
  - du chargement des programmes,
  - des opérations classiques d'un OS (Operating System)
- **USER et GDI font appel aux services de KERNEL**
- **USER s'occupe de la création et de la gestion des objets d'interface utilisateur :**
  fenêtres, menus, boîtes de dialogues, curseurs, icônes
- → l'objet primordial est la fenêtre

Principe de programmation Windows
Architecture de Windows

- **GDI : Graphical Device Interface**
  - assure l'indépendance vis-à-vis des périphériques
  - gère les dessins de fenêtres, les menus, les boîtes de dialogue, etc …
  - USER utilise les services de GDI pour les tracés de ses objets



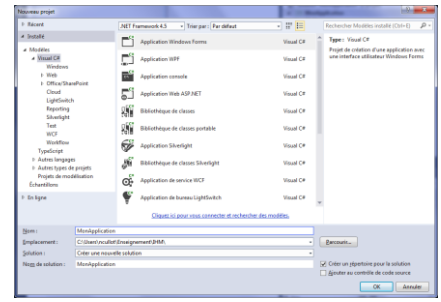
Création d'une application Windows

- Avec Visual Studio, en C#
- Une application minimale consiste en :
  - un objet "Application" (non visible à l'écran) qui
    - Contient les fonctions de base nécessaires au démarrage et au contrôle de l'application jusqu'à sa phase finale.
  - au moins un objet "Fenêtre"

13

Création d'une application Windows

- Les étapes de création



14

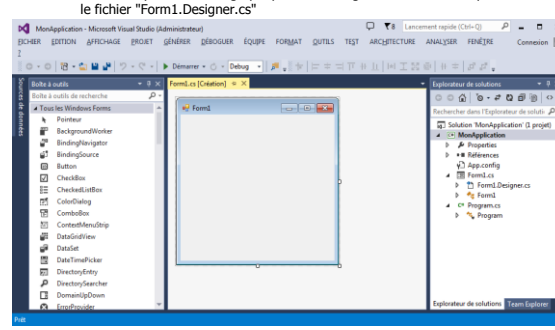
Création d'une application Windows

- Plusieurs fichiers sont automatiquement créés :
  - *Program.cs* qui contient la fonction principale de l'application.
  - *Form1.Designer.cs* qui contient la description des éléments graphiques de l'interface.
  - *Form1.cs* qui contiendra les descriptions des traitements à réaliser.
- Les fichiers Program.cs et Form1.Designer.cs sont directement gérés par l'EDI de Visual.

15

Création d'une application Windows

- La classe "Form1" correspond à la fenêtre principale de l'application qui peut être construite par l'interface graphique avec une génération automatique du code dans le fichier "Form1.Designer.cs"



16

Le squelette de programme généré

Program.cs

Description des librairies usuelles

Programme Principal Main

Boucle d'exécution du programme

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace MonApplication
{
   static class Program
   {
      /// <summary>
      /// Point d'entrée principal de l'application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new Form1());
      }
   }
}
```

17

Le squelette de programme généré

Form1.cs

Librairies

Description de la classe "Form1" qui dérive de la classe "Form"

Classe partielle

Constructeur

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace MonApplication
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
   }
}
```

18

Le squelette de programme généré

Form1.designer.cs

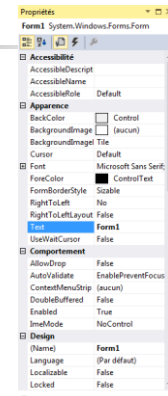
```
namespace MonApplication
{   partial class Form1
   { private System.ComponentModel.IContainer components = null;
    protected override void Dispose(bool disposing)
     {  if (disposing && (components != null)) components.Dispose();
          base.Dispose(disposing);
     }

     #region Code généré par le Concepteur Windows Form
     private void InitializeComponent()
     {
        this.components = new System.ComponentModel.Container();
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.Text = "Form1";
     }
     #endregion
   }
}
```

19

Les fenêtres

- Les propriétés des fenêtres ou formulaires (classe Form)
  - Object -> Control -> ScrollableControl -> ContainerControl -> Form
  - Nom de la fenêtre (classe)
    - Name : Form1
  - Titre de la fenêtre
    - Text : Form1
  - Police d'écriture
    - Font : Microsoft ….
  - Couleurs
    - BackColor, ForeColor
  - Etc.



