Lecture Outline

- Development process
- Planning an OOPED application in VB using TOE Charts
- Assigning values to properties at run time
- Inbuilt functions & methods
- Convert and Random Class

The Process of Development

- **Steps:**
  - A. Plan (TOE charts, sketch of user interface, pseudocode)
  - B. Build user interface
  - C. Code the Application
  - D. Test and Debug the Application
  - E. Write the Documentation

Planning VB.NET Application

1. Identify the **tasks** the application needs to perform
2. Identify the **objects** to which you will assign those tasks
3. Identify the **events** required to trigger an object into performing its assigned task
4. Draw a **TOE** chart
5. Draw a **sketch** of the user interface

Step 1: Identify the Application’s Tasks

- What will the user need to enter?
- What will the application need to calculate?
- Will previous information need to be cleared from the screen?
- How will the user end the application?
- **Example Tasks**
  - **Get employee information:** Name, Address, Personnel Number, Hours worked
  - **Calculate salary:** Hourly Rate, Hourly Wage, Hours
  - **Print Salary Slip:** Name, Personnel Number, Gross, Net
  - **Clear:** Clear screen/number, reset
  - **End:** Exit

Step 2: Identify the Objects

- After completing the Task column of the TOE chart, you then assign each task to an object in the user interface
  - Use a Text Box for data the user must enter
  - Use a Label for output produced by program
  - Use a Button to perform an action
Step 3: Identifying the Events

- Text boxes and Label controls display their contents automatically, so no special event is necessary for them to do their assigned task.
- You will have the buttons perform their assigned tasks when they are clicked by the user.
- For each task you have identified, list the event in the Event column.

Step 4: Sample TOE Chart

<table>
<thead>
<tr>
<th>Task</th>
<th>Object</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 5: Design the User Interface

- Look at the objects in the TOE chart.
- Using pen and paper, sketch the layout of the objects.
- Organize the graphical user interface (GUI) so that the information flows either vertically or horizontally, with the most important information always located in the upper-left corner of the screen.

Exercise - Order Form

A timber supplier needs a computer program which allows customers to enter their details, and the number of metres of Oak, Pine or Redwood they wish to purchase, then calculate and display the total purchase cost.

- Oak = $7, Pine = $5, Redwood = $10 per metre

To do:
- Identify tasks of the application
- Objects needed in the application
- Events for objects

Designing the User Interface

- Info flows vertically or horizontally
- Group related controls – group box, white space
- Command buttons - up to 6, bottom of screen or stacked in corners, most commonly used first
- Meaningful captions on buttons
- Sentence capitalisation for identifying labels
- Book title capitalisation for buttons
- Label controls - 1-3 words only
- Aligning and sizing controls
- Use of color, graphics and different fonts

Task, Object, Event

<table>
<thead>
<tr>
<th>Task</th>
<th>Object</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sketch user interface, naming all objects (even additional ones like labels used next to text boxes) - Example: TimberOrder.sln
Assign a value to an object’s property during runtime

- Syntax assignment statement
  - `Me.[objectName].property = newValue`
  - `Me.txtCity.Text = “Melbourne”`
  
  Enables properties to be set or changed during run-time, rather than at design-time.

- What do these assignments do?
  - `lblLabel.Text = “Name”`
  - `txtBox.Text = “”`
  - `lblLabel.Text = txtBox.Text`

- In summary,
  - Properties can be set during design time using the property window.
  - Properties can be set or changed during run-time using the assignment statement in code (usually in response to the user’s actions).

Mathematical Operators

- Exponentiation
- Negation
- `/` Multiplication, division
- `%` Integer division
- `Mod` Modulus arithmetic
- `+`, `-` Addition and subtraction

Note that you can use brackets to overcome precedence orders.

- Provided you are dealing with numeric types (not Strings) you can use standard maths operