

www.Mcours.com

Site N°1 des Cours et Exercices Email: contact@mcours.com

*Introduction à la
programmation en
Visual Basic® .NET avec
Microsoft® .NET*

Plan du cours

- **Module 1 : Mise en route**
- **Module 2 : Utilisation de formulaires et de contrôles**
- **Module 3 : Utilisation de variables et de tableaux**
- **Module 4 : Utilisation de procédures**
- **Module 5 : Structures de décision et boucles**
- **Module 6 : Validation des entrées de l'utilisateur**

Plan du cours (suite)

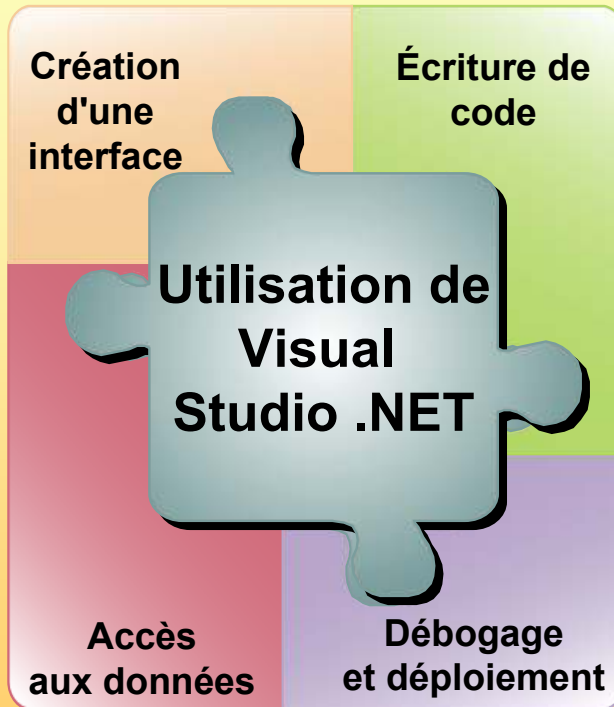
- **Module 7 : Programmation orientée objet en Visual Basic .NET**
- **Module 8 : Gestion des erreurs et des exceptions**
- **Module 9 : Amélioration de l'interface utilisateur**
- **Module 10 : Déploiement d'applications**

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Module 1 :

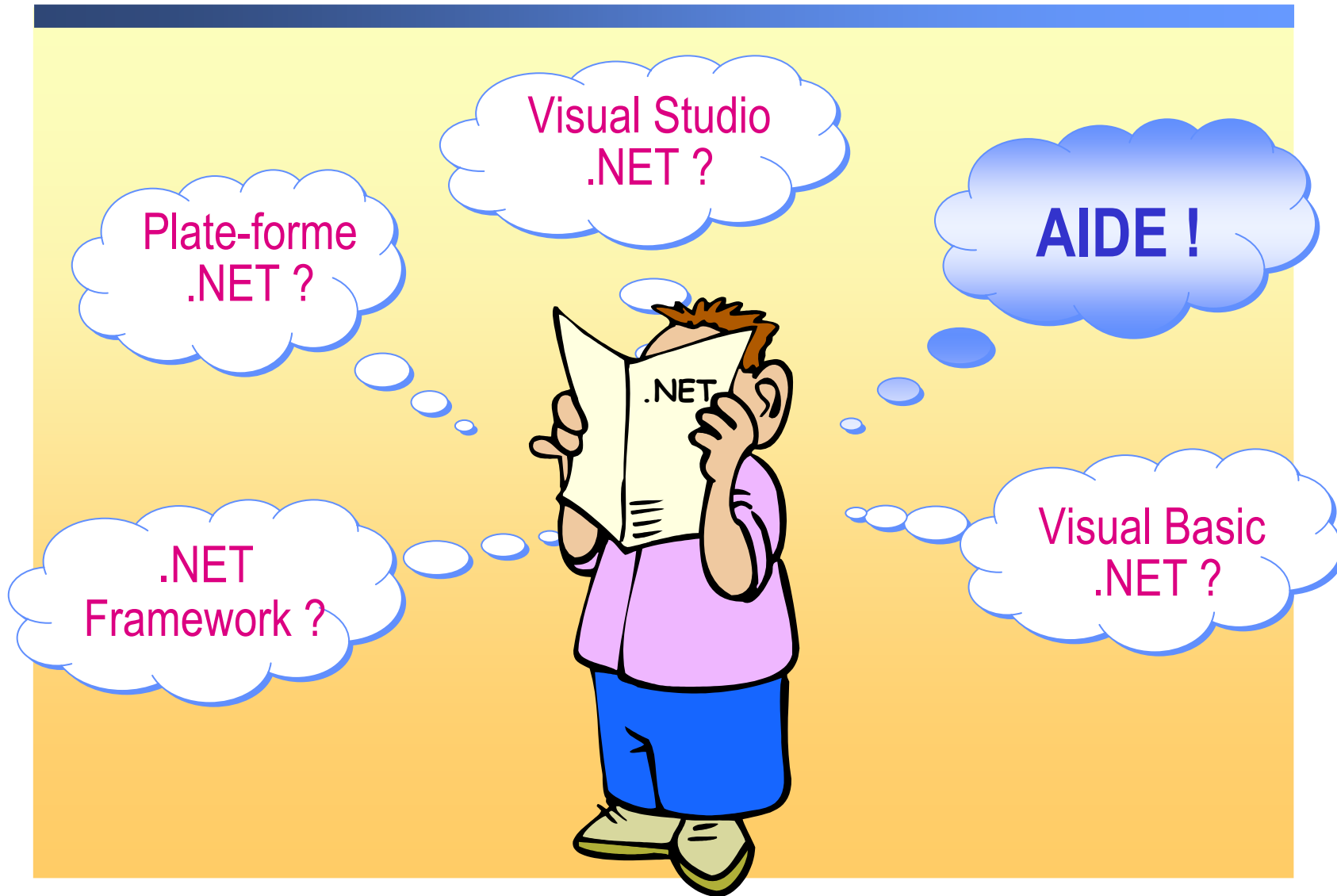
Mise en route

Vue d'ensemble

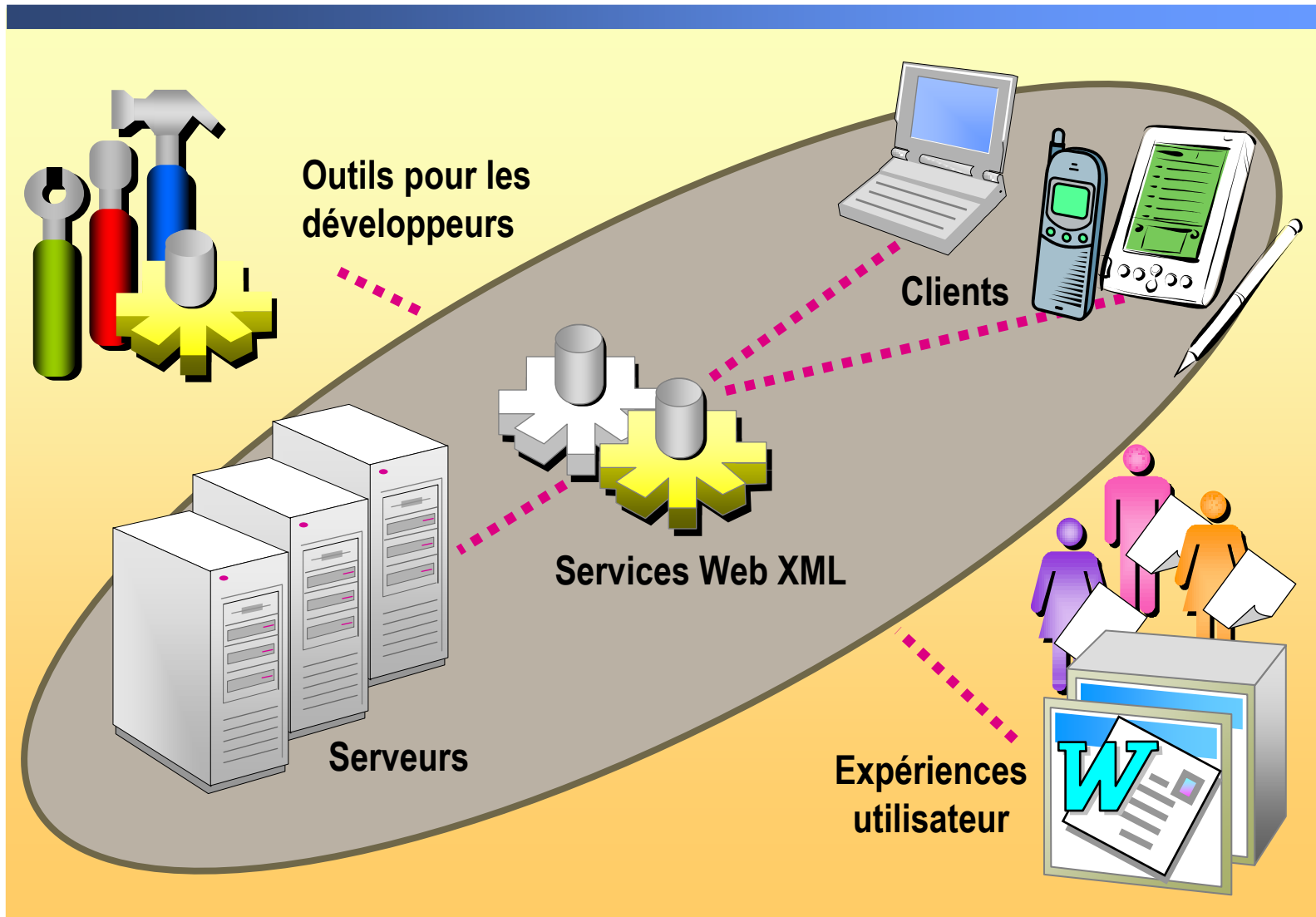


- Concepts de base de l'environnement .NET
- Exploration de l'environnement de développement
- Création d'un projet Visual Basic .NET

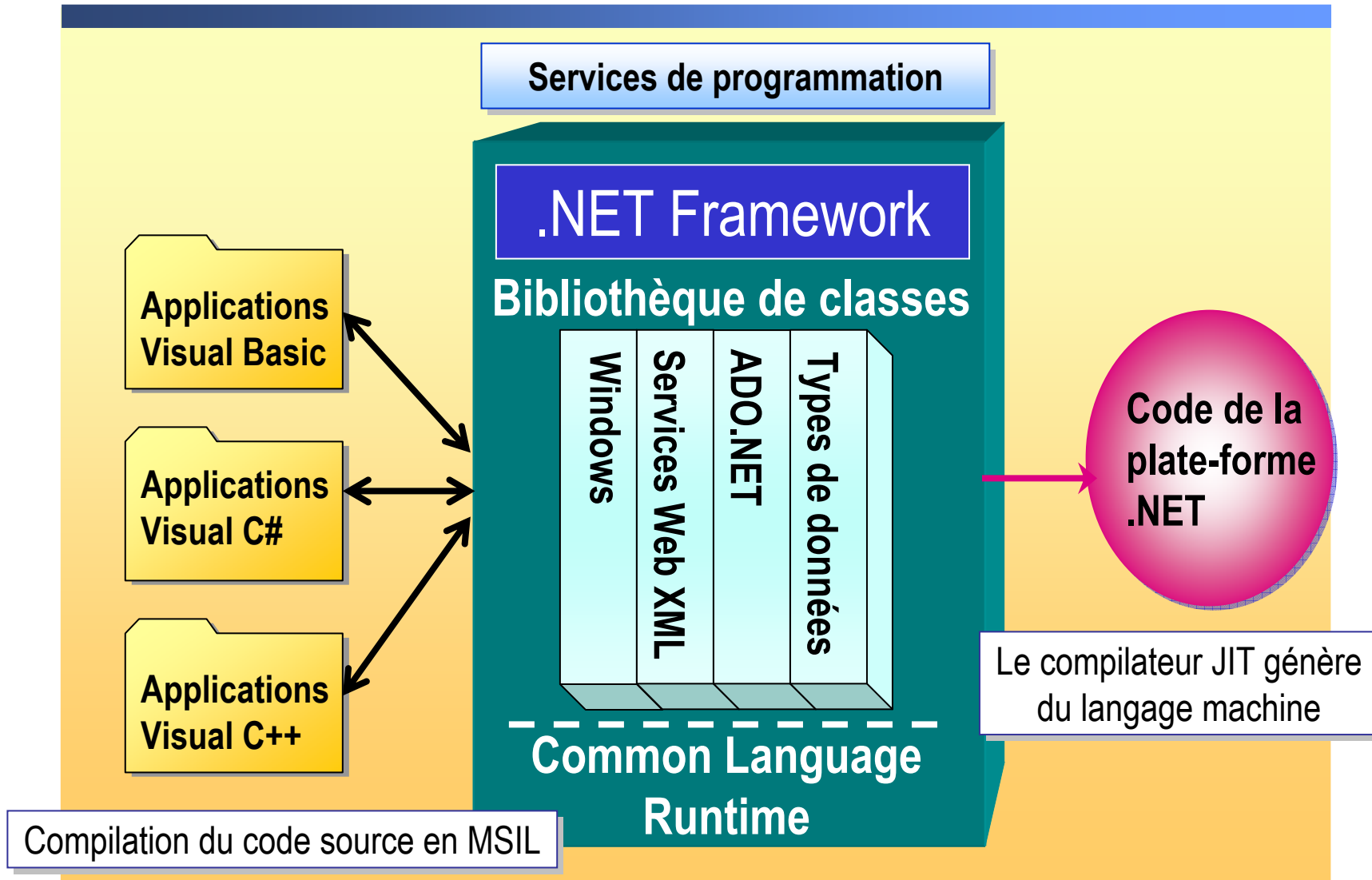
Leçon : Concepts de base de l'environnement .NET



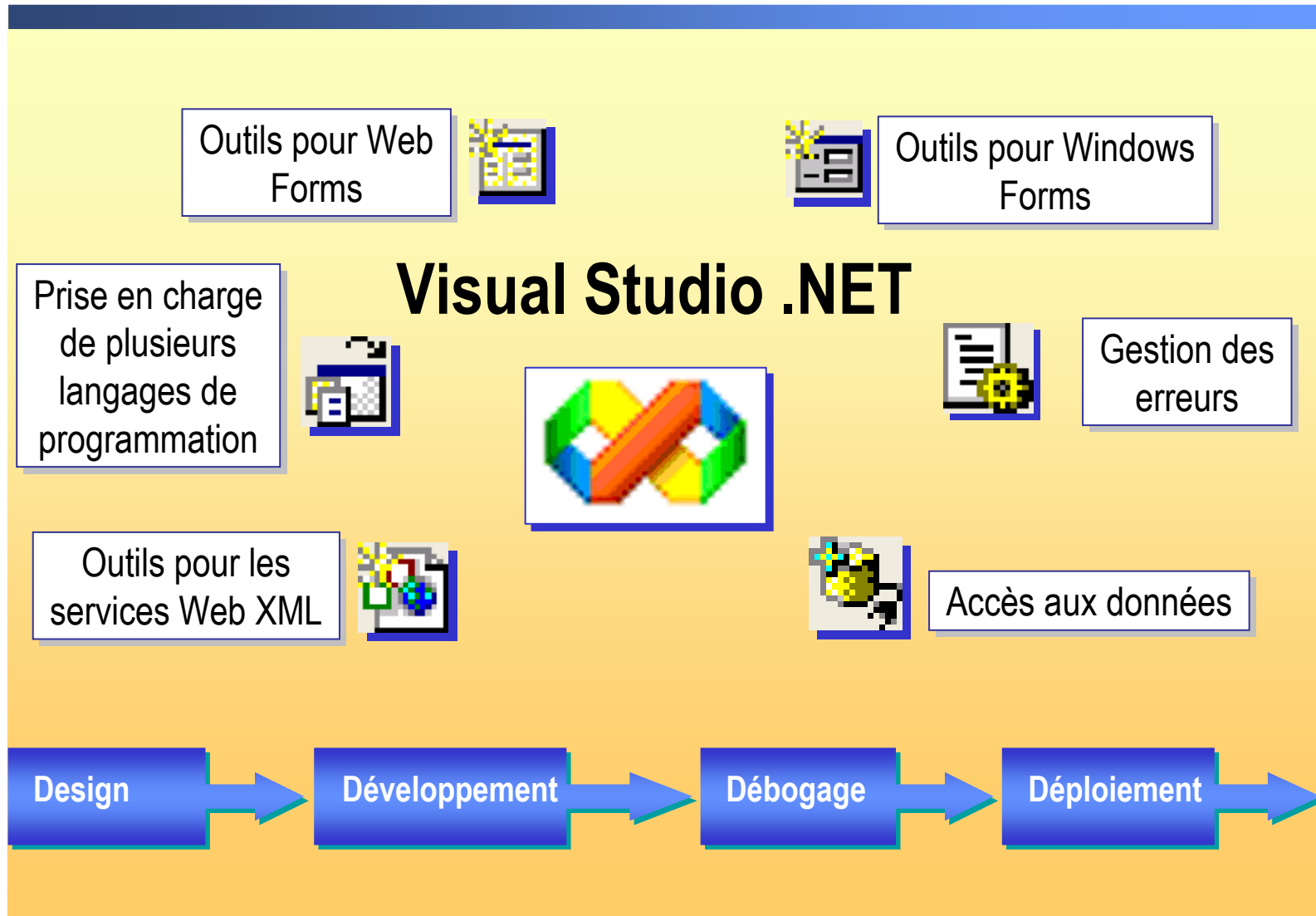
Présentation de la plate-forme .NET



Fonctionnement de Microsoft .NET Framework



Présentation de Microsoft Visual Studio .NET



Utilisation de la page de démarrage

The image shows a screenshot of a web application's start page. A dark blue navigation menu is overlaid on the left side, listing various options. The background page is white with a blue header and footer. The header contains the word 'Dém' and a search bar. The footer contains two buttons: 'Ouvrir un projet' and 'Nouveau projet'. A table with five rows is visible in the center-right area, with the first row highlighted in purple. The table has two columns: the first column contains the word 'Modifié' and the second column contains dates.

Commencer
Nouveautés
Communauté en ligne
Actualités
Recherche en ligne
Téléchargements
Services Web XML
Hébergement Web
Mon profil

Comme
Nouve
Com
Actu
Re
T

cher des exemples

Modifié	
Aujourd'hui	
Aujourd'hui	
Hier	
Hier	

[first applic](#)

Ouvrir un projet Nouveau projet

Présentation d'un modèle d'application

Fournit les fichiers de démarrage,
la structure de projet et les
paramètres d'environnement



Application
Windows



Bibliothèque de
classes



Bibliothèque de
contrôles Wi...



Application Web
ASP.NET

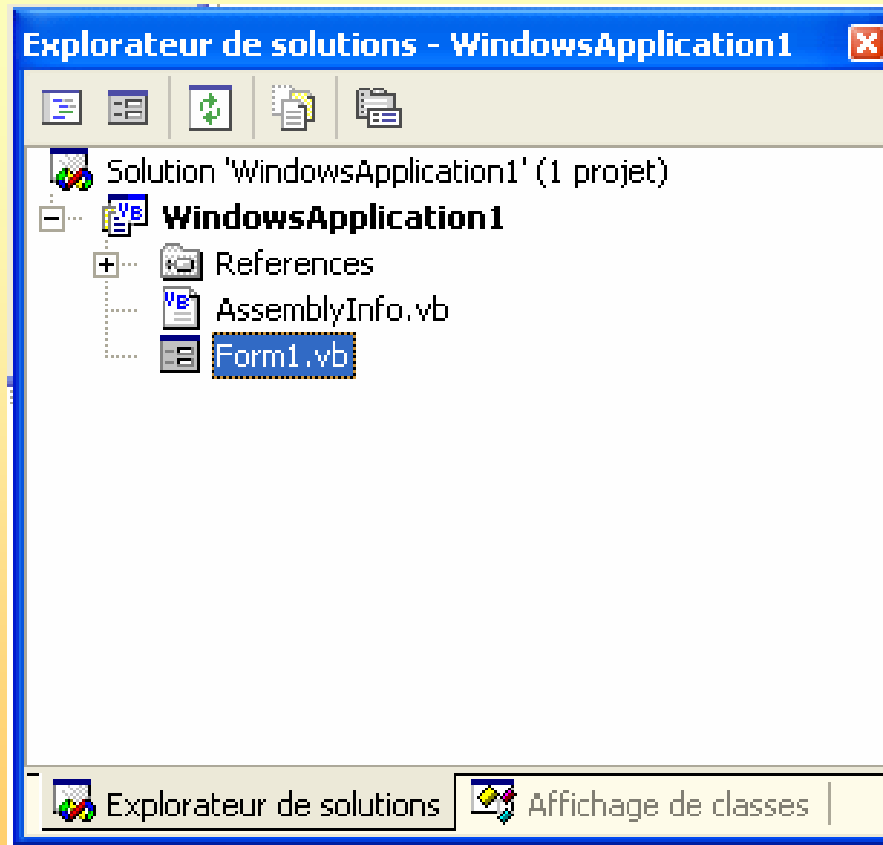


Service Web
ASP.NET



Bibliothèque de
contrôles Web

Utilisation de l'Explorateur de solutions



Application pratique : Utilisation d'un projet Visual Basic .NET



- 1** Démarrez un projet Visual Basic .NET à partir d'un modèle d'application Windows
- 2** Exécutez le projet dans l'environnement de développement
- 3** Générez un fichier exécutable
- 4** Exécutez l'application en dehors de l'environnement de développement
- 5** Visualisez les fichiers dans l'Explorateur de solutions
- 6** Enregistrez le projet et quittez Visual Studio .NET

Leçon : Exploration de l'environnement de développement

The image shows the Microsoft Visual Basic .NET IDE in design mode. The main window is titled "WindowsApplication1 - Microsoft Visual Basic .NET [design] - Form1.vb [Design]*". The interface includes a menu bar (Fichier, Edition, Affichage, Projet, Générer, Débugger, Données, Format, Outils, Fenêtre), a toolbar, and a "Boîte à outils" (Toolbox) on the left. The central area is the "Concepteur Windows Forms" (Windows Forms Designer) showing a form named "Form1" on a grid. On the right, the "Propriétés" (Properties) window is open, displaying the properties of the selected form. A "Fenêtre Propriétés" (Properties Window) callout points to this window.

Callouts in the image identify the following components:

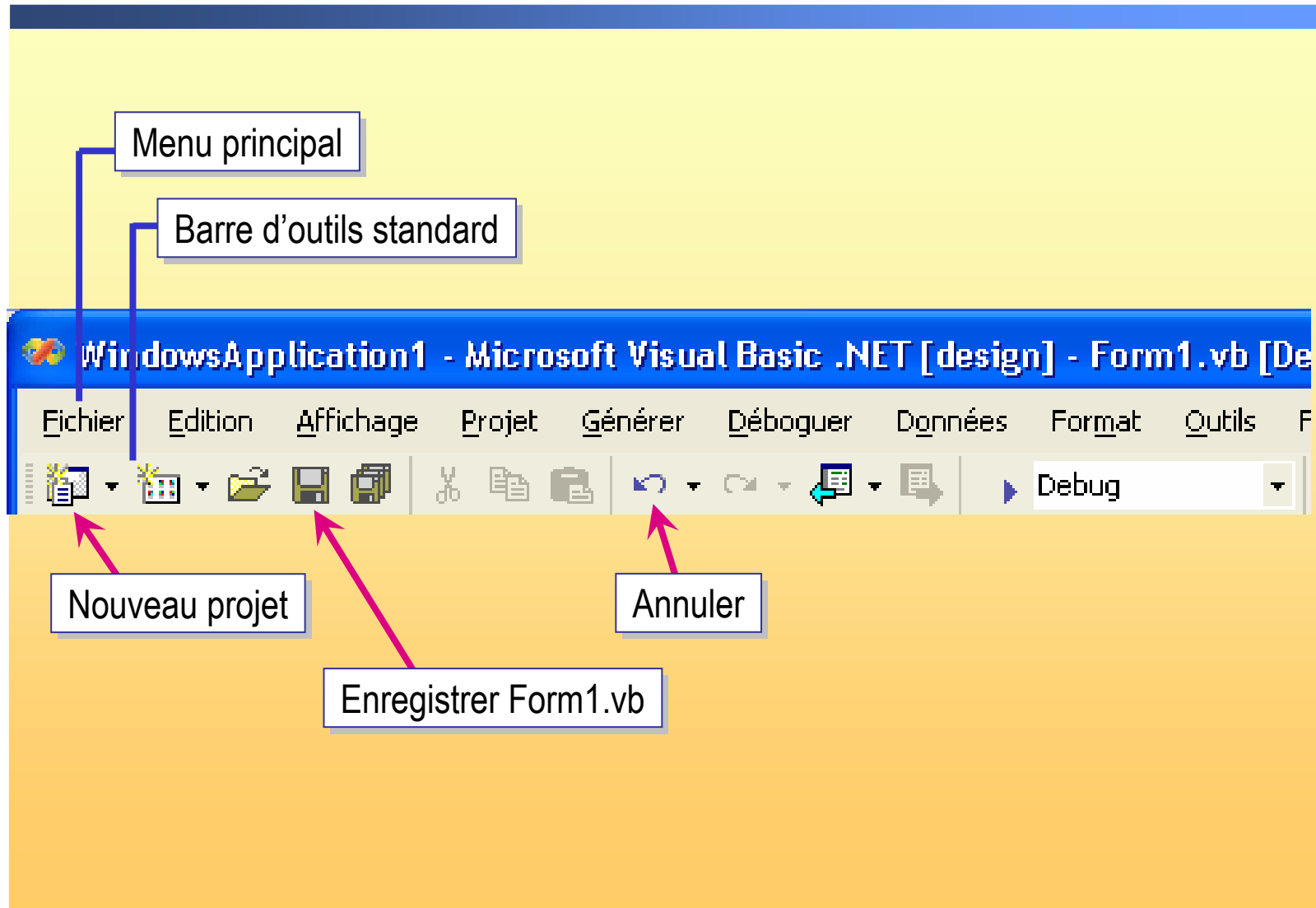
- Menu principal
- Barre d'outils
- Boîte à outils
- Concepteur Windows Forms
- Fenêtre Propriétés

The "Propriétés" window displays the following properties for "Form1" (System.Windows.Forms.Form):

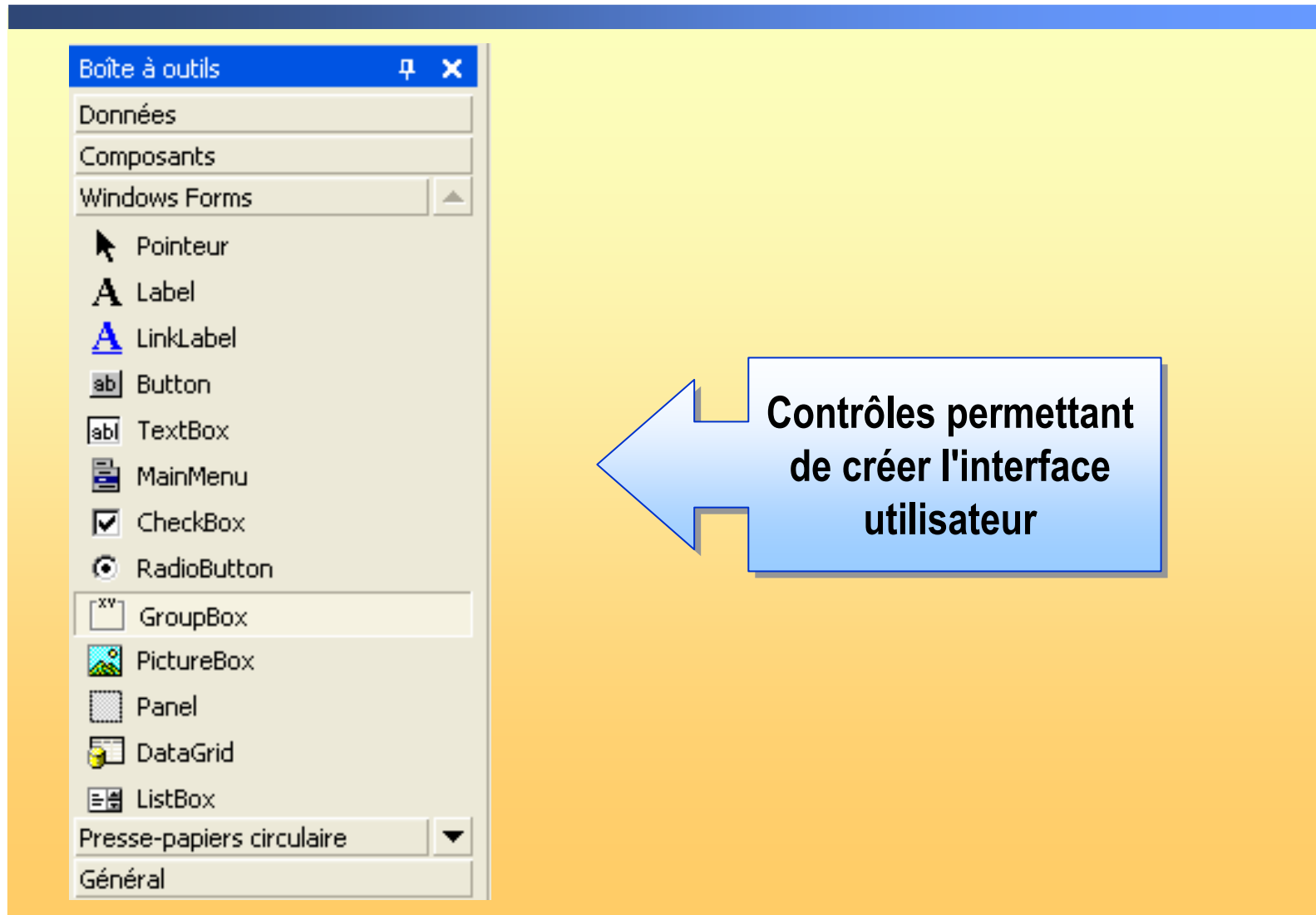
Propriété	Valeur
DockPadding	
Location	0; 0
MaximumSize	0; 0
MinimumSize	0; 0
Size	300; 300
StartPosition	WindowsDefaultLocation
WindowState	Normal

The "Divers" (Miscellaneous) section is currently empty.