20

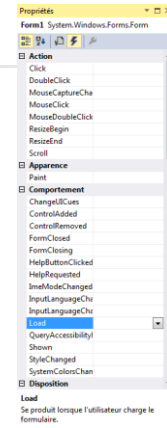
Les fenêtres

- Les propriétés de la fenêtre peuvent être fixées :
  - De manière interactive, lors de sa conception sous Visual Studio
  - Par le programme, notamment pour changer des propriétés pendant l'exécution.
- Certaines propriétés dites run-time ne peuvent être manipulées que pendant l'exécution
  - ActiveControl indique le contrôle de la fenêtre qui a le focus,
  - DisplayRectangle, coordonnées de l'aire client de la fenêtre (Rectangle)
  - etc.

21

Les fenêtres

- Les événements
- Plusieurs événements sont liés aux fenêtres
  - Activated : la fenêtre devient active
  - Desactivate : la fenêtre a perdu son état de fenêtre active
  - Closed : la fenêtre a été fermée
  - Click : l'utilisateur a cliqué sur le bouton gauche de la souris
  - DoubleClick, KeyDown, KeyPress, KeyUp,
  - MouseDown, MouseEnter, MouseMove, MouseUp,
  - Paint : la fenêtre doit être reaffichée,
  - Resize : la fenêtre est en train d'être redimensionnée
  - etc.



22

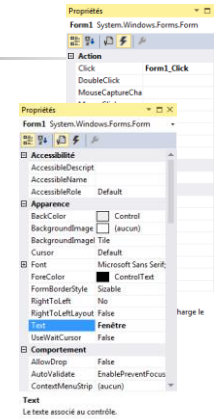
Les méthodes liées aux fenêtres

- La classe "Form" comporte un certain nombre de méthodes permettant de gérer une fenêtre (ou composant)
  - void Close() – fermeture de la fenêtre
  - void Invalidate() – force le réaffichage de la fenêtre par signalement de l'événement Paint à la fenêtre
    - Le rafraîchissement peut être différé
  - void Refresh() – force le rafraîchissement immédiat
  - void Show() – fait apparaître le composant
  - void Hide() – cache le composant
  - etc.

23

1ère application

- Création d'une 1ère application qui affiche une fenêtre, et qui affiche dans le titre de la fenêtre la position de la souris quand on clique dans le fenêtre.
- Etapes de création
  - Création de l'application Windows Forms
  - Modification de la propriété « Text » de la fenêtre
  - Ajout du gestionnaire de clic dans la fenêtre



24

1ère application

- Code généré automatiquement
- Dans le fichier "Form1.Designer.cs", la méthode "InitializeComponent" est complétée pour indiquer que la fenêtre répond à l'événement "Click"

```
private void InitializeComponent()
    {
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    ...
            this.Name = "Form1";
            this.Text = "Fenêtre";
            ...
            this.Click += new System.EventHandler(this.Form1_Click);
            ...
    }
```


1ère application

- Code généré automatiquement
- Dans le fichier "Form1.cs", le gestionnaire d'événement est généré, il doit être complété;

```
private void Form1_Click(object sender, EventArgs e)
    {
    ...
    }
```

- Le gestionnaire a deux arguments :
  - **sender** qui est une référence au composant qui est à l'origine de l'action
  - **EventArgs** qui est la classe de base des classes qui contiennent les informations liées à un événement.

1ère application

- Dans l'application, l'objet sender est la fenêtre, qui dérive de la classe Form, il peut être utilisé pour modifier la fenêtre
  - ((Form) sender).Text = "Titre";
- L'événement qui est géré est un événement de type souris,
  - MouseEventArgs est la sous-classe de EventArgs qui contient des informations pour un événement de type souris
    - Par exemple, les coordonnées du point de clic

1ère application

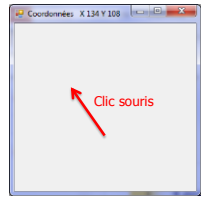
- On compète le gestionnaire.

```
private void Form1_Click(object sender, EventArgs e)
 {  string s="";
    s = "Coordonnées ";
    MouseEventArgs me =(MouseEventArgs) e;
     s  += "  X " + me.X + " Y " + me.Y;
    ((Form) sender).Text=s;
 }
```

Les arguments "sender" et "e" sont transtypés.

1ère application

- Un clic souris dans la fenêtre fait afficher
  - Sa position X, Y
  - Dans le titre de la fenêtre

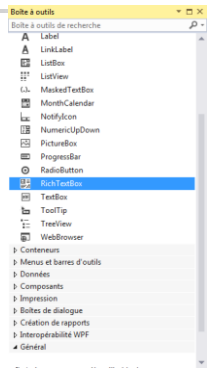


Composants et hiérarchie de classes

- Les composants appelés également contrôles sont
  - Des fenêtres particulières
- Les composants sont les boutons de commandes, les cases à cocher, les boîtes de listes, etc.
- Plusieurs opérations sont communes à l'ensemble des composants
  - (hiérarchie de classes)

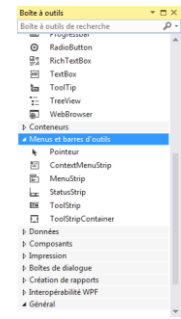
Composants et hiérarchie de classes

- Les boutons et les cases à cocher
  - Button, RadioButton, CheckedBox
- Les listes
  - CheckedListBox, ComboBox, ListBox, ListView
- Les messages
  - Label, LinkLabel
- Les zones d'édition
  - TextBox, MaskedTextBox, RichTextBox
- Etc.



Composants et hiérarchie de classes

- Les menus
  - MenuStrip
  - ContextStrip
- Les barres à outis
  - ToolStrip
- Les barres d'états
  - StatusStrip, StatusLabel
- etc.



Composants et hiérarchie de classes

- Toutes les classes de la hiérarchie de classe de .net dérivent de la classe Object.
- La classe de base des composants au sens large (visuels ou non) est la classe "Component"
- La classe "Control" dérivée de "Component" est la 1ère classe de base pour tous les composants (visuels).

Composants et hiérarchie de classes

- La classe "Control" comprend des propriétés et des méthodes liées
  - aux caractéristiques (couleur, …)
  - à la position (emplacement, taille, …)
  - à la manipulation

  des composants.
- Propriétés de la classe "Control"
  - Couleurs, image d'arrière plan
    - ForeColor, BackColor (Color), BackgroundImage (Image)

Composants et hiérarchie de classes

- Propriétés de la classe "Control"
  - Position et taille
    - Bottom, Right, Left, Top (coordonnées bord intérieur)
    - Bounds (Rectangle) (coord. par rapport fenêtre mère)
    - ClientRectangle (Rectangle) (aire client du composant)
    - ClientSize (Size) (taille de l'aire client)
    - Etc.
  - Autres informations
    - Anchor (ancrage du composant : Bottom, Right, Left, Top, None)
    - Cursor (curseur de la souris quand elle survole le composant)

Composants et hiérarchie de classes

- Propriétés de la classe "Control"
  - ContextMenu (ContextMenu), menu contextuel associé au composant (sur clic droit)
  - Capture (boolean) vrai si le composant a capturé la souris (i.e. il informé des opérations sur la souris même en dehors du composant)
  - Enabled (boolean), le composant est actif
  - Visible (boolean), le composant est visible
  - etc.
- Collections des contrôles dépendant du composant
  - Controls (collection)
- Etc.

Composants et hiérarchie de classes

- Evénements traités par la classe "Control"
  - Clic sur le composant (Click, DoubleClick)
  - Ajout ou suppression d'un composant (ControlAdded, ControlRemoved)
  - Frappe Clavier (KeyDown, KeyUp)
  - Réception et perte du focus d'entrée (Enter, Leave)
  - Mouvements souris (MouseDown, MouseEnter, MousHover, MouseLeave, MouseUp, MouseWheel)
  - Déplacement du composant (Move)
  - Redimensionnement du composant (Resize)

Composants et hiérarchie de classes

- La classe "Control" a plusieurs classes dérivées dont :
  - **ScrollableControl** comme classe de base pour les composants avec défilement possible
  - **ContainerControl** comme classe de base pour les composants créés à partir de plusieurs composants

Composants et hiérarchie de classes

- Les méthodes de la classe "Control"
- Obtention d'un objet "Graphics" pour le composant
  - Graphics CreateGraphics();
- Affichage et masquage du composant
  - void Show(); et void Hide();
- Rafraîchissement
  - void Invalidate(), void Invalidate(Recatngle)
- Méthode de transformation de coordonnées
  - Rectangle RectangleToClient (Rectangle r) (coord écran vers coord. client)
  - Rectangle RectangleToscreen (Rectangle r) (coord. client vers coord. écran)

Composants et hiérarchie de classes

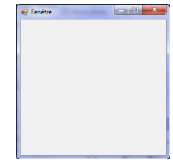
- Les boutons et cases à cocher dérivent de la classe "Control"
  - Object -> Component -> Control -> ButtonBase
    - Button , CheckBox, RadioButton
- Les boîtes de liste dérivent également de la classe Control
  - Object -> Component -> Control -> ListControl
    - ListBox (liste standard)
      - CheckedListBox (liste avec cases à cocher)
    - ComboBox (liste déroulante)
  - Object -> Component -> Control -> TreeView (arbre)
  - Object -> Component -> Control -> ListView (fenêtre de liste)

Composants et hiérarchie de classes

- Les zones d'affichage et d'édition dérivent également de la classe "Control
  - Component -> Control
    - -> Label (message)
    - -> TextBoxBase ->TextBox (édition simple)
    - -> TextBoxBase -> MaskedTextBox (édition masquée)
- D'autres classes dérivent également de la classe "Control".

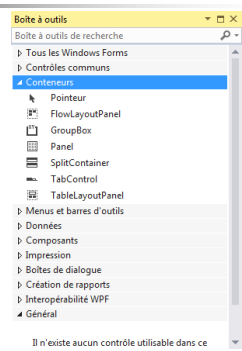
Organisation des composants

- Une fenêtre ou une boîte de dialogue peut être organisée en utilisant des conteneurs
  - Qui peuvent à nouveau contenir de conteneurs
  - Ou bien des composants/ contrôles de l'interface (boutons, liste, zones d'édition, etc.)
- Lors de la création d'une application, la fenêtre principale est vide :



Organisation des composants

- Elle peut être « organisée » en utilisant les conteneurs proposés dans la bibliothèque

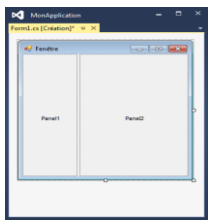


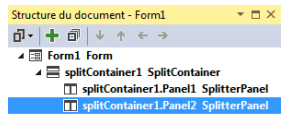
Organisation des composants

- Plusieurs types de conteneurs sont disponibles
  - Panel : Panneau simple
  - SplitContainer qui permet de diviser une zone en 2 panneaux redimensionnables
  - TableLayoutPanel qui permet une organisation sous forme de tableau
  - TabControl qui définit un panneau à onglets

Organisation des composants

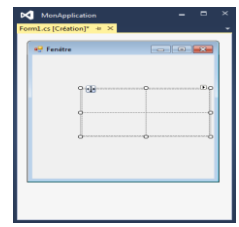
- L'ajout d'un composant SplitContainer partage la fenêtre
  - en 2 panneaux de type SplitterPanel

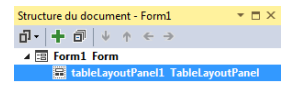




Organisation des composants

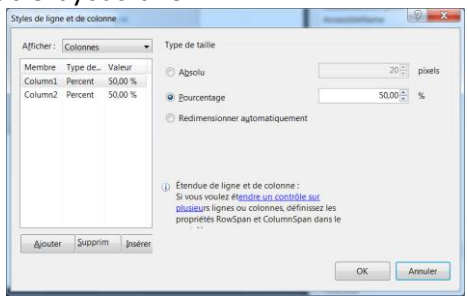
- Un composant TableLayoutPanel propose une organisation en tableau,
  - Le tableau doit être positionné dans la fenêtre
  - Et paramétrer (nb de lignes, colonnes, largeur, hauteur, etc.)





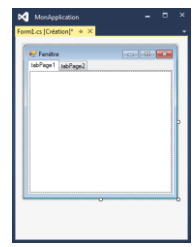
Organisation des composants

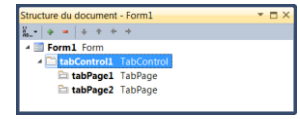
- Fenêtre de paramétrage d'un conteneur TableLayoutPanel



Organisation des composants

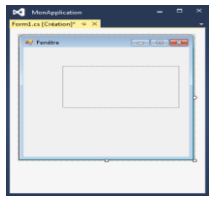
- Il est possible de créer des panneaux sous forme d'onglets, à l'aide d'un conteneur TabControl,
  - 2 pages sont créées par défaut, de type TabPage

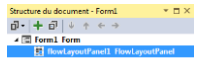




Organisation des composants

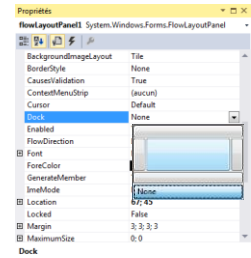
- Un conteneur FlowLayoutPanel fournit un panneau organisé en FlowLayout
  - ses composants seront positionnés automatiquement les uns à côté des autres





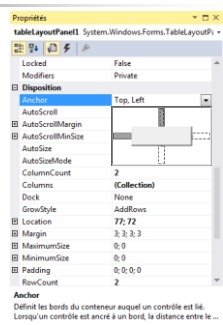
Organisation des composants

- Les conteneurs doivent être positionnés dans la fenêtre ou dans le conteneur dans lequel ils ont été définis.
- Un positionnement en haut, en bas, à droite, à gauche ou au centre (avec remplissage maximum) est possible en fixant la propriété Dock
  - *(semblable à un BorderLayout de java)*
  - *Top, Bottom, Left, Rigth, Fill, None*



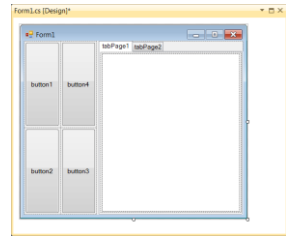
Organisation des composants

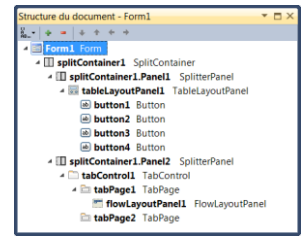
- Les composants peuvent également être positionnés en fixant leur mode d'ancrage avec la propriété Anchor
  - Exemple : ancrage Top, Left



Organisation des composants

- Exemple de composition d'une fenêtre avec la structure du document associé





Organisation des composants

- Au fur et à mesure de la construction interactive de l'interface,
  - le code source de la méthode
    - InitializeComponent est complété
- Par exemple, pour le conteneur SplitContainer

```
this.splitContainer1 = new System.Windows.Forms.SplitContainer();
this.splitContainer1.Dock = System.Windows.Forms.DockStyle.Fill;
this.splitContainer1.Name = "splitContainer1";
// splitContainer1.Panel1
this.splitContainer1.Panel1.Controls.Add(this.tableLayoutPanel1);
// splitContainer1.Panel2
this.splitContainer1.Panel2.Controls.Add(this.tabControl1);
```


Organisation des composants

- Les méthodes SuspendLayout, ResumeLayout sont utilisées lors de la spécification d'un composant
  - Au début de la méthode InitializeComponent

```
this.splitContainer1.Panel1.SuspendLayout();
this.splitContainer1.Panel2.SuspendLayout();
this.splitContainer1.SuspendLayout();
```

  - A la fin de la méthode

```
this.splitContainer1.ResumeLayout(false);
Etc.
```

- La méthode PerformLayout force le contrôle à appliquer le Layout

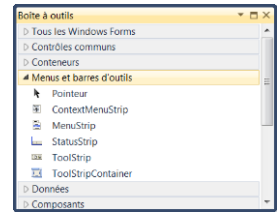
Organisation des composants

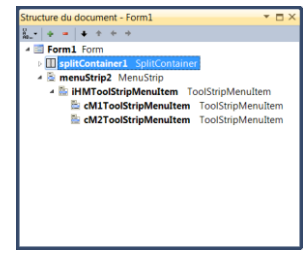
- Les composants contenus dans un conteneur sont ajoutés dans la collection de « Controls » de ce conteneur
- Exemple : ajout des panneaux à onglets dans le deuxième panneau du SplitContainer
  - this.splitContainer1.Panel2.Controls.Add(this.tabControl1);

55

Les contrôles des menus

- Une barre de menu est de type MenuStrip, les options et les sous-options sont de type ToolStripMenuItem





56

Les composants - Conclusion

- La bibliothèque Windows Forms fournit
  - un ensemble riche de composants
  - pour construire et gérer des interfaces graphiques
- Visual Studio 2013 fournit
  - un environnement dynamique de création des interfaces
  - avec de la génération de code source automatique

57