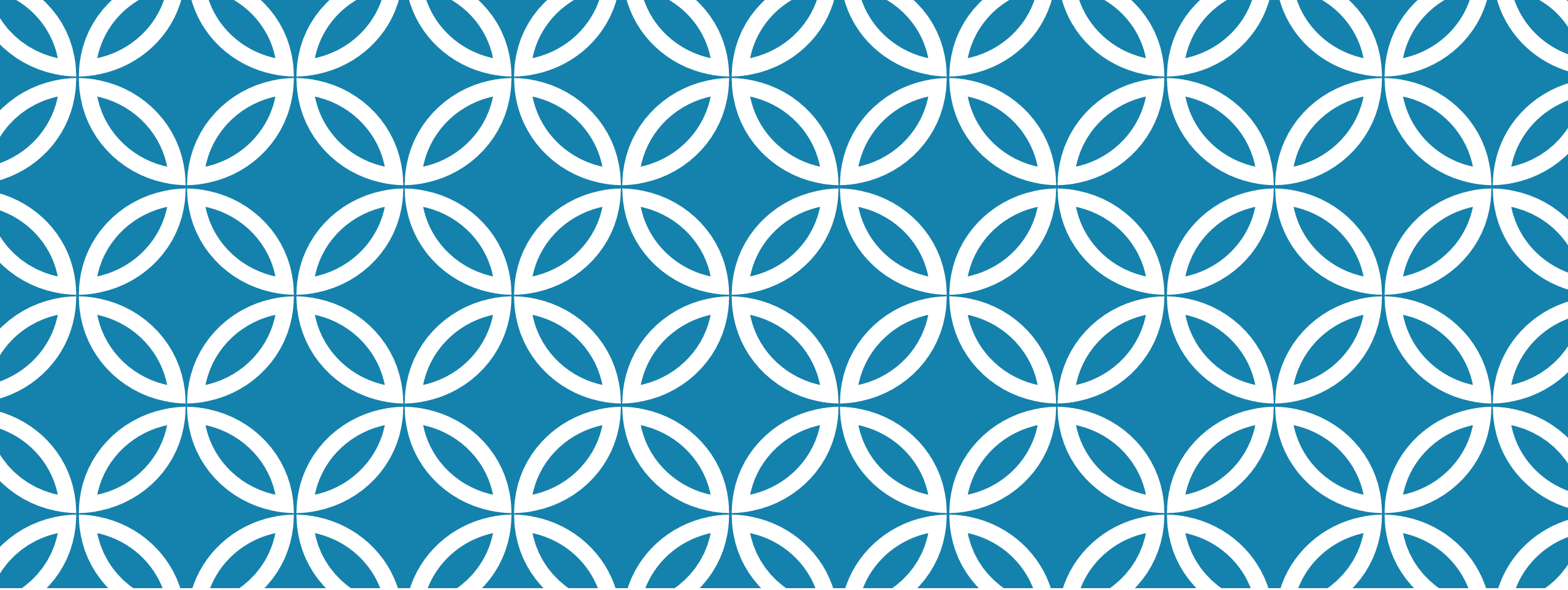


STRUCTURE UNE SOLUTION .NET

STRUCTURE D'UN PROGRAMME VISUAL BASIC

- Une solution peut contenir un ou plusieurs projets
- Un *projet* peut contenir un ou plusieurs assemblys
- Chaque *assembly* est compilé à partir d'un ou de plusieurs fichiers sources
- Un *fichier source* fournit la définition et l'implémentation des
  - classes,
  - structures,
  - Modules,
  - Interfaces

qui contiennent l'ensemble de votre code.

ÉLÉMENTS DE PROGRAMMATION AU NIVEAU FICHIER

instructions **Option ;**

- Explicit
- Strict
- Compare
- Infer

instructions **Imports**

- Pour référencer des classes et d'autres types définis à l'intérieur de l'espace de noms importé sans devoir les qualifier.

instructions **Namespace** et éléments au niveau de l'espace de noms.

ÉLÉMENTS DE PROGRAMMATION AU NIVEAU FICHIER

Directives de compilation

```
#If FrenchVersion Then
   ' <code spécifique à la version française du langage >.
#ElseIf GermanVersion Then
   ' < code spécifique à la version allemande du langage >.
#Else
        ' < code spécifique aux autres versions du langage >.
#End If
```

ÉLÉMENTS DE PROGRAMMATION AU NIVEAU NAMESPACE

1. Classes
2. Structures
3. Modules
   - contiennent l'ensemble du code de votre fichier source
4. Interface
   - définissent les signatures d'éléments mais ne fournissent aucune implémentation
5. Énumérations
6. Délégués
   - éléments de données

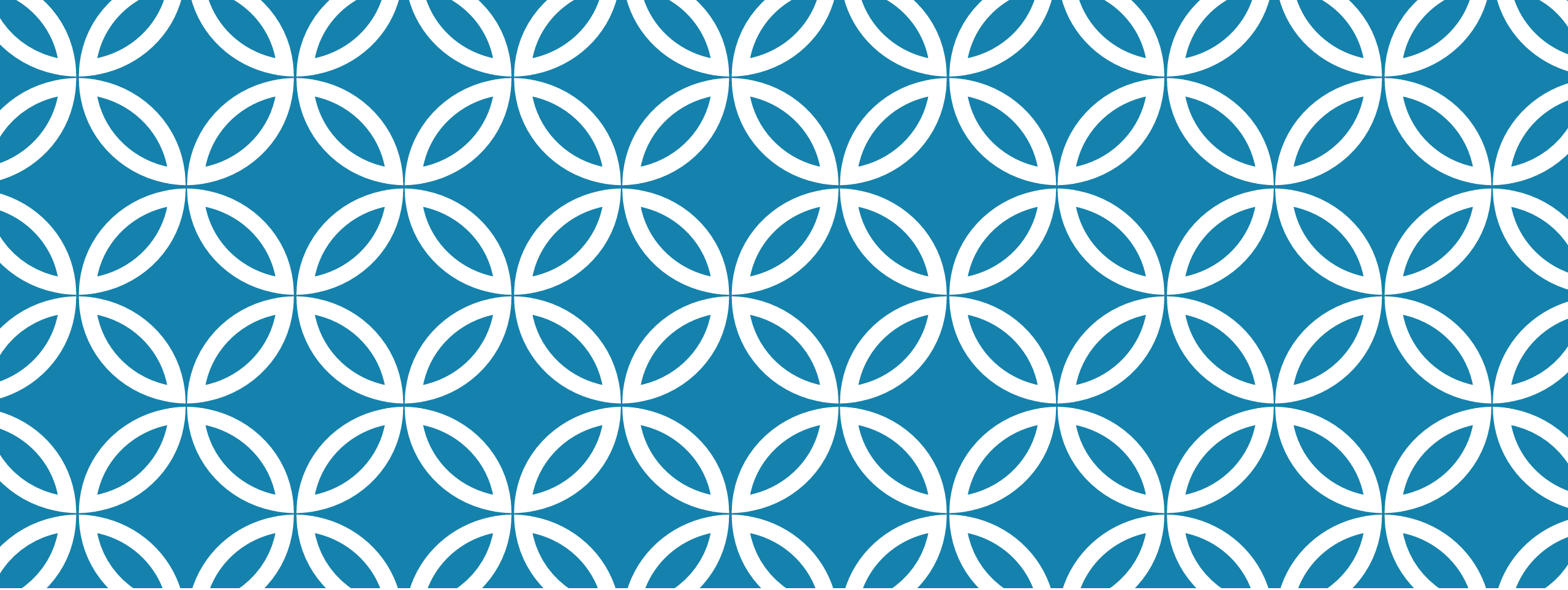
ÉLÉMENTS DE PROGRAMMATION AU NIVEAU MODULE

1. Function
2. Sub
3. Get
4. Set
5. Operator (Attention : Il doit être publique et partagé)
   - les seuls éléments de programmation qui peuvent contenir du code exécutable
6. Event
7. Delegate

Les éléments de données au niveau du module sont les variables, les constantes, les énumérations et les délégués.