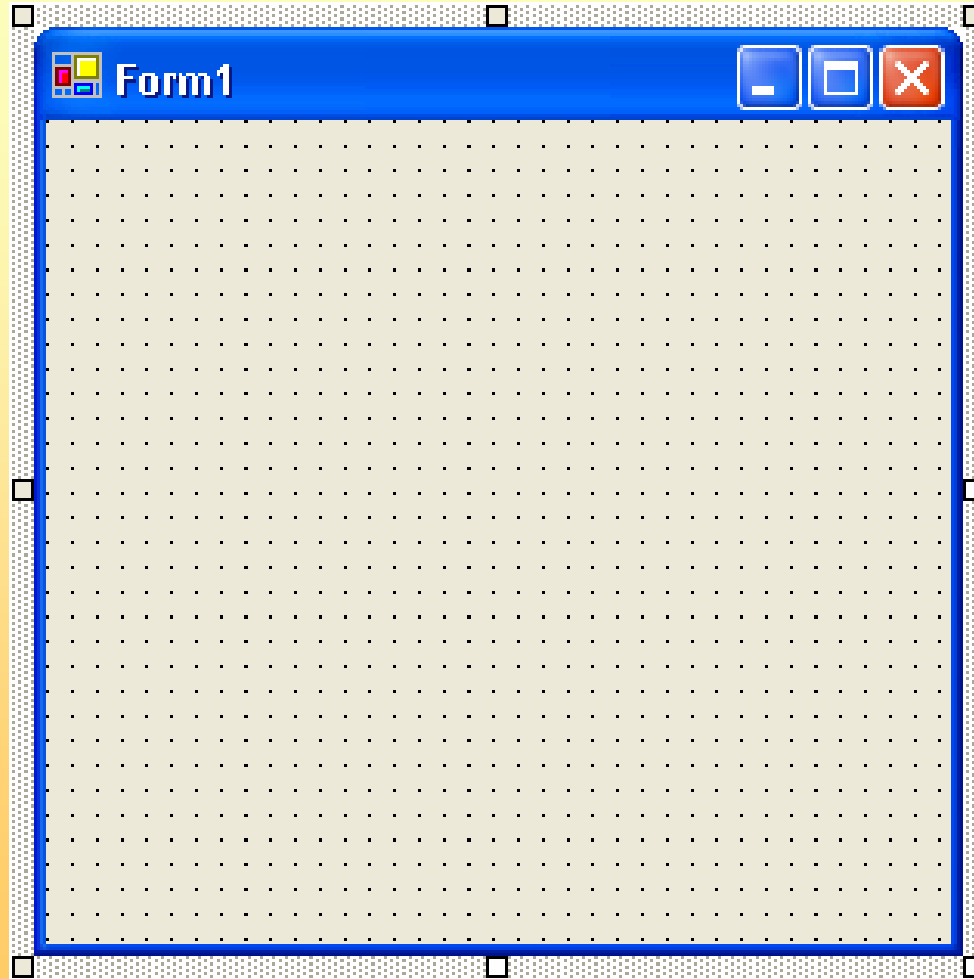
Menus et barres d'outils



La boîte à outils



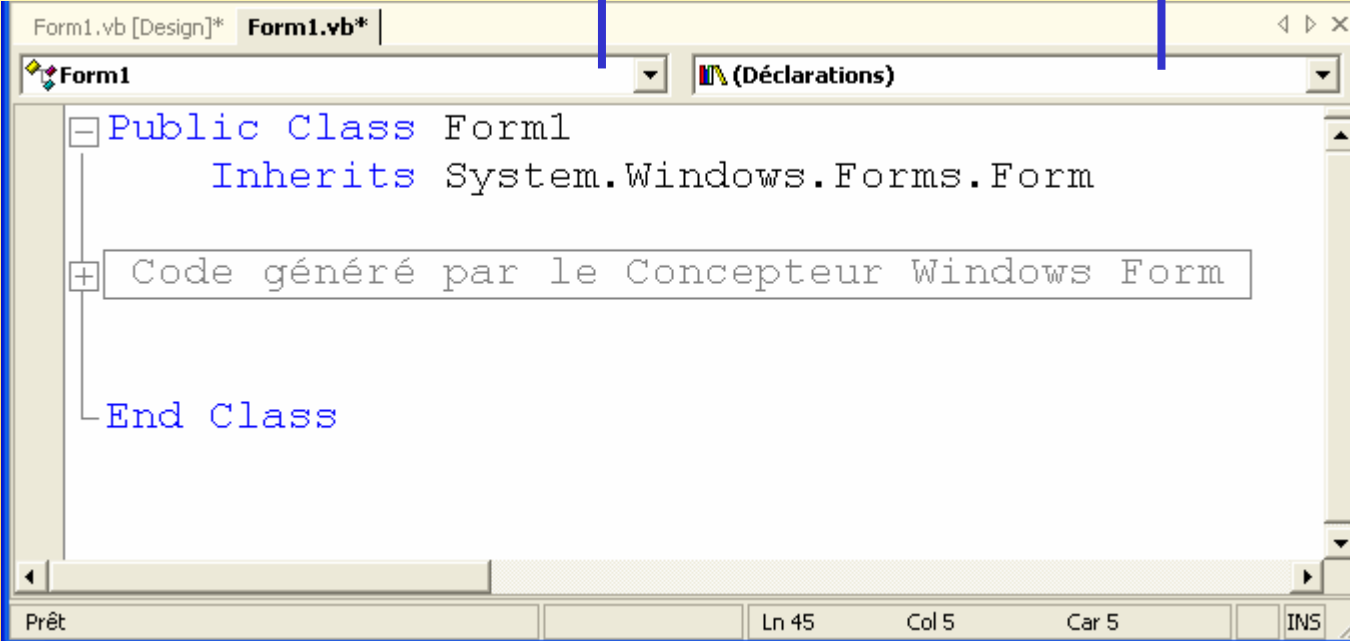
Concepteur Windows Forms



Éditeur de code

Liste Nom de la classe

Liste Nom de la méthode



The screenshot shows a Visual Basic code editor window with the following content:

```
Form1.vb [Design]* Form1.vb*  
Form1 (Déclarations)  
Public Class Form1  
    Inherits System.Windows.Forms.Form  
    Code généré par le Concepteur Windows Form  
End Class
```

The status bar at the bottom indicates: Prêt, Ln 45, Col 5, Car 5, and INS.

Fenêtre Propriétés

The screenshot shows the 'Propriétés' (Properties) window in Visual Studio. The window title is 'Propriétés' and it displays the properties for 'Form1' of type 'System.Windows.Forms.Form'. The properties list includes:

(Name)	Form1
AcceptButton	(aucun)
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
AutoScale	True
AutoScroll	False
AutoScrollMargin	0; 0
AutoScrollMinSize	0; 0
BackColor	<input type="color"/> Control
BackgroundImage	<input type="image"/> (aucun)

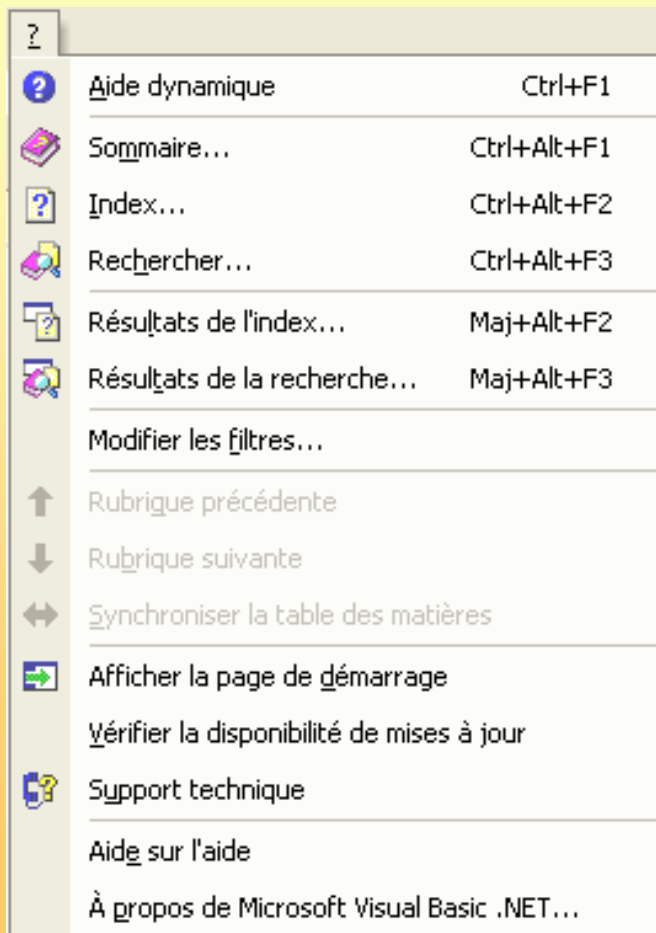
A blue callout box with an arrow points to the properties list, containing the text: "Définit les propriétés comme la taille, la légende ou la couleur".

Autres fenêtres de programmation

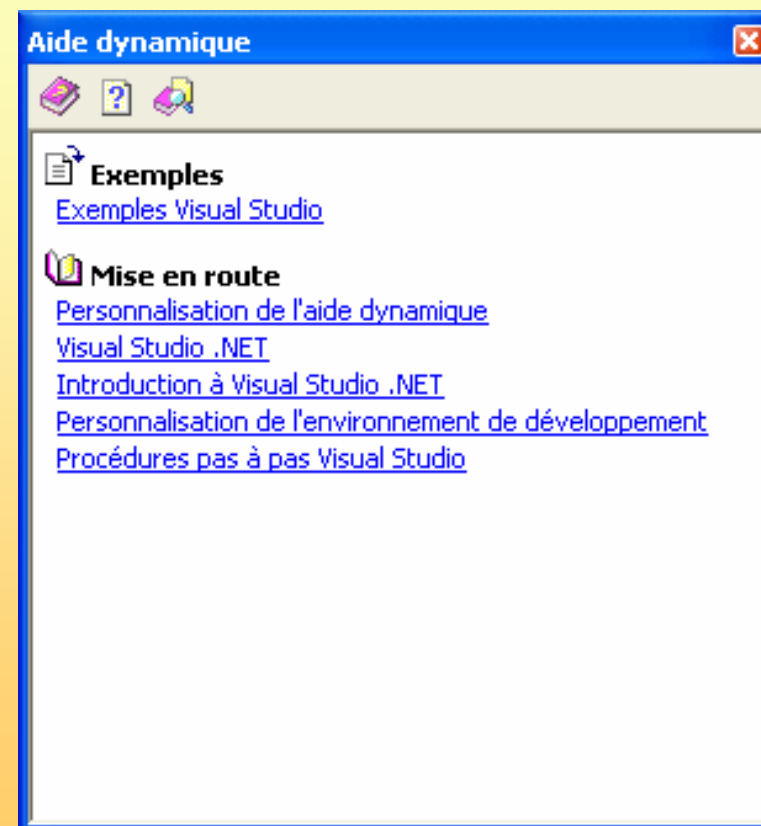
Fenêtre de programmation	Objectif
Liste des tâches	Permet d'organiser et de gérer les tâches liées à la création d'une application
Sortie	Affiche des messages d'état pour diverses fonctionnalités de l'environnement de développement
Affichage de classes	Permet d'analyser du code et de retrouver les symboles qui représentent des éléments de programmation dans votre solution
Commande	Permet d'exécuter des commandes ou de déboguer et de tester des expressions dans l'environnement de développement
Explorateur d'objets	Permet d'afficher des objets et leurs membres

Systeme d'aide en ligne

Menu ? (Aide)



Aide dynamique

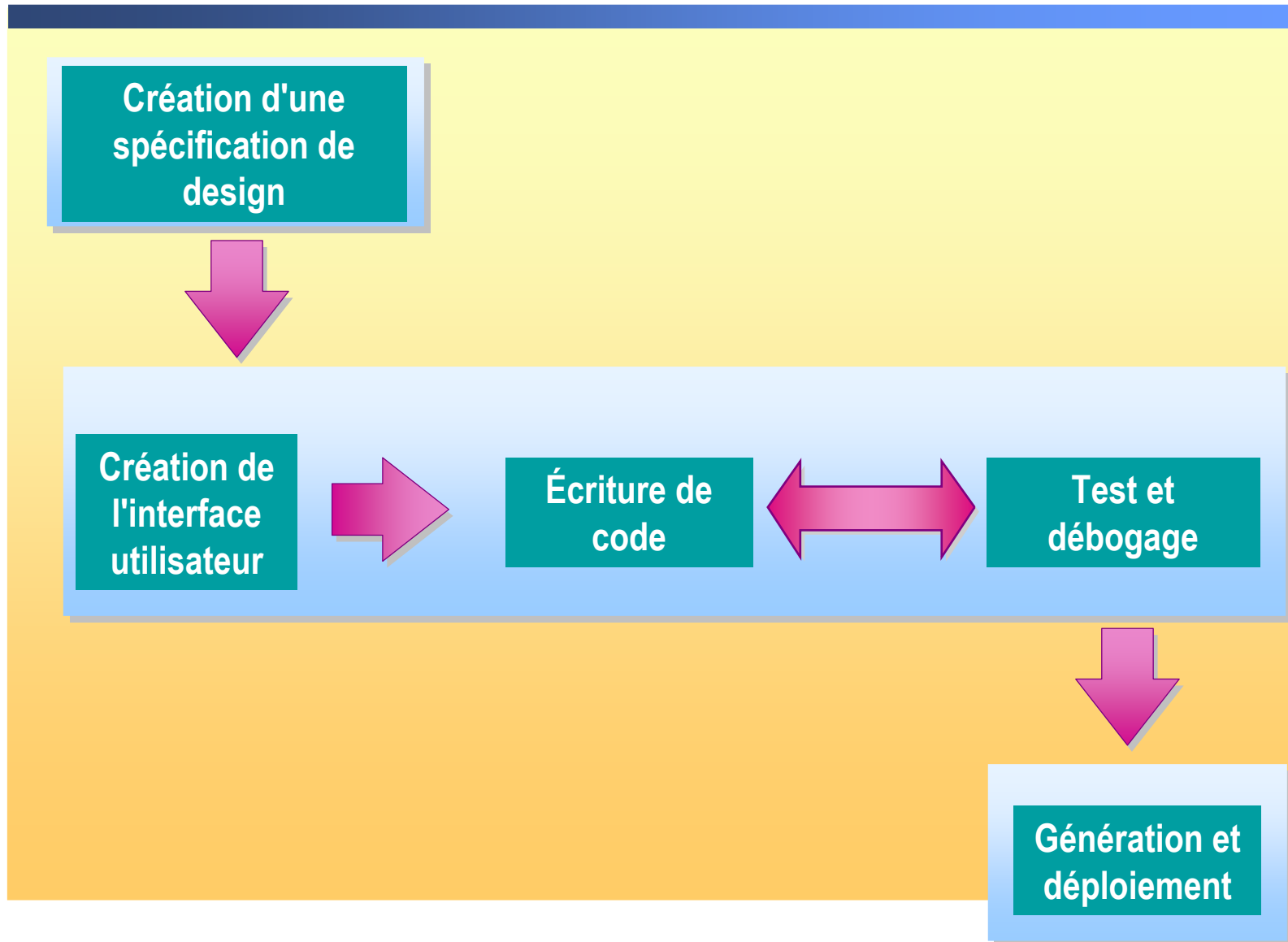


Application pratique : Utilisation de l'environnement de développement

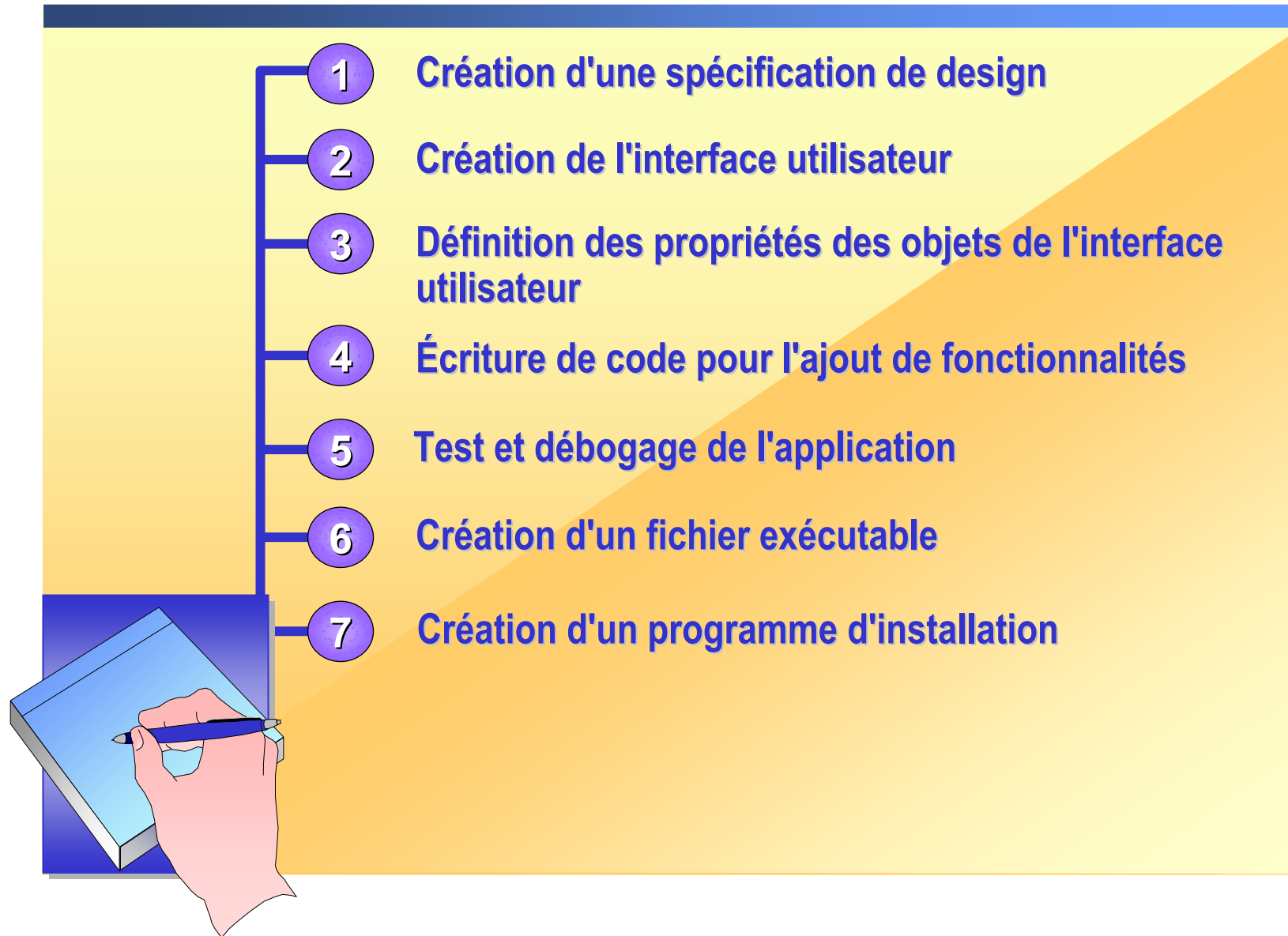


- 1** Ouvrez et exécutez une application existante
- 2** Examinez un formulaire dans le Concepteur Windows Forms et l'éditeur de code
- 3** Ouvrez, fermez, rouvrez et masquez la boîte à outils
- 4** Examinez les paramètres des propriétés des contrôles
- 5** Utilisez l'Aide dynamique

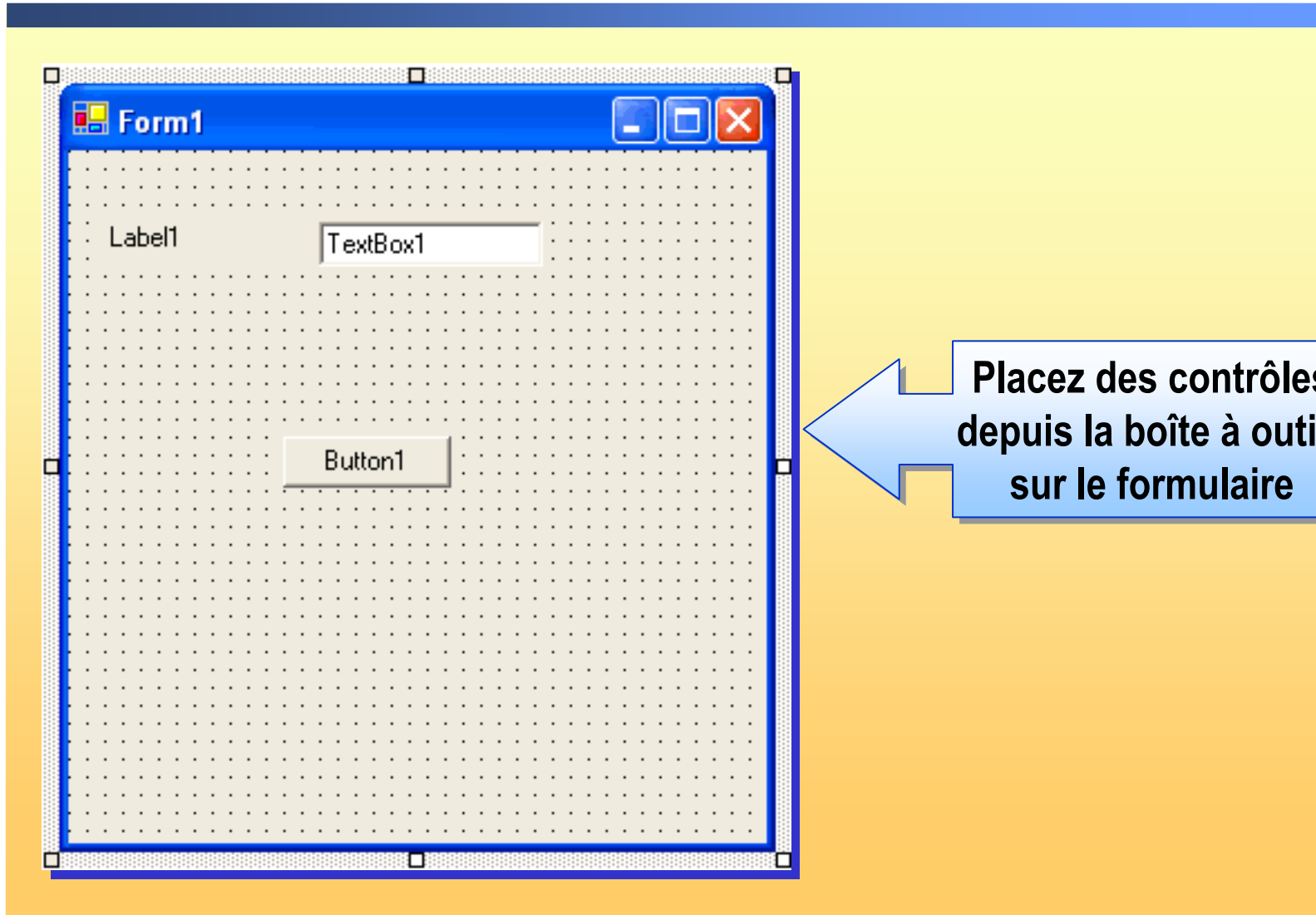
Leçon : Création d'un projet Visual Basic .NET



Processus de développement



Création de l'interface utilisateur



Définition des propriétés des contrôles

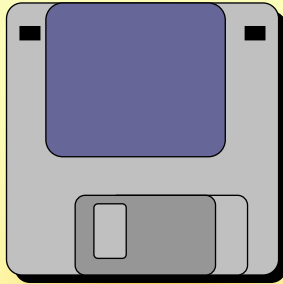
Propriétés	Valeurs
(Name)	Textbox1
BackColor	Blue
Autosize	True
Visible	True
Border	Fixed 3D
Font	Microsoft SanSerif, 8.2 pt
Text	Textbox1

Ajout de code pour les contrôles

- Dans la liste Nom de la classe, cliquez sur le contrôle
- Dans la liste Nom de la méthode, cliquez sur l'événement
- Ajoutez du code entre Private Sub et End Sub

```
Private Sub Button1_Click(. . .)Handles Button1.Click  
    'Ajoutez votre code ici  
End Sub
```

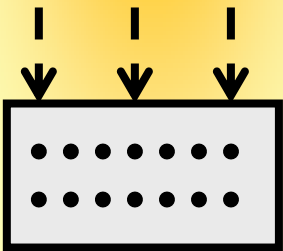
Enregistrement, génération et exécution de l'application



Enregistrement de l'application



**Exécution de la solution dans
l'environnement de développement**



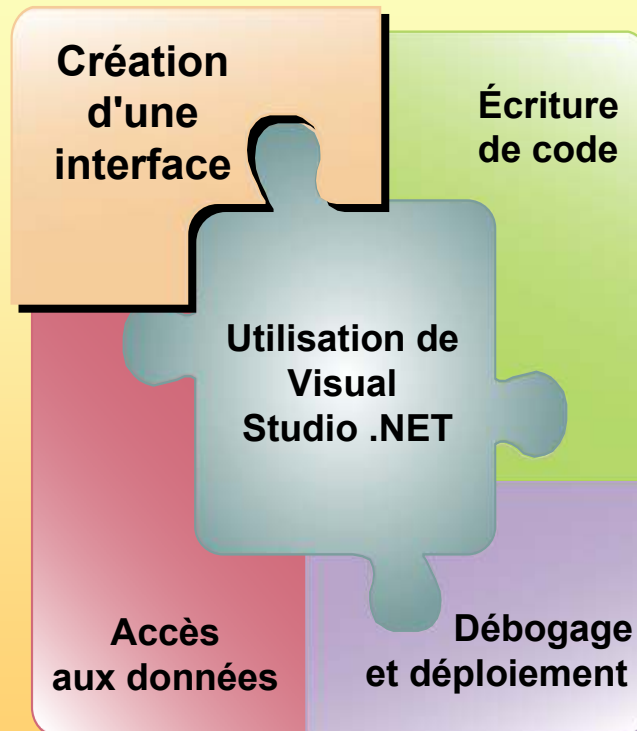
Génération d'un fichier exécutable

www.Mcours.com

Site N°1 des Cours et Exercices Email: contact@mcours.com

Module 2 : Utilisation de formulaire et de contrôles

Vue d'ensemble



- Compréhension des concepts de programmation
- Utilisation de Windows Forms
- Utilisation des contrôles
- Présentation de votre code

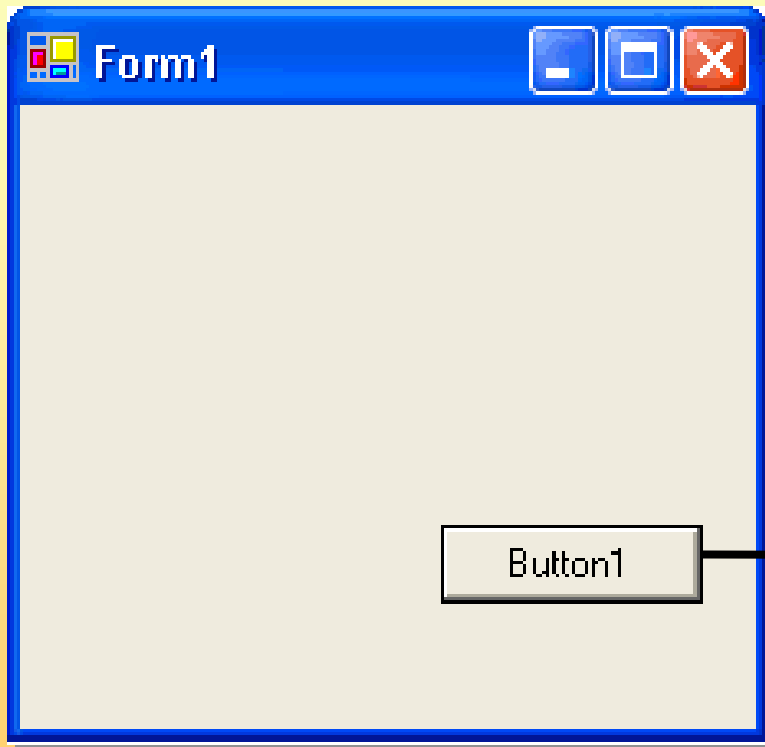
Leçon : Compréhension des concepts de programmation

- Programmation événementielle
- Classes
- Événements
- Méthodes
- Objets
- Propriétés

www.Mcours.com

Site N°1 des Cours et Exercices Email: contact@mcours.com

Présentation de la programmation événementielle



```
Sub Button1_Click (...)  
    'insérer le code de  
    'l'événement  
End Sub
```


Classes : Modèles servant à la création des objets

Classe

Une représentation symbolique d'un objet

Analogie : un plan



Objet

Une instance d'une classe

Analogie : une maison conçue à partir du plan

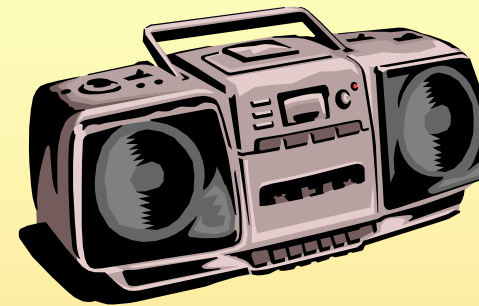
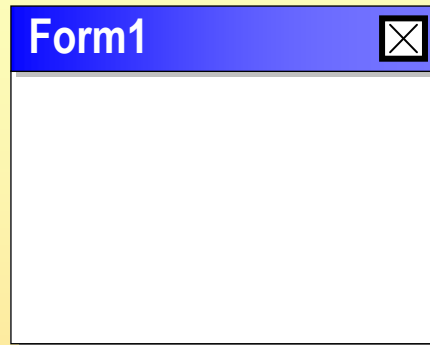


Exemple

Chaque formulaire d'un projet Visual Basic est un objet distinct

Chaque formulaire est une instance de la classe Form

Présentation des propriétés, des méthodes et des événements



Propriétés	Size Text	Volume Basses
Méthodes	Close Hide	Rechercher une station
Événements	Click	Niveau de pile faible

Application pratique : Propriétés, méthodes et événements



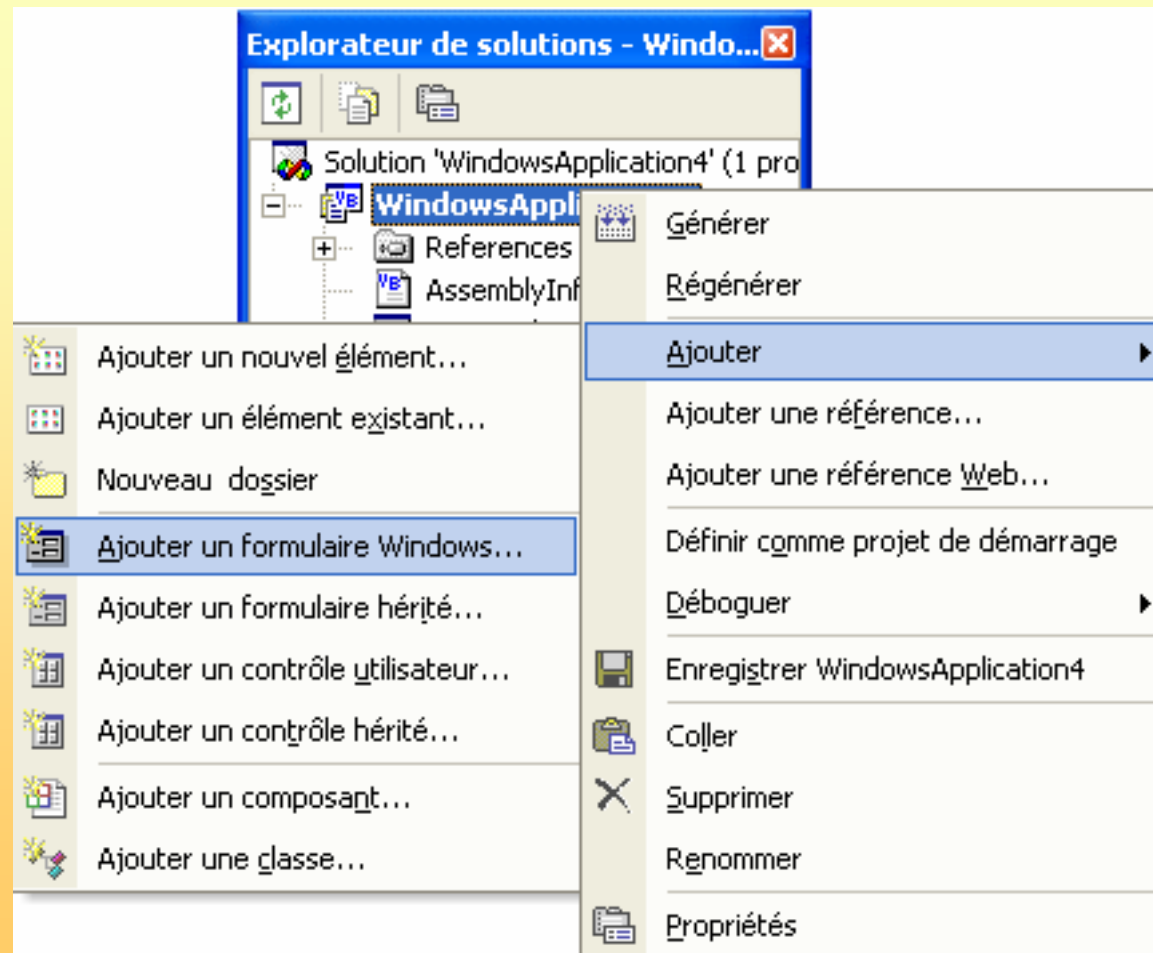
Associez à chacun des objets suivants les propriétés, les méthodes et les événements appropriés :

- Porte
- Voiture
- *Exemple : Téléphone*
 - *Propriétés : Largeur, Couleur, Volume*
 - *Méthodes : Sonner, RéglerVolume*
 - *Événements : RéglageVolume, FinNumérotation*

Leçon : Utilisation de Windows Forms

- **Création d'un formulaire**
- **Définition des propriétés d'un formulaire**
- **Appel des méthodes**
- **Gestion des événements de formulaire**
- **Formulaires modaux et non modaux**
- **Gestion de plusieurs formulaires**

Création d'un formulaire



Définition des propriétés d'un formulaire

The screenshot shows the Visual Studio Properties window for a form named **Form1** (System.Windows.Forms.Form). The window displays a list of properties for the selected **AcceptButton** control. The properties listed include:

(Name)	Form1
AcceptButton	(aucun)
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False

Annotations with arrows point to the following elements:

- Nom de l'objet**: Points to the object name **Form1** in the top dropdown.
- Bouton Par catégorie**: Points to the 'Sort by category' button (A 2 ↓).
- Bouton Alphabétique**: Points to the 'Sort by alphabetically' button (A ↓).
- Volet Description**: Points to the description pane for the **AcceptButton** property, which reads: "Le bouton d'acceptation du formulaire. S'il est défini, le bouton est "cliqué" lorsque l'utilisateur appuie sur la touche "ENTRÉE"."

- Si vous modifiez la propriété Name de Form1, vous devez aussi donner le nouveau nom à l'objet de démarrage de votre projet

Appel des méthodes

```
Code généré par le Conce  
Private Sub Form1_Load  
Me.  
End S  
End Class
```

Méthodes

- CenterToScreen
- ClientSize
- Close
- components
- Container
- Contains
- ContextMenu
- ControlBox
- ControlCollection
- Controls

```
Sub Form1_Click  
    Me.CenterToScreen( )  
End Sub
```

Gestion des événements de formulaire

Page de démarrage | Form1.vb [Design]* | Form1.vb*

(Événements de la classe de base)

```
Inherits System.  
Code généré par le  
End Class
```

(Déclarations)

- Click
- Activated
- BackColorChanged
- BackgroundImageChanged
- BindingContextChanged
- CausesValidationChanged
- ChangeUICues
- Click
- Closed
- Closing
- ContextMenuChanged
- CursorChanged

Zone de liste
Nom de la classe

Événements

Application pratique : Création du code associé à des événements de formulaire



1 Ouvrez une nouvelle application Windows dans Visual Basic .NET

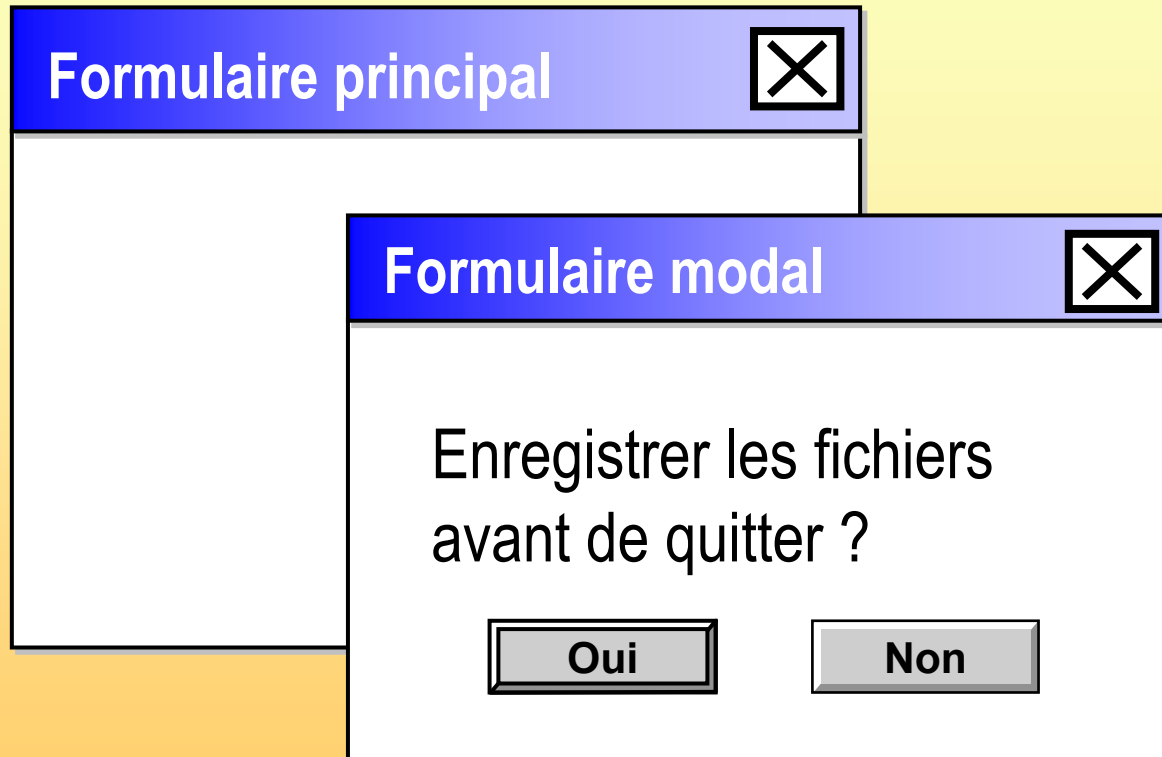
2 Ouvrez l'Éditeur de code correspondant à ce formulaire

3 Créez le gestionnaire d'événement Form1_Click

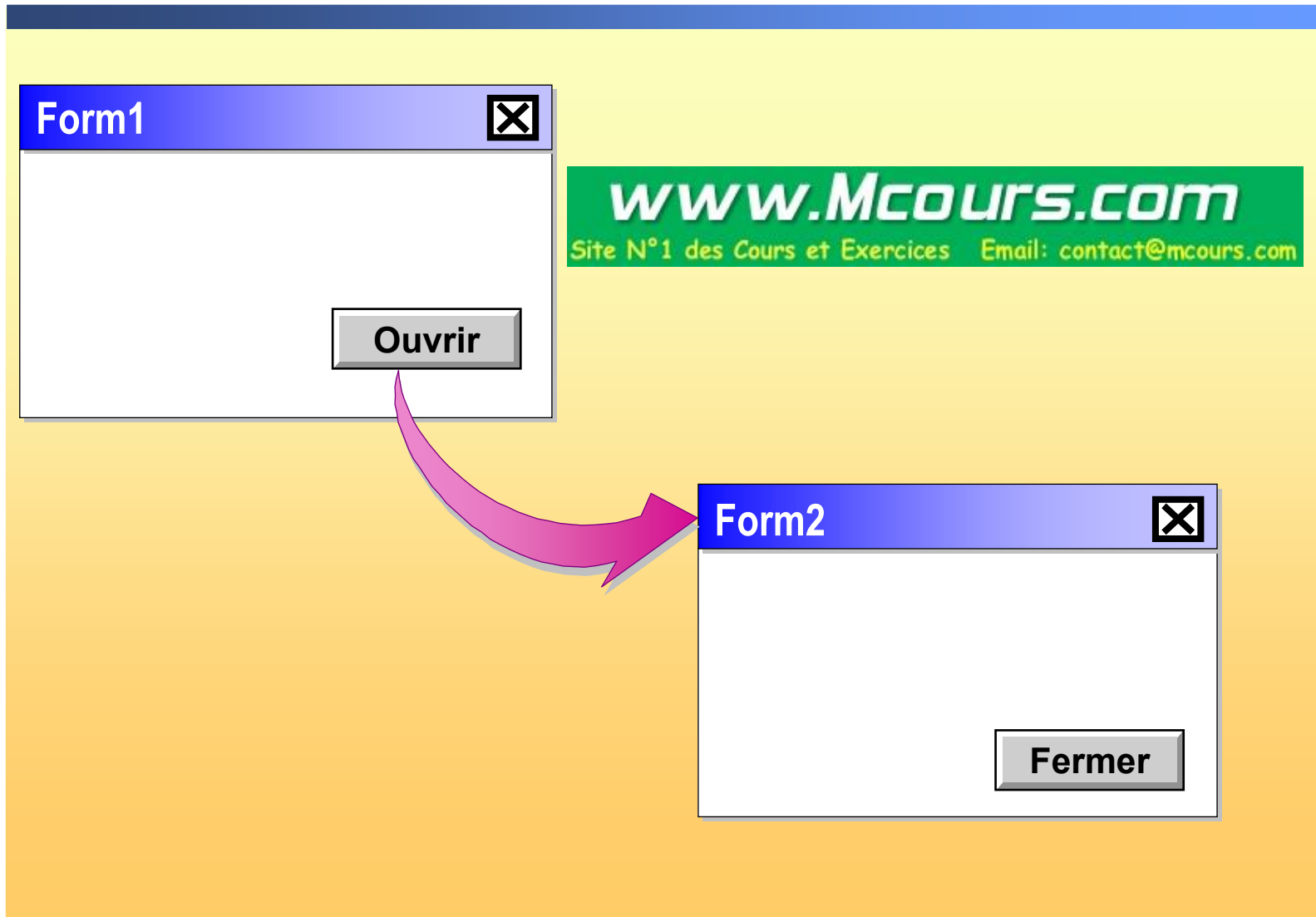
4 Ajoutez du code au gestionnaire d'événements

5 Exécutez l'application et testez votre code

Formulaires modaux et non modaux



Gestion de plusieurs formulaires



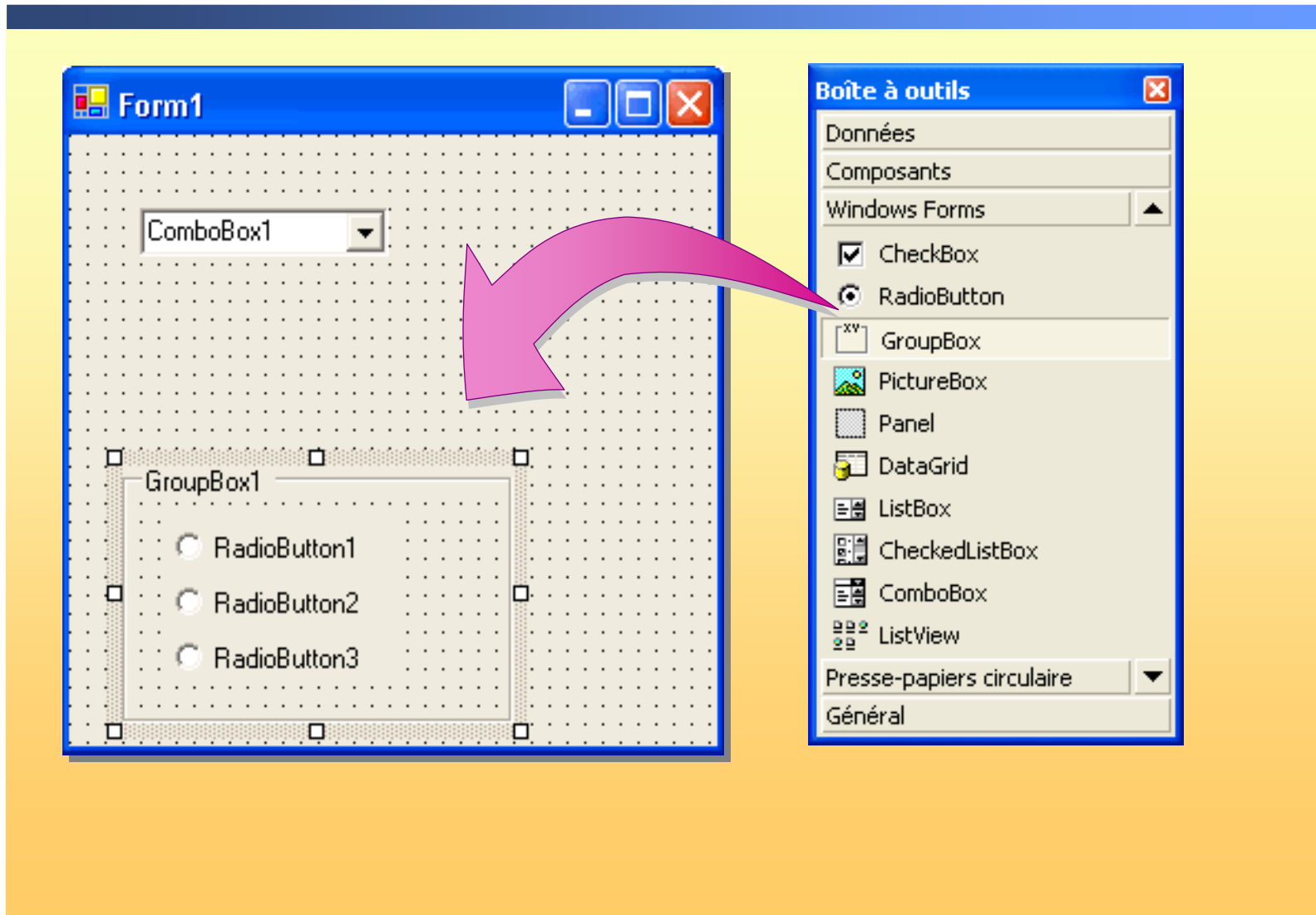
Leçon : Utilisation des contrôles

- Ajout de contrôles à un formulaire
- Définition des propriétés d'un contrôle
- Association de code à des événements de contrôle
- Utilisation de la fonction MessageBox

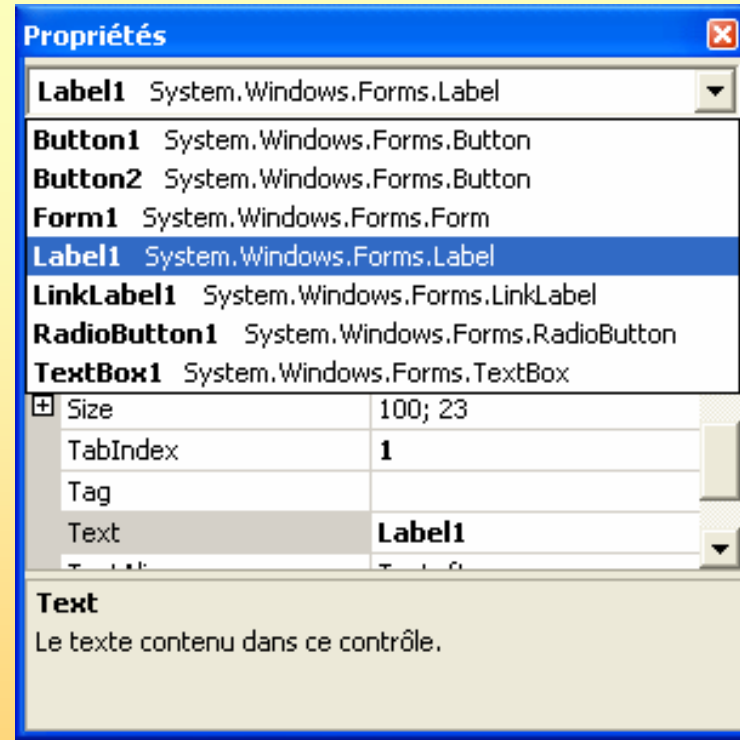
www.Mcours.com

Site N°1 des Cours et Exercices Email: contact@mcours.com

Ajout de contrôles à un formulaire

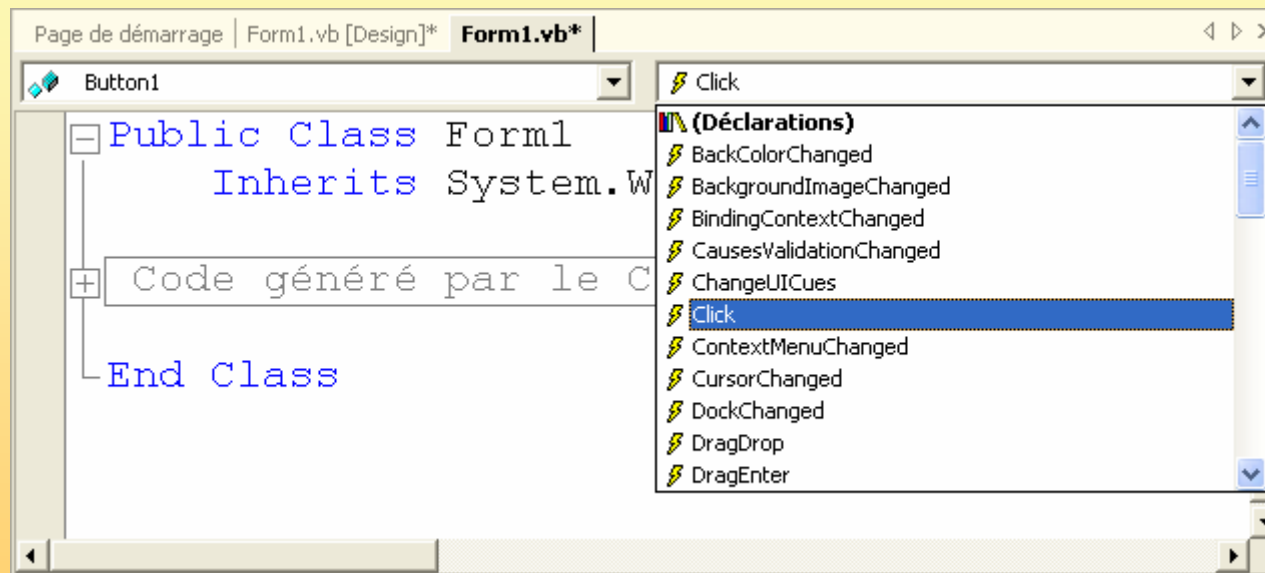


Définition des propriétés d'un contrôle



- Vous pouvez choisir la même valeur de propriété pour plusieurs contrôles à la fois
- Définissez les propriétés de contrôles individuels pour faciliter l'accessibilité de l'application

Association de code à des événements de contrôle



Utilisation de la fonction MessageBox

The diagram illustrates a standard Windows MessageBox window titled "Question". The window contains a question mark icon, the text "Voulez-vous enregistrer ce fichier ?", and two buttons: "Oui" (highlighted with a dotted border) and "Non".

Callouts identify the following components:

- Légende en tant que chaîne**: Points to the title bar text "Question".
- Icône**: Points to the question mark icon.
- Texte en tant que chaîne**: Points to the main text "Voulez-vous enregistrer ce fichier ?".
- Bouton par défaut**: Points to the "Oui" button.
- Boutons**: Points to both the "Oui" and "Non" buttons.

