Langage VB.Net

Initiation

Mourad MAHRANE

DAWAN Paris, Tour CIT Montparnasse - 3,rue de l'Arrivée, 75015 PARIS
DAWAN Nantes, Le Sillon de Bretagne - 26e étage - 8, avenue des Thébaudières, 44800 ST-HERBLAIN
DAWAN Lyon, Le Britannia, 4ème étage - 20, boulevard Eugène Deruelle, 69003 LYON
formation@dawan.fr

Objectifs

- Apprendre à développer avec VB.Net

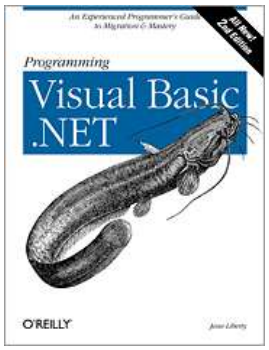
- Créer des interfaces de gestion de bases

- Manipuler des objets de la plate-forme .NET

Bibliographie

Visual Basic .NET, Frédéric Baurand – Ellipses – Fév. 2014

Programming Visual Basic .Net, Jesse Liberty – O'Reilly - Avr. 2003



MSDN : http://msdn.microsoft.com/fr-fr/default.aspx
Centre .NET Framework :
http://msdn.microsoft.com/fr-fr/netframework/default.aspx

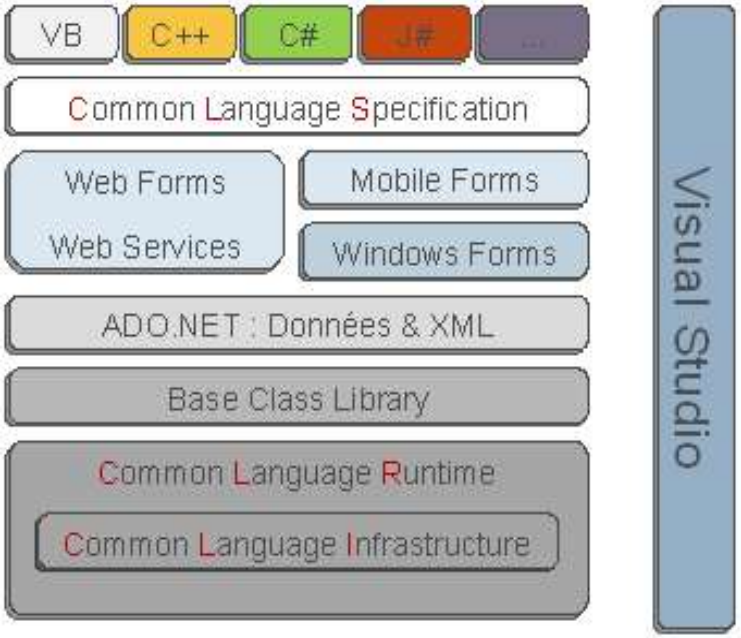
Plan

- Framework .NET
- Langage VB.Net
- Syntaxe du langage
- Tableaux
- Méthodes et paramètres
- Gestion des exceptions
- Classes fondamentales
- Applications graphiques
- Programmation orientée objet
- ADO.NET

Framework .NET

Plateforme .NET

- **Utilisation :** Développement – Déploiement – Exécution
- **Applications :** Web, Windows, Mobile, serveurs, jeux
- **Langages supportés :** VB .NET, J#, C#, etc.
- **Gratuite**
- **Installation :** intégrée à certaines éditions Windows
  Téléchargeable via MSDN ou Windows Update



Versions

- Juin 2000 : Lancement du développement

- Février 2002 : .NET Framework 1

- Mars 2003 : .NET Framework 1.1

- Novembre 2006 : .NET Framework 2.0

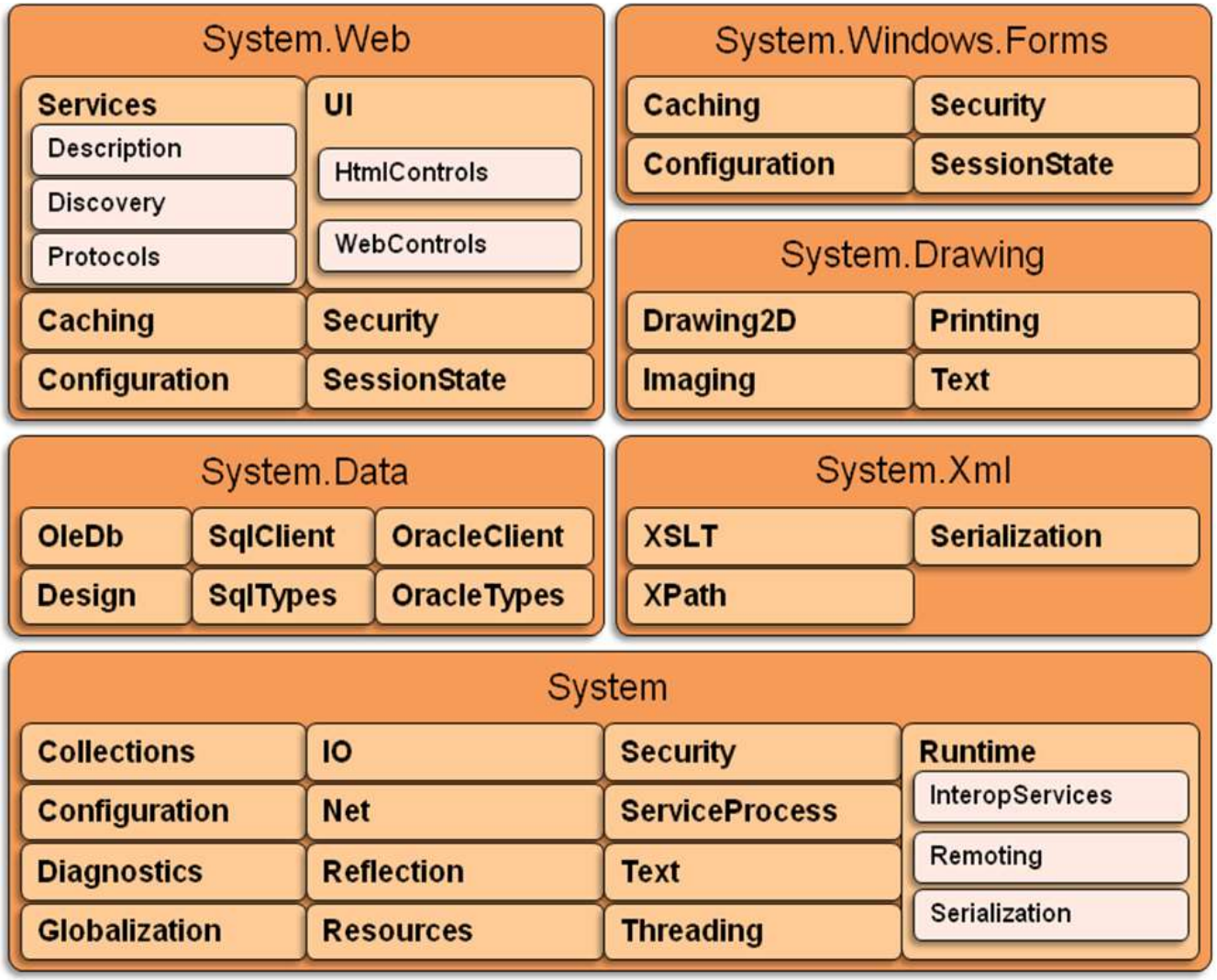
- Novembre 2007 : .NET Framework 3.0

- 2008 : .NET Framework 3.5
- 2010 : .NET Framework 4.0 => 2012 : v4.5

Common Language Runtime

- Implémentation du standard « Common Language Infrastructure »
- Concepts : Debug - Typage (Common Type System) - Exceptions...
- Fonctions : Gestion du contexte d'exécution et de la mémoire - Versioning des applications – Sécurité et intégrité des applications (signatures) - Interopérabilité COM

Bibliothèque de classes



Avantages

- Normes et pratiques du web.
- Modèles d'application unifiés.
- Classes extensibles.

N.B. : le Framework .NET ne fonctionne que sous Windows

Utiliser Mono ou DotGNU pour d'autres plateformes

Présentation

- Langage orienté objet de type sécurisé
- Très proche du Visual Basic
- RAD
- Multi-plateformes (IL)
- Plusieurs versions successives

Développement VB.Net

- Applications Windows
- Pages ASP.NET
- Services Web
- Services Windows

VB.Net Multi-plateformes :

Win32 - Win64 - WinCE – WinMobile

IDE :

- Microsoft Visual Studio payant ou version Community
- SharpDevelop ou MonoDevelop gratuits mais moins performants


Programme VB.Net

- Module
- Méthode main
- Classe Console (ReadLine et WriteLine)

```
Module Module1

    Sub Main()

        Console.WriteLine("Hello World!")

    End Sub

End Module
```

Débogage et Exécution

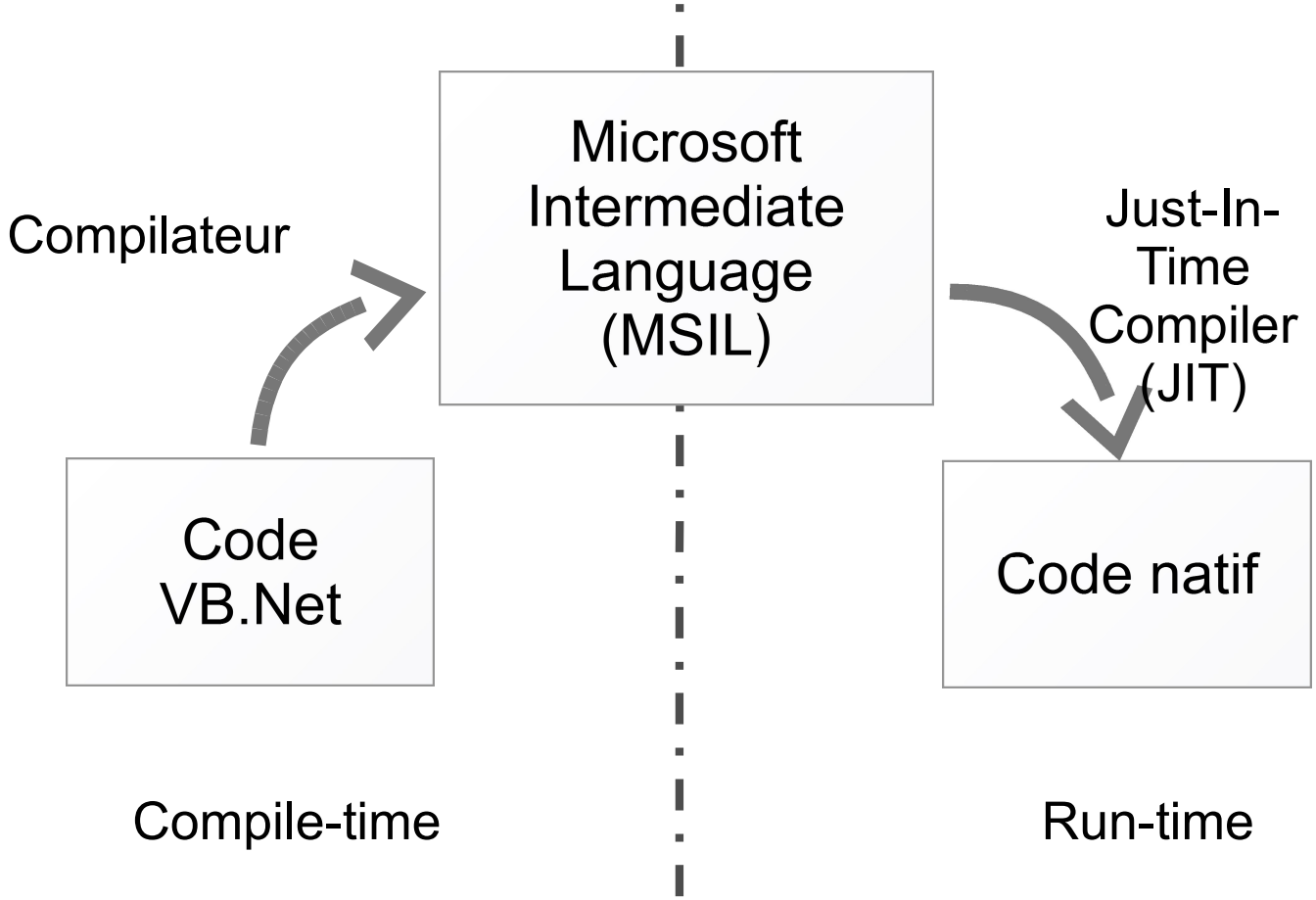
Localisation et correction des erreurs :

- Erreurs et débogage JIT
- Points d'arrêts et pas-à-pas
- Examen et modifications des variables

Exécution :

- IDE (Start Without Debugging)
- Ligne de commande (nom de l'application)

Etapes de compilation



En ligne de commande :

vbc.exe

msbuild.exe

Syntaxe du langage

Base du langage

- Les erreurs de syntaxes sont interdites et vérifiées à la compilation
- Instructions les unes après les autres sans séparateur de ligne, code en UTF-8
- Pas de différences entre majuscules et minuscules
- Commentaires :
  - `'fin de la ligne`
  - `''' documentation`

Types de données

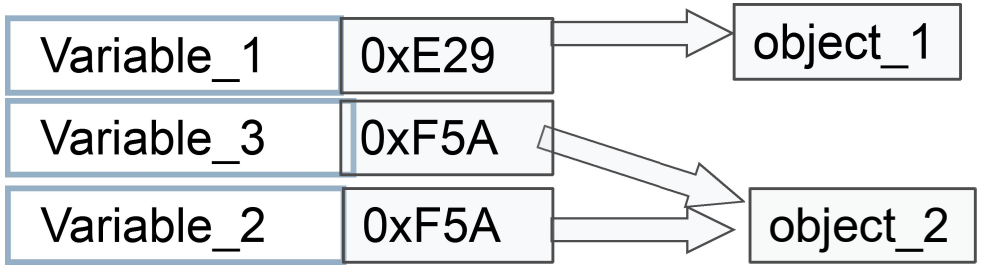
- Types communs (System)
- Types simples et Types références
- Déclaration de variables, affectation de valeurs, constantes …
- Tableaux
- Enumérations (enum)
- Structures (struct)

Types de données

- Types communs (System)
  - Entiers : Byte/SByte (8 bits), Short/UShort (16 bits), Integer/UInteger (32 bits), Long/ULong (64 bits)
  - Flottants : Single (32 bits), Double (64 bits)
  - Decimal (128 bits)
  - Autres : Boolean, Date, Char, String

- Types simples et Types références

Variable_1	45
Variable_2	true
Variable_3	c
Variable_4	56.8



Types de données

- Déclaration de variables, affectation de valeurs, constantes …

```
'Declaration de variables
Dim Variable1 as Integer
Dim Variable2 as String
'Affectation de valeurs
Variable1 = 15
Variable2 = "ceci est une chaine"
'Affectation lors de la declaration
Dim Variable3 as Integer = 7
'Declaration de constante
Const VARIABLECONSTANTE as String = "constante"
```

Types de données

- Enumérations

```
Enum CouleurCarte
    Pique
    Coeur
    Carreau
    Trefle
End Enum
Sub Main()
    Dim MaCouleur as CouleurCarte
    MaCouleur = CouleurCarte.Coeur
End Sub
```

Types de données

- Structures

```
Structure Automobile
    Dim Puissance as Integer
    Dim Couleur as String
    Dim Assurance as Boolean
End Structure
Dim MaVoiture as Automobile
MaVoiture.Puissance = 6
```

- Nullable/HasValue

```
'Declaration de variables
Dim Variable1? as Integer
Dim Variable2 as Integer?
Variable1 = Nothing
Variable1.HasValue() 'retourne faux
```

Opérateurs

- Arithmétiques : + - * / \ MOD
- Égalité/Inégalité : = <>
- Relationnels : < <= => >
- Logiques : And Or Not Xor AndAlso OrElse IsFalse IsTrue
- Assignation : = += -= *= /= \= ^= <<= >>= &=
- Concaténation de chaines : &

Transtypage

- Conversions implicites (automatique)

```
Dim MonEntier As Integer = 25
Dim MonLong as Long = MonEntier
```

- Conversions explicites (cast)

```
Dim MonEntier As Integer = 25
If Ctype(MonEntier, String).Equals("25") Then ...
```

Dérivés : CBool,CByte,CChar,CDate, CInt, CStr ...

Transtypage

- DirectCast

```
Dim MonEntier As Integer = DirectCast(obj, Integer)
```

- TryCast

```
Dim MonEntier As Integer = TryCast(obj, Integer)
If MonEntier is Nothing Then ...
```

- Fonctions de conversions

Str(25), Val("25 32bis"), Int(15.7), Fix(15.7)

- Classe System.Convert

Structures Conditionnelles

- if / else

```
Dim MonEntier As Integer = 25
If MonEntier = 22 Then
    'Instruction 1
ElseIf MonEntier = 25 Then
    'Instruction 2
Else
    'Instruction 3
End If
```

Structures Conditionnelles

- Select/Case

```
Dim MonEntier As Integer = 25
Select Case MonEntier
    Case 22
        'Instruction 1
    Case 23,24
        'Instruction 2
    Case 25
        'Instruction 3
    Case 26 To 30
        'Instruction 4
    Case Else
        'Instruction 5
End Select
```

Structures Conditionnelles

- opérateur ternaire

```
Dim MonEntier As Integer = 25

Dim ResultatTest As String

ResultatTest = IIF (MonEntier < 25, "Variable
inferieure a 25", "Variable superieure a 25")
```

Équivalent à

```
Dim MonEntier As Integer = 25

Dim ResultatTest As String

If MonEntier < 22 Then

    ResultatTest = "Variable inferieure a 25"

Else

    ResultatTest = "Variable superieure a 25"

End If
```

Boucles

- for

```
For i As Integer = 0 To 10
        'Instructions
Next
```

Instructions : Exit For, Continue For

- while

```
Dim MaCondition As Boolean
While MaCondition
        'Instructions
End While
```

Instructions : Exit While, Continue While

Boucles

- do/loop

```
Dim MaCondition As Boolean
Do
        'Instructions
Loop While MaCondition
```

```
Dim MaCondition As Boolean
Do
        'Instructions
Loop Until MaCondition
```

```
Dim MaCondition As Boolean
Do While MaCondition
        'Instructions
Loop
```

```
Dim MaCondition As Boolean
Do Until MaCondition
        'Instructions
Loop
```

NB : Test de la condition en début ou en fin de boucle

Instructions : Exit Do, Continue Do

Tableaux

Tableaux

- Déclaration d'un tableau

```
Dim Tableau(5) as Integer
Dim Tableau() as Integer =
```

- Valeur d'un élément d'un tableau

```
Tableau(indice)
```

- Taille d'un tableau

```
Tableau.Length
```

Dans la déclaration, Taille = nb de cases – 1

Index des cases débute à 0

Tableaux

- Tableau à 2 dimensions

```
Dim Tableau(10,10) as Integer
```

- Tableau à 3 dimensions (cube)

```
Dim Tableau(10,10,10) as Integer
```

Si besoin de plus de 3 dimensions : tableau de tableaux

```
Dim TableauDeCubes(1)(10,10,10) as Integer
```

Tableaux

- for

```
Dim UnTableau(10) As Integer
For compteur As Integer = 0 To UnTableau.Length - 1
        'Instruction sur les cases du tableau
Next
```

- foreach

```
Dim UnTableau(10) As String
For Each element As String In UnTableau
        'Instruction sur les cases du tableau
Next
```

Tableaux

- Redimensionner

```
Dim UnTableau(10,10) As Integer
Redim Preserve UnTableau(10,15)
Redim UnTableau(10,10)
```

- Retourner

```
Array.Reverse(UnTableau)
```

- Vider

```
Array.Clear(UnTableau, caseDebut, nbCases)
```

- Copier

```
Array.Copy(UnTableau, UnAutreTableau, nbCases)
```

- Trier, rechercher, ...

Méthodes et paramètres

Méthodes

- Une méthode permet de regrouper des instructions :
  - Diviser le code en morceaux (réutilisabilité, clarté, travail en groupe)
  - Factoriser le code (maintenabilité, clarté)
- Une méthode (« procédure », « fonction ») a un nom, des paramètres et peut retourner une valeur (dans le cas d'une fonction)
- Un appel de méthode indique son nom, envoie et/ou utilise les paramètres, utilise la valeur de retour éventuelle

Méthode

- Déclaration

```
Sub MaProcedure()
        'Instructions
End Sub
```

- Appel

```
MaProcedure()
```

- Passage de paramètres

```
Sub MaProcedure(ByVal Param1 As Integer)
        'Instructions
End Sub
```

Fonction

- Déclaration

```
Function MaFonction() As Double
        Dim monResultat As Double = 3.14
        Return monResultat
End Function
```

- Appel

```
Dim MaVariable As Double = MaFonction()
```

- Passage de paramètres

```
Function MaFonction(ByVal Param1 As Integer) As
   Double
        Return Param1 * 3.14
End Function
```

Passage de paramètres

Passage de paramètres

- En entrée (ByVal)
- En entrée/sortie (ByRef)

```
Function MaFonction(ByVal Param1 As Integer, ByRef
   Param2 As Double) As Double

       Param2 = Param1 + Param2

       Return Param1 * Param2

End Function
```

Appel :

```
Dim param1 As Integer = 10

Dim param2 As Double = 2.5

Dim resultat As Double = MaFonction(param1, param2)

'param1=10, param2=12.5, resultat=125
```

Passage de paramètres

Paramètre facultatif (Optional)

```
Sub MaFonction(ByVal Param1 As Integer,
   Optional ByVal Param2 As Integer,
   Optional ByVal Param3 As Integer)

...

End Sub
```

Appel :

```
MaFonction(2,,5)
```

Valeur par défaut

```
Sub MaFonction(ByVal Param1 As Integer,
   Optional ByVal Param2 As Integer,
   Optional ByVal Param3 As Integer = 100)

...

End Sub
```

Passage de paramètres

Tableau de paramètres (ParamArray)

```
Sub MaFonction(ByVal Param1 As String,
   ByVal ParamArray Param2()  As String)

...

End Sub
```

Appel :

```
MaFonction("Test","Nb","De","Parametres","Inconnu")

Dim tableauParametres As String() =
   {"Nb","De","Parametres","Inconnu"}

MaFonction("Test", tableauParametres)
```

Passage de paramètres

Passage de paramètres

```
Sub MaFonction(ByVal param1 As Integer,
   Optional ByVal param2 As Integer,
   Optional ByVal param3 As Integer)

...

End Sub
```

- Par position

```
MaFonction(2,10,5)
```

- Par nom

```
MaFonction(param1:=2,param2:=10,param3:=5)
```

- Mixte

```
MaFonction(2,param3:=5)
```

Surcharge de méthode

Plusieurs méthodes peuvent avoir le même nom et des arguments différents. Pour une fonction, le type de retour doit être identique.

```
Sub MaFonction(ByVal param1 As Integer)
...
End Sub
Sub MaFonction(ByVal param1 As Integer,
   ByVal param2 As Integer)
...
End Sub
Sub MaFonction(ByVal param1 As Integer,
   ByVal param2 As Integer,
   ByVal param3 As Integer)
...
End Sub
```

Récursivité

Capacité d'une méthode à s'appeler elle-même.

```
Function factorial(ByVal n As Integer) As Integer
    If n <= 1 Then
        Return 1
    Else
        Return factorial(n - 1) * n
    End If
End Function
```

Règles d'écriture

- Conventions de codage :
  - Laisser faire l'IDE
- Conventions de nommage : uneVariable, unArgument, uneMethodePrivee, UneMethodePublique, UnAttributPublique, UneConstantePublique, UnType ...
- Commentaires interprétés :

```
''' <summary>...
```

Exercices

- Déclarer un tableau de 10 nombres.
- Écrire des méthodes qui calculent le *minimum*, le *maximum et* la *moyenne* des valeurs de ce tableau.
- Tester ces méthodes.

Exceptions

Définition

Situations inattendues ou exceptionnelles qui surviennent pendant l'exécution d'un programme, interrompant le flux normal d'exécution

Types d'Exceptions

- **Checked Exceptions**
  - Le développeur doit les anticiper et coder des lignes pour les traiter.
- Exemple : on peut essayer de charger un fichier qui n'existe pas.

- **Errors**
  - On ne doit pas les identifier et le programme s'arrête en les rencontrant.

- **Runtime exceptions**
  - Ne peuvent être prévues (dans certains cas)

Gestion des exceptions

- try / catch / finally : récupération
- throw : lancement
- Classe System.Exception et ses dérivées
- Il est possible de définir des exceptions

```
Try
        'Lancement d'une exception
        Throw new Exception()
Catch ex As Exception
        'Gestion de l'exception
Catch ex As Exception When (Expr)
        'Gestion de l'exception si Expr vaut true
Finally
        'Traitement effectué en fin de try
End Try
```

Exercices

- Déclarer un tableau de 10 nombres pouvant contenir des valeurs nulles.
- Modifier les méthodes *minimum*, le *maximum* et *moyenne* pour renvoyer une exception si une valeur est nulle.
- Tester ces méthodes.

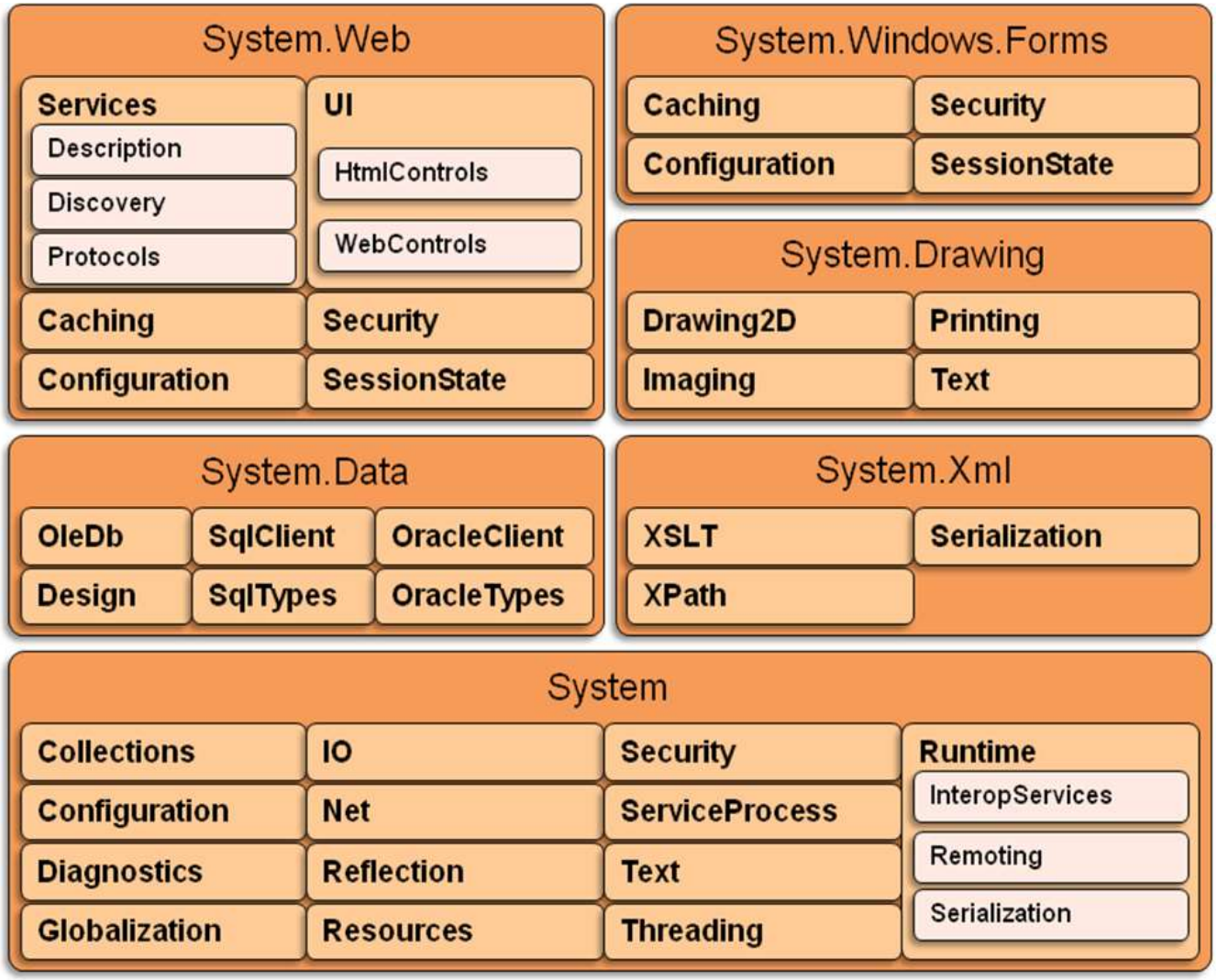
Bibliothèque de classes .Net

Définition

BCL : Base Class Library

Il s'agit des classes fondamentales sur lequel les applications .NET sont construites.

Bibliothèque de classes



Chaines de caractères

System.String :

- Comparaison : Compare, Equals
- Concaténation : Concat, Join
- Découpage : Split, Substring
- Recherche : StartsWith, EndsWith, IndexOf, ...
- Mise en forme : Format, PadLeft/Right, TrimStart/End, …
- Longueur : Length
- Opérateurs : = <>

Date

System.DateTime :

- Comparaison : Compare, Equals
- Opération : Add, Substract
- Conversion : Parse, ToString
- Recherche : Date, Day, Hour, Month, …
- Opérateurs : + - = <> > >= < <=

Collections

System.Collections :
Regroupe des classes pour gérer des ensembles d'objets :

- Faiblement typée : ArrayList, Hashtable, Queue, Stack, ...
- Fortement typée (System.Collections.Generic) : List, Dictionnary, HashSet, Queue, Stack, ...

Génériques

Utilisés pour typer :

- une classe :

```
Public Class classHolder(Of t)
    Public Sub processNewItem(ByVal newItem As t)
        Dim tempItem As t
        ' Insert code that processes an item of data
   type t.
    End Sub
End Class
```

- une collection :

```
Dim stagiaires As New List(Of String)
```

Collection

VisualBasic propose sa propre classe pour gérer des collections : Microsoft.VisualBasic.Collection

Méthodes

- Add(item, *clé, before,after*)
- Remove(index|clé)
- Item(index,clé)
- Count()

Attention : faiblement typée, nécessite un cast

Enumérateurs

```
Dim maCollection As New Collection
' Insert code to add elements to the customers
collection.
Dim monEnum As IEnumerator = maCollection.GetEnumerator()
monEnum.Reset()
Dim monElement As Object
While monEnum.MoveNext()
    monElement = monEnum.Current()
    ' Insert code to process this element of the
collection.
End While
```

Itérateurs

Permet un parcours personnalisé d'une collection

```
For Each letter In Letters()
    Console.Write(letter)
Next
…
Private Iterator Function Letters()As IEnumerable(Of
Char)
    Dim currentCharacter As Char = 'a'
    Do
        Yield currentCharacter
    Loop While (currentCharacter++ < 'z')
End Function
```

Entrées/Sorties

System.IO :
Regroupe des classes pour lire et écrire des données dans des fichiers ou des flux de données

Entrées/Sorties

- **Stream** : transfert de données
- **Principe d'utilisation d'un flux**:
  - Ouverture du flux
  - Identification de l'information (lecture/écriture)
  - Fermeture du flux

```
StreamReader sr AS
   new StreamReader(filename)
string s = sr.ReadToEnd()
sr.Close()
```

Exercices

- Écrire une méthode qui lit un fichier contenant un nombre par ligne et stocker ces nombres dans une collection de votre choix.
- Modifier les méthodes *minimum*, le *maximum* et *moyenne* pour effectuer les calculs sur la collection nouvellement créée.
- Écrire une méthode qui trie la collection et écrit le résultat dans un fichier.
- Tester ces méthodes.

Interfaces Graphiques (Windows Forms)

Interfaces graphiques

- Création d'un projet & Architecture de l'application
- Les références
- Le point d'entrée
- Les objets ApplicationContext et Application
- Lancement et arrêt de l'application
- Le fichier AssemblyInfo.cs

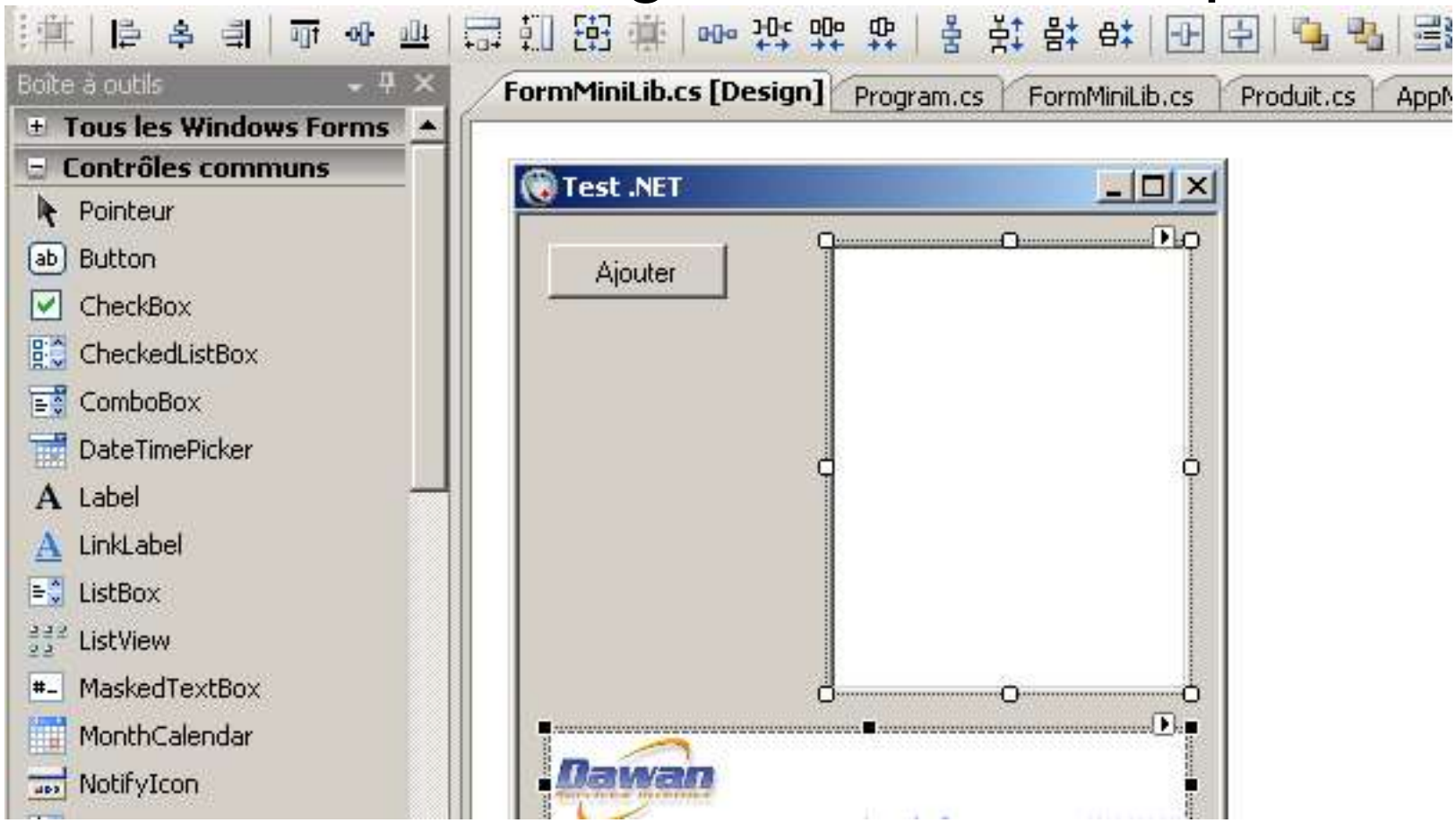
Types d'interfaces

Deux approches :

- WinForms : encapsulation des objets Win32
- WPF (Windows Presentation Foundation) : abstraction totale des contraintes de l'OS

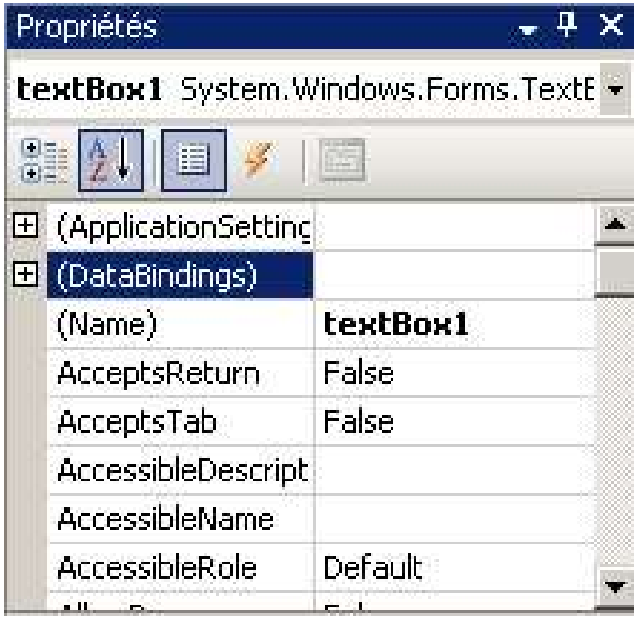
Manipulation

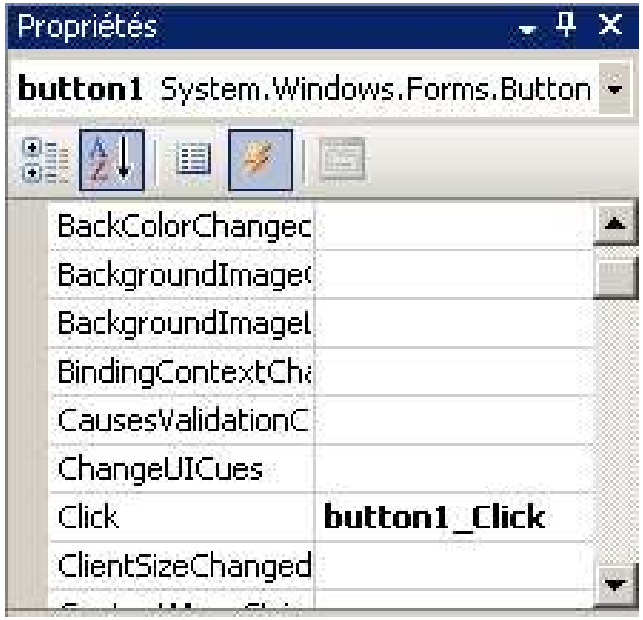
- Un éditeur complet est utile
- Le code est alors généré automatiquement



Manipulation : propriétés, événements

- Chaque composant a des propriétés et événements à disposition
- Par défaut les propriétés ont des valeurs standards (sauf : Name et Location), et les événements aucun délégué

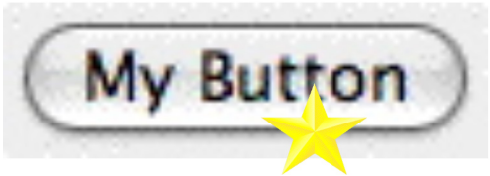




Manipulation : propriétés, événements

- Pour intercepter un événement, un listener doit être associé au composant
- Le listener est appelé lors de l'évènement
- Il peut y avoir plusieurs listener pour un même événement sur un même composant

Listener 1



Créations

- Base : les forms (fiche / fenêtre), associées à des fichiers de code
- Contrôles simples : Label, TextBox, Button etc...
- Gestions des événements (standards et spécifiques)
- Listes (ex : ListBox : Items, SelectedIndex)
- Menus (habituel et contextuel)
- OpenFileDialog et SaveFileDialog
- FontDialog, ColorDialog, FolderBrowserDialog, etc...

Configuration et Déploiement

- Fichier de configuration

```
<configuration>
  <appSettings>
    <add key="myLabel" value="Label prédéfini" />
  </appSettings>
</configuration>
```

```
Dim label As String =
   System.Configuration.ConfigurationManager.AppSett
   ings("myLabel")
```

- Déploiement WindowsInstaller
- ClickOnce