ÉLÉMENTS DE PROGRAMMATION AU NIVEAU PROCÉDURE

1. Les instructions de déclaration
   - Les éléments de données au niveau de la procédure se limitent aux variables et aux constantes locales.
2. Les instructions d'exécution
   1. Appels aux procédures
   2. Assignations
   3. AddHandler
   4. RemoveHandler
   5. RaiseEvent
   6. Structure de contrôle



VB.NET
LE LANGAGE

IGI

PLAN

Les types de données

Variables

Les constantes et énumération

Les structures de contrôle

Les procédure et leurs paramètres

Portée de variables

Chaines de caractères

Tableaux

LES TYPES DE DONNÉES

Un type de données est un **type valeur** s'il contient des données dans l'espace qui lui est alloué en mémoire. Un **type référence** contient un pointeur vers un autre emplacement en mémoire contenant les données.

Sont des variables par 'Valeur' :

- Les Integer, les Long les Short ;
- Les Single, Double, Decimal ;
- Les Booleans, Char, Date ;
- Les Structures ;
- Les énumérations.

LES TYPES DE DONNÉES

La variable 'par Référence' ne contiennent pas la valeur de l'objet mais son adresse en mémoire, sa référence.

Sont des variables par référence :

- Les Objets ;
- Les Strings ;
- Les tableaux ;
- Les Classes.

LES TYPES DE DONNÉES

Le cas particulier des 'String'.

- Bien que les Strings soit par référence, B=A affecte simplement la valeur de A à B.
- c'est juste la définition de l'opérateur d'affectation "=" qui a été redéfinie pour la classe String.

Déclaration avec New ?

- En théorie, il faut utiliser New quand on déclare une variable 'par référence‘

Valeur après déclaration

- Apres création (avant initialisation) une variable numérique 'par Valeur' contient 0
- Par contre une String (par référence) qui a été créée par Dim et non initialisée contient 'Nothing'.

LES TYPES DE DONNÉES

Valeur après déclaration

- On peut tester par **If IsNothing(O)** then..

ou

- **If O Is Nothing.**

Comparaisons:

- Les variables par Valeur peut être comparée entre elles par les opérateurs de comparaison usuels (= ,…)
- Par contre une variable par référence peut être comparée à une autre par "Is".
- NB: pour les String '=' et 'Is' peuvent être utilisés

LES TYPES DE DONNÉES

Comparaisons:

- Equals peut être utilisé pour comparer les 2 types de données
  - **O1.Equals(O2) → Boolean**
- Il existe une instruction permettant de voir si une variable est de type 'Par référence‘: **IsReference**

LES TYPES DE DONNÉES

Le Type 'Object':

- Parfois on ne sait pas ce que va contenir une variable: un Integer? une String? un Single?
- on utilise le type 'Object'.
- Il remplace le type 'Variant' de VB6

Comment savoir quel type de variable contient la variable 'Objet'?

- **TypeOf O1 Is**
- **O1.GetType.ToString**

LES TYPES DE DONNÉES

Le Type Boolean:

- On a parfois besoin de savoir si une assertion est vraie ou Fausse.
- Pour stocker une information de ce type, on utilise une variable de type boolean.
- Une variable de ce type ne peut contenir que True ou False.

Les opérateurs logiques

- A **And** B retourne True si A et B sont vrais
- A **Or** B retourne True si une des 2 est vrai
- A **Xor** B retourne True si une et une seule est vrai
- **Not** A retourne True si A est faux et vice versa **IsNot** A

Les opérateurs **AndAlso** et **OrElse** sont plus rapides car ils n'évaluent pas la seconde expression si ce n'est pas nécessaire.

LES VARIABLES

Une variable permet de stocker une **valeur** (un nombre, du texte , une date, un objet…)

Une variable a

- un **nom**
- un **type**
- une **'portée**‘

LES VARIABLES

Avant d'utiliser une variable, il faut la déclarer, la créer, pour cela on utilise l'instruction Dim:

```
Dim a As Integer
```

```
Dim a, b, c As Integer
```

Il faut aussi parfois l'**initialiser,** `Dim a As Integer =3`

LES VARIABLES

Nom :	Contient :
Boolean	Contient une valeur Booléenne (logique): True ou False.
Byte	Contient les nombres entiers de 0 à 255 (sans signe)
Short	Entier signé sur 16 bits (-32768 à 32768)
Integer	Entier signé sur 32 bits (-2147483648 à 2147483647)
Long	Entier signé sur 64 bits (-9223372036854775808 à 9223372036854775807)
BigInteger	Entier signé très grand (sans limite supérieure ou inférieure) (VB2010)
Single	Nombre réel en virgule flottante (-1,401298 *10^-45 à 1,401298 10^45)
Double	Nombre réel en virgule flottante double précision. (..puissance 324)
Decimal	Nombre réel en virgule fixe grande précision sur 16 octets.
Char	1 caractère alphanumérique
String	Chaîne de caractère de longueur variable (jusqu'a 2 milliards de caractères)
DateTime	Date plus heure
Object	Peut contenir tous les types de variables mais aussi des contrôles, des fenêtres..
Structure	Ensemble de différentes variables définies par l'utilisateur.
UInteger	Entier codé sur 32 bits pouvant prendre les valeurs 0 à 4 294 967 295.
ULong	Entier codé sur 64 bits :0 à 18 446 744 073 709 551 615
Ushort	Entier sur 16 bits 0 à 65 535.
SByte	Byte mais signé. Codé sur 1 octet, valeur de -128 à 127
Complex	Nombre complexe (en VB2010)

LES VARIABLES

Type Nullable.

Les types Par Valeur peuvent être étendus afin d'accepter une valeur normale habituelle ou une valeur Null (Nothing en VB). On peut déclarer un type Nullable de 3 manières:

```
Dim MyInteger As Nullable (Of Integer)
Dim MyInteger? As Integer
Dim MyInteger As Integer?
```

LES CONSTANTES ET ÉNUMÉRATIONS

Constantes

- Comme les variables, elles ont un **nom** et un **type,** mais leurs valeurs sont '**constantes**'.
- On les déclare par le mot **Const,** on peut les initialiser en même temps avec =
- Exemple: Const NOMDUPROGRAMME= "LDF"

LES CONSTANTES ET ÉNUMÉRATIONS

Énumération

- Les énumérations sont utilisées lorsque l'on a un jeu de constantes liées logiquement.
- Un bloc **Enum** permet de créer une liste (une énumération) de constantes:

```
Enum TypeFichier
    DOC
    RTF
    TEXTE
End Enum
```

- Chaque constante littérale de l'énumération a une valeur par défaut.

LES CONSTANTES ET ÉNUMÉRATIONS

Exemples d'énumérations

- ConsoleColor
- Keys
- HorizontalAlignment
- VerticalAlignment
- …

LES STRUCTURES DE CONTRÔLE

Alternatives

- Permettent de créer des **structures décisionnelles**

  If Condition Then

  ….

  End if

- On peut écrire sur la même ligne après Then (Pas besoin de End If).
- Alternative complete

  If Condition then

  …

  Else

  …

  End If

- Alternative imbriquée

LES STRUCTURES DE CONTRÔLE

Alternatives

- **syntaxe avec 'Elseif':**

```
If Condition1 Then
…
ElseIf condition2 Then
…
ElseIf condition3 Then
…
end if
```

LES STRUCTURES DE CONTRÔLE

Select case

- Créer une structure décisionnelle permettant d'exécuter un grand nombre de blocs de code différents en fonction de la valeur d'une expression

```
Select Case expression
    Case v1'code effectué si expression=valeur1
    Case v2, v3 'code effectué si expression=v2 ou v3
    Case is > v8 'code effectué si expression est > à v8
    Case v5 To v20 'code effectué si expression=v5 jusqu'à v20
    Case Else 'code effectué dans tous les autres cas
End Select
```

LES STRUCTURES DE CONTRÔLE

For Next

- Les boucles sont très utilisées pour parcourir une plage de valeur qui permet par exemple de parcourir tous les éléments d'un tableau ou pour effectuer de manière itérative un calcul.
- Le nombre de boucle va être déterminé par une variable qui sert de compteur: la variable de boucle.
- **Le nombre d'exécution est déterminé au départ de la boucle** car le compteur a une valeur de départ, une valeur d'arrêt.

LES STRUCTURES DE CONTRÔLE

For Next

```
For variable=début To fin
…
Next variable.
```

- Il peut y avoir un pas (**Step**), le compteur s'incrémente de la valeur du pas à chaque boucle

```
For variable=début To fin  step p
…
Next variable.
```

- On peut quitter prématurément la boucle avec **Exit For**.
- Le nom de la variable de boucle est facultatif après Next
- **Continue For** permet de sauter au prochain Next et de poursuivre la boucle.

LES STRUCTURES DE CONTRÔLE

Do Loop

- Permet aussi de faire des boucles mais **sans que le nombre de boucle (d'itération) soit déterminé au départ**

```
Do
…
Loop
```

On doit mettre **Until** (Jusqu'à ce que) ou **While** (Tant que) avant la condition d'arrêt pour sortir de la boucle.

On peut mettre la condition après Do:

```
Do
…
Loop while|Until Condition.
```

- Il y a **Exit Do** pour sortir de la boucle;
- il existe aussi **Continue Do** qui permet de sauter au prochain Loop et de poursuivre la boucle..

LES STRUCTURES DE CONTRÔLE

While End While

```
While Condition
...
End While
```

- Il y a **Exit while** pour sortir de la boucle;
- il existe aussi **Continue While** qui permet de sauter au prochain Loop et de poursuivre la boucle..

LES STRUCTURES DE CONTRÔLE

For Each

- C'est une variante de la boucle For mais elle **permet de parcourir les objets d'une collection.** Elle n'utilise pas l'indice

```
Dim mystring As String
For Each item As Objet in ListBox.items
   mystring=mystring+item
Next
```

- **Attention, dans une boucle For Each, on peut parcourir la collection mais on ne peut pas modifier un élément de la collection.**

LES STRUCTURES DE CONTRÔLE

Switch

- Switch est utilisé avec des couples d'arguments, si le premier est vrai, le second est retourné.

  Réponse=Switch(Expression1, Reponse1, Expression2, Reponse2)

- Si Expression2 est vrai Reponse2 est retourné.

Monnaie= Microsoft.VisualBasic.Switch(Pays = "USA", "Dollar", Pays = "Europe", "Euro", Pays = "Angleterre", "Livre")

- Si Pays="FRANCE", cette expression est vrai, le second objet du couple est retourné.
- Retourne Euro

LES STRUCTURES DE CONTRÔLE

IIF

- IIf est utilisé avec 3 arguments.
- Si le premier argument est vrai, le second est retourné.
- Si le premier argument est faux c'est le troisième qui est retourné.

Reponse = IIf(Nombre > 0, "Positif", "Négatif ou nul")

LES PROCÉDURES ET LEURS PARAMÈTRES

En VB les procédures sont des **Sub** ou des **Function**

Les procédures Sub: débutent par le mot Sub et se terminent par End Sub. Elles peuvent avoir des paramètres mais ne 'retournent' rien.

Les procédures Function: Si on a besoin que la procédure retourne un résultat (un seul), on utilise une Fonction. Elles débutent par Function et se terminent par End Function et contiennent obligatoirement le mot Return, au moins une fois, suivi de l'objet à retourner

LES PROCÉDURES ET LEURS PARAMÈTRES

Par Valeur, Par Référence:

- Il y a 2 manières d'envoyer des paramètres :
- **Par valeur** : (ByVal)c'est la valeur (le contenu de la variable) qui est envoyée. (La variable est copiée dans une autre partie de la mémoire pour être utilisée par la routine appelée.)
- **Par référence** :(ByRef) c'est l'adresse (le lieu physique ou se trouve la variable) qui est envoyée. Si la Sub modifie la variable, cette modification sera visible dans la procédure appelante après le retour.

LES PROCÉDURES ET LEURS PARAMÈTRES

Par Valeur, Par Référence:

- Un paramètre ou argument peut être Optional, c'est à dire facultatif.

Sub MaRoutine (Optional X As Integer=0)

- Parfois il faut envoyer des paramètres de même type mais dont on ne connaît pas le nombre, dans ce cas on utilise**ParamArray** (Liste de paramètres):

Function Somme (ByVal ParamArray Valeurs() as Integer) As Integer

PORTÉE DES PROCÉDURES

Le terme Sub ou Function peut être précédé d'une indication de portée; la procédure sera t-elle visible uniquement dans le module où elle se trouve ou partout?

La procédure peut être **Private**. Dans ce cas on ne peut l'appeler qu'à partir du module qui la contient.

La procédure peut être **Public**. Dans ce cas on pourra l'appeler à partir de la totalité du programme.

PORTÉE DES VARIABLES

Il est souvent important de rendre une variable visible dans une procédure mais pas les autres, dans un seul module ou dans la totalité du programme

Dans les procédures

- Si on déclare une variable dans une procédure (une Sub ou une Function) à l'aide du mot clé Dim, elle est visible uniquement dans cette procédure, c'est une variable locale.
- Si à la place de Dim on utilise **Static**, la variable est dite 'Statique': A la sortie de la procédure, la variable et sa valeur continue d'exister et on garde sa valeur en mémoire; lors des appels suivants de la procédure, on retrouve la valeur de la variable.

PORTÉE DES VARIABLES

Dans un bloc d'instructions

- Si vous déclarez une variable dans un bloc, elle ne sera visible que dans ce bloc.
- Si à la place de Dim on utilise **Static**, la variable est dite 'Statique': A la sortie de la procédure, la variable et sa valeur continue d'exister et on garde sa valeur en mémoire; lors des appels suivants de la procédure, on retrouve la valeur de la variable.

