CHƯƠNG 3: NHỮNG CẢI TIẾN VỀ NGÔN NGỮ VÀ CÂU LỆNH VB.NET

Phan Trọng Tiến

BM Công nghệ phần mềm

Khoa Công nghệ thông tin, VNUA

Email: phantien84@gmail.com

Website: http://timoday.edu.vn

Nội dung chính

1. Tổng quan
2. Các kiểu dữ liệu
3. Sử dụng các biến
4. Demo: Sử dụng biến và các cấu trúc dữ liệu
5. Functions, Subroutines, and Properties
6. Demo: Làm việc với biến và các thủ tục
7. Xử lý ngoại lệ (Exception Handing)
8. Demo: Cấu trúc xử lý ngoại lệ
9. Lab: Thực hiện các cấu trúc xử lý ngoại lệ

1. Tổng quan

- VB.Net giới thiệu nhiều cải tiến về ngôn ngữ và cú pháp giúp phát triển một cách tốt nhất:
  - Kết hợp chặt chẽ với các kiểu hệ thống trong .Net Framework làm VB.Net tương thích với các ngôn ngữ khác.
  - Các cải tiến về cú pháp với các biến làm tăng sự sáng sủa và thực thi code.
  - Các thay đổi về **Functions**, **Subroutines** và **Properties** làm code dễ đọc và bảo trì
  - Cấu trúc xử lý ngoại lệ, làm ngôn ngữ VB.Net trở nên mạnh mẽ

Kết quả cần đạt được

- Mô tả thay đổi các kiểu dữ liệu trong VB.Net
- Khai báo và khởi tạo các biến và mảng
- Dùng câu lệnh rút gọn để khởi gán giá trị cho biến
- Thực thi Functions và Subroutines
- Gọi các Properties của một đối tượng
- Sử dụng cú pháp ***Try … Catch … Finally*** để thực hiện xử lý ngoại lệ

2. Các kiểu dữ liệu

- Kiểu dữ liệu hệ thống
- So sánh tham biến (*ByRef*) và tham trị (*ByVal*)
- Các kiểu dữ liệu mới
- Thay đổi với các kiểu dữ liệu tồn tại
- Sử dụng *CType* để chuyển đổi từ kiểu dữ liệu này sang kiểu dữ liệu khác

Kiểu dữ liệu hệ thống

- Được tích hợp trong CLS
- Chia sẻ lúc chạy (Runtime), biên dịch (compiler) và các Tool.
- Điều khiển cách khai báo, sử dụng và quản lý các kiểu dữ liệu lúc chạy
- Bao gồm tập hợp các kiểu dữ liệu tự định nghĩa
- Các kiểu dữ liệu hệ thống thông thường được kế thừa từ lớp System.Object

So sánh tham biến và tham trị

- Các biến kiểu tham trị (Value – Type)
  - Chứa dữ liệu trực tiếp
  - Có bản sao là dữ liệu của chính nó
  - Các thao tác trên 1 biến không ảnh hưởng đến biến khác
  - Câu lệnh gán tạo một bản sao dữ liệu
- Các biến kiểu tham biến (Reference – Type)
  - Lưu trữ tham chiếu tới dữ liệu (các đối tượng tham chiếu đã biết)
  - Hai tham chiếu có thể tham chiếu đến cùng một đối tượng
  - Các thao tác trên một biến có thể ảnh hưởng đến biến khác

Các kiểu dữ liệu mới

Visual Basic .NET data type	Storage size	Value range
Char	2 bytes	0 to 65535 (unsigned)
Short	2 bytes	-32,768 to 32,767
Decimal	12 bytes	Up to 28 digits on either side of decimal (signed)

Thay đổi với các kiểu dữ liệu tồn tại

Visual Basic 6.0	Visual Basic .NET
Integer	**Short**
Long (32 bits, signed)	**Integer**
(none)	**Long** (64 bits, signed)
Variant	Not supported: use **Object**
Currency	Not supported: use **Decimal**
Date	No longer stored as a **Double**
String (fixed length)	Not supported

Sử dụng *CType* để chuyển đổi từ kiểu dữ liệu này sang kiểu dữ liệu khác

- Dùng CType để chuyển đổi các giá trị từ một kiểu này sang một kiểu khác.
- Tương tự như hàm CStr và CInt trong VB6
- Cú pháp:
  - CType(*expression, typename*)
- Ví dụ:

```
Dim x As String, y As Integer
x = "34"
y = CType(x, Integer)
```

3. Sử dụng các biến

- Khai báo và sử dụng biến và mảng
- Khai báo nhiều biến
- Phạm vi của biến
- Tạo các kiểu dữ liệu có cấu trúc
- Các tùy chọn biên dịch
- Các toán tử gán

Dùng biến

- Để lưu trữ dữ liệu, một ngôn ngữ lập trình dùng biến. Biến là vị trí bộ nhớ tạm trong máy. Một biến có tên biến và kiểu dữ liệu của nó.
- VB.Net cung cấp các kiểu dữ liệu khác nhau để giúp lưu trữ dữ liệu.
- Ví dụ:

Dim iCount As Integer

Khai báo và sử dụng biến và mảng

- Khởi tạo biến khi bạn khai báo chúng
- Khởi tạo mảng với kích thước xác định hoặc không có kích thước
  - Thay đổi kích thước mảng bằng cách dùng từ khóa ReDim

```
Dim i As Integer = 21
Dim dToday As Date = Today( )

'Array declarations
Dim Month(12) As Integer  'Creates array with 13 elements
'Initialize the array with 12 elements
Dim aMonth( ) As Integer = {1,2,3,4,5,6,7,8,9,10,11,12}
```

Khai báo sử dụng biến và mảng

- Cú pháp:

```
Dim varname [As [New] type] [= initexpr]
```

- Khai báo mảng

```
Dim x( ) As String
ReDim x(5) 'Correct in Visual Basic .NET
Dim y(2) As String
ReDim Preserve y(5) 'Allowed in Visual Basic .NET
```

Khai báo nhiều biến

- Khai báo nhiều biến trong VB6

```
Dim I, J, X As Integer
'Results in I and J As Variant, X As Integer
```

- Khai báo nhiều biến trong VB.Net

```
Dim I, J, X As Integer
'Results in I, J, and X As Integer
```

Dạng viết ngắn khai báo

- Có thể khai báo dùng ký tự nhận dạng
- Ví dụ:

```
Dim strVar$
```

Bảng ký tự nhận dạng

Data Type	Identifier Type Character
Integer	%
Long	&
Single	!
Double	#
Decimal	@
String	$

Các kiểu dữ liệu

Kiểu dữ liệu	Kích thước(Mô tả)
Integer	4 bytes
Long	8 bytes
Short	2 bytes
Byte	1 byte
Double	8 bytes
Single	4 bytes
Decimal	12 bytes
Boolean	2 bytes
Char	2 bytes
DateTime	8 bytes
String	10 bytes + 2 bytes cho mỗi chữ
Object	Chứa bất kỳ loại dữ liệu nào

Cách đặt tên biến

- Bắt đầu bằng một ký tự
- Không có dấu chấm hoặc ký tự nhận dạng, không khoảng trắng
- <= 255 ký tự
- Phải là duy nhất trong cùng phạm vi

Khởi tạo biến

- Mặc định, một biến mang một giá trị khi khai báo. Integer mặc định là 0.
- Bạn có thể khởi tạo giá trị của biến khi bắt đầu.
- Ví dụ:

```
Dim iNum as Integer
iNum = 20 ‘hoặc
Dim iNum as Integer = 20
```

Từ khóa *New*

- Biến thực sự được tạo khi bạn dùng chúng trong code hoặc khởi tạo chúng. Có thể dùng từ khóa *New* để tạo một biến ngay khi bạn khai báo.
- Ví dụ

```
Dim iNum as Integer
iNum = new Integer()  'hoặc
Dim iNum as Integer = New Integer()  'hoặc
Dim iNum as New Integer()
```

Từ khóa *Nothing* và *Null*

- *Nothing* - Giải phóng một biến đưa nó về giá trị mặc định
- Ví dụ:

```
Dim iNum as Integer = 10
iNum = Nothing
' Lúc này iNum = 0
```

- *Null* – Không hỗ trợ trong VB.Net.