At the bottom, a code block shows the function signature:

```
MessageBox.Show( )
```


Application pratique : Création d'un message



1 Ouvrez une nouvelle application Windows dans Visual Basic .NET

2 Créez l'interface utilisateur

3 Définissez les propriétés du formulaire et du contrôle

4 Ouvrez un gestionnaire d'événements et ajoutez du code pour créer une boîte de message

5 Exécutez l'application et testez votre code

Leçon : Présentation de votre code

- Conventions d'affectation de noms
- Mise en forme et documentation du code

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Conventions d'affectation de noms

■ Règles d'affectation de noms

- Utilisez uniquement des lettres, des chiffres et des soulignés (_)
- Commencez par une lettre ou un souligné
- N'utilisez pas de mots clés

Answer42
42Answer



OpenButton
True



■ Principe d'affectation de noms

- **Casse** : Utilisez PascalCasing ou camelCasing, selon l'élément que vous nommez
- **Mécanique** : Utilisez des substantifs pour les objets, des verbes pour les méthodes, etc.
- **Choix des mots** : Soyez cohérent, utilisez les mêmes termes dans les différents segments de code

BADSTYLE
_poorstyle
BestStyle



Mise en forme et documentation du code

- Mise en retrait du code

```
Sub Button1_Click
    Me.Close
End Sub
```

- Signe de continuation et concaténation

```
MessageBox.Show("Nom Utilisateur = " & UserName.Text & _
    ", Mot de passe = " & Password.Text)
```

- Ajout de commentaires dans le code

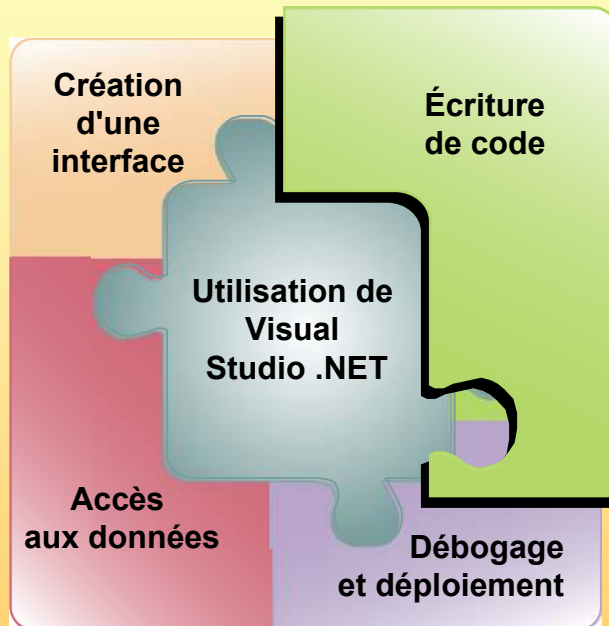
```
'Rendre CalculationForm visible
Dim CalculationForm as new Form2( )
CalculationForm.Show( )
```

- Ajoutez des commentaires dans le code pour en faciliter la lecture et la mise à jour

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Module 3 : Utilisation de variables et de tableaux

Vue d'ensemble



- Introduction aux types de données
- Utilisation de variables
- Portée des variables
- Conversion des types de données
- Création et utilisation de structures
- Stockage de données dans des tableaux

Leçon : Introduction aux types de données

■ **Systeme de types
communs**

■ **Types valeur**

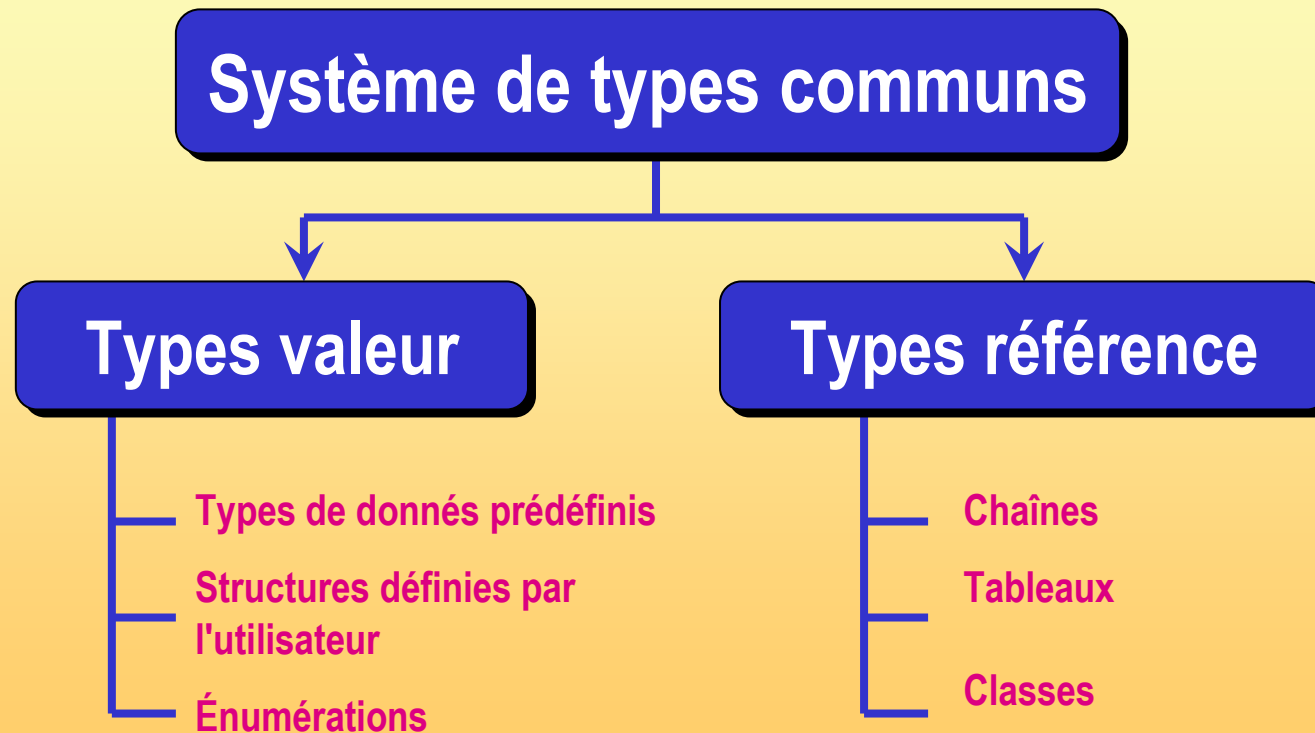
■ **Types référence**

www.Mcours.com

Site N°1 des Cours et Exercices Email: contact@mcours.com

Présentation du système de types communs

Définit la manière dont les types fonctionnent dans le Common Language Runtime



Types de données

Type Visual Basic .NET	Taille de stockage	Plage des valeurs
Boolean	2 octets	True ou False
Date	8 octets	De 0:00:00, le 1er janvier 0001, à 23:59:59, le 31 décembre 9999
Decimal	16 octets	Jusqu'à 29 chiffres significatifs avec des valeurs comprises jusqu'à $7,9228 \times 10^{28}$ (signées)
Double	8 octets	De -4,94065645841246544E-324 à +1,79769313486231570E+308 (signées)
Integer	4 octets	De -2 147 483 648 à +2 147 483 647 (signées)
Single	4 octets	De -3,4028235E+38 à 1,401298E-45 (signées)
String	Varie	De 0 à environ 2 milliards de caractères Unicode

Choix d'un type de données

Type de données	Gestion	Type	Exemple
Boolean	Conditions True ou False	Valeur	True
Short, Integer, Long et Byte	Tous les entiers	Valeur	23 (Integer)
Single, Double et Decimal	Nombres composés d'entiers et de fractions	Valeur	9456,72 (Decimal)
Date	Valeurs horaires et de date	Valeur	12/02/2003 12:30:42
String	Caractères pouvant être imprimés et affichés	Référence	« Maison »
Object	Pointeur vers la valeur d'un objet	Référence	myClass myPerson

Application pratique : Choix du type de données



1 Analysez les exemples de données

2 Tenez compte de la taille et du type des données

3 Sélectionnez le type de données le plus dense

Leçon : Utilisation de variables

Tâches

- 1** Attribution d'un nom aux variables
- 2** Déclaration de variables
- 3** Affectation de valeurs aux variables
- 4** Utilisation de variables

Présentation des variables

- Les variables stockent des valeurs sujettes à modification lorsqu'une application s'exécute
- Une variable est composée des six éléments suivants :

Élément	Description
Nom	Mot que vous utilisez pour vous référer à la variable dans le code
Adresse	Emplacement de la mémoire où la valeur de la variable est conservée
Type de données	Type et taille d'origine des données pouvant être stockées par la variable
Valeur	Valeur au niveau de l'adresse de la variable
Portée	Ensemble du code pouvant accéder à la variable et l'utiliser
Durée de vie	Période au cours de laquelle une variable est valide et utilisable

Attribution d'un nom aux variables

■ Règles

- Commencer chaque nom de variable par un caractère alphabétique ou un trait de soulignement (_)
- Ne pas utiliser d'espaces ni de symboles
- Ne pas utiliser de mots clés tels que **Integer** ou **Date**

■ Exemples

- `NomClient` (PascalCasing)
- `soldeCompte` (camelCasing)

Déclaration de variables

- **Syntaxe**

- `Dim nomDeLaVariable As Type`

- **Exemples de type valeur**

```
Dim numberBooks As Integer  
Dim squareFootage As Single
```

- **Exemples de type référence**

```
Dim myForm As Form  
Dim userInput As String
```

Incidences de Option Explicit sur les variables

■ Lorsque Option Explicit est On (réglage par défaut)

- Vous devez déclarer explicitement les variables avant de les utiliser
- Réduit les erreurs de logique et facilite la gestion du code
- Entraîne une exécution du code plus rapide

■ Lorsque Option Explicit est Off

- Vous pouvez déclarer implicitement une variable par le simple fait de l'utiliser dans votre code
- Accroît le risque de conflits de noms et de comportements inattendus provoqués par des erreurs d'orthographe
- Entraîne une exécution du code plus lente

Affectation de valeurs aux variables

Vous pouvez effectuer les opérations suivantes :

- Affecter une valeur à une variable après la déclaration de cette dernière

```
Dim birthday As Date  
birthday = #3/9/1974#
```

- Affecter une valeur au moment de la déclaration de cette dernière

```
Dim birthday As Date = #3/9/1974#
```

Utilisation de variables

Vous pouvez utiliser des variables pour :

- **Stocker des valeurs issues d'une expression**
- **Stocker des informations saisies par l'utilisateur**
- **Stocker des objets**
- **Stocker des valeurs de propriété**
- **Renvoyer des valeurs**
- **Afficher des informations en sortie**

Comparaison Variables / Constantes

Variables	Constantes
Déclaration à l'aide de l'expression Dim	Déclaration à l'aide de l'expression Const
Les valeurs changent lorsque l'application s'exécute	Les valeurs restent identiques lorsque l'application s'exécute
Utilisation de la mémoire plus importante qu'avec des constantes	Utilisation de la mémoire plus réduite qu'avec des variables

Syntaxe :

`Const nomDeLaConstante As Type`

Application pratique : Recherche de bogues



1 Dim 12Count As Integer

2 Dim Number For Double

3 Const Son's Birthday As Day

4 Dim Error.Message As Text

5 Dim \$CurrentExpenses With Decimal

Leçon : Portée des variables

Module ou classe Public
Public A As Integer

La variable *A* est accessible à partir de n'importe quel projet de la solution

Module ou classe Friend
Friend B As Date

La variable *B* est accessible n'importe où dans le projet

Module ou classe Private
Private c As String

La variable *c* est accessible n'importe où dans le module

Procédure ou bloc
Dim d As Integer

La variable *d* est accessible uniquement dans la procédure ou le bloc

Présentation de la notion de portée

Définition : La portée est l'ensemble de tout le code se référant à une variable par son nom

Facteurs qui affectent la portée

Endroit de déclaration de la variable

Bloc

Procédure

Module,
classe ou
structure

Niveau d'accès du conteneur de la variable

Private

Public

Niveau d'accès de la variable

Friend

Déclaration de variables locales

Endroit de déclaration	Mot clé	Modificateur d'accès	Portée
Bloc	Dim	Aucun	Niveau du bloc
Procédure	Dim	Aucun	Niveau de la procédure

Exemple de variable locale : au niveau du bloc

```
If x < > 0 Then  
    Dim blockNumber As Integer  
    blockNumber = x + 1  
End If
```

Exemple de variable locale : au niveau de la procédure

```
Sub ShowMessage_Click( )  
    Dim myVariable As String  
    ' Insérer du code pour ajouter des fonctionnalités  
End Sub
```

Déclaration de variables statiques

- **Endroit** : Déclarez les variables dans un bloc ou une procédure
- **Syntaxe** : Utilisez un mot clé statique (aucun modificateur d'accès)
 - *Static nomDeLaVariable As Type*
- **Exemple**

```
Sub AddItem_Click( )  
    Static items As Integer  
    ' Ajouter 1 au compteur  
    items += 1  
    MessageBox.Show (" Le compte est actuellement de " & items)  
End Sub
```


Déclaration de variables de module

- Déclarez la variable dans un module, une classe ou une structure

Modificateur d'accès	Portée
Private	Module
Friend	Projet
Public	Solution

- Exemples

```
Private myModuleMessage As String  
Friend myProjectMessage As String  
Public mySolutionMessage As String
```

Application pratique : Définition de niveaux d'accès aux variables



1 Examinez le code de démarrage pour rechercher une variable non déclarée

2 Déclarez la variable dans différents endroits pour obtenir différents niveaux de portée

3 Déterminez quel modificateur d'accès utiliser lors de la déclaration d'une variable

Leçon : Conversion des types de données

- **Présentation des fonctions de conversion**
- **Conversion explicite de types de données**
- **Mécanisme de conversion implicite de données**

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Présentation des fonctions de conversion

Définition : Les fonctions de conversion permettent de convertir explicitement une valeur d'un type de données en un autre

Valeur Integer
1234

Devient

CStr →

Valeur String
"1234"

Valeur Double
567,9894

CInt →

Valeur Integer
568

Valeur String
"12 février 1992"

CDate →

Valeur Date
#2/12/92#

Conversion explicite de types de données

**Syntaxe : *nomDeLaVariable* =
*FonctionDeConversion(Expression)***

Exemple

- 1 Déclarez une variable en tant que type de données String
`Dim myString As String`
- 2 Déclarez une autre variable en tant que type de données Integer
`Dim myInteger As Integer`
- 3 Affectez une valeur à la variable String
`myString = "1234"`
- 4 Convertissez la valeur String en valeur Integer
`myInteger = CInt(myString)`

Mécanisme de conversion implicite de données

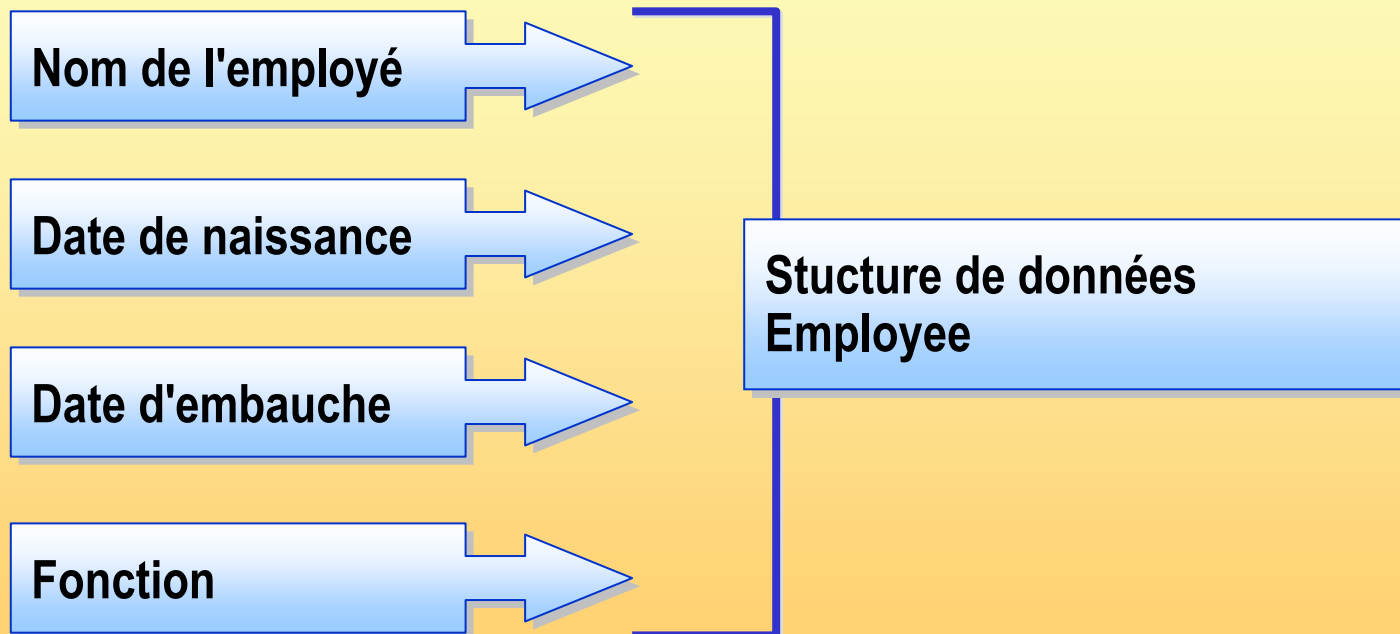
- Les types de données sont convertis automatiquement
- Aucune syntaxe spéciale n'est nécessaire dans le code
- Exemple de conversion implicite de données :

```
Dim sequence As String
Dim number As Integer
' ...
sequence = "1234"
number = sequence
' La valeur de la séquence est convertie implicitement
' en valeur Integer
```

- Inconvénients :
 - Susceptibles de produire des résultats inattendus
 - Exécution du code ralentie
- L'option **Strict** ne permet aucun type de conversion implicite restrictive

Leçon : Création et utilisation de structures

Informations connexes du groupe → structure unique



Présentation des structures

- Types de données composites
- Utilisées pour créer des types de valeurs définis par l'utilisateur
- Les membres peuvent être des variables, des propriétés, des méthodes ou des événements
- Exemple de structure définie par l'utilisateur :

```
Public Structure Employee  
    Public FirstName As String  
    Public LastName As String  
    Public HireDate As Date  
    Public JobTitle As String  
    Private Salary As Decimal  
End Structure
```

- Exemples de structures prédéfinies : *Point*, *Size* et *Color*

Déclaration de structures

- Dans un module, un fichier ou une classe (pas dans une procédure)
- Syntaxe :

```
ModificateurAccès Structure nomDeLaStructure  
    ' Déclarer les membres de la structure ici  
End Structure
```

- Le modificateur d'accès utilisé est :
 - **Public** pour un accès illimité
 - **Protected** pour un accès uniquement au sein de sa propre classe
 - **Friend** pour un accès à partir de n'importe quel emplacement de l'application ou de l'assembly
 - **Private** pour un accès uniquement au sein de son contexte de déclaration
- N'affectez pas de valeurs aux membres des données dans la déclaration

Utilisation de structures

Procédure

- 1** Déclaration d'une structure
- 2** Déclaration d'une variable du type de la structure déclarée
- 3** Affectation de valeurs aux données membres
- 4** Écriture du code permettant d'utiliser les membres de la structure

Application pratique : Création et utilisation de structures



- 1** Déclarez une structure
- 2** Déclarez une variable en tant que type de la structure
- 3** Affectez des valeurs aux membres de la structure
- 4** Écrivez le code pour utiliser les membres de la structure
- 5** Exécutez et testez l'application

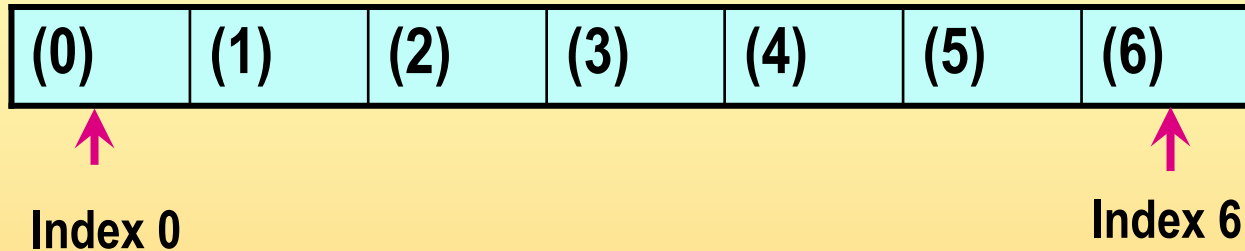
Leçon : Stockage de données dans des tableaux

- Présentation d'un tableau
- Déclaration d'un tableau unidimensionnel
- Utilisation de tableaux multidimensionnels
- Redimensionnement des tableaux

Présentation d'un tableau

- **Définition : Un tableau est une séquences d'éléments de données**

- Tous les éléments d'un tableau possèdent le même type de données
- Les éléments individuels sont accessibles en utilisant des index d'entiers



- **Exemple**

- Pour déclarer un tableau d'entiers composé de sept éléments :

```
Dim countHouses(6) As Integer
```

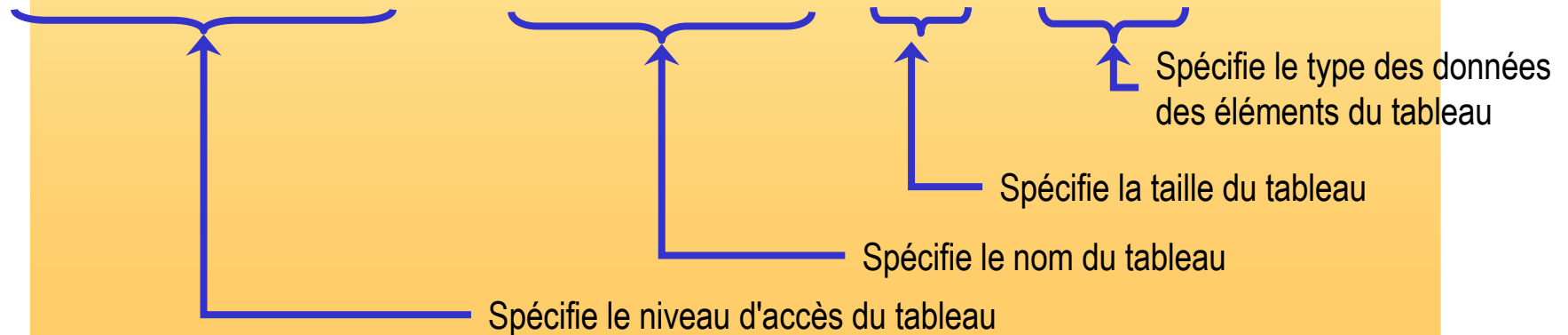
- Pour accéder au troisième élément du tableau :

```
TextBox1.Text = CStr(countHouses(2))
```

Déclaration d'un tableau unidimensionnel

- Vous déclarez un tableau en spécifiant les éléments suivants :
 - Nom du tableau
 - Taille (nombre d'éléments)
 - Type de données des éléments du tableau
 - Modificateur d'accès (si nécessaire)

ModificateurAccès NomDuTableau(Size) As Type



Utilisation des tableaux multidimensionnels

- Spécifiez tous les éléments et les dimensions
- Total des éléments = produit de toutes les tailles
- Pour déclarer une variable de tableau multidimensionnel :
 - Ajoutez une paire de parenthèses après le nom de la variable
 - Utilisez des virgules à l'intérieur des parenthèses pour séparer les dimensions
 - Utilisez l'instruction **Dim** ou un modificateur d'accès au début de la déclaration
- Exemple :

```
Public ThreeDimensions(3,9,14) As Double  
    ' Tableau tridimensionnel
```

Redimensionnement des tableaux

- Vous pouvez redimensionner un tableau à tout moment
- Utilisez l'instruction ReDim
- Syntaxe :

```
ReDim tableauExistant(NouvelleGrandeur)
```

- Exemple :

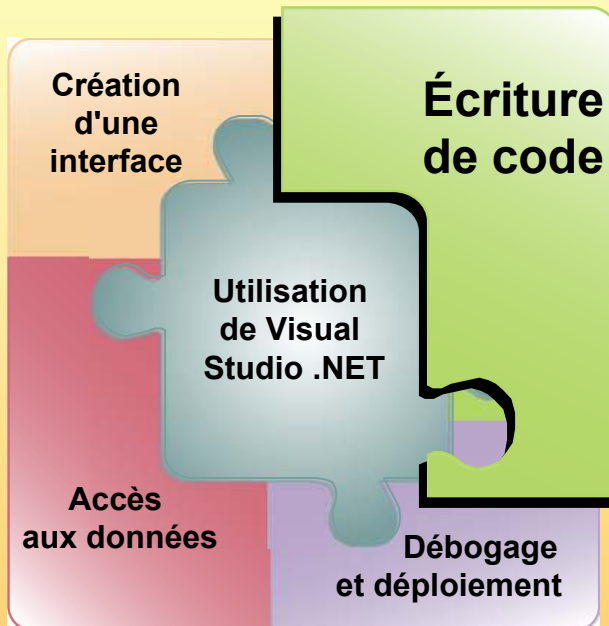
```
Dim myArray(,) ' Déclarer tableau  
ReDim myArray(3, 5) ' Redimensionner tableau
```


www.Mcours.com

Site N°1 des Cours et Exercices Email: contact@mcours.com

Module 4 : Utilisation de procédures

Vue d'ensemble



- **Création de procédures**
- **Utilisation de procédures**
- **Utilisation de fonctions prédéfinies**

Leçon : Création de procédures

- **Présentation des procédures**
- **Création de procédures Sub**
- **Création de procédures Fonction**
- **Déclaration d'arguments dans les procédures**
- **Utilisation d'arguments facultatifs**
- **Réutilisation du code**

Présentation des procédures

- Les procédures représentent les instructions du code exécutable dans un programme. Elles sont insérées entre une instruction de déclaration et une instruction End
- Il existe trois types de procédures :
 - Procédures **Sub**
(y compris les procédures d'événement **Sub**)
 - Procédures **Function**
 - Procédures **Property**
- Permettent la réutilisation du code
- Sont déclarées en tant que procédures publiques par défaut

Création de procédures Sub

Les procédures Sub effectuent des actions mais ne renvoient pas de valeur à la procédure appelante

```
[Modificateur d'accès] Sub NomDeLaProcédure _  
    [(Liste des arguments)]  
' Instructions de la procédure Sub  
End Sub
```

Exemple :

```
Private Sub AboutHelp( )  
    MessageBox.Show("MonProgramme V1.0", "Aide de  
    MonProgramme")  
End Sub
```

Création de procédures Function

Les procédures Function effectuent des actions et peuvent retourner une valeur au programme appelant

```
[Modificateur d'accès] Function NomDeLaFonction _  
  [(Liste des arguments)] As TypeDeDonnées  
  ' Instructions de la fonction, y compris l'instruction  
  ' Return facultative  
End Function
```

Exemple :

```
Public Function DoubleTheValue(ByVal J As Double) As _  
  Double  
  . . .  
  Return J*2  
  . . .  
End Function
```

Déclaration d'arguments dans les procédures

- Les *Arguments* sont les données transmises aux procédures
- Vous pouvez transmettre des arguments via *ByVal* ou *ByRef*
 - **ByVal** : La procédure ne peut pas modifier la valeur de la variable d'origine
 - **ByRef** : La procédure peut modifier la valeur de la variable d'origine
 - **Exception** : Les arguments non variables ne sont jamais modifiés dans le code appelant, même s'ils sont transmis par référence
- Dans Visual Basic .NET, la méthode de transmission par défaut est la transmission par valeur (ByVal)
- Syntaxe et exemple :

```
([ByVal | ByRef] nomArgument As TypeDeDonnées)
```

```
(ByVal Name As String)
```

Utilisation d'arguments facultatifs

- Règles de déclaration des arguments facultatifs :
 - Vous devez spécifier une valeur par défaut
 - La valeur par défaut doit être une expression de constante
 - Chaque argument qui suit un argument facultatif doit également être facultatif
- Syntaxe :

```
(Optional [ByVal|ByRef] nomArgument As _  
    TypeDeDonnées=valeurParDefaut)
```

- Exemple :

```
Function Add (ByVal value1 As Integer, ByVal value2 As _  
    Integer, Optional ByVal value3 As Integer = 0) As Integer
```


Réutilisation du code

Éléments	Fonction	Exemples
Structure	Crée des objets qui ne doivent pas être étendus	Size Point
Module	Fournit des fonctions d'utilitaire et des données globales	Conversion de températures
Classe	Étend des objets, ou pour les objets dont les ressources doivent être nettoyées	Forms Button

- **Création d'un module :**

```
[Public|Friend] Module NomDuModule  
    .  
    .  
    .  
End Module
```

Application pratique : Création d'une fonction dans un module



1 Ouvrez un nouveau projet

2 Ajoutez un nouveau module au projet

3 Créez une fonction dans le module

4 Écrivez le code de la fonction


Leçon : Utilisation de procédures

- Utilisation des procédures Sub
- Utilisation des procédures Function
- Transmission de tableaux à des procédures
- Création d'une procédure Sub Main

Utilisation des procédures Sub

```
Public Sub Hello(ByVal name As String)
    MessageBox.Show("Bonjour " & Name)
End Sub
```

```
Sub Test( )
    Hello("John")
End Sub
```



Utilisation des procédures Function

■ Appel d'une fonction

- Vous devez inclure le nom de la fonction à droite d'une instruction d'assignation

```
Dim celsiusTemperature As Single  
celsiusTemperature = FtoC(80)
```

- Vous devez utiliser le nom de la fonction dans une expression

```
If FtoC(userValue) < 0 Then  
    ...  
End If
```

Application pratique : Utilisation de la valeur de retour d'une fonction



1 Créez l'interface utilisateur

2 Écrivez le code de l'application

A screenshot of a Windows application window titled "Aire". The window has a blue title bar with standard minimize, maximize, and close buttons. The main content area is light gray and contains three text labels: "Hauteur", "Largeur", and "Aire". Each label is followed by a white text input field. At the bottom center of the window, there is a button labeled "Button1".

Transmission de tableaux à des procédures

- Transmettez un tableau comme vous transmettez d'autres arguments :

```
Sub PassArray(ByVal testScores As Integer( ))  
    ...  
End Sub  
  
Dim scores( ) As Integer = {80, 92, 73}  
PassArray(scores)
```

- Déclarez un tableau de paramètres :

```
Sub StudentScores(ByVal name As String, ByVal _  
    ParamArray scores( ) As String)  
    ' Instructions de la procédure Sub  
End Sub
```

- Appelez une procédure comportant un tableau de paramètres :

```
StudentScores("Anne", "10", "26", "32", "15", "22", "16")
```

Création d'une procédure Sub Main

- **Sub Main : Point de départ de votre application**
- **Application.Run : Démarre l'application**
- **Application.Exit : Quitte l'application**

Application pratique : Création d'une procédure Sub Main



- 1** Déclarez des variables au niveau du module
- 2** Créez une procédure Sub Main et définissez-la comme objet de démarrage
- 3** Écrivez le code correspondant au formulaire Selection
- 4** Écrivez le code pour quitter l'application
- 5** Testez l'application

Leçon : Utilisation de fonctions prédéfinies

- Utilisation de la fonction InputBox
- Utilisation des fonctions de date et d'heure
- Utilisation des fonctions de chaîne
- Utilisation des fonctions de format
- Utilisation des fonctions financières

www.Mcours.com

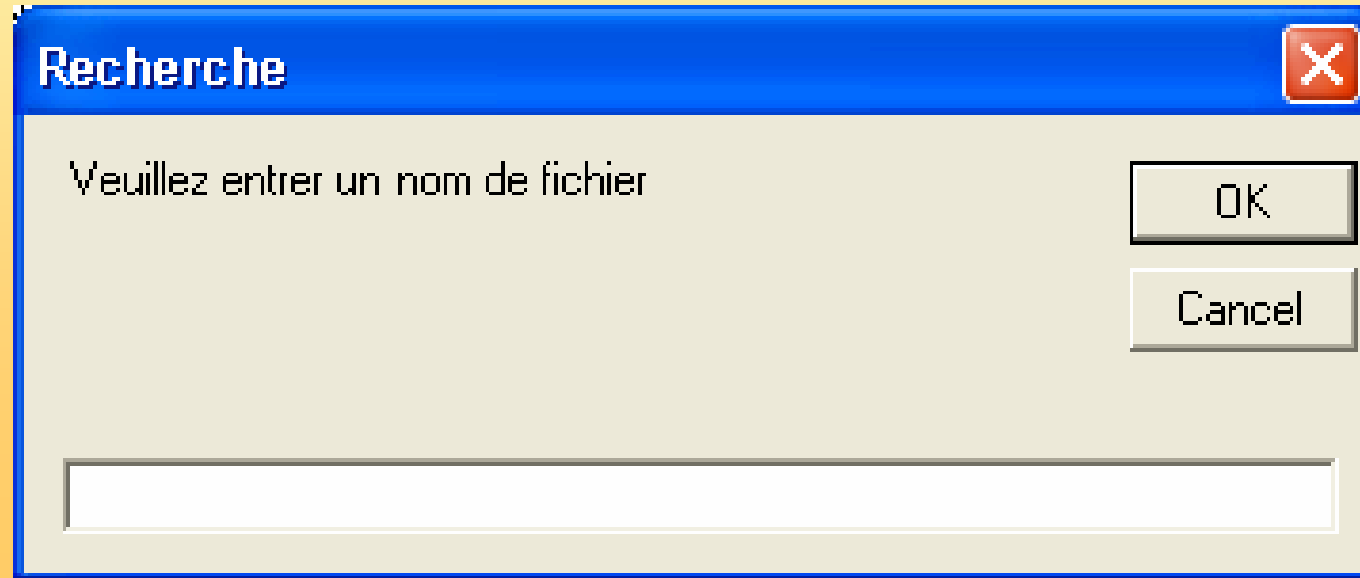
Site N°1 des Cours et Exercices Email: contact@mcours.com

Utilisation de la fonction InputBox

- Affiche une invite dans une boîte de dialogue, puis renvoie l'entrée utilisateur sous la forme d'une chaîne

```
Dim FileName As String
```

```
FileName = InputBox("Veuillez entrer un nom de  
fichier", "Recherche")
```



Utilisation des fonctions de date et d'heure

- Effectuent des calculs impliquant des dates et des heures

- Exemples :

- **DateAdd** : Ajoute ou retranche une durée donnée à une date

```
DateAdd(DateInterval.Day, 10, billDate)
```

- **DateDiff** : Détermine le nombre de durées spécifiées qui existent entre deux valeurs date/heure

```
DateDiff(DateInterval.Day, Now, secondDate)
```

Utilisation des fonctions de chaîne

- Extraient uniquement une partie d'une chaîne
- Renvoient des informations concernant une chaîne
- Affichent des informations dans un format donné
- Exemples :

- **Trim**

```
NewString = Trim(MyString)
```

- **Len**

```
Length = Len(customerName)
```

- **Left**

```
Microsoft.VisualBasic.Left(customerName, 5)
```

Utilisation des fonctions de format

- Mettent en forme les nombres, les dates et les heures selon les standards reconnus
- Affichent les différents formats sans qu'il soit nécessaire de modifier le code pour chaque nationalité ou pays
- Exemples :
 - **FormatCurrency**

```
FormatCurrency(amountOwed, , , TriState.True, TriState.True)
```

- **FormatDateTime**

```
FormatDateTime(myDate, DateFormat.LongDate)
```

Utilisation des fonctions financières

- Effectuent des calculs et réalisent des opérations liés aux finances, tels que les taux d'intérêt
- Exemples :
 - Pmt

```
payment = Pmt(0.0083, 24, -5000, 0, DueDate.BegOfPeriod)
```

- Rate

```
ratePerPeriod = Rate(24, 228, -5000, 0, DueDate.BegOfPeriod, _  
0.8)*100
```

Module 5 : Structures de décision et boucles

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Vue d'ensemble



- Utilisation d'expressions conditionnelles
- Utilisation de structures de décision
- Utilisation de structures de boucles conditionnelles

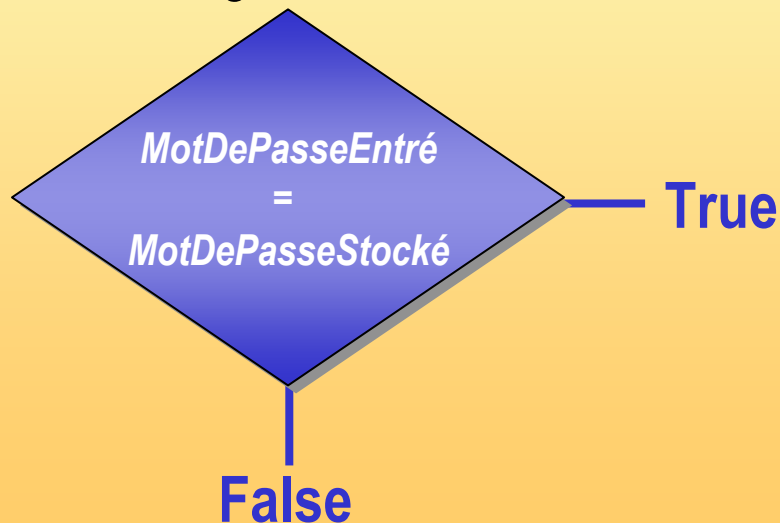
Leçon : Utilisation d'expressions conditionnelles

- **Présentation des expressions conditionnelles**
- **Utilisation d'opérateurs arithmétiques**
- **Utilisation d'opérateurs de comparaison**
- **Utilisation d'opérateurs logiques**
- **Association d'opérateurs logiques et d'opérateurs de comparaison**

Présentation des expressions conditionnelles

■ Expressions conditionnelles :

- Incluent une condition à tester **True** ou **False**
- Incluent un opérateur permettant de spécifier de quel test de la condition il s'agit



Si le mot de passe est correct, la condition est True

Utilisation d'opérateurs arithmétiques

- Symboles évaluant des expressions conditionnelles
- Peuvent effectuer des opérations arithmétiques
- Syntaxe :

expression1 opérateur arithmétique expression2

- Exemple :

```
Dim x As Integer  
x = 52 * 17  
x = 120 / 4  
x = 67 + 34  
x = 32 - 12  
x = 23 ^ 3
```

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Utilisation d'opérateurs de comparaison

- Symboles évaluant des expressions conditionnelles et renvoyant une valeur de type Boolean
- Peuvent comparer des nombres et des chaînes
- Syntaxe :

expression1 opérateur de comparaison expression2

- Exemple :

```
Dim Quantity As Integer  
Dim LargeOrder As Boolean  
LargeOrder = Quantity > 1000
```

Utilisation d'opérateurs logiques

- Les opérateurs logiques procèdent à une évaluation logique des expressions et renvoient une valeur de type Boolean
- Syntaxe :

expression1 opérateur logique *expression2*

- Exemple :

OrderedLastMonth **And** OrderDelivered

Association d'opérateurs logiques et d'opérateurs de comparaison

- Vous pouvez combiner des opérateurs de comparaison avec des opérateurs logiques à l'aide d'instructions conditionnelles
- Exemple :

Opérateurs de
comparaison

Opérateur logique

```
LateActiveCustomer = DaysOverDue >= 60 And ActiveCustomer
```

Application pratique : Évaluation d'expressions conditionnelles



- Utilisez l'application exemple pour calculer les résultats de ces expressions :

TestString = TestString	0 And 0
TestString = Teststring	-1 And 0
TestString < TestString	-1 And -1
Test < TestString	-1 Or -1
100 > 10	-1 Xor -1
10 < 10	-1 Xor 0
10 <= 10	0 Xor 0

Leçon : Utilisation de structures de décision

■ If...Then

```
If Ventes > 10000 Then
    Prime = 0.10 *
Ventes
End If
```

■ If...Then...Else

```
If Ventes > 10000 Then
    Prime = 0.10 * Ventes
Else
    Prime = 0
End If
```

■ If...Then...ElseIf

```
If Ventes > 10000 Then
    Prime = 0.10 * Ventes
ElseIf Ventes > 5000 Then
    Prime = 0.05 * Ventes
Else
    Prime = 0.02 * Ventes
End If
```

■ Select Case

```
Select Case Rank
    Case 1
        Prime = 0
    Case 2, 3
        Prime = 0.05 * Ventes
    Case 4 to 6
        Prime = 0.10 * Ventes
    Case Else
        Prime = 0.15 * Ventes
End Select
```

Utilisation d'instructions If...Then

- À utiliser pour une décision True ou False
- Si la condition est True, les instructions suivant l'instruction If sont exécutées
- Si la condition est False, les instructions suivant l'instruction If ne sont pas exécutées

```
If Ventes > 10000 Then  
    Prime = 0.10 * Ventes  
End If
```

Utilisation d'instructions If...Then...Else

- À utiliser avec une décision offrant au moins deux choix
- Chaque instruction If doit avoir une instruction End If correspondante
- Si la condition est True, les instructions suivant l'instruction If sont exécutées
- Si la condition est False, les instructions suivant l'instruction If ne sont pas exécutées

```
If Ventes > 10000 Then
    Prime = 0.10 * Ventes
Else
    Prime = 0
End If
```

Utilisation d'instructions If...Then...Elseif

- À utiliser pour imbriquer des instructions de décision
- Chaque instruction If doit avoir une instruction End If correspondante
- Les instructions Elseif n'ont pas leur propre End If
- Les instructions Elseif ne peuvent pas suivre Else
- Si la condition est True, les instructions suivant If sont exécutées

```
If Ventes > 10000 Then
    Prime = 0.10 * Ventes
Elseif Ventes > 5000 Then
    Prime = 0.05 * Ventes
Else
    Prime = 0.02 * Ventes
End If
```

Utilisation d'instructions Select Case

- Sélectionnent un bloc de code à exécuter à partir d'une liste de choix possibles
- À utiliser comme alternative aux instructions imbriquées et complexes If...Then...Else
- Si plusieurs instructions Case sont vraies, seules les instructions appartenant à la première instruction Case vraie sont exécutées

```
Select Case Rank
```

```
  Case 1
```

```
    Prime = 0
```

```
  Case 2, 3
```

```
    Prime = 0.05 * Ventes
```

```
  Case 4 to 6
```

```
    Prime = 0.10 * Ventes
```

```
  Case Else
```

```
    Prime = 0.15 * Ventes
```

```
End Select
```

Principes de sélection d'une structure de décision

- **Utiliser les instructions If...Then pour contrôler l'exécution d'un bloc de code unique**
- **Utiliser les instructions If...Then...Else pour contrôler l'exécution de deux sections de code s'excluant mutuellement**
- **Utiliser les instructions Select Case lorsque vous avez une liste de valeurs possibles à votre disposition**

Leçon : Utilisation de structures de boucles conditionnelles

- Utilisation d'instructions For...Next
- Utilisation d'instructions For Each...Next
- Utilisation d'instructions Do...Loop
- Utilisation de l'instruction Exit

Utilisation d'instructions For...Next

- À utiliser lorsque vous savez le nombre de fois que vous voulez répéter l'exécution du code

```
For NamePos = 0 to 4  
    MessageBox.Show(Names(NamePos))
```

```
Next
```

```
' Dans l'ordre inverse
```

```
For NamePos = 4 to 0 Step -1  
    MessageBox.Show(Names(NamePos))
```

```
Next
```


Utilisation d'instructions For Each...Next

- Une collection est un ensemble d'objets regroupés auquel on fait référence en tant qu'unité. Par exemple :
 - les éléments d'une zone de liste font partie d'une collection **Items**
 - chaque formulaire possède une collection **Controls** représentant tous les contrôles du formulaire
- Les instructions **For Each ... Next** sont utilisées pour effectuer des boucles parmi les éléments d'une collection

```
Sub LightBlueBackground (. . .)
    Dim ThisControl As System.Windows.Forms.Control
    For Each ThisControl In ThisForm.Controls
        ThisControl.BackColor = System.Drawing.Color.LightBlue
    Next ThisControl
End Sub
```

Utilisation d'instructions Do...Loop

■ Do...Loop Until

- Exécute le code dans la boucle, puis évalue la condition. Se répète jusqu'à ce que la condition soit déclarée **True**

■ Do Until...Loop

- Exécute le code dans la boucle uniquement si la condition est déclarée **False** ; elle se répète jusqu'à ce que l'expression en test soit déclarée **True**

■ Do...Loop While

- Exécute le code dans la boucle, puis évalue la condition. Se répète jusqu'à ce que la condition soit déclarée **False**

■ Do While...Loop

- Exécute le code dans la boucle uniquement si la condition est déclarée **True** ; elle se répète jusqu'à ce que l'expression en test soit déclarée **False**

Utilisation de l'instruction Exit

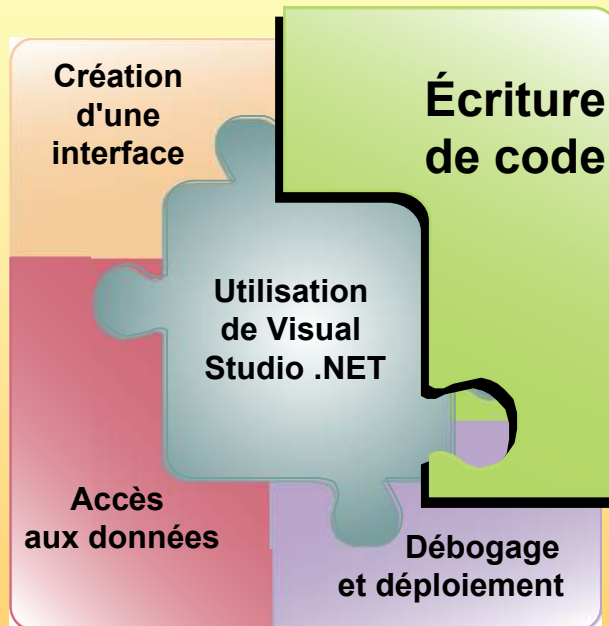
- À utiliser pour quitter les boucles Do ou For plus tôt lorsqu'une condition est remplie

```
Do Until y = -1
  If x < 0 Then Exit Do
  x = Sqrt (x)
  If y > 0 Then Exit Do
  y = y + 3
  If z = 0 Then Exit Do
  z = x / y
Loop
```

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Module 6 : Validation des entrées de l'utilisateur

Vue d'ensemble



- Limitation des entrées de l'utilisateur
- Validation des données de champs
- Validation des données de formulaires

Leçon : Limitation des entrées de l'utilisateur

- Directives pour la validation des entrées de l'utilisateur
- Présentation de la validation intrinsèque
- Utilisation des propriétés TextBox
- Utilisation du contrôle Masked Edit

Directives pour la validation des entrées de l'utilisateur

Directives

- **Empêcher les utilisateurs d'entrer des données non valides lorsque cela est possible**
- **Guider les utilisateurs tout au long du processus de saisie des données valides**
- **Permettre aux utilisateurs d'entrer leurs données quand et comme ils le souhaitent**
- **Penser aux implications liées à la validation lors de la conception de l'application**
- **Placer le code de validation à l'emplacement approprié selon le design de votre application**

Présentation de la validation intrinsèque

- **Définition : Propriétés et méthodes intégrées au contrôle que vous pouvez utiliser pour restreindre et valider les entrées de l'utilisateur**
- **Contrôles courants qui offrent une validation intrinsèque :**

Contrôle	Technique de validation
RadioButton	Restreint les entrées à On ou Off
CheckBox	Restreint les entrées à Checked ou Unchecked
CheckedListBox	Fournit une liste d'entrées valides
ListBox	Fournit une liste d'entrées valides (graphiques et texte)
DateTimePicker	Restreint les entrées à des dates ou à des heures
MonthCalendar	Restreint les entrées à une plage de dates
TextBox	Définit des propriétés pour restreindre ou modifier les entrées

Utilisation des propriétés TextBox

- Vous pouvez utiliser les propriétés ci-dessous pour restreindre ou modifier les entrées utilisateur des contrôles TextBox :

Contrôle	Technique de validation
PasswordChar	Cache ou masque les caractères entrés dans une zone de texte
MaxLength	Définit un nombre maximal de caractères à entrer dans une zone de texte
ReadOnly	Fournit une réponse prédéterminée valide
CharacterCasing	Définit tous les caractères d'une zone de texte en majuscules ou en minuscules

Utilisation du contrôle Masked Edit

Procédure

- 1** Ajout du contrôle Masked Edit à la boîte à outils

```
##| MaskedTextBox
```

- 2** Ajout d'un contrôle Masked Edit à un formulaire

- 3** Définition des propriétés du contrôle

- 4** Accès aux données entrées par l'utilisateur et mise en forme

Application pratique : Utilisation du contrôle Masked Edit



1 Ajoutez le contrôle Masked Edit à la boîte à outils

2 Ajoutez le contrôle Masked Edit à un formulaire

3 Modifiez la propriété Mask

4 Modifiez la propriété Format

5 Accédez aux données utilisateur et affichez-les

Leçon : Validation des données de champs

- Utilisation de fonctions booléennes
- Utilisation du composant `ErrorProvider`
- Réglage du focus sur les contrôles et le texte
- Modification des entrées de l'utilisateur
- Utilisation des événements de validation

Utilisation de fonctions booléennes

Fonctions booléennes courantes

Fonction	Description
IsNumeric	Renvoie une valeur booléenne qui indique si une expression est reconnue en tant que nombre
IsDate	Renvoie une valeur booléenne qui indique si une expression évalue une date valide

Exemple

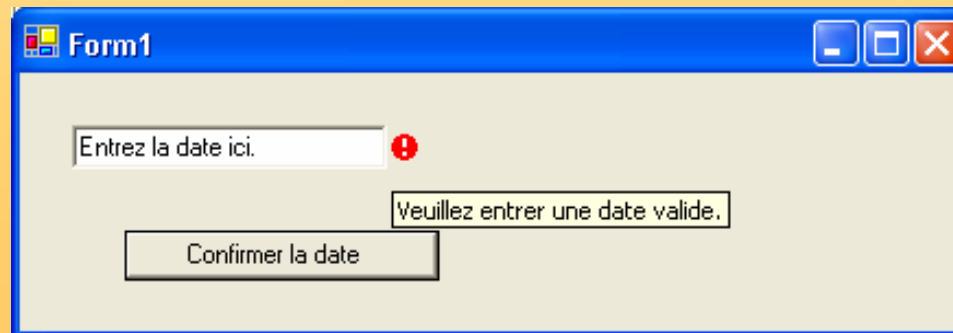
```
If IsNumeric(TextBox1.Text) Then  
    MessageBox.Show("La zone de texte contient un  
    nombre.")  
End If
```

Utilisation du composant ErrorProvider

- Ajout d'un composant ErrorProvider à un formulaire
 - Disponible sous l'onglet **Windows Forms** de la boîte à outils
- Appel de la méthode **setError**
 - Le premier paramètre spécifie l'emplacement de l'icône et le second, le message d'erreur à afficher :

```
ErrorProvider1.SetError (Textbox1, "Veuillez entrer une date valide.")
```

- Si l'utilisateur entre des données non valides, une icône d'erreur apparaît sur le formulaire :



Réglage du focus sur les contrôles et le texte

■ Pour quelles raisons définir le focus ?

- Lorsqu'un contrôle possède le focus, l'utilisateur peut y entrer des données à l'aide de la souris ou du clavier
- Lorsqu'un utilisateur entre des données non valides, vous pouvez conserver le focus sur le contrôle approprié jusqu'à ce que l'erreur soit corrigée

■ Exemples

- Pour définir le focus sur un contrôle **TextBox**, utilisez la méthode **Focus** :

```
TextBox1.Focus( )
```

- Pour sélectionner tout le texte d'un contrôle, utilisez **SelectAll** :

```
TextBox1.SelectAll( )
```

Modification des entrées de l'utilisateur

- Vous pouvez modifier des entrées de l'utilisateur à l'aide des fonctions suivantes :

Fonction	Description
UCase	Convertit une chaîne spécifiée en majuscules
LCase	Convertit une chaîne spécifiée en minuscules
Trim	Élimine les espaces à gauche et à droite dans une chaîne spécifiée

- Exemple

```
Dim LowerCase, UpperCase As String
LowerCase = " Bonjour le monde 1234" ' Chaîne à convertir
UpperCase = UCase(LowerCase) ' Retourne "BONJOUR LE MONDE
1234"
```


Utilisation des événements de validation

- Utilisez la propriété `CausesValidation` pour déclencher l'événement `Validating`
- Événement `Validating`

```
Private Sub WarehouseTextbox_Validating(. . .)
    If WarehouseTextbox.Text = "" Then
        infoErrorProvider.SetError(WarehouseTextbox, _
            "Veuillez entrer un emplacement pour
            l'entrepôt.")
        e.Cancel = True
    End If
End Sub
```

- Événement `Validated`

```
Private Sub WarehouseTextbox_Validated(. . .)
    infoErrorProvider.SetError(WarehouseTextbox, "")
End Sub
```

Application pratique : Validation des données de champs



1 Écrivez du code pour l'événement Validating afin de tester les données

2 Utilisez le composant ErrorProvider pour avertir l'utilisateur d'une erreur

3 Écrivez du code pour l'événement Validated afin de réinitialiser le composant ErrorProvider

4 Testez l'application

Leçon : Validation des données de formulaires

- Validation de plusieurs champs d'un formulaire
- Désignation des boutons d'acceptation et d'annulation
- Questions sur la sécurité

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Validation de plusieurs champs d'un formulaire

- Repères visuels destinés à l'utilisateur

- Exemple

Désactivation du bouton **OK** jusqu'à ce que l'utilisateur ait entré des données dans tous les champs requis

- Validation de tous les champs du formulaire en même temps

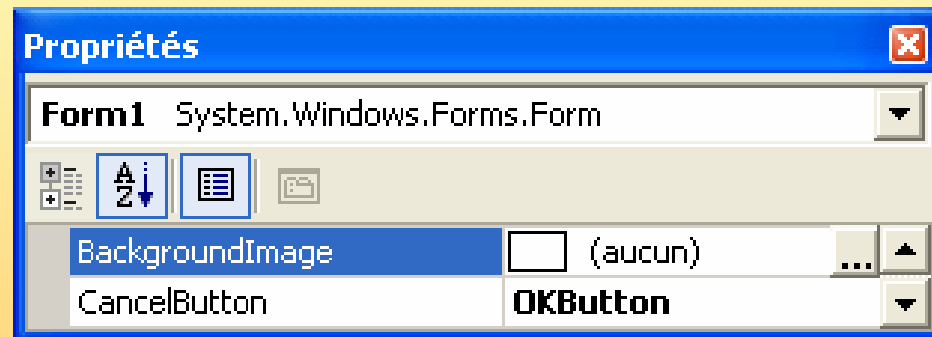
- Exemple

Placement de tout le code de validation dans le gestionnaire d'événements **Click** du bouton **OK**

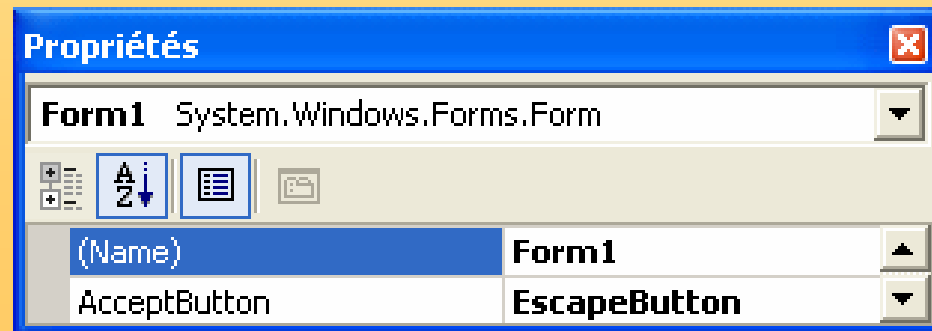
Désignation des boutons d'acceptation et d'annulation

- Définissez la propriété **AcceptButton** du formulaire
- Définissez la propriété **CancelButton** du formulaire
- Exemples

- Désignation du bouton **OKButton** en tant que bouton d'acceptation



- Désignation du bouton **EscapeButton** en tant que bouton d'annulation



Questions sur la sécurité

- **Authentification des utilisateurs**
- **Vérification de l'utilisateur Windows actuel**
 - Utilisez la propriété **UserName** de l'objet **SystemInformation**
 - Exemple

```
MessageBox.Show(" Vous êtes " & SystemInformation.UserName)
```

- **Sécurisation de votre code**

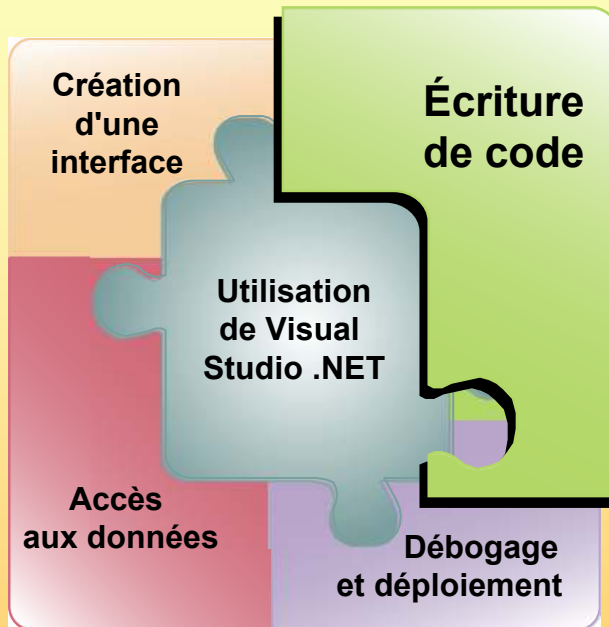
Module 7 :

Programmation orientée

objet en

Visual Basic .NET

Vue d'ensemble



- **Compréhension des classes**
- **Utilisation des classes**
- **Utilisation des membres partagés**
- **Héritage, polymorphisme et espaces de noms**

Leçon : Compréhension des classes

- abstraction
- classe
- encapsulation
- objet

www.Mcours.com

Site N°1 des Cours et Exercices Email: contact@mcours.com

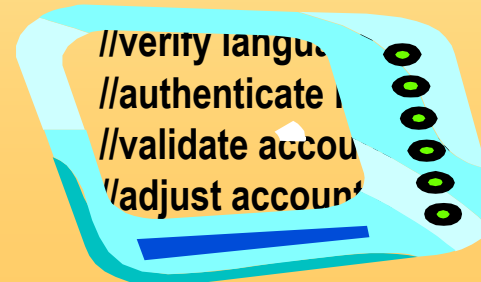
Présentation d'une classe

- Une *classe* est un plan qui décrit un objet et qui définit des attributs et des opérations pour celui-ci
- Les classes utilisent l'*abstraction* pour rendre uniquement disponibles les objets essentiels à la définition de l'objet
- Les classes utilisent l'*encapsulation* pour appliquer une abstraction

Ce que l'utilisateur voit :

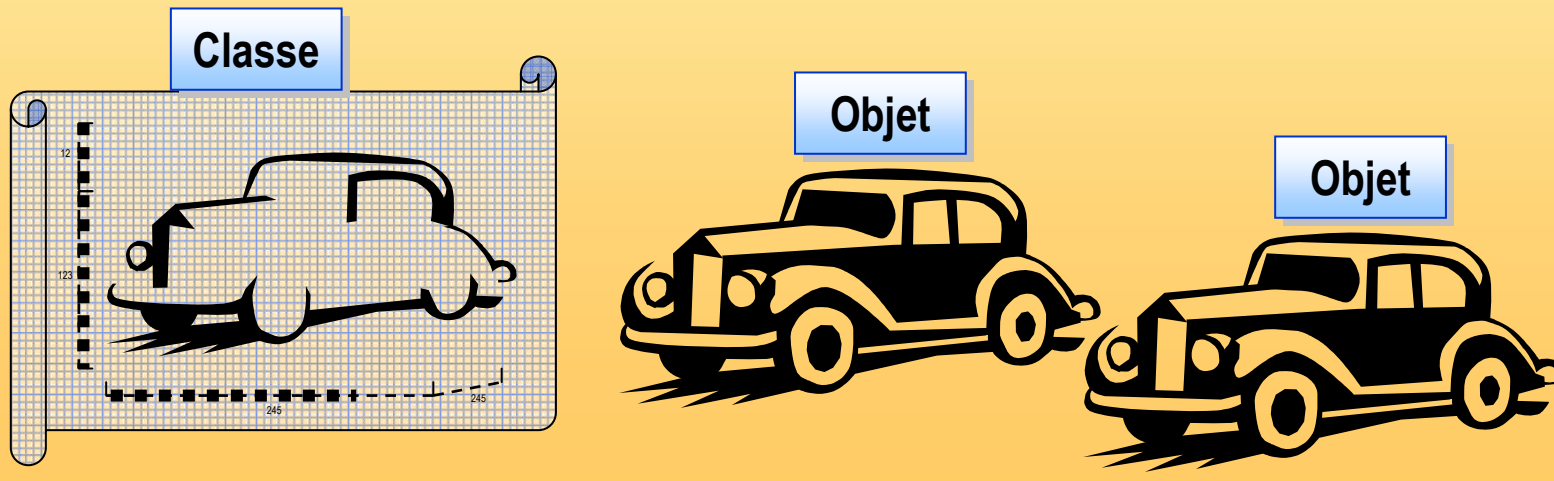


Ce qui est encapsulé :



Présentation d'un objet

- Un objet est une instance spécifique d'une classe
- Les objets possèdent les caractéristiques suivantes :
 - Identité : Les objets peuvent être distingués les uns des autres
 - Comportement : Les objets peuvent effectuer des tâches
 - État : Les objets stockent des informations qui peuvent changer au fil du temps



Utilisation de l'Explorateur d'objets

The screenshot displays the Visual Studio Object Explorer window. The title bar reads "Explorateur d'objets". The breadcrumb path is "Parcourir : Composants sélectionnés". The window is divided into three main sections:

- Objets:** A tree view showing the project structure. The "AccessibleObject" class is selected and highlighted with a blue box. A callout box labeled "Volet Objets" points to this tree view.
- Membres de 'AccessibleObject':** A list of methods and properties for the selected class, including `DoDefaultAction()`, `GetChildCount()`, `GetChild(Integer)`, `GetFocused()`, `GetHelpTopic(String)`, `GetSelected()`, `HitTest(Integer, Integer)`, `Navigate(System.Windows.Forms.Accessit`, and `New()`. A callout box labeled "Volet Membres" points to this list.
- Class Description:** A text area showing the class signature: `<System.Runtime.InteropServices.ComVisibleAttribute(True)>`
`Public Class AccessibleObject`
Inherits `System.MarshalByRefObject`
Membre de `System.Windows.Forms`
Summary:
Fournit des informations utilisées par les applications d'accessibilité pour adapter l'interface utilisateur aux personnes handicapées. A callout box labeled "Volet Description" points to this section.

Leçon : Utilisation des classes

- **Création d'une classe**
- **Ajout de données membres d'instance**
- **Ajout de méthodes**
- **Ajout de propriétés**
- **Création d'une instance d'une classe**
- **Utilisation des constructeurs**
- **Utilisation des destructeurs**

Création d'une classe

- Créez une classe à l'aide de la commande Ajouter une classe du menu Projet
- Exemple de classe appelée BankAccount :

```
Public Class BankAccount  
  
End Class
```

Ajout de données membres d'instance

- Ajout d'une donnée membre appelée *balance*

```
Public Class BankAccount
  Private balance As Double
End Class
```

Mot clé	Définition
Public	Accessible à partir de n'importe quel emplacement
Private	Accessible uniquement au sein du type
Protected	Accessible uniquement par les classes qui héritent de la classe

Ajout de méthodes

- Ajout d'une méthode appelée Deposit

```
Public Class BankAccount  
  
    Private balance As Double  
  
    Public Sub Deposit(ByVal amount As Double)  
        balance +=amount  
    End Sub  
  
End Class
```

- Méthodes surchargées : deux ou plusieurs méthodes portant le même nom, mais des signatures différentes
Exemple : `MessageBox.Show`

Ajout de propriétés

- Ajout d'une propriété :

```
Public Class BankAccount
    Private customerName As String

    Public Property Name( ) As String
        Get
            Return customerName
        End Get
        Set(ByVal Value As String)
            customerName = Value
        End Set
    End Property

End Class
```

Création d'une instance d'une classe

- Utilisation du mot clé **New** pour créer une instance de la classe **BankAccount** :

```
Module Bank  
  
  Sub Main  
    Dim account As New BankAccount( )  
    account.Deposit(500.00)  
  End Sub  
  
End Module
```

Application pratique : Création d'une classe

- Au cours de cette application pratique, vous allez créer une classe **BankAccount** et y ajouter des méthodes et des propriétés



www.Mcours.com

Site N°1 des Cours et Exercices Email: contact@mcours.com

Utilisation des constructeurs

- Exécute le code lorsque l'objet est instancié

```
Public Sub New( )  
    ' Effectue une initialisation simple  
    intValue = 1  
End Sub
```

- Peut surcharger, mais n'utilise pas le mot clé **Overloads**

```
Public Sub New(ByVal i As Integer)  
    ' Surchargé sans le mot clé Overloads  
    ' Effectue une initialisation plus complexe  
    value = i  
End Sub
```

Utilisation des destructeurs

- Utilisés pour le nettoyage des ressources
- Appelés au moment de l'exécution avant la destruction de l'objet
 - Important : La destruction peut ne pas avoir lieu immédiatement

```
Protected Overrides Sub Finalize( )
```

```
    ' Peut fermer des connexions ou d'autres ressources  
    conn.Close
```

```
End Sub
```

Leçon : Utilisation des membres partagés

- Utilisation des données membres partagées
- Utilisation des méthodes partagées

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Utilisation des données membres partagées

- Les données membres partagées permettent à plusieurs instances d'une classe de faire référence à une même variable de niveau classe

```
Class SavingsAccount
  Public Shared InterestRate As Double
  Public Name As String, Balance As Double
  . . .
End Class
```

```
SavingsAccount.InterestRate = 0.03
```

Utilisation des méthodes partagées

- **Peuvent être utilisées sans déclarer une instance de classe**
- **Ne peuvent accéder qu'aux données partagées**

```
'Code de classe de test  
Public Shared Function GetComputerName( ) As String  
...  
End Function
```

```
'Code client  
MessageBox.Show(TestClass.GetComputerName( ))
```


Leçon : Héritage, polymorphisme et espaces de noms

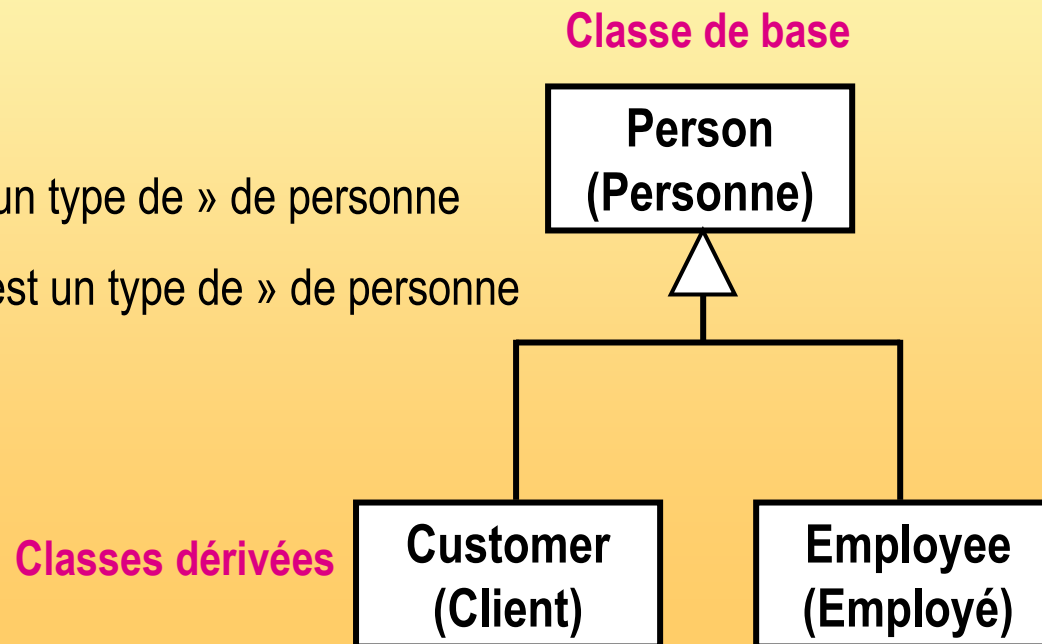
- héritage
- polymorphisme
- structures et classes
- espaces de noms

Présentation de l'héritage

- L'héritage spécifie une relation « de type »
- Plusieurs classes partagent les mêmes attributs et opérations, ce qui permet une réutilisation efficace du code

- Exemples :

- Un client « est un type de » de personne
- Un employé « est un type de » de personne

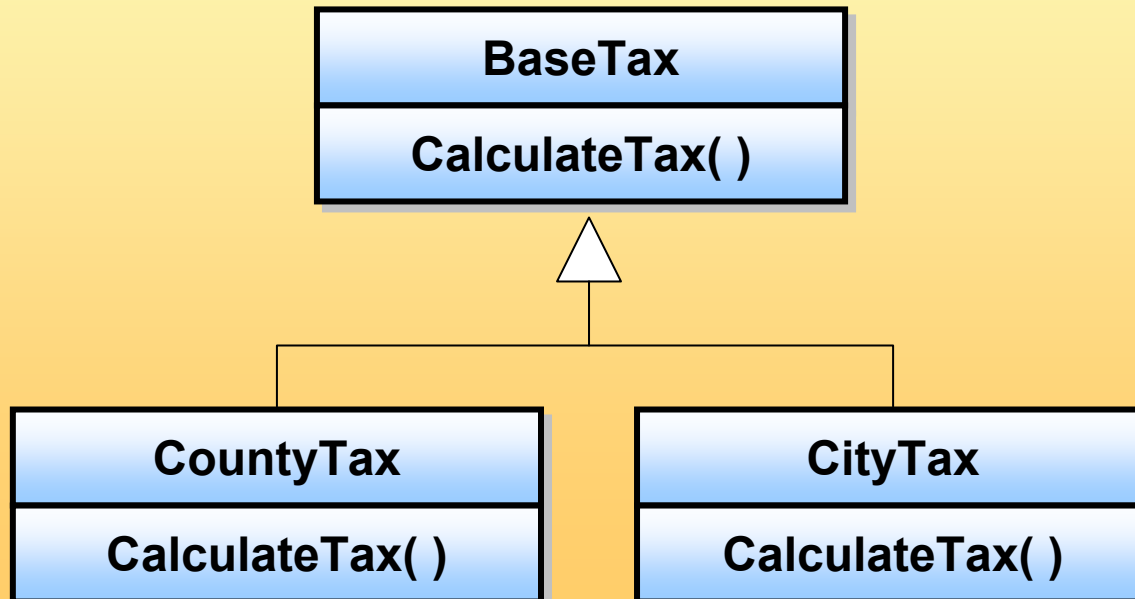


Héritage d'une classe

- Une classe dérivée hérite d'une classe de base
- Les propriétés, méthodes, données membres, événements et gestionnaires d'événements peuvent être hérités (selon la portée)
- Mots clés
 - **Inherits** : Hérite de la classe de base
 - **NotInheritable** : Ne peut pas être héritée
 - **MustInherit** : Les instances de la classe ne peuvent pas être créées ; doit être héritée comme une classe de base

Présentation du polymorphisme

- Le nom de la méthode réside dans la classe de base
- Les implémentations de la méthode résident dans les classes dérivées



Comparaison entre classes et structures

Classes	Structures
Peuvent définir des données membres, des propriétés et des méthodes	Peuvent définir des donnée membres, des propriétés et des méthodes
Prennent en charge les constructeurs et l'initialisation des membres	Aucun constructeur par défaut et aucune initialisation des membres
Prennent en charge la méthode Finalize	Ne prennent pas en charge la méthode Finalize
Extensibles par héritage	Ne prennent pas en charge l'héritage
Type de données référence	Type de données valeur

Organisation des classes en espaces de noms

- Les espaces de noms servent de système d'organisation
- Les espaces de noms fournissent des noms qualifiés complets pour les classes
 - Exemple : **System.Windows.Forms.Button**
- Pour importer un espace de noms :
 - Au niveau du projet, vous devez inclure une référence à la DLL qui contient l'espace de noms
 - Vous devez utiliser le mot clé **Imports**

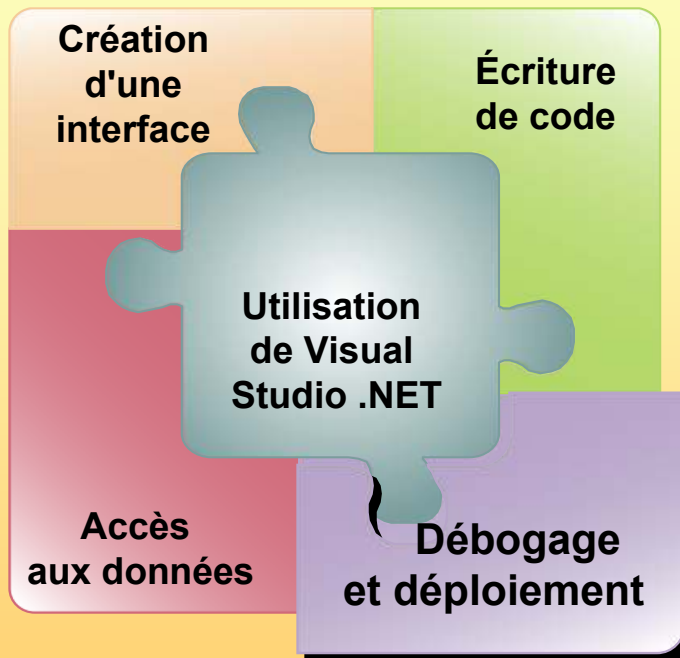
www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Module 8 :

Gestion des erreurs

et des exceptions

Vue d'ensemble



- Types d'erreurs
- Utilisation du débogueur
- Gestion des exceptions

Leçon : Types d'erreurs

- Présentation des erreurs de syntaxe
- Présentation des erreurs d'exécution
- Présentation des erreurs de logique

Présentation des erreurs de syntaxe



```
Private Sub PerformAction()  
    Dim newValue As XYZ  
    MessageBox.Show("Test")  
End Sub  
End Class
```


Erreur de syntaxe

Erreur de syntaxe

Le nom 'MessageBox' n'est pas déclaré.

Affichage des erreurs dans la fenêtre Liste des tâches :

Liste des tâches - 2 tâche(s) Erreur de génération affichée(s) (filtrée(s))				
!	<input checked="" type="checkbox"/>	Description	Fichier	Ligne
Cliquez ici pour ajouter une nouvelle tâche				
!		Le nom 'MessageBox' n'est pas déclaré.	C:\Program ...\Form1.vb	59
!		Type 'XYZ' non défini.	C:\Program ...\Form1.vb	58

Liste des tâches |  Sortie

Présentation des erreurs d'exécution

Vitesse = Kilomètres/Heures

- ' Si la variable *Heures* = 0, la division n'est pas une opération valide, même si l'instruction elle-même est syntaxiquement correcte

Microsoft Development Environment



Une exception non gérée du type 'System.OverflowException' s'est produite dans RuntimeError.exe.

Informations supplémentaires : L'opération arithmétique a provoqué un dépassement de capacité.

Arrêter

Continuer

Ignorer

Aide

Présentation des erreurs de logique

- **Définition : Une erreur qui entraîne la production de résultats incorrects par une application**
 - Peut ne pas générer de message d'erreur
 - Peut être recherchée en testant l'application et en analysant ses résultats

```
Dim x As Integer = 2  
Do While x < 10  
    ' Instructions du code  
    x -= 1  
Loop
```

Leçon : Utilisation du débogueur

- **Présentation du mode arrêt**
- **Utilisation des points d'arrêt**
- **Modification des points d'arrêt**
- **Barre d'outils Débogueur**
- **Procédure d'analyse pas à pas du code**
- **Utilisation des fenêtres de débogage**
- **Utilisation de la fenêtre Commande**

Présentation du mode arrêt

- **Permet d'interrompre le fonctionnement d'une application**
- **Vous pouvez effectuer les opérations suivantes lorsque vous êtes en mode arrêt :**
 - Parcourir votre code ligne après ligne
 - Déterminer les procédures actives qui ont été appelées
 - Contrôler les valeurs des variables, des propriétés et des expressions
 - Utiliser les fenêtres de débogage pour modifier les valeurs des variables et des propriétés
 - Modifier le flux du programme
 - Exécuter les instructions du code

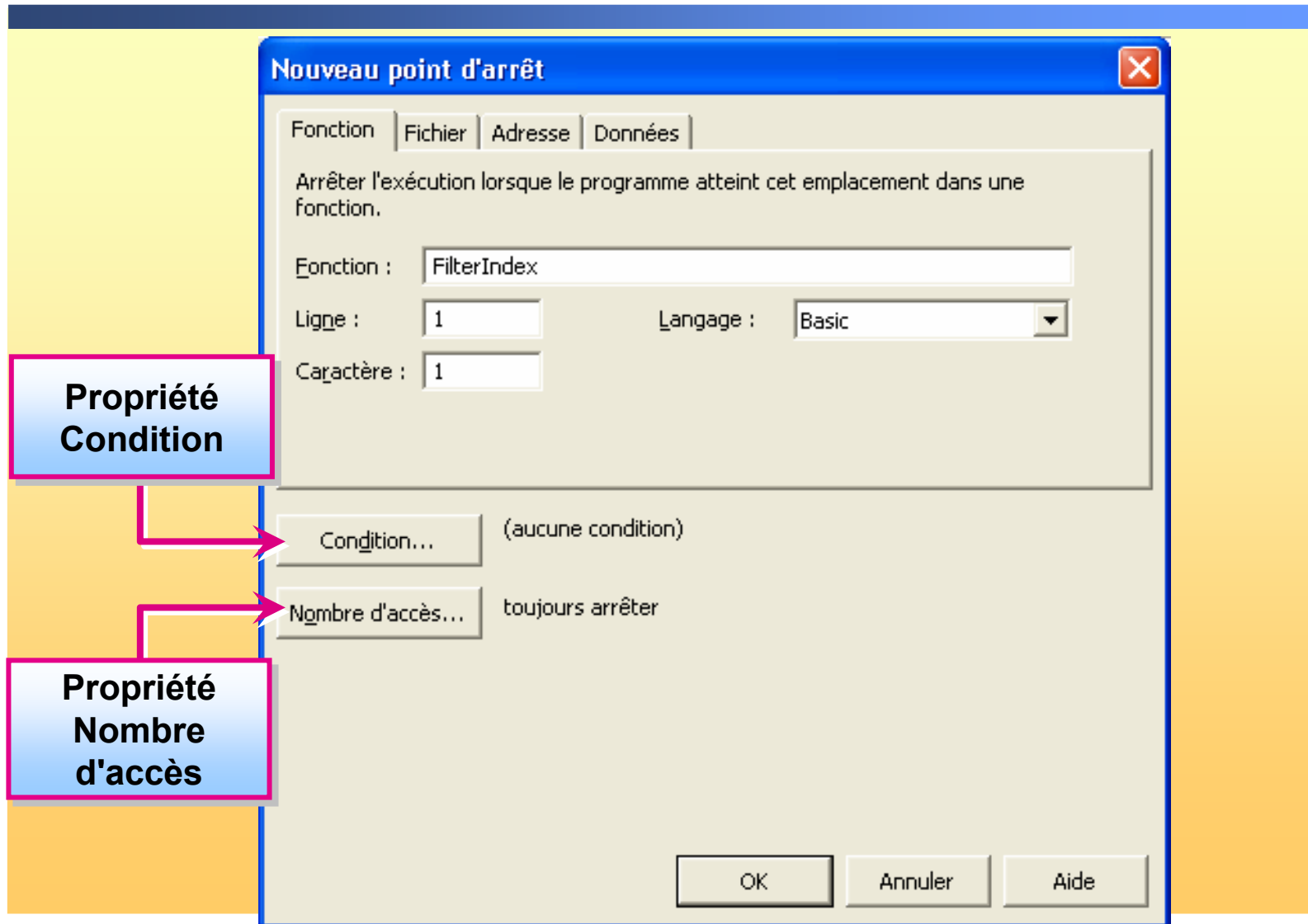
Utilisation des points d'arrêt

- Un point d'arrêt est un marqueur intégré à votre code qui permet à Visual Basic d'interrompre l'exécution du code sur une ligne spécifique
- Vous ne pouvez pas placer un point d'arrêt dans du code non exécutable

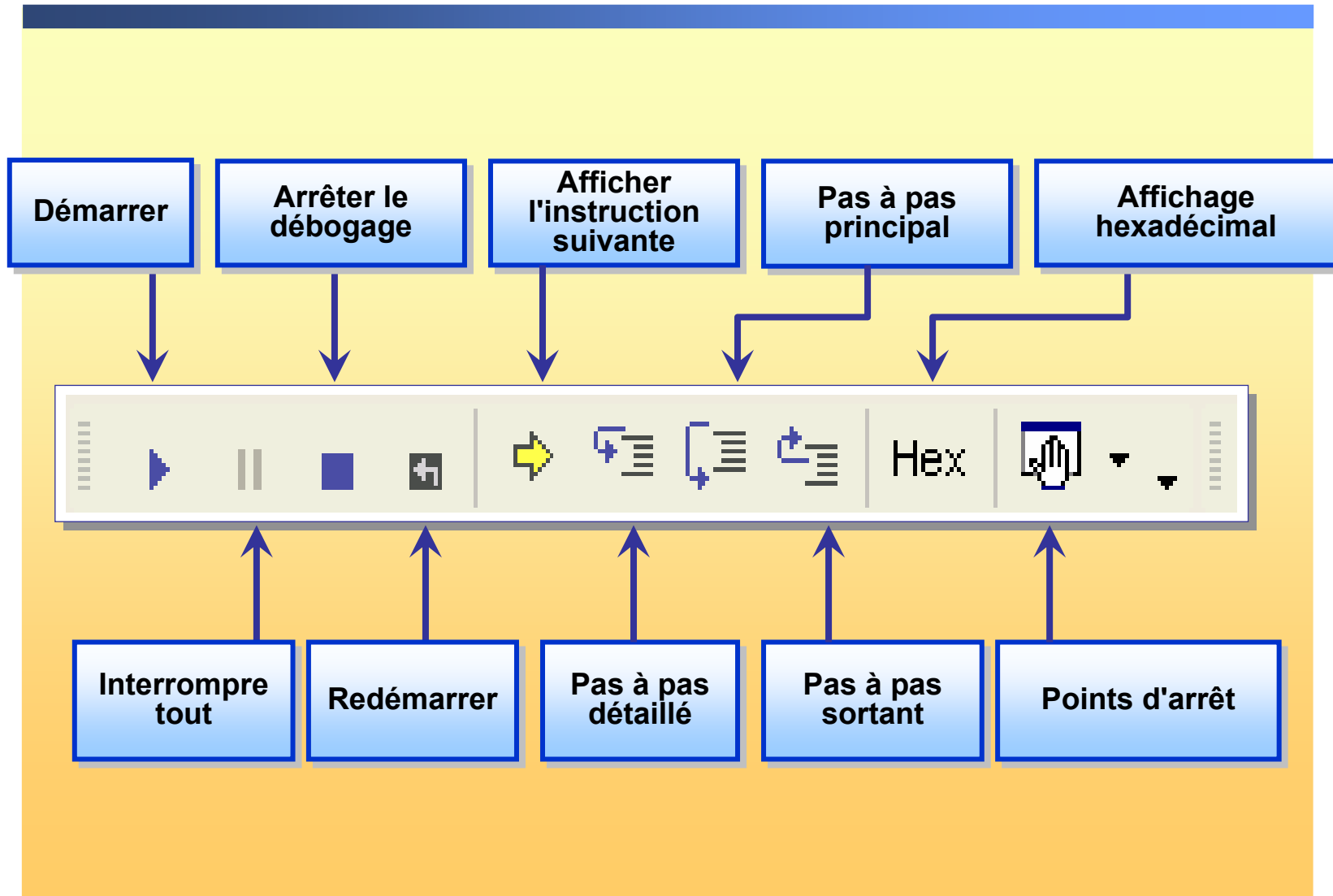
Points d'arrêt

```
Private Sub Button1_Click(ByVal sender As System.Object,  
    Dim maxLen As Integer = 0  
    Dim longestName As String = ""  
    Dim index As Integer  
  
    For index = 0 To names.GetLength(0)  
        If names(index).Length > maxLen Then  
            longestName = names(index)  
        End If  
    Next  
    MsgBox("Le prénom le plus long est " & longestName)  
End Sub  
End Class
```


Modification des points d'arrêt



Barre d'outils Déboguer



Procédure d'analyse pas à pas du code

- **Pas à pas détaillé et Pas à pas principal : Exécutent la ligne de code suivante. Si la ligne contient un appel de procédure :**
 - **Pas à pas détaillé :** Exécute uniquement l'appel, puis s'arrête à la première ligne de code dans la procédure
 - **Pas à pas principal :** Exécute l'intégralité de la procédure, puis s'arrête à la première ligne de code hors procédure
- **Pas à pas sortant :** Reprend l'exécution de votre code jusqu'au retour de la procédure, puis s'arrête au point de retour dans la procédure appelante
- **Exécuter jusqu'au curseur :** Le débogueur exécute votre application jusqu'à ce qu'il atteigne le point d'insertion que vous avez défini

Utilisation des fenêtres de débogage

Fenêtre	Utilisation
Automatique	Permet d'afficher des variables dans l'instruction en cours et dans les trois instructions suivant et précédant l'instruction en cours
Pile des appels	Permet d'afficher l'historique des appels à la ligne de code en cours de débogage
Variables locales	Permet d'afficher et de modifier les variables locales
Espion	<ul style="list-style-type: none">▪ Permet de créer votre liste personnalisée de variables ou d'expressions à contrôler▪ Permet de visualiser et manipuler toutes les expressions espionnes

Utilisation de la fenêtre Commande

- **Utilisez la fenêtre Commande pour :**
 - Émettre des commandes (mode Commande)
 - Déboguer et évaluer des expressions (mode Immédiat)

Tâche	Solution	Exemple
Évaluation des expressions	Faites précéder l'expression d'un point d'interrogation (?)	?myVariable
Passage du mode Commande en mode Immédiat	Tapez immed dans la fenêtre sans le signe Supérieur à (>)	immed
Passage du mode Immédiat en mode Commande	Tapez >cmd dans la fenêtre	>cmd
Passage provisoire en mode Commande lors d'un travail en mode Immédiat	Tapez la commande dans la fenêtre et faites-la précéder d'un signe Supérieur à (>)	>alias

Application pratique : Débogage du code



1 Examinez le code dans le gestionnaire d'événements Click

2 Générez et exécutez l'application

3 Utilisez les outils de débogage pour localiser l'erreur de logique

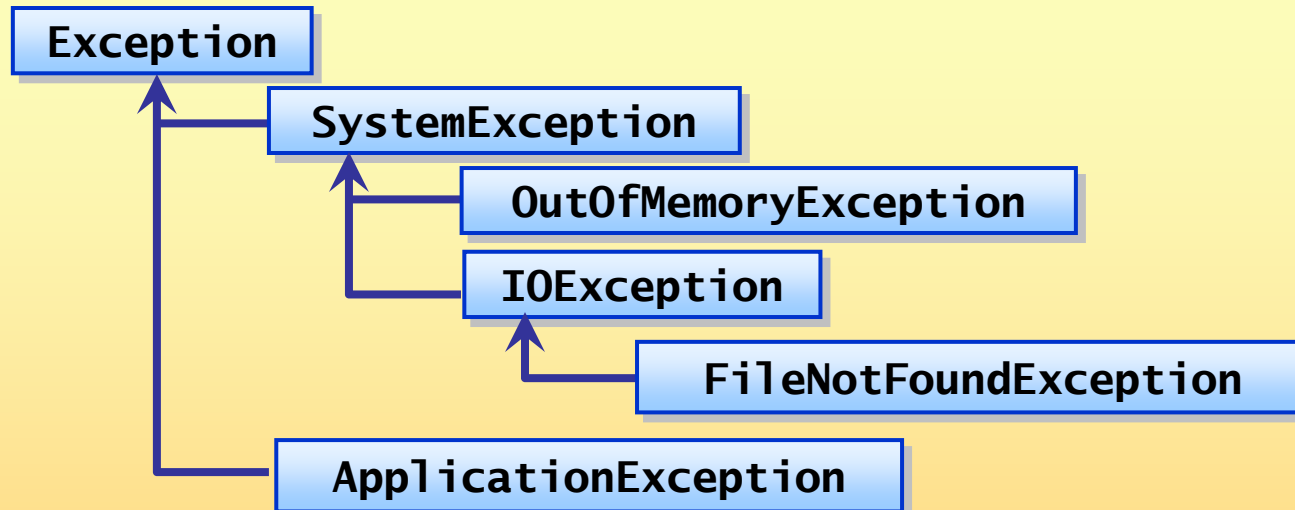
4 Suggérez une méthode pour corriger l'erreur

Leçon : Gestion des exceptions

- **Classe Exception**
- **Présentation de la gestion structurée des exceptions**
- **Utilisation de l'instruction Try...Catch**
- **Utilisation du bloc Finally**
- **Procédure de déclenchement des exceptions**
- **Principes d'utilisation de la gestion structurée des exceptions**

Classe Exception

- .NET Framework fournit le modèle objet d'exception suivant :



- Les classes d'exception vous permettent d'extraire des informations sur toutes les exceptions rencontrées
- Les propriétés de la classe de base **Exception** vous permettent d'analyser les exceptions
 - Propriétés principales : **StackTrace**, **Message**, **HelpLink** et **Source**

Présentation de la gestion structurée des exceptions

- **Détecte et répond à des erreurs lors de l'exécution d'une application**
- **Utilisez l'instruction Try...Catch...Finally pour encapsuler et protéger des blocs de code pouvant potentiellement provoquer des erreurs**
 - Chaque bloc possède un ou plusieurs gestionnaires associés
 - Chaque gestionnaire spécifie une forme de condition de filtre sur le type d'exception qu'il gère
- **Avantages :**
 - Permet de séparer le code de gestion des erreurs du code de logique
 - Facilite la lecture, le débogage et la gestion du code

Utilisation de l'instruction Try...Catch

- Placez les sections du code susceptibles de lever des exceptions dans un bloc Try
- Gérez les exceptions dans un bloc Catch distinct

Try

```
fs = New FileStream("data.txt", _  
    FileMode.Open)
```

← Logique du programme

Catch ex As FileNotFoundException

```
    MessageBox.Show("Fichier introuvable")
```

← Gestion des exceptions

Catch ex As Exception

```
    MessageBox.Show(ex.Message)
```

End Try

Utilisation du bloc Finally

- Bloc de code facultatif qui, s'il est inclu, est toujours exécuté
- Placez dans le bloc Finally le code de nettoyage, tel que le code pour la fermeture de fichiers

Try

```
fs = New FileStream("data.txt", FileMode.Open)
```

Catch ex As FileNotFoundException

```
MessageBox.Show("Fichier de données manquant")
```

Catch ex As Exception

```
MessageBox.Show(ex.Message)
```

Finally

```
If Not (fs Is Nothing) Then fs.Close( )
```

End Try

Procédure de déclenchement des exceptions

- L'instruction **Throw** crée une exception que vous pouvez gérer à l'aide du code de gestion structurée des exceptions

```
If (day < 1) Or (day > 365) Then  
    Throw New ArgumentOutOfRangeException( )  
Else  
    ...  
End If
```

Principes d'utilisation de la gestion structurée des exceptions

N'utilisez pas la gestion structurée des exceptions pour les erreurs qui ne sont pas susceptibles de se produire régulièrement. Utilisez d'autres blocs de code pour résoudre ces erreurs

- **If...End If**, etc.

Utilisez la valeur de retour pour les erreurs courantes

- Exemple : Les méthodes de lecture d'E/S sur fichier ne lèvent pas une exception de fin de fichier

Levez toujours des exceptions dans les blocs **Catch** qui vont de la plus spécifique à la moins spécifique

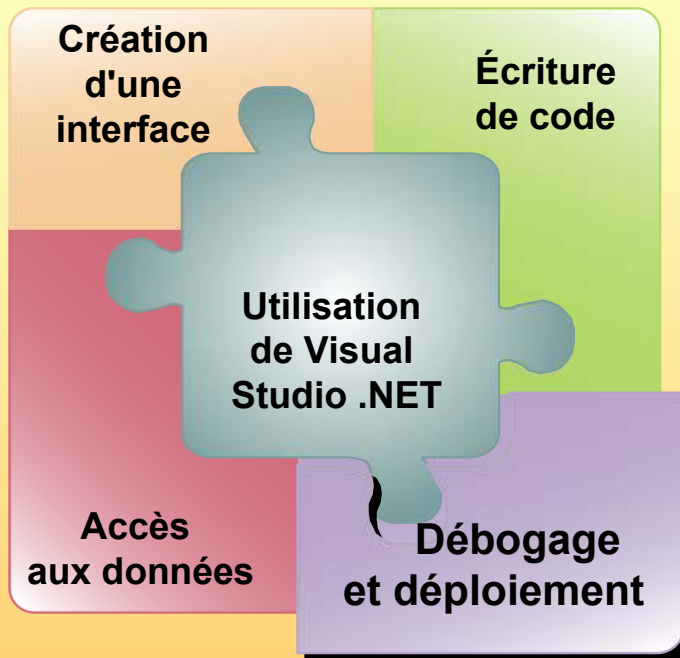
www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Module 8 :

Gestion des erreurs

et des exceptions

Vue d'ensemble



- Types d'erreurs
- Utilisation du débogueur
- Gestion des exceptions

Leçon : Types d'erreurs

- Présentation des erreurs de syntaxe
- Présentation des erreurs d'exécution
- Présentation des erreurs de logique

Présentation des erreurs de syntaxe



```
Private Sub PerformAction()  
    Dim newValue As XYZ  
    MessageBox.Show("Test")  
End Sub  
End Class
```

Erreur de syntaxe

Erreur de syntaxe

Le nom 'MessageBox' n'est pas déclaré.

Affichage des erreurs dans la fenêtre Liste des tâches :

Liste des tâches - 2 tâche(s) Erreur de génération affichée(s) (filtrée(s))				
!	<input checked="" type="checkbox"/>	Description	Fichier	Ligne
		Cliquez ici pour ajouter une nouvelle tâche		
!		Le nom 'MessageBox' n'est pas déclaré.	C:\Program ...\Form1.vb	59
!		Type 'XYZ' non défini.	C:\Program ...\Form1.vb	58

Liste des tâches | Sortie

Présentation des erreurs d'exécution

Vitesse = Kilomètres/Heures

- ' Si la variable *Heures* = 0, la division n'est pas une opération valide, même si l'instruction elle-même est syntaxiquement correcte

Microsoft Development Environment



Une exception non gérée du type 'System.OverflowException' s'est produite dans RuntimeError.exe.

Informations supplémentaires : L'opération arithmétique a provoqué un dépassement de capacité.

Arrêter

Continuer

Ignorer

Aide

Présentation des erreurs de logique

- **Définition : Une erreur qui entraîne la production de résultats incorrects par une application**
 - Peut ne pas générer de message d'erreur
 - Peut être recherchée en testant l'application et en analysant ses résultats

```
Dim x As Integer = 2  
Do While x < 10  
    ' Instructions du code  
    x -= 1  
Loop
```

Leçon : Utilisation du débogueur

- **Présentation du mode arrêt**
- **Utilisation des points d'arrêt**
- **Modification des points d'arrêt**
- **Barre d'outils Débogueur**
- **Procédure d'analyse pas à pas du code**
- **Utilisation des fenêtres de débogage**
- **Utilisation de la fenêtre Commande**

Présentation du mode arrêt

- **Permet d'interrompre le fonctionnement d'une application**
- **Vous pouvez effectuer les opérations suivantes lorsque vous êtes en mode arrêt :**
 - Parcourir votre code ligne après ligne
 - Déterminer les procédures actives qui ont été appelées
 - Contrôler les valeurs des variables, des propriétés et des expressions
 - Utiliser les fenêtres de débogage pour modifier les valeurs des variables et des propriétés
 - Modifier le flux du programme
 - Exécuter les instructions du code

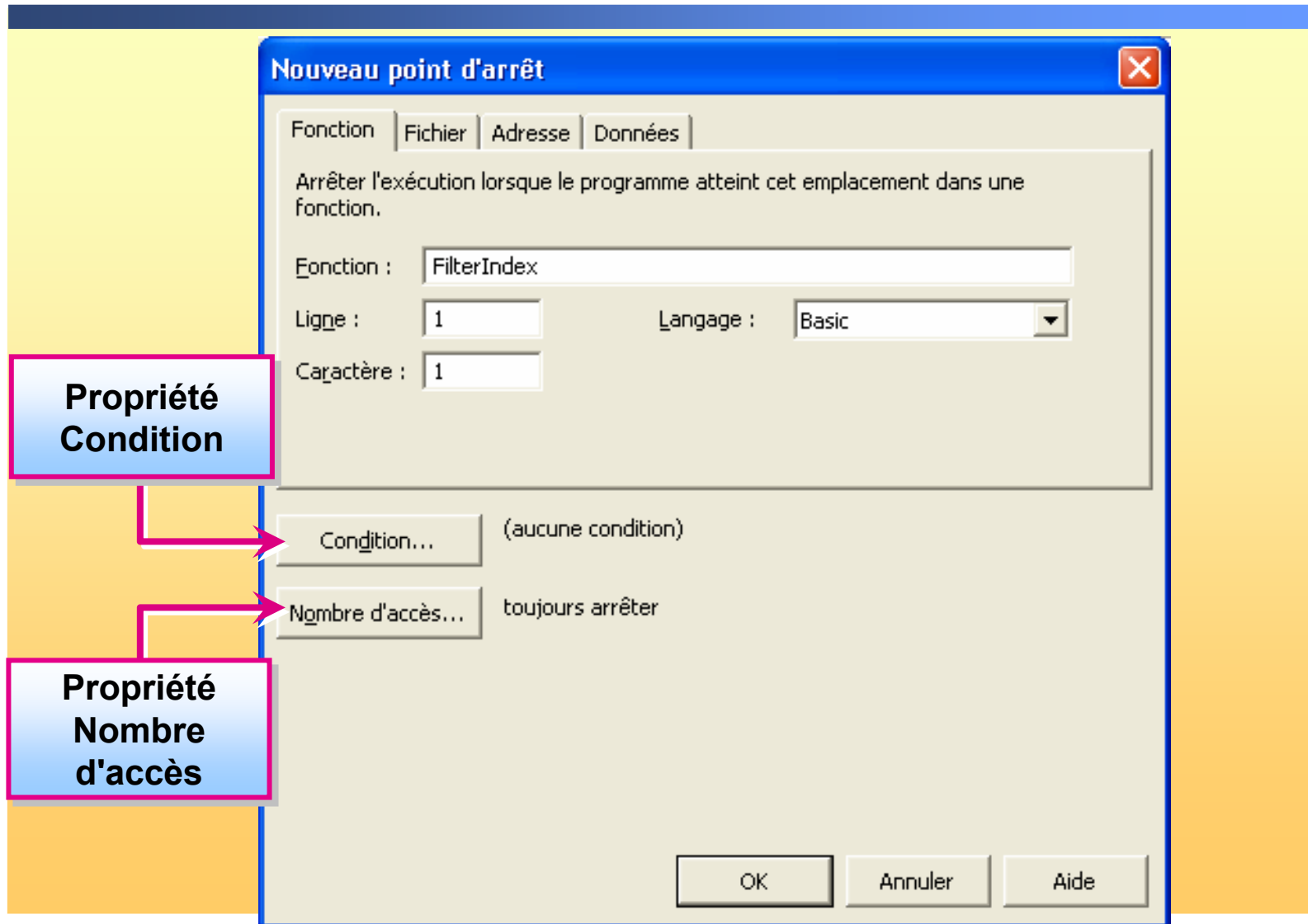
Utilisation des points d'arrêt

- Un point d'arrêt est un marqueur intégré à votre code qui permet à Visual Basic d'interrompre l'exécution du code sur une ligne spécifique
- Vous ne pouvez pas placer un point d'arrêt dans du code non exécutable

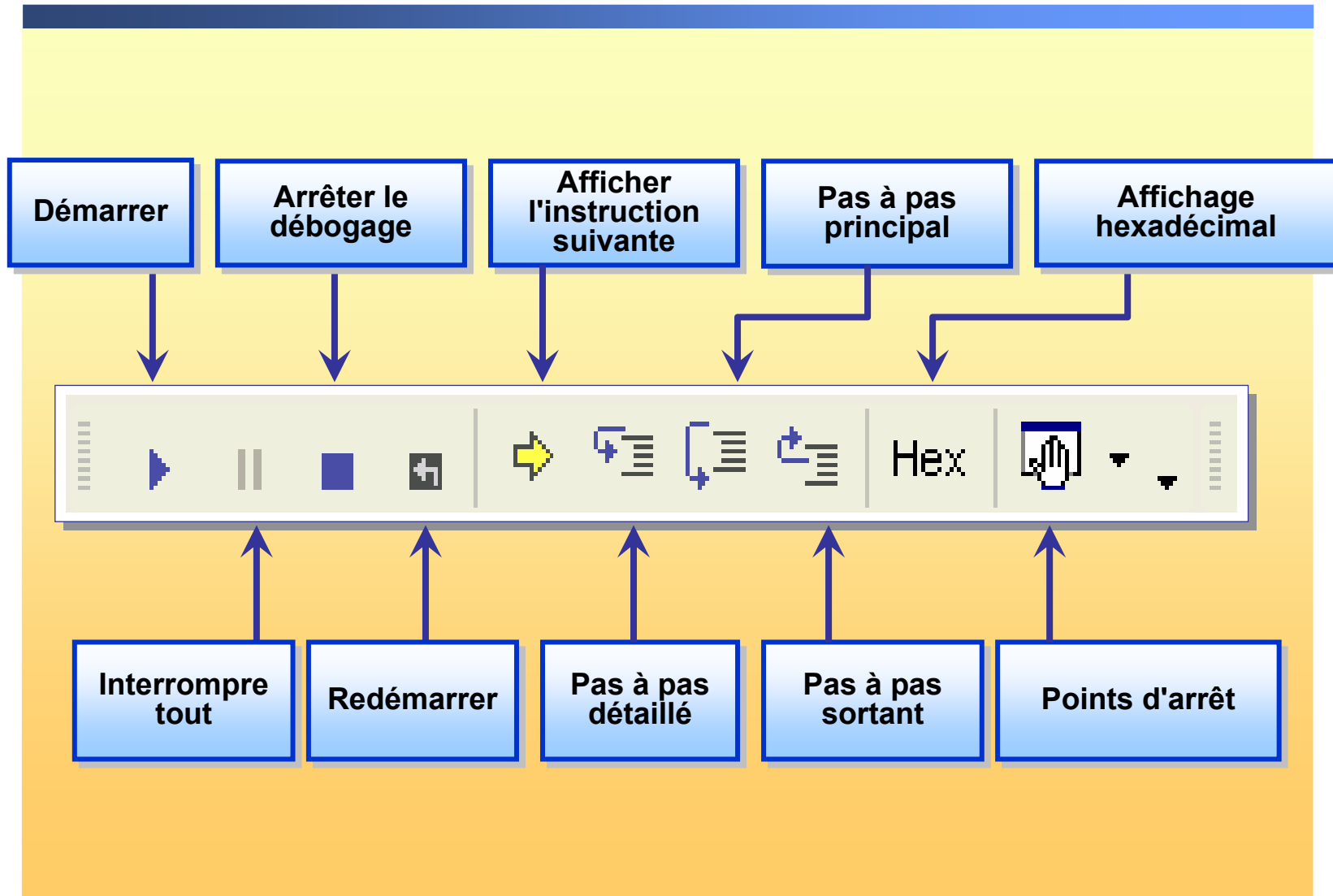
Points d'arrêt

```
Private Sub Button1_Click(ByVal sender As System.Object,  
    Dim maxLen As Integer = 0  
    Dim longestName As String = ""  
    Dim index As Integer  
  
    For index = 0 To names.GetLength(0)  
        If names(index).Length > maxLen Then  
            longestName = names(index)  
        End If  
    Next  
    MsgBox("Le prénom le plus long est " & longestName)  
End Sub  
End Class
```

Modification des points d'arrêt



Barre d'outils Déboguer



Procédure d'analyse pas à pas du code

- **Pas à pas détaillé et Pas à pas principal : Exécutent la ligne de code suivante. Si la ligne contient un appel de procédure :**
 - **Pas à pas détaillé :** Exécute uniquement l'appel, puis s'arrête à la première ligne de code dans la procédure
 - **Pas à pas principal :** Exécute l'intégralité de la procédure, puis s'arrête à la première ligne de code hors procédure
- **Pas à pas sortant :** Reprend l'exécution de votre code jusqu'au retour de la procédure, puis s'arrête au point de retour dans la procédure appelante
- **Exécuter jusqu'au curseur :** Le débogueur exécute votre application jusqu'à ce qu'il atteigne le point d'insertion que vous avez défini

Utilisation des fenêtres de débogage

Fenêtre	Utilisation
Automatique	Permet d'afficher des variables dans l'instruction en cours et dans les trois instructions suivant et précédant l'instruction en cours
Pile des appels	Permet d'afficher l'historique des appels à la ligne de code en cours de débogage
Variables locales	Permet d'afficher et de modifier les variables locales
Espion	<ul style="list-style-type: none">▪ Permet de créer votre liste personnalisée de variables ou d'expressions à contrôler▪ Permet de visualiser et manipuler toutes les expressions espionnes

Utilisation de la fenêtre Commande

- **Utilisez la fenêtre Commande pour :**
 - Émettre des commandes (mode Commande)
 - Déboguer et évaluer des expressions (mode Immédiat)

Tâche	Solution	Exemple
Évaluation des expressions	Faites précéder l'expression d'un point d'interrogation (?)	?myVariable
Passage du mode Commande en mode Immédiat	Tapez immed dans la fenêtre sans le signe Supérieur à (>)	immed
Passage du mode Immédiat en mode Commande	Tapez >cmd dans la fenêtre	>cmd
Passage provisoire en mode Commande lors d'un travail en mode Immédiat	Tapez la commande dans la fenêtre et faites-la précéder d'un signe Supérieur à (>)	>alias

Application pratique : Débogage du code



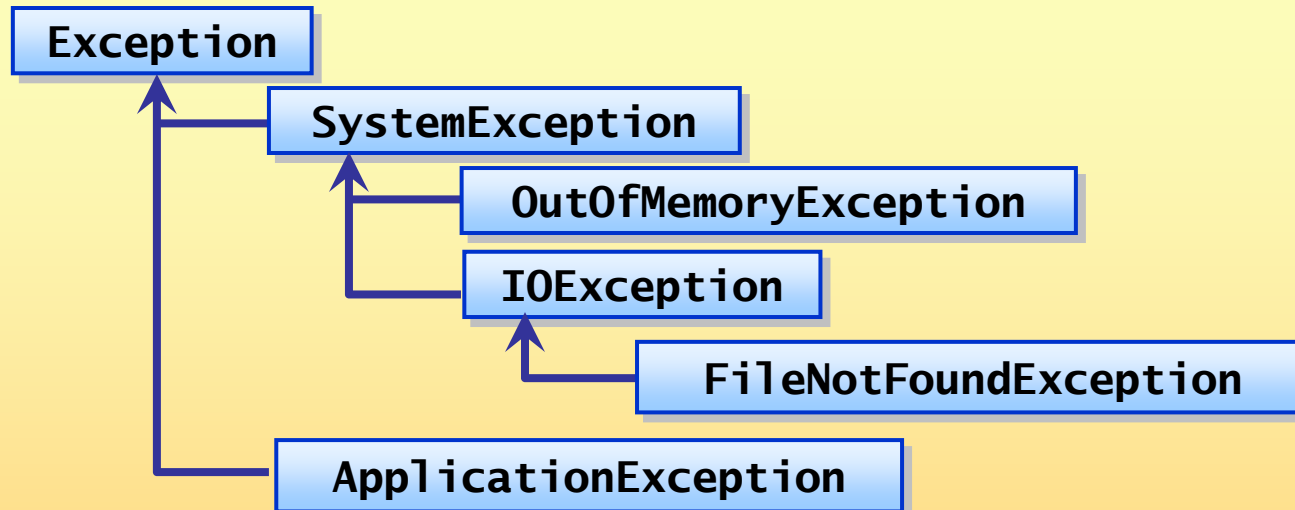
- 1** Examinez le code dans le gestionnaire d'événements Click
- 2** Générez et exécutez l'application
- 3** Utilisez les outils de débogage pour localiser l'erreur de logique
- 4** Suggérez une méthode pour corriger l'erreur

Leçon : Gestion des exceptions

- **Classe Exception**
- **Présentation de la gestion structurée des exceptions**
- **Utilisation de l'instruction Try...Catch**
- **Utilisation du bloc Finally**
- **Procédure de déclenchement des exceptions**
- **Principes d'utilisation de la gestion structurée des exceptions**

Classe Exception

- .NET Framework fournit le modèle objet d'exception suivant :



- Les classes d'exception vous permettent d'extraire des informations sur toutes les exceptions rencontrées
- Les propriétés de la classe de base **Exception** vous permettent d'analyser les exceptions
 - Propriétés principales : **StackTrace**, **Message**, **HelpLink** et **Source**

Présentation de la gestion structurée des exceptions

- **Détecte et répond à des erreurs lors de l'exécution d'une application**
- **Utilisez l'instruction Try...Catch...Finally pour encapsuler et protéger des blocs de code pouvant potentiellement provoquer des erreurs**
 - Chaque bloc possède un ou plusieurs gestionnaires associés
 - Chaque gestionnaire spécifie une forme de condition de filtre sur le type d'exception qu'il gère
- **Avantages :**
 - Permet de séparer le code de gestion des erreurs du code de logique
 - Facilite la lecture, le débogage et la gestion du code

Utilisation de l'instruction Try...Catch

- Placez les sections du code susceptibles de lever des exceptions dans un bloc Try
- Gérez les exceptions dans un bloc Catch distinct

Try

```
fs = New FileStream("data.txt", _  
    FileMode.Open)
```

← Logique du programme

Catch ex As FileNotFoundException

```
    MessageBox.Show("Fichier introuvable")
```

← Gestion des exceptions

Catch ex As Exception

```
    MessageBox.Show(ex.Message)
```

End Try

Utilisation du bloc Finally

- Bloc de code facultatif qui, s'il est inclu, est toujours exécuté
- Placez dans le bloc Finally le code de nettoyage, tel que le code pour la fermeture de fichiers

Try

```
fs = New FileStream("data.txt", FileMode.Open)
```

Catch ex As FileNotFoundException

```
MessageBox.Show("Fichier de données manquant")
```

Catch ex As Exception

```
MessageBox.Show(ex.Message)
```

Finally

```
If Not (fs Is Nothing) Then fs.Close( )
```

End Try

Procédure de déclenchement des exceptions

- L'instruction **Throw** crée une exception que vous pouvez gérer à l'aide du code de gestion structurée des exceptions

```
If (day < 1) Or (day > 365) Then  
    Throw New ArgumentOutOfRangeException( )  
Else  
    ...  
End If
```

Principes d'utilisation de la gestion structurée des exceptions

N'utilisez pas la gestion structurée des exceptions pour les erreurs qui ne sont pas susceptibles de se produire régulièrement. Utilisez d'autres blocs de code pour résoudre ces erreurs

- **If...End If**, etc.

Utilisez la valeur de retour pour les erreurs courantes

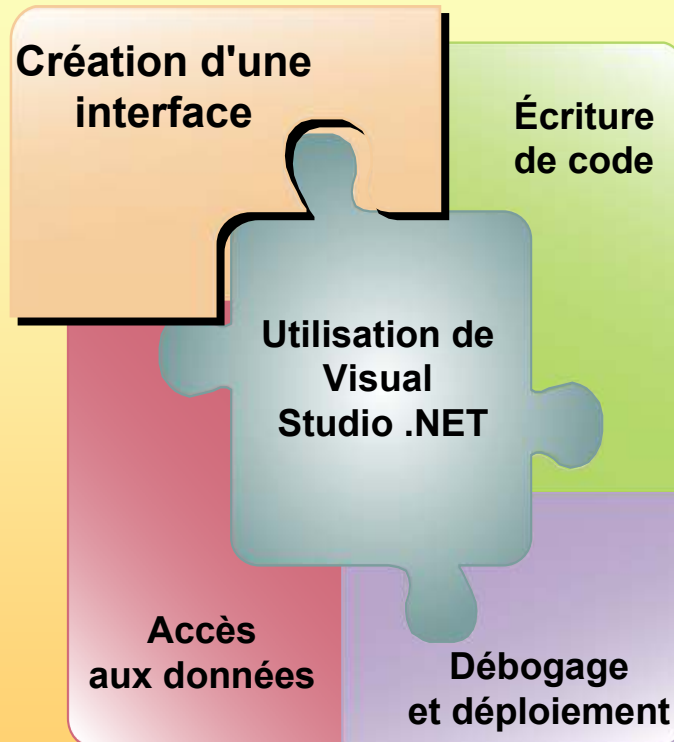
- Exemple : Les méthodes de lecture d'E/S sur fichier ne lèvent pas une exception de fin de fichier

Levez toujours des exceptions dans les blocs **Catch** qui vont de la plus spécifique à la moins spécifique

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Module 9 : Amélioration de l'interface utilisateur

Vue d'ensemble

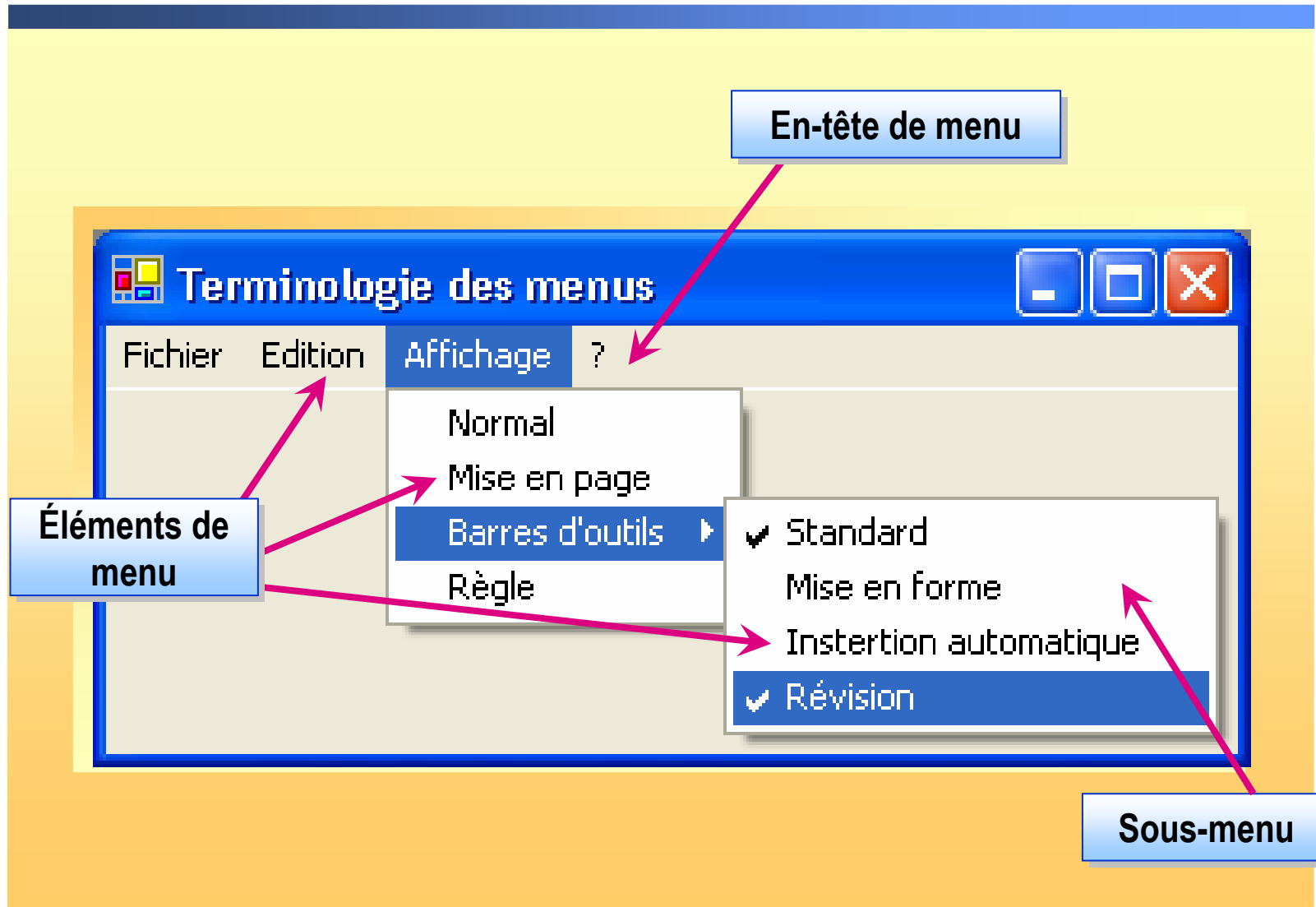


- **Création de menus**
- **Création de barres d'état**
- **Création de barres d'outils**

Leçon : Création de menus

- Terminologie des menus
- Principes de création des menus
- Création d'un menu
- Modification des éléments de menu
- Amélioration de votre menu

Terminologie des menus



Principes de création des menus

Principe	Exemple
Utilisez une lettre majuscule comme première lettre du nom des éléments de menu	Fichier, Edition, ? (Aide)
Affectez une touche d'accès rapide unique à chaque élément de menu	<u>F</u> ichier, <u>E</u> dition, <u>?</u> (Aide)
Utilisez des conventions d'appellation cohérentes	FileItem, EditItem, HelpItem
Placez des points de suspension (. . .) derrière les commandes de menu qui requièrent plus d'informations de la part de l'utilisateur	Enregistrer sous...

Création d'un menu

1 Ajoutez un contrôle MainMenu au formulaire

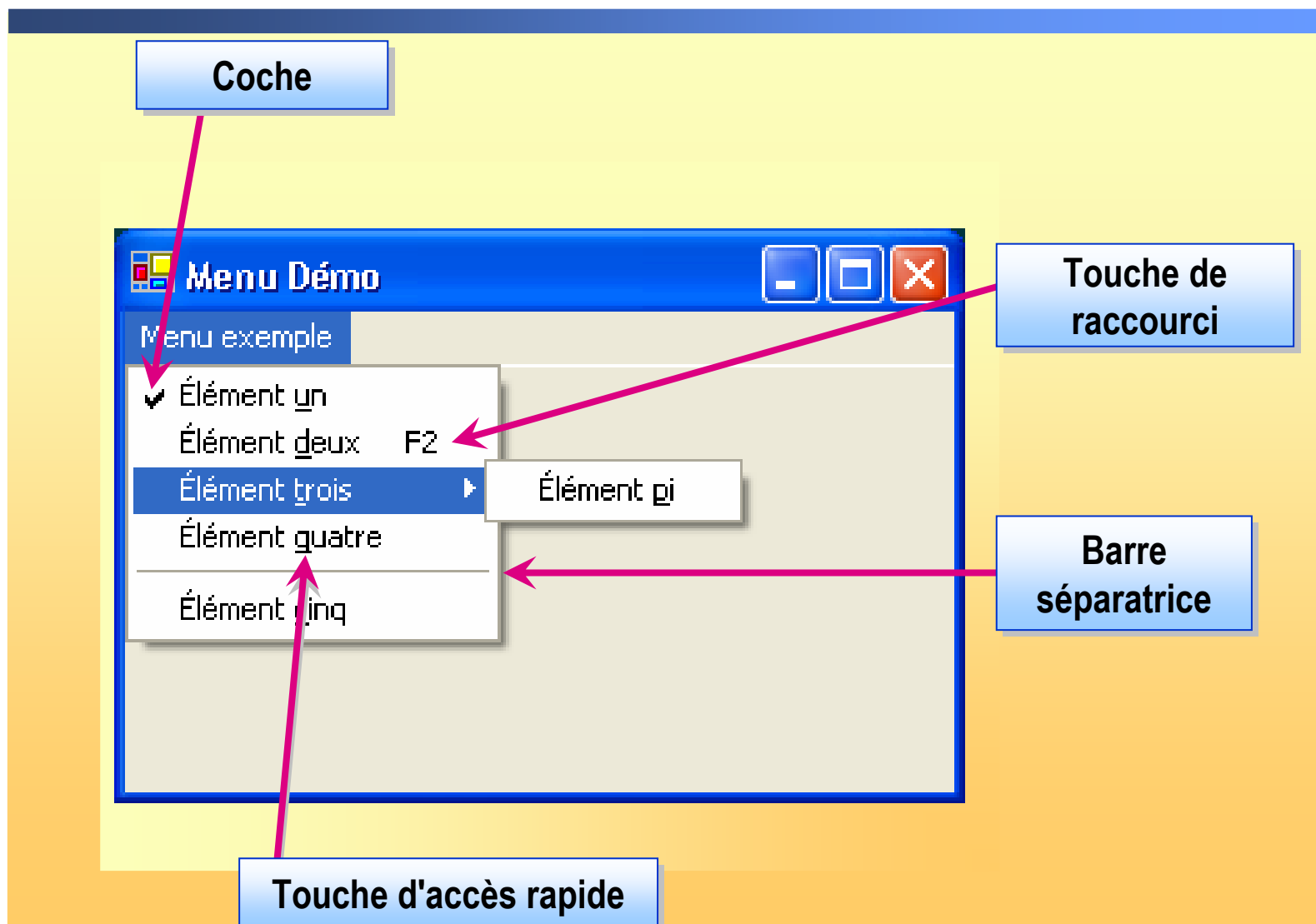
2 Créez la structure du menu en ajoutant des éléments de menu

3 Ajoutez des fonctionnalités aux éléments de menu

Modification des éléments de menu

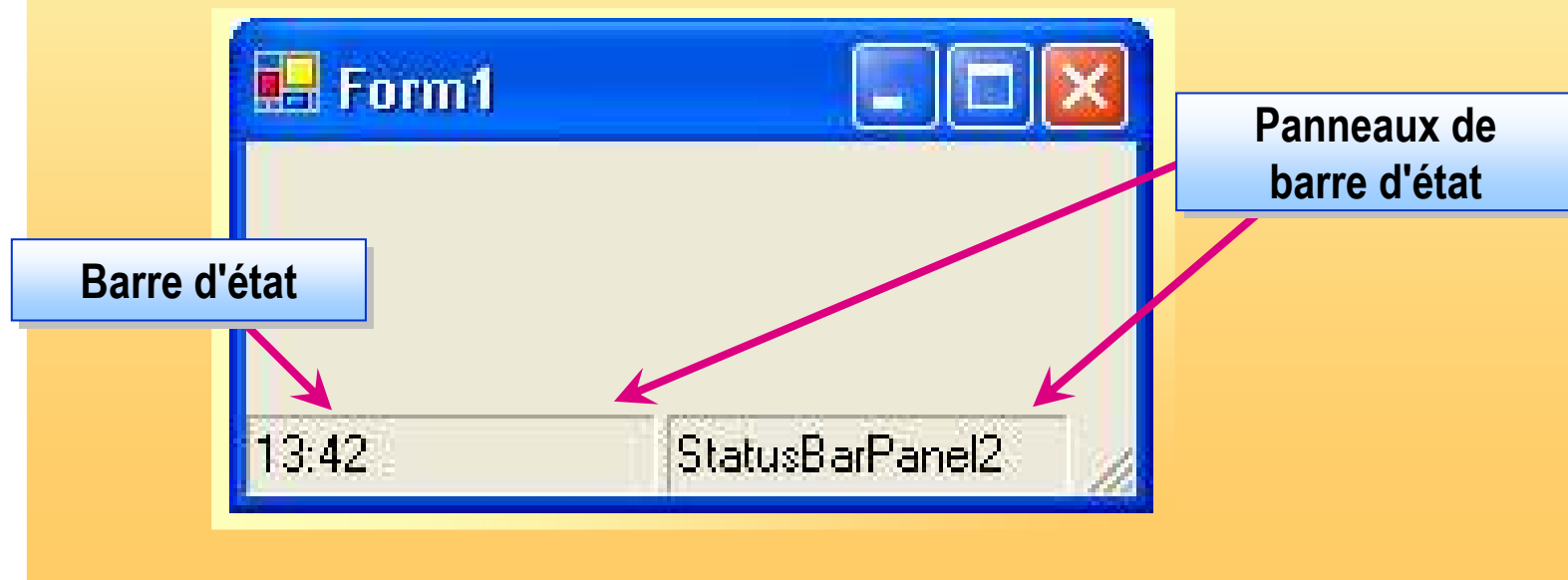
Modification souhaitée	Étapes
Déplacer	Dans le Concepteur de menus, cliquez sur l'élément de menu et déplacez-le vers son nouvel emplacement
Supprimer	Cliquez avec le bouton droit sur l'élément de menu, puis cliquez sur Supprimer
Modifier	Utilisez la fenêtre Propriétés pour modifier la propriété Text, et utilisez l'Éditeur de noms pour modifier la propriété Name
Désactiver	Définissez la propriété Enabled de l'élément de menu à la valeur False
Masquer	Définissez la propriété Visible de l'élément de menu à la valeur False

Amélioration de votre menu



Leçon : Création de barres d'état

- Création d'une barre d'état
- Ajout de panneaux à une barre d'état
- Définition du texte des panneaux au moment de l'exécution



Création d'une barre d'état

Procédure

- 1** Ouvrez le formulaire auquel vous souhaitez ajouter la barre d'état
- 2** À partir de la boîte à outils, ajoutez un contrôle StatusBar au formulaire
- 3** Définissez les propriétés StatusBar
- 4** Ajoutez le nombre souhaité de panneaux à la barre d'état

Ajout de panneaux à une barre d'état

Procédure

- 1** Ouvrez la fenêtre Propriétés pour le contrôle StatusBar
- 2** Définissez la propriété ShowPanels à la valeur True
- 3** Dans la propriété Panels, ouvrez l'Éditeur de collections StatusBarPanel
- 4** Utilisez les boutons Ajouter et Supprimer pour ajouter à et supprimer des boutons du contrôle StatusBar
- 5** Définissez les propriétés des panneaux

Définition du texte des panneaux au moment de l'exécution

- Référez-vous au panneau de la propriété que vous voulez définir à l'aide de l'index du panneau

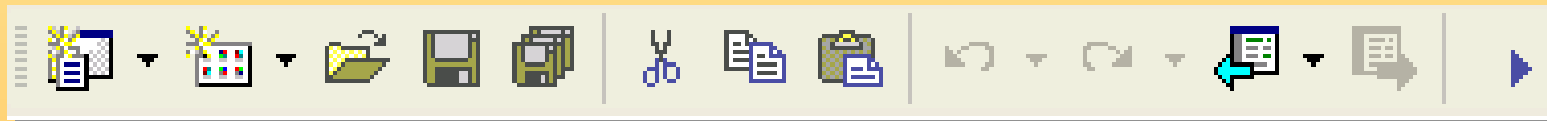
```
StatusBar1.Panels(0).Text = Now( )
```

```
StatusBar1.Panels(1).Text = "Prêt"
```

```
StatusBar1.Panels(0).Alignment = _  
HorizontalAlignment.Center
```

Leçon : Création de barres d'outils

- Création d'une barre d'outils
- Ajout de boutons à une barre d'outils
- Ajout d'icônes aux boutons d'une barre d'outils
- Écriture de code pour l'événement ButtonClick



Création d'une barre d'outils

Procédure

- 1** Ajoutez un contrôle ToolBar au formulaire à partir de la boîte à outils
- 2** Définissez les propriétés ToolBar
- 3** Ajoutez des boutons à la barre d'outils
- 4** Définissez les propriétés ToolBarButton
- 5** Si vous le souhaitez, affectez des images à chacun des boutons de la barre d'outils
- 6** Écrivez le code pour les boutons de la barre d'outils

Ajout de boutons à une barre d'outils

Procédure

- 1** Ouvrez la fenêtre Propriétés du contrôle ToolBar
- 2** Dans la propriété Buttons, ouvrez l'Éditeur de collections ToolBarButton
- 3** Utilisez les boutons Ajouter et Supprimer pour ajouter des boutons de barre d'outils et en supprimer
- 4** Dans l'Éditeur de collections ToolBarButton, définissez les propriétés des boutons

Ajout d'icônes aux boutons d'une barre d'outils

Procédure

- 1** Ajoutez un composant ImageList au formulaire à partir de la boîte à outils
- 2** Ajoutez des images au composant ImageList
- 3** Définissez la propriété ImageList du contrôle ToolBar
- 4** Ajoutez des boutons au contrôle ToolBar
- 5** Dans l'Éditeur de collections ToolBarButton, définissez la propriété ImageIndex de chaque bouton

Écriture de code pour l'événement ButtonClick

- Tous les boutons d'une barre d'outils partagent un seul événement Click
- Utilisez une commande Select Case ainsi que la classe ToolBarButtonEventArgs afin d'identifier le bouton sur lequel l'utilisateur a cliqué

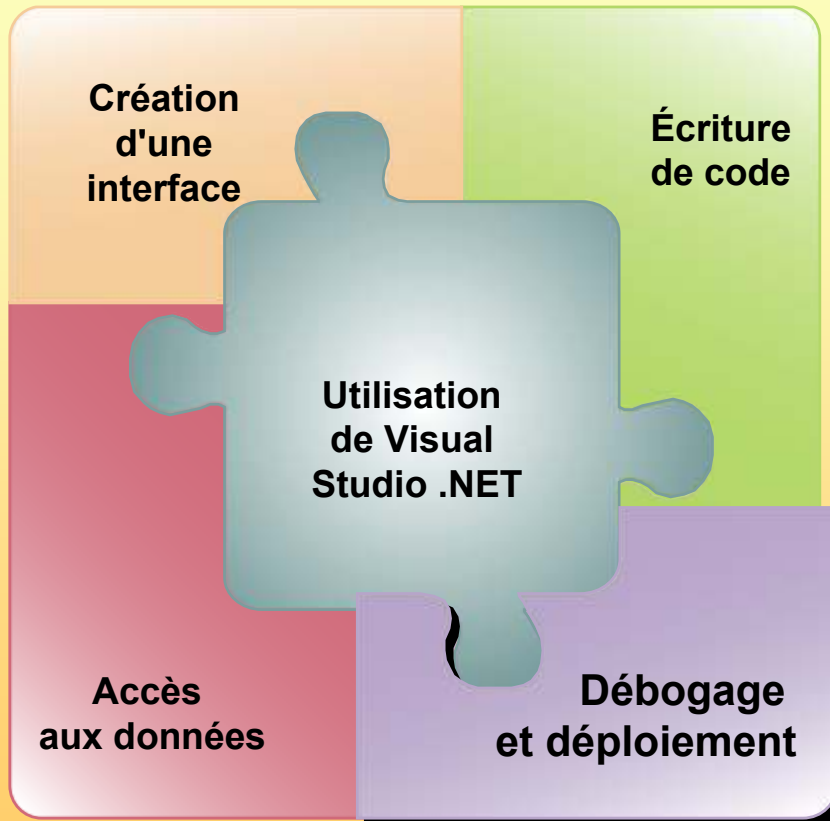
```
Protected Sub ToolBar1_ButtonClick(ByVal sender _ As  
Object, ByVal e As ToolBarButtonEventArgs)  
    Select Case ToolBar1.Buttons.IndexOf(e.Button)  
        Case 0  
            MessageBox.Show("L'utilisateur a cliqué sur le  
premier bouton de la barre d'outils")  
        Case 1  
            MessageBox.Show("L'utilisateur a cliqué sur le  
deuxième bouton de la barre d'outils")  
    End Select  
End Sub
```

www.Mcours.com

Site N°1 des Cours et Exercices Email: contact@mcours.com

Module 12 : Déploiement d'applications

Vue d'ensemble



- Introduction au déploiement
- Déploiement d'une application Windows

Leçon : Introduction au déploiement

- **Présentation des assemblies**
- **Déroulement du déploiement dans Visual Studio .NET**
- **Choix du type de projet**

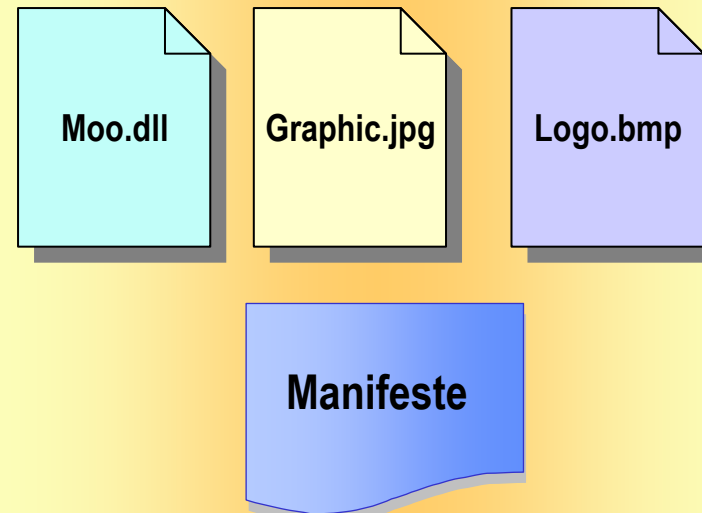
Présentation des assemblies

- Le code doit être empaqueté dans une unité fonctionnelle, appelée un *assembly*, avant d'être exécuté par le Common Language Runtime
- Les assemblies contiennent toutes les informations nécessaires au déploiement et à la gestion des versions

Assembly sous forme
d'un seul fichier



Assembly sous forme
de plusieurs fichiers



Déroulement du déploiement dans Visual Studio .NET

■ Options d'empaquetage

- Sous forme de fichiers exécutables portables (fichiers .dll et .exe)
- Sous forme de fichiers cabinet (.cab)
- Sous forme de package Windows Installer

■ Déploiement

- Repose sur la technologie Windows Installer

■ Autres modes de déploiement

- XCOPY
- Commande Copier un projet

Choix du type de projet

Projet de configuration

- Crée un fichier Windows Installer (fichier .msi)

Projet de configuration Web

- Crée un fichier Windows Installer pour une application Web (fichier .msi)

Projet de module de fusion

- Empaquette des composants qui peuvent être partagés par plusieurs applications (fichier .msm)

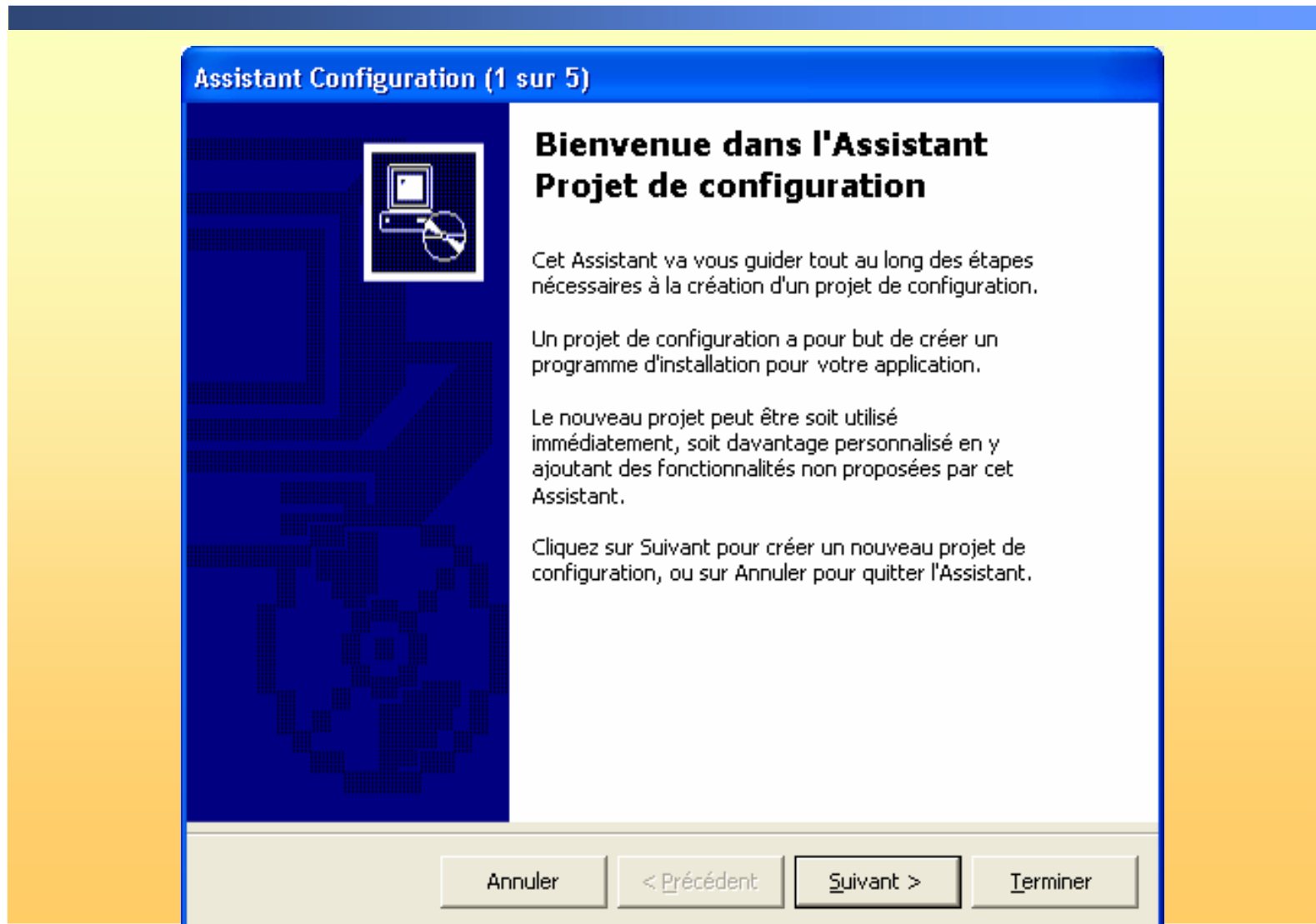
Projet CAB

- Crée un fichier .cab pour le téléchargement vers un navigateur Web

Leçon : Déploiement d'une application Windows

- 1** Créez un projet de configuration
- 2** Définissez les propriétés du projet
- 3** Personnalisez le projet de configuration
- 4** Générez l'application
- 5** Distribuez et installez l'application

Création d'un projet de configuration Windows Installer



Définition des propriétés d'un projet

■ Fenêtre Propriétés

- Définissez des propriétés générales
- Exemples : **Author, Manufacturer, ProductName**

■ Boîte de dialogue Propriétés de déploiement

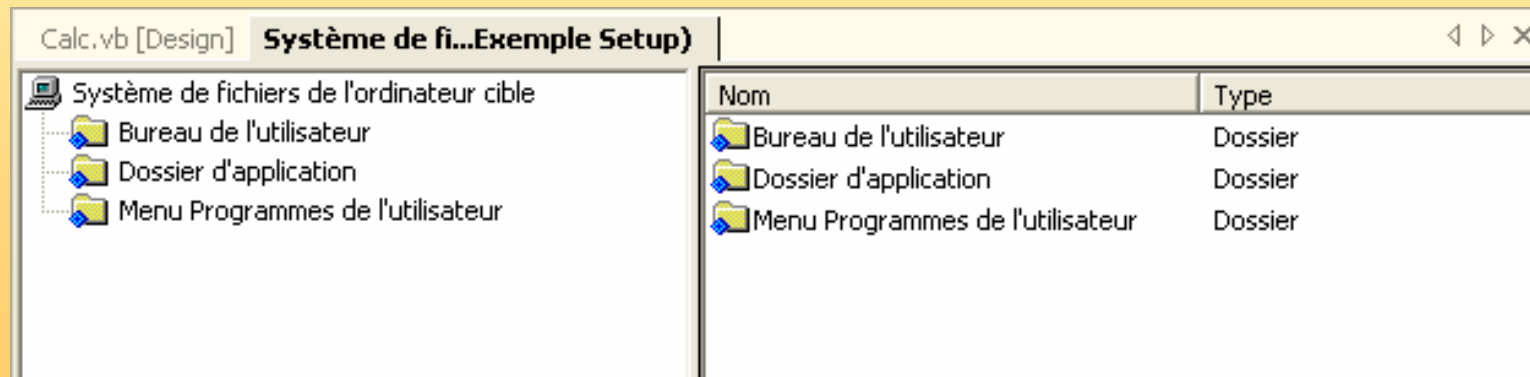
- Définissez des propriétés spécifiques à la configuration
- Exemples : **Programme d'amorçage, Compression, Nom du fichier de sortie et Fichiers du package**
- Lorsque vous effectuez une première installation sur des versions de Windows antérieures à Windows XP, vous devez inclure le programme d'amorçage dans le programme d'installation

Éditeurs de déploiement

- **Utilisez les éditeurs de déploiement pour configurer le programme d'installation que vous créez**
 - Éditeur du système de fichiers
 - Éditeur du Registre
 - Éditeur des types de fichiers
 - Éditeur de l'interface utilisateur
 - Éditeur des actions personnalisées
 - Éditeur des conditions de lancement

Utilisation de l'Éditeur du système de fichiers

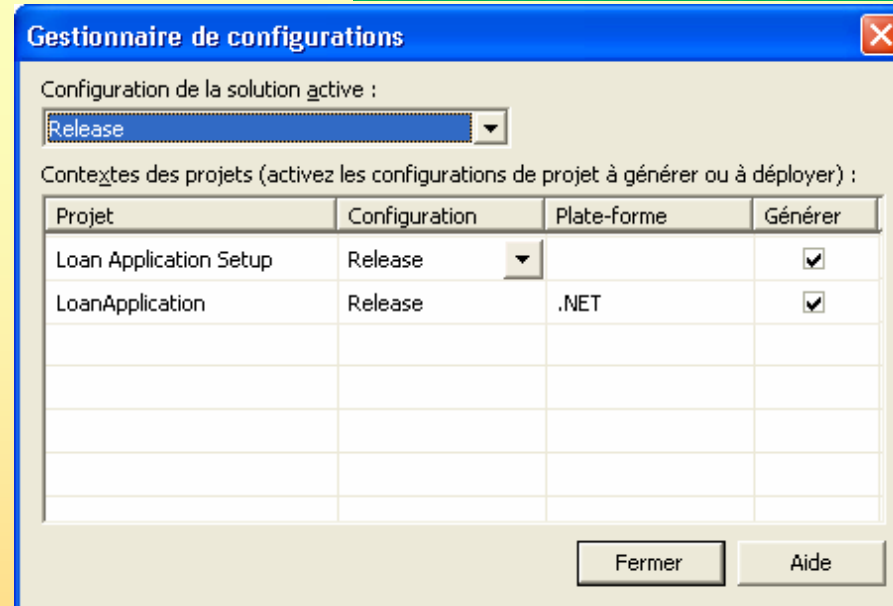
- Offre une représentation du système de fichiers sur un ordinateur de destination
- S'appuie sur le concept de dossiers virtuels pour s'assurer que les fichiers sont installés à l'endroit précis où vous le voulez



Génération du projet de configuration

www.Mcours.com

Site N°1 des Cours et Exercices Email: contact@mcours.com



- .NET Framework doit être installé sur n'importe quel ordinateur qui exécutera une application créée à l'aide de Visual Studio .NET
- Pour installer .NET Framework, utilisez le programme d'installation redistribuable Dotnetfx.exe