LES CHAÎNES ET LES CARACTÈRES

Déclaration de chaînes

```
Dim str As String
```

Déclaration & initialisation

```
Dim A As String= "Visual"
```

Le type System.String ou String est une Classe du Framework, qui a des méthodes.

- **ToUpper**
- **ToLower**
- **Trim**
- …

LES CHAÎNES ET LES CARACTÈRES

Comparaison

- On peut comparer 2 String avec: =, <>, <, >

Comparaison avec Equals et String.Compare

- **Equals** retourne un Boolean égal à True si les 2 chaines sont égales.
- **String.Compare** compare 2 chaines et retourne un Integer qui prend la valeur:
  0 si les 2 chaines sont égales.
  inférieur à 0 si string1 est inférieur à string2.
  supérieur à 0 si string1 est supérieur à string2.

LES CHAÎNES ET LES CARACTÈRES

Une variable Char contient un caractère et un seul stocké sous forme d'un nombre sur16 bits.

Après déclaration, une variable Char contient " c'est à dire un caractère vide.

L'ajout du caractère 'c' à un littéral de chaîne force ce dernier à être un type Char.

```
Option Strict On '
Dim C As Char
C = "A"c
'Autre manière de faire:
C=CChar("A") 'On converti la String "A" en Char
```

LES CHAÎNES ET LES CARACTÈRES

String.ToCharArray: Permet de passer une string dans un tableau de Char.

```
Dim maString As String = "abcdefghijklmnop"
Dim maArray As Char() = maString.ToCharArray
```

Methodes de la classe Char

Isdigit

Isletter

IsNumber

IsControl

…

LES CHAÎNES ET LES CARACTÈRES

Conversion Numérique vers String:

- i.ToString()
- *Cstr(i)*

Conversion String vers numérique

- *Cint(s)*
- *Parse : Integer.Parse(s)*

Utilisation de Ctype

- *Ctype(ObjetTypeOrigine, TypeCible)* → *ObjetTypeCible*

LES TABLEAUX

Comment déclarer un tableau ?

Dim Tableau(3) As Integer

- Cette declaration crée un tableau à 4 éléments.
- Noter que comme c'est un tableau d'entier, juste après la création du tableau les éléments sont initialisés à 0.
- Le tableau commence toujours par l'indice 0.
- Tableau(1)= 12 permet d'affecter le nombre 12 au 2eme élément du tableau.
- Un tableau peut avoir plusieurs dimensions :

Dim T(2,2) ' 3 X 3 éléments

- On peut créer des tableaux de tableaux:

Dim T(2),(2)

LES TABLEAUX

Comment initialiser un tableau ?

```
Dim Mois() As String ={"Janvier","Février","Mars"}
' Crée un tableau de type String().
Dim winterMonths = {"December", "January", "February"}
' Crée un tableau de type Integer()
Dim numbers = {1, 2, 3, 4, 5}
'Crée un tableau de Double
Dim b = {1, 2, 3.5}
'Attention création d'un tableau d'OBJECT
Dim d = {1, "123"}
'Création de tableau à plusieurs dimensions et de tableau de tableau
Dim e = {{1, 2, 3}, {4, 5, 6}}              'Integer(,)
Dim f = {({1, 2, 3}), ({4, 5, 6})}          'Integer()()
```

LES TABLEAUX

Autres syntaxes ?

```
'Déclaration
Dim t As String()
'On instancie et on initialise
t = New String(1) {"One", "Two"}
' on affecte au tableau un nouveau tableau de String contenant "One" et "Two"
Dim R(,) as Integer ={{0, 1}, {1, 2}, {0, 0}, {2, 3}}
```

LES TABLEAUX

Redim permet de redimensionner un tableau (modifier le nombre d'éléments d'un tableau existant), si on ajoute **Preserve** les anciennes valeurs seront conservées.

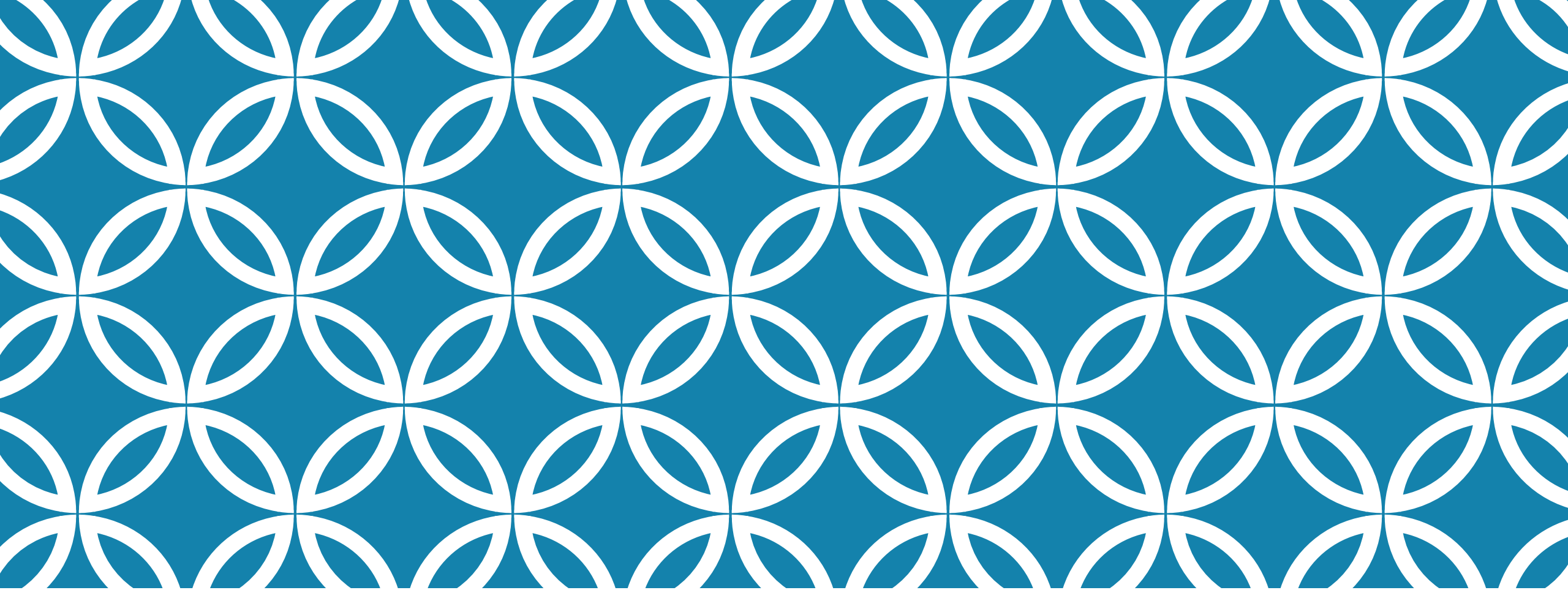
Attention, on ne peut pas modifier le nombre de dimension, ni le type des données.

Dim T(20,20) As String

...

Redim Preserve T(30,30)

.



POO

Visual Basic .net

QU'EST CE QU'UNE CLASSE

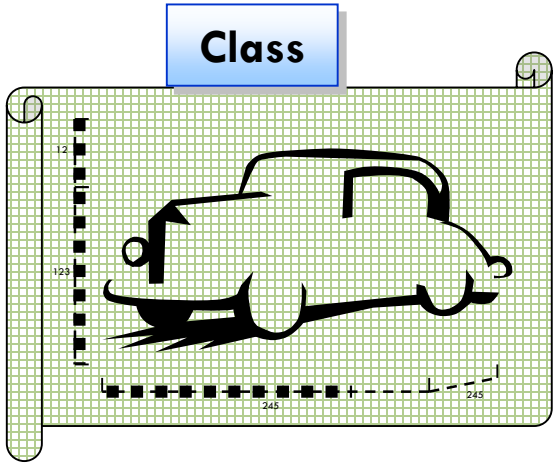
- Une classe est un modèle qui décrit un type d'objets et définit les attributs et les opérations de l'objet
- Les Classes utilisent l'abstraction pour ne mettre à disposition que les éléments essentiels à la définition de l'objet
- Les Classes utilisent l'encapsulation pour appliquer une abstraction

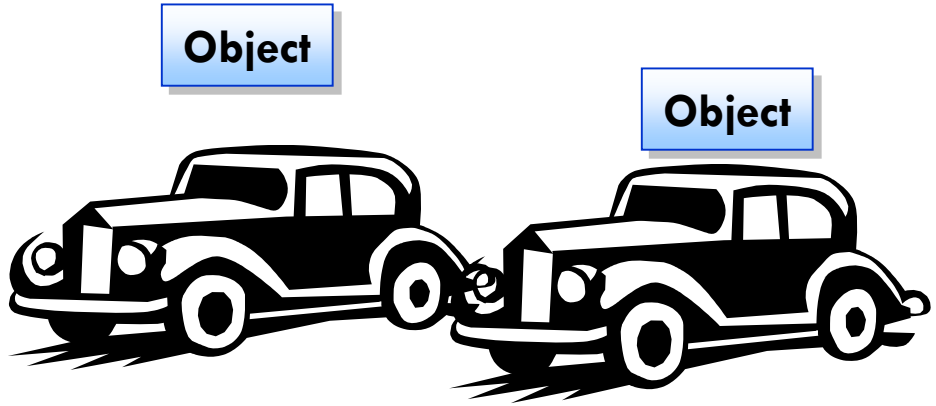
QU'EST CE QU'UN OBJET?

Un objet est une instance d'une classe

Les objets ont les qualités suivantes:

- Identité: objets se distinguent les uns des autres
- Comportement: Les objets peuvent effectuer des tâches
- État: Objets stockent des informations qui peuvent varier dans le temps





PROCÉDURE DE DÉFINITION DE CLASSES

Ajouter une classe

Assigner un nom approprié

Créer les constructeur au besoin

Créer un destructeur au besoin

Déclarer les propriétés

Déclarer les méthodes et événements

UTILISATION DES MODIFICATEUR D'ACCÈS

Spécifier les droit d'Accès aux Variables et Procédures

Public : Accessible partout dans la solution.

Private : Accessible seulement dans le type lui même.

Friend : Accessible dans le type lui même et tous les namespaces et code du même assembly.

Protected : Accessible dans la classe et toute classe qui en dérive.

Protected Friend : La reunion de **Protected** et **Friend.**

UTILISATION DES ATTRIBUTS

- Métadonnées supplémentaires fournies au travers de "<>"
- Pris en charge par:
  - Assemblées, classes, méthodes, propriétés, et plus
- Utilisations courantes:
  - versions des Assemblée , les services Web, les composants, la sécurité, et la personnalisation

```
<Obsolete("SVP utilizez la méthode M2")> _
Public Sub M1( )
   'le message apparait dans l'DE une fois utilisée dans le code
End Sub
```

LA SURCHARGE DES MÉTHODES

Méthodes avec le même nom peuvent accepter des paramètres différents

```
Public Function Display(s As String) As String
    MsgBox("String: " & s)
    Return "String"
End Sub
Public Function Display(i As Integer) As Integer
    MsgBox("Integer: " & i)
    Return 1
End Function
```

Les paramètres spécifiés déterminent la méthode appelée

Le mot-clé Overloads est facultatif à moins de surcharger les méthodes héritées

UTILISATION DES CONSTRUCTEURS

- **Sub New(…)**
- **Code exécuté quand l'objet est Instancié**

```
Public Sub New( )
    'Perform simple initialization
    intValue = 1
End Sub
```

- **Peut être surchargé, mais sans utiliser le mot-clé OverLoads**

```
Public Sub New(ByVal i As Integer) 'Overloaded without Overloads
    'Perform more complex initialization
    intValue = i
End Sub
```

UTILISATION DES DESTRUCTEURS

- **Sub Finalize()**
- Utilisé pour faire le “ménage” si nécessaire
- Code exécuté quand le Garbage Collector détruit l’objet
  - Attention: la destruction peut ne pas se produire immédiatement après le déclin de l’objet

```
Protected Overrides Sub Finalize( )
    'Can close connections or other resources
    conn.Close
End Sub
```

INSTANTIATION & INITIALISATION D'OBJECTS

- Déclarer puis instancier

```
Dim C1 As TestClass
c1 = New TestClass () 'Instantiation
```

- Déclarer, instancier et initialiser en utilisant le constructeur par défaut

```
Dim c2 As TestClass = New TestClass ()
```

- Déclarer, instancier et initialiser en utilisant constructeur par défaut

```
Dim c3 As New TestClass ()
```

- Déclarer, instancier et initialiser en utilisant constructeur de remplacement

```
C4 Dim As New TestClass (10)
C5 Dim As TestClass = New TestClass (10)
```

GARBAGE COLLECTION

Processus de fond qui nettoie les variables inutilisées

- Utilisez x = Nothing pour Activer le GC

Détecte les objets en mémoire qui ne sont pas référencés

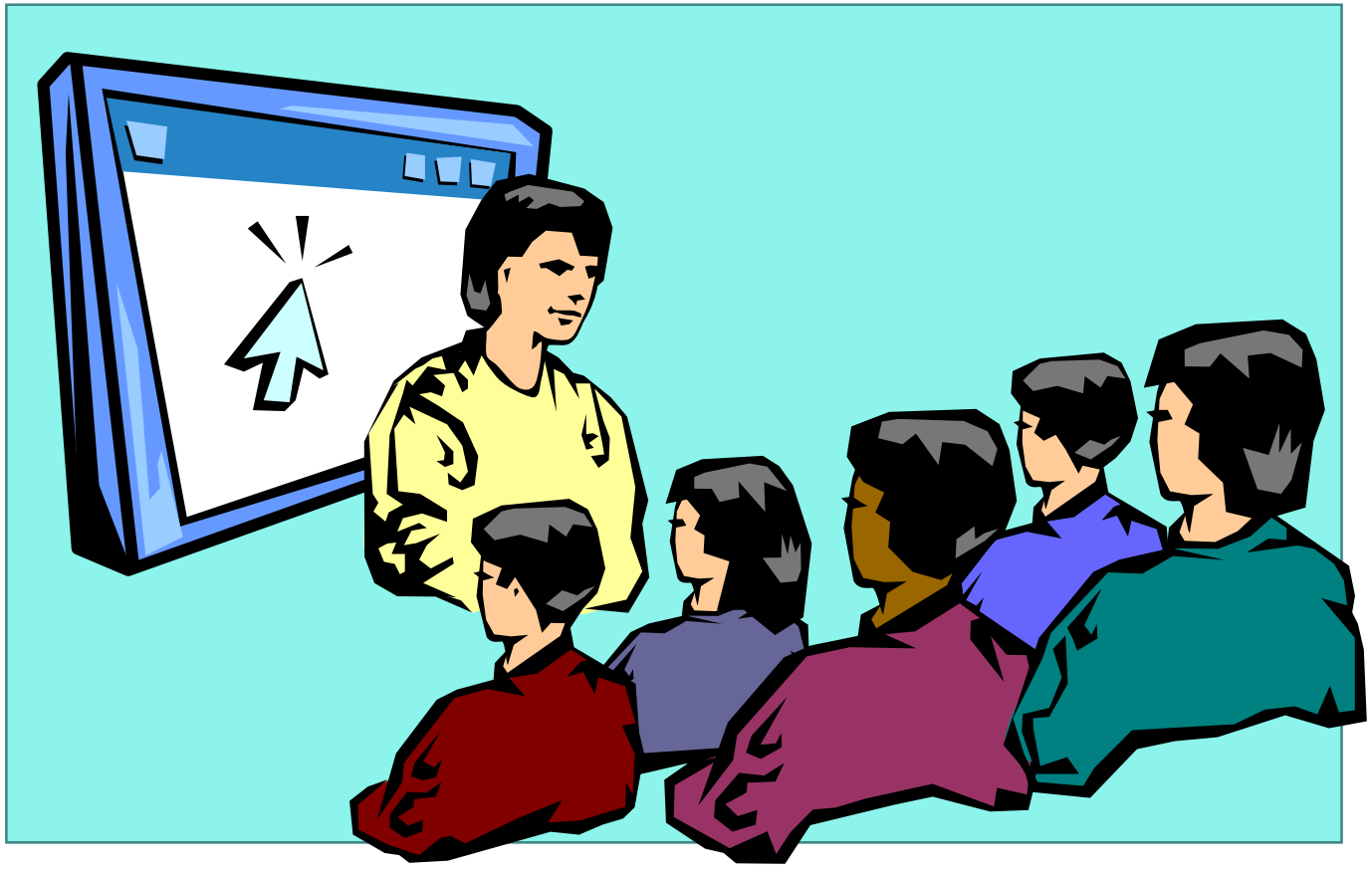
Appelle le destructeur de l'objet

- quand cela se produira n'est pas bien déterminé
- Les ressources potentielles peuvent être attachées pendant de longues périodes de temps (connexions de base de données, fichiers, etc.)

Vous pouvez forcer le nettoyage en utilisant la classe GC du system

- System.GC.Collect()

DEMONSTRATION: CRÉATION DE CLASSES



L'HÉRITAGE

C'est quoi l'héritage?

La redéfinition et la surcharge

Exemple

Masquage (shadowing)

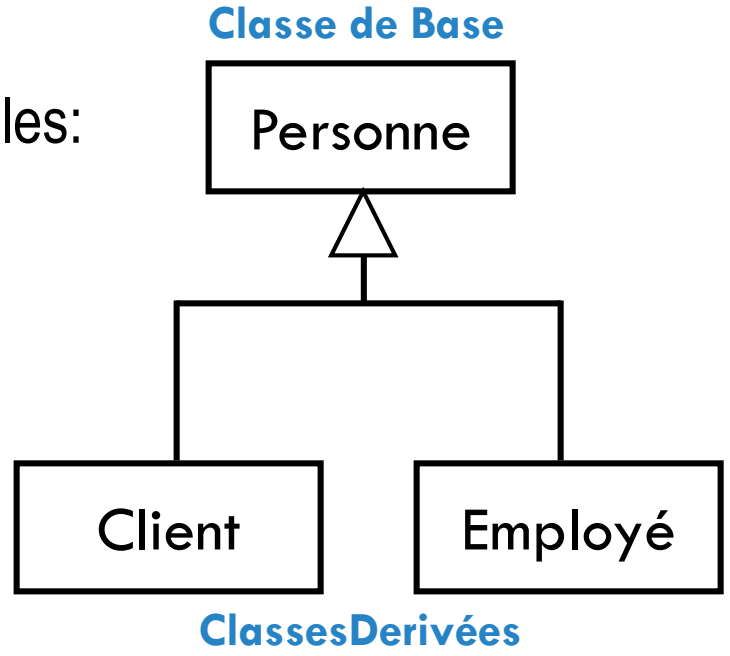
Utilisation du mot-clé MyBase

Utilisation du mot-clé MyClass

L'HÉRITAGE?

- **Héritage spécifie une relation «est-une-sorte-de"**
- **Plusieurs classes partagent les mêmes attributs et les opérations, permettant la réutilisation efficace du code**

Exemples:



- **Un client "est une sorte" de "personne"**
- **Un employé "est une sorte" de "personne"**

L'HÉRITAGE?

- **Classe dérivée hérite d'une classe de base**
- **Propriétés, méthodes, données membres, événements et des gestionnaires d'événements peuvent être hérités (selon la portée)**
- **Mots clés**
  - **Inherits** hérite d'une classe de base
  - **NotInheritable -** ne peut être classe de base
  - **MustInherit -** instances de la classe ne peuvent pas être créé; elle doit être obligatoirement une classe de base
  - **Protected -** Portée membre qui permet d'utiliser seulement en dérivant des classes -

REDÉFINITION ET SURCHARGE

- Classe dérivée peut remplacer une propriété ou une méthode héritée
    - **Overridable** - peut être redéfinit
    - **MustOverride** - doit être redéfini dans la classe dérivée
    - **Overrides-** redéfinit la méthode de la classe héritée
    - **NotOverridable** - ne peut être redéfini (par défaut)
- Utilisez le mot-clé **OverLoads –** pour surcharger ou une méthode ou une propriété héritée

EXEMPLE

```
Public Class BaseClass

    Public Overridable Sub OverrideMethod( )
        MsgBox("Base OverrideMethod")
    End Sub

    Public Sub Other( )
        MsgBox("Base Other method - not overridable")
    End Sub
End Class
```

```
Public Class DerivedClass
    Inherits BaseClass

    Public Overrides Sub OverrideMethod( )
        MsgBox("Derived OverrideMethod")
    End Sub
End Class
```

```
Dim x As DerivedClass = New DerivedClass( )
x.Other          'Displays "Base Other method - not overridable"
x.OverrideMethod   'Displays "Derived OverrideMethod"
```

MASQUAGE (SHADOWING)

- **Masque Membres de la Class de Base même s'ils sont surcharges**

```
Class CBase
    Public Sub M1( )   'Non-overridable pae défault
    ...
    End Sub
End Class

Class CMasquante
    Inherits aBase
    Public Shadows Sub M1(ByVal i As Integer)
    'Clients peut voir seulement cette methode
    ...
    End Sub
End Class
```

```
Dim x As New CMasquante( )
x.M1( )    'Génère une erreur
x.M1(20)   'No error
```

MYBASE

- Se réfère à la classe de base immédiate
- Peut accéder seulement aux Membres publiques, protégés, ou ami de la classe de base
- N'est pas un objet réel (ne peut pas être stocké dans une variable)

```
Public Class DerivedClass
    Inherits BaseClass

    Public Overrides Sub OverrideMethod( )
        MsgBox("Derived OverrideMethod")
        MyBase.OverrideMethod( )
    End Sub
End Class
```

MYCLASS

Assure que la classe de base est appelée, et pas la classe dérivée

```
Public Class BaseClass
    Public Overridable Sub OverrideMethod( )
        MsgBox("Base OverrideMethod")
    End Sub

    Public Sub Other( )
        MyClass.OverrideMethod( )'Will call above method
        OverrideMethod( )     'Will call derived method
    End Sub
End Class
```

```
Dim x As DerivedClass = New DerivedClass( )
x.Other( )
```

INTERFACES

Definition des Interfaces

La mise en œuvre du Polymorphisme

DEFINING INTERFACES

- Interfaces définissent les signatures publiques des procédure, propriétés et événements
- Utilise le mot-clé Interface pour définir une Interface
- Peut surcharger ses Membres comme pour les Classes

```
Interface IMyInterface
    Function Method1(ByRef s As String) As Boolean
    Sub Method2( )
    Sub Method2(ByVal i As Integer)
End Interface
```

- Utilise le mot-clé **Inherits** pour hériter d'une autre Interfaces

POLYMORPHISME

Polymorphisme

- Plusieurs classes fournissent une même propriété ou méthode
- Le client n'a pas besoin de savoir le type de la classe sur lequel s'est basé un objet

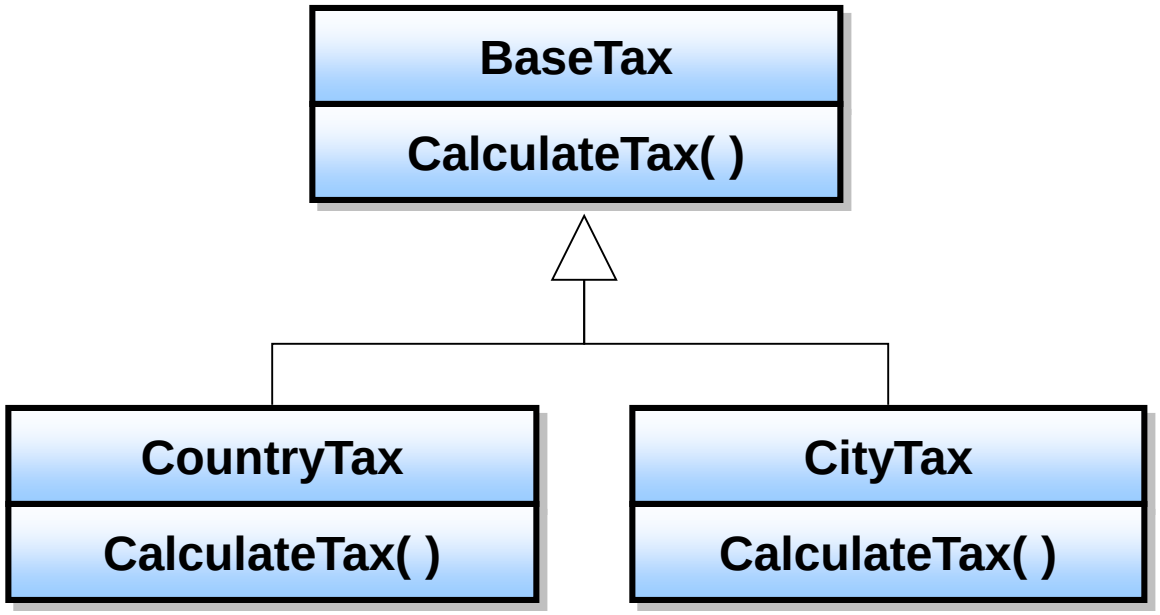
Deux Approches

- Interfaces
  - Classes implémentent les membres d'une interface
- Héritage
  - Les classes dérivées redéfinissent les membres de la classe de base

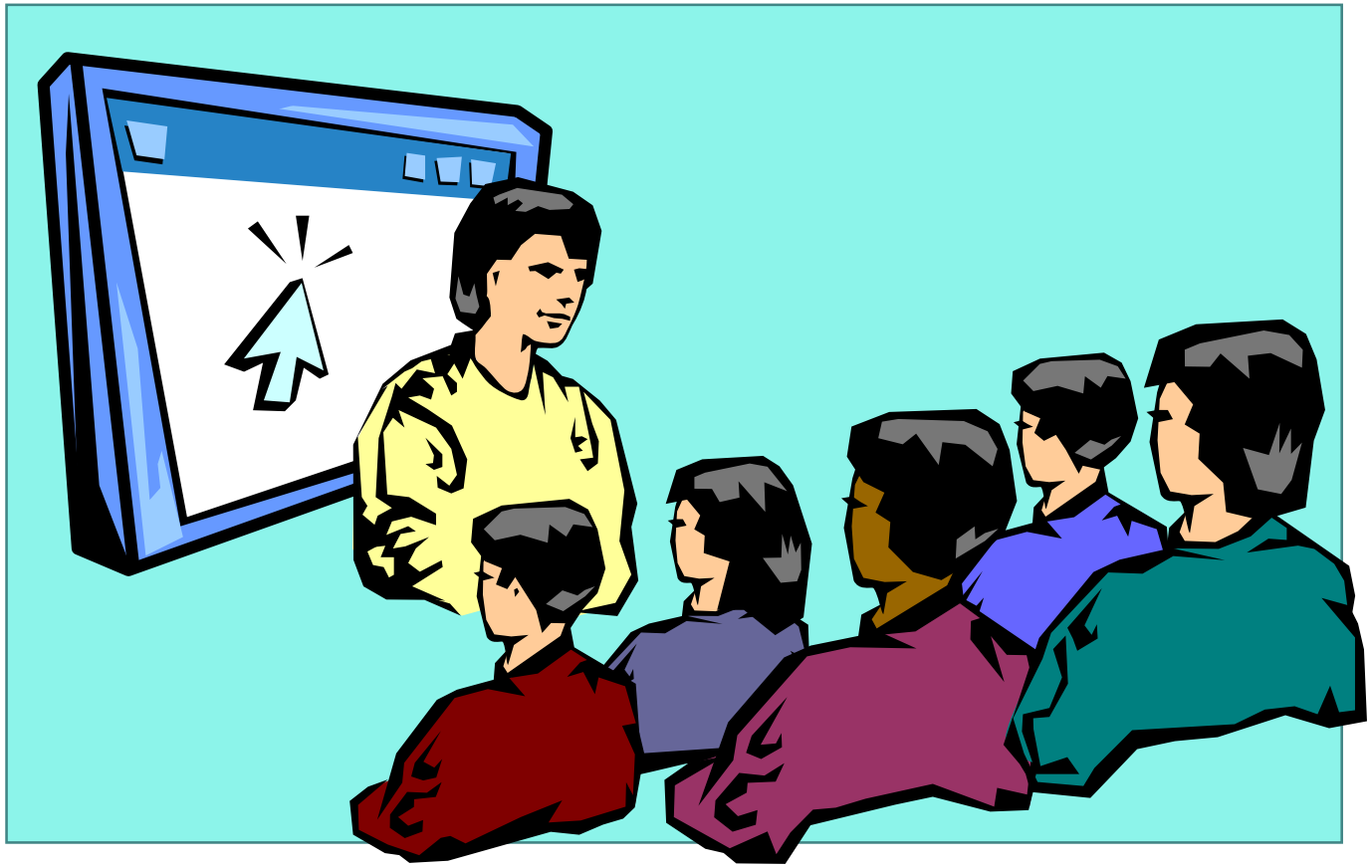
POLYMORPHISME?

Le nom de la méthode réside dans la classe de base

L'implémentation de la méthode réside dans les classes dérivées



DEMO: INTERFACES & POLYMORPHISME



◆ LES CLASSES

Utilisation des données membres partagées

Utilisation des méthodes membres partagées

La gestion des événements

Délégués?

Utilisation des Délégués

Comparaison entre Classes et Structures

DONNÉES MEMBRES PARTAGÉES

Permet à plusieurs Instances d'une classe de se référer à une variable en commun

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

    Sub New(ByVal strName As String, ByVal dblAmount As Double)
        Name = strName
        Balance = dblAmount
    End Sub

    Public Function CalculateInterest( ) As Double
        Return Balance * InterestRate
    End Function
End Class
```

```
SavingsAccount.InterestRate = 0.003
Dim acct1 As New SavingsAccount("Joe Howard", 10000)
MsgBox(acct1.CalculateInterest, , "Interest for " & acct1.Name)
```

MÉTHODES MEMBRES PARTAGÉES

- Permet d'utiliser une méthode membre sans instancier la classe
- Un méthode partagée ne peut pas accéder qu'aux données membres partagées

```
'TestClass code
Public Shared Function GetComputerName( ) As String
    ...
End Function
```

```
'Client code

MsgBox(TestClass.GetComputerName( ))
```

GESTION D'ÉVÉNEMENTS

Définir et déclenche des événements

WithEvents et **Handles** :

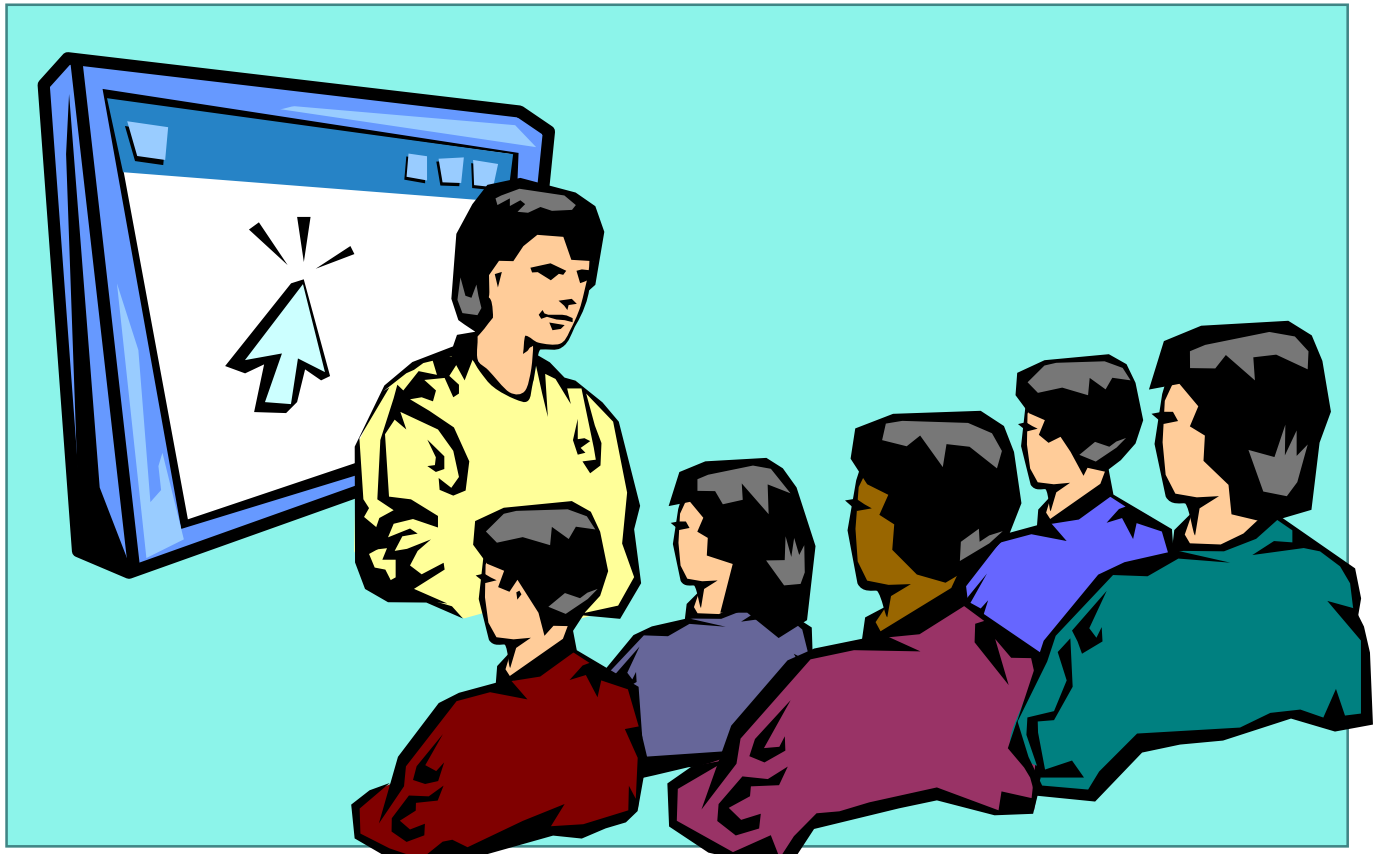
AddHandler : Connexion Dynamique à un événement

```
Dim x As New TestClass( ), y As New TestClass( )
AddHandler x.anEvent, AddressOf HandleEvent
AddHandler y.anEvent, AddressOf HandleEvent
...

Sub HandleEvent(ByVal i As Integer)
    ...
End Sub
```

RemoveHandler : Déconnecte d'une source d'événement

DEMONSTRATION: GÉRER LES ÉVÉNEMENTS



DÉLÉGUÉS?

Objets qui appellant les Methodes d'un autre Objets

Basé sur la classe System.Delegate

Type-safe, Secure, Managed Objects

Example:

- Utile comme intermédiaire entre une procédure d'appel et la procédure appellée

DÉLÉGUÉS

- Delegate déclare un Délégué et définit ses paramètres et le type retourné

```
Delegate Function CompareFunc( _
    ByVal x As Integer, ByVal y As Integer) As Boolean
```

- Méthodes doivent avoir la même signature
- Utilise la Méthode Invoke du Delegate pour appeler les Méthodes

CLASSES VS STRUCTURES

Classes	**Structures**
Peut définir des données membres, propriétés et méthodes	Peut définir des données membres, propriétés et méthodes
Prise en charge des constructeurs et de l'initialisation des membres	Pas de constructeur par défaut ou l'initialisation des membres
Supporte la méthode **Finalize**	Ne supporte pas la méthode **Finalize** ; implémente **IDisposable**
Extensible par héritage	Pas d'héritage
Type Référence	type Valeur