Phạm vi của biến

- Xác định phạm vi truy cập của biến
- Biến có thể chỉ dùng trong khối lệnh hoặc toàn bộ chương trình.
- Một biến khai báo trong thủ tục thì chỉ dùng được trong thủ tục
- Đôi khi cần các biến khai báo biến dùng ***khắp chương trình*** như các biến tham chiếu ở mức module. Biến ở mức module phân làm hai loại private và *public*
  - *Private : chỉ dùng trong module khai báo*
  - *Public : dùng trong tất cả các module*

Phạm vi của biến

- Các biến trong thủ tục
  - Được sử dụng trong phạm thủ tục đó.
- Các biến trong khối lệnh
  - Các biến chỉ được truy cập trong khối lệnh đó
  - Thời gian tồn tại biến trong khối lệnh là kết thúc khối lệnh

```
Dim iLooper As Integer    'Procedure level variable

For iLooper = 1 to 10
    Dim iMax As Integer   'Block level variable
    iMax = iLooper
Next
MsgBox (iMax)     'This line generates a compiler error
```

Tạo các kiểu dữ liệu có cấu trúc

- Kiểu cấu trúc người dùng tự định nghĩa
- Kiểu cấu trúc hỗ trợ nhiều đặc điểm trong Class
- Sử dụng khai báo Structure … End Structure
- Khai báo cấu trúc các thành viên với thuộc tính truy cập

```
Structure Customer
  Public CustID As Integer
  Dim CustDayPhone As String              'Defaults to public
  Private CustNightPhone As String        'Private allowed
End Structure
```

Các tùy chọn biên dịch

- Option Explicit [On/Off]
  - On là tùy chọn mặc định, bạn phải khai báo biến trước khi sử dụng
- Option Strict [On/Off]
  - Cho phép/không cho phép chuyển đổi ngầm định
  - Chuyển đổi sang kiểu dữ liệu Object không cho phép khi **On**
- Option Base 1 ***không hỗ trợ***
  - Chỉ số mảng luôn bắt đầu là 0

Khai báo tường minh và không tường minh (Implicit and Explicit)

- Khai báo không tường minh -> Một biến sử dụng không cần khai báo
- Các nhà lập trình khuyên nên khai báo tường minh
- Cú pháp:

  Option Explicit [On|Off]

  Mặc định

  Option Explicit On

Các kiểu chuyển đổi (Convertion)

- VB.Net cung cấp hai kiểu chuyển dổi
  - *Chuyển đổi rộng: kết quả không bị mất, và luôn thành công. Ví dụ : Short -> Integer.*
  - *Chuyển đổi hẹp: kết quả có thể bị mất và có thể không thành công. Ví dụ Integer -> Short*
- VB.Net cung cấp cho phương pháp chuyển đổi dữ liệu.
- Câu lệnh:

  Option Strict [On | Off]

  Nếu là On kiểu dữ liệu các biến sẽ kiểm tra trước khi chuyển đổi.

Khai báo Hằng

- Hằng là một biến mà giá trị của nó không thay đổi trong suốt quá trình chạy chương trình.
- Ví dụ:

  Const MAX_SCORE As Integer =100

  or

  Const MAX_SCORE = 100
- Xử lý một hằng nhanh hơn một biến
- Đặt tên hằng bằng các từ hoa, gạch chân dưới (_) với mỗi từ.

Các toán tử

- Toán tử toán học
- Toán tử gán
- Toán tử so sánh
- Toán từ logic
- Toán tử cộng chuỗi

Các toán tử tóan học

- Toán tử số mũ: ^ (dùng cho dữ liệu số)
- Toán tử nhân: * (dùng cho dữ liệu số)
- Toán tử chia: / (dùng cho dữ liệu số)
- Toán tử chia nguyên: \ (chia hai số nguyên)
- Toán tử Mod: phép chia dư (dùng cho dữ liệu số)
- Toán tử +: cộng hai số(dùng cho kiểu số và kiểu ký tự)
- Toán tử -: Trừ hai số (dùng cho kiểu số)

Các toán tử gán

- Toán tử +=
- Toán tử -=
- Toán tử *=
- Toán tử /=
- Toán tử \=
- Toán tử &=
- Toán tử ^=

Các toán tử so sánh

- Các toán tử quan hệ
- Toán tử *Is*
- Toán tử *Like*

Các toán tử quan hệ

- Dùng so sánh hai biểu thức

Result = Bieu_Thuc1 ToanTuSoSanh *Bieu_Thuc2*

Operator	Result Is True If	Result Is False If
< (less than)	Expression1 < Expression2	Expression1 >= Expression2
<= (less than or	Expression1 <= Expression2	Expression1 > Expression2 equal to)
> (greater than)	Expression1 > Expression2	Expression1 <= Expression2
>= (greater than or	Expression1	Expression1
	>= Expression2	< Expression2 equal to)
= (equal to)	Expression1 = Expression2	Expression1 <> Expression2
<> (not equal to)	Expression1 <> Expression2	Expression1 = Expression2

Toán tử *Is*

- Tóan tử *Is* được dùng để so sánh hai đối tượng tham chiếu.
- Cú pháp:

Result = Object1 Is Object2

Kết quả trả về giá trị kiểu Boolean. Toán tử xác định khi nào cả hai đối tượng Object1 và Object2 là giống nhau.

- Ví dụ:

Ví dụ

```
Dim Object1, Object2 As New Object
Dim MyObjectA, MyObjectB, MyObjectC As Object
Dim MyResult As Boolean
MyObjectA = Object1
MyObjectB = Object2
MyObjectC = Object2
MyResult = MyObjectA Is MyObjectB
'Returns False
MyResult = MyObjectB Is MyObjectC
'Returns True

MyResult = MyObjectA Is MyObjectC
'Returns False
```

Toán tử *Like*

- Dùng để so sánh các chuỗi
- Cú pháp:

  Result = Chuỗi Like Mẫu

  Kết quả trả về giá trị kiểu Boolean.
- Khi dùng toán tử *Like* cũng có thể dùng các ký tự đại diện.
- Kết quả trả về True nếu *Chuỗi* được so khớp với *Mẫu*

Bảng ký tự đại diện

Character in Pattern	Matches
?	Any one character
*	Zero or more characters
#	Any single digit (0–9)
[list]	Any one character in the specified list
[!list]	Any one character other than the characters in list

Ví dụ

```
Dim MyValue As Boolean
MyValue = "A" Like "A"
                                        'Returns True
MyValue = "A" Like "a"
                                        'Returns False
MyValue = "C" Like "[A-F]"
                                        'Returns True

MyValue = "H" Like "[A-F]"
                                        'Returns False
MyValue = "D" Like "[!A-F]"
                                        'Returns False
MyValue = "zxyz" Like "z*z"
                                        'Returns True
MyValue = "GFdAT13h4g" Like "GF?A*"
                                        'Returns True
```

Các toán tử logic

- Toán tử And
- Toán tử Not
- Toán tử Or
- Toán tử Xor
- Toán tử AndAlso
- Toán tử OrElse

Toán tử AndAlso

- Là một toán tử mới VB.Net
- Cú pháp:

  Result = Bieu_Thuc1 AndAlso *Bieu_Thuc2*

- Toán tử AndAlso làm việc giống như toán tử And nhưng nó mạnh hơn And.
- Hoạt động: Trước hết nó kiểm tra giá trị của Bieu_Thuc1, nếu trả về true thì nó mới kiểm tra Bieu_Thuc2

Ví dụ

```
Dim A As Integer = 15
Dim B As Integer = 10
Dim C As Integer = 5
Dim MyResult As Boolean
MyResult = A > B AndAlso B > C
                                        'Returns True
MyResult = B > A AndAlso B > C
                                        'Returns False
MyResult = A > B AndAlso C > B
                                        'Returns False
```

Toán tử OrElse

- Là một toán tử mới VB.Net
- Cú pháp:

  Result = Bieu_Thuc1 OrElse *Bieu_Thuc2*

- Toán tử này hoạt động cũng giống như toán tử Or nhưng mạnh hơn toán tử Or.
- Hoạt động: Trước hết kiểm tra *Bieu_Thuc1*, nếu nó trả về True thì gán luôn cho kết quả, nếu không thì nó mới kiểm tra *Bieu_Thuc2*.

Ví dụ

```
Dim A As Integer = 15
Dim B As Integer = 10
Dim C As Integer = 5
Dim MyResult As Boolean
MyResult = A > B OrElse B > C
                                        'Returns True
MyResult = B > A OrElse B > C
                                        'Returns True
MyResult = B > A OrElse C > B
                                        'Returns False
```

4. Demo: Sử dụng biến và các cấu trúc dữ liệu

5. Functions, Subroutines, and Properties

- Gọi hàm và thủ tục
- Đặt các đối số ByRef và ByVal
- Các đối số tùy chọn
- Hàm và thủ tục tĩnh (Static)
- Trả lại giá trị từ hàm
- Sử dụng các thuộc tính mặc định

Gọi hàm và thủ tục

- Trong VB.Net
  - Phải dùng dấu ngoặc đơn đi kèm với các biến trong hàm hoặc thủ tục
  - Bạn phải dùng dấu ngoặc đơn rỗng cho các hàm hoặc thủ tục không có tham số.
- Ví dụ

```
DisplayData(1, 21) 'Subroutine
Call DisplayData(1, 21)
```

Đặt các đối số ByRef và ByVal

- VB6
  - Mặc định tham số truyền vào hàm và thủ tục là ByRef
- VB.Net
  - Mặc định tham số truyền vào hàm và thủ tục là ByVal
- ByRef và ByVal khác gì nhau?

Các đối số tùy chọn

- VB6
  - Bạn không cần cung cấp giá trị mặc định cho tham số tùy chọn
  - Bạn có thể dùng hàm IsMissing
- VB.Net
  - Bạn phải cung cấp giá trị mặc định cho tham số tùy chọn

```
Function Add(Value1 As Integer, Value2 As Integer,
Optional Value3 As Integer = 0) As Integer
```

Hàm tĩnh và thủ tục tĩnh (Static)

- VB6
  - Bạn có thể đặt Static trước Funtion và Sub
  - Các biến cục bộ trong hàm tĩnh hoặc thủ tục tĩnh giữ nguyên giá trị của chúng qua nhiều lần gọi
- VB.Net
  - Hàm tĩnh và thủ tục tĩnh không hỗ trợ
  - Bạn phải khai báo một cách rõ ràng tất cả các biến là tĩnh

Ví dụ cách dùng biến tĩnh

```
Dim iLooper As Integer
Static iMax As Integer
  For iLooper = 1 To 10
    iMax += 1
  Next
MsgBox(iMax)
```

Trả lại giá trị từ hàm

- VB6
  - Dùng tên hàm để trả về kết quả
- VB.Net
  - Bạn có thể dùng tên hàm
  - Bạn có thể dùng câu lệnh Return để trả về kết

```
Function GetData( ) As String
    ...
    GetData = "My data"
End Function
```

```
Function GetData( ) As String
    ...
    Return "My data"
End Function
```

Sử dụng các thuộc tính mặc định

- VB6
  - Hỗ trợ thuộc tính trên mỗi đối tượng
  - Dùng Set để xác định khi nào khởi gán đang tham chiếu tới đối tượng hoặc thuộc tính mặc định
- VB.Net
  - Chỉ hỗ trợ thuộc tính mặc định cho các biến
  - Đồng nhất giữa phép gán đối tượng và thuộc tính mặc định
  - Thuộc tính mặc định thường được sử dụng vào chỉ số trong các tập hợp ví dụ như **Fields.Item**

Sử dụng các thuộc tính mặc định

- Bạn có thể gọi thuộc tính mặc định chỉ khi thuộc tính làm các biến

```
Dim rs As ADODB.Recordset, Lab1 As Label
'…initialization

rs.Fields.Item(1).Value = Lab1.Text
'Valid because no defaults used
rs.Fields(1).Value = Lab1.Text
'Valid because Item is parameterized

rs.Fields(1) = Lab1.Text
'Not valid because Value is not parameterized
Lab1 = "Data Saved"
'Not valid because Text is not parameterized
```

6. Demo: Làm việc với biến và các thủ tục

7. Xử lý ngoại lệ

- Cấu trúc xử lý ngoại lệ
- Try … Catch … Finally
- Sử dụng Try … Catch … Finally
- Lớp *System.Exception*
- Lọc các ngoại lệ (Filtering Exception)
- Xử lý các ngoại lệ (Throwing Exception)

Cấu trúc xử lý ngoại lệ

- Những nhược điểm khi xử lý các lỗi không theo cấu trúc
  - Code khó đọc, gỡ rối và bảo trì
  - Dễ bỏ xót các lỗi
- Những thuận lợi khi xử lý lỗi có cấu trúc
  - Được hỗ trợ bởi nhiều ngôn ngữ
  - Cho phép bạn tạo các khối bảo vệ code
  - Cho phép bạn lọc các *ngoại lệ* tương tự như câu lệnh ***Select Case***
  - Cho phép bạn tạo nhiều xử lý ngoại lệ
  - Code bạn dễ đọc, dễ gỡ rối và bảo trì

Try… Catch… Finally

```
...
Try
' code của bạn làm ở đây
' Có thể dùng Exit Try để kết thúc khối lệnh và
' khôi phục  sau End Try
Catch
' Định nghĩa các kiểu ngoại lệ và hành động xử lý
' Có thể dùng một dãy các câu lệnh (quản lý nhiều lỗi)
Finally
' là khối tùy chọn
' Định nghĩa các hành động cuối cùng làm ở đây
End Try
...
```

Sử dụng Try … Catch … Finally

```
Sub TrySimpleException
  Dim i1, i2, iResult As Decimal
  i1 = 22
  i2 = 0
  Try
    iResult = i1 / i2       ' Cause divide-by-zero error
    MsgBox (iResult)        ' Will not execute

  Catch eException As Exception  ' Catch the exception
    MsgBox (eException.Message)  ' Show message to user

  Finally
    Beep
  End Try
End Sub
```

Lớp *System.Exception*

- Cung cấp các thông tin về Ngoại lệ

Property or Method	Information provided
Message property	Why the exception was thrown
Source property	The name of the application or object that generated the exception
StackTrace property	Exception history
InnerException property	For nested exceptions
HelpLink property	The appropriate Help file, URN, or URL
ToString method	The name of the exception, the exception message, the name of the inner exception, and the stack

Lọc các ngoại lệ

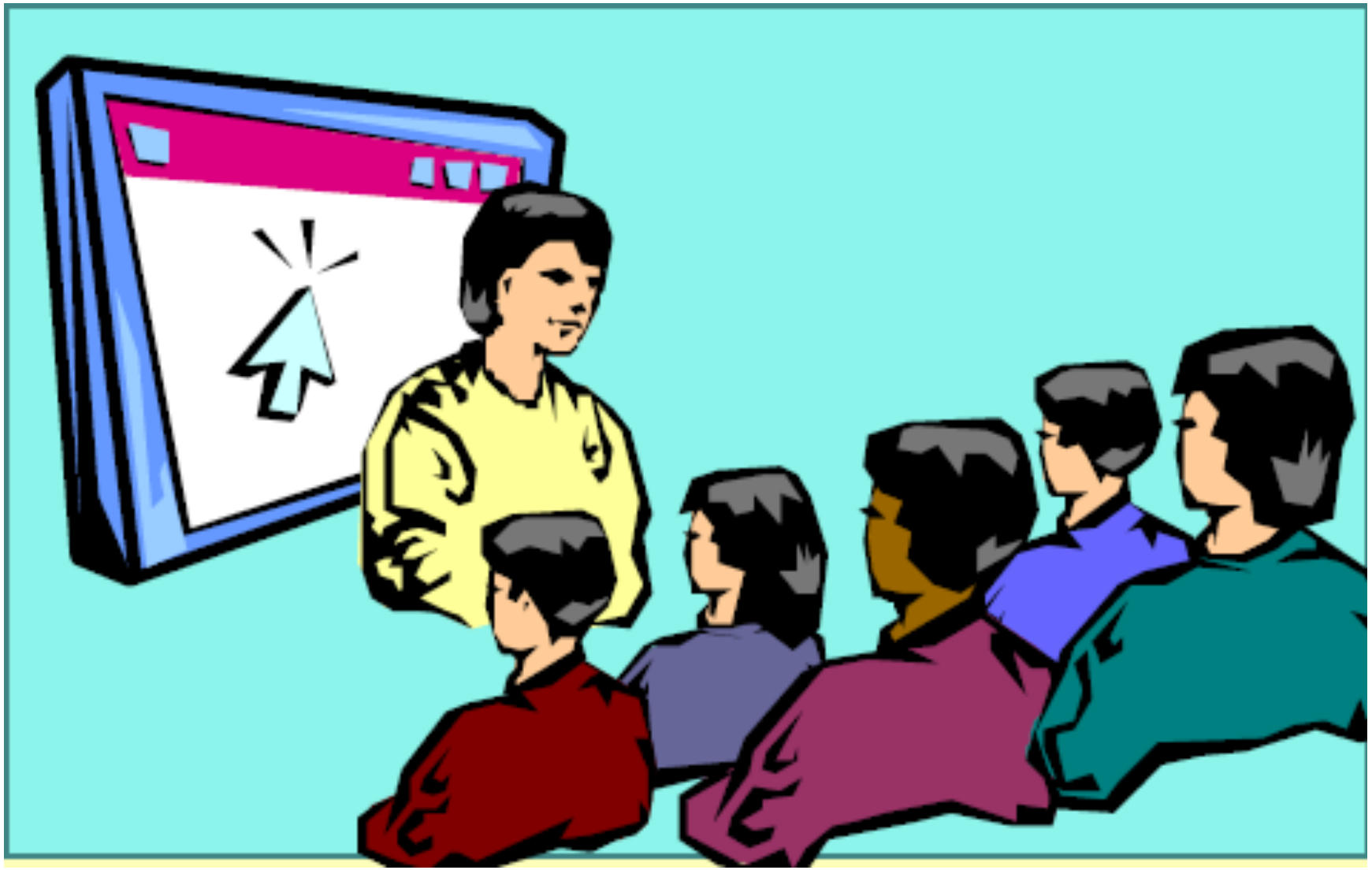
```
Dim x, y, z As Integer, bSucceeded As Boolean = True
Try
    'Perform various operations on variables
    ...
Catch eException As DivideByZeroException
    MsgBox("You have attempted to divide by zero.")
    bSucceeded = False
Catch eException As OverflowException
    MsgBox("You have encountered an overflow.")
    bSucceeded = False
...
Catch When Err.Number = 11
    MsgBox("Error occurred.")
    bSucceeded = False
Finally
    If bSucceeded Then
        ...
    End If
End Try
```

Xử lý các ngoại lệ (Throwing Exception)

- Dùng từ khóa *Throw* thay cho phương thức Err.Raise trong VB6.

```
Try
  If x = 0 Then
      Throw New Exception("x equals zero")
  Else
      Throw New Exception("x does not equal zero")
  End If
Catch eException As Exception
  MsgBox("Error: " & eException.Message)
Finally
  MsgBox("Executing finally block")
End Try
```

8. Demo: Cấu trúc xử lý ngoại lệ



Lab: Thực hiện cấu trúc xử lý ngoại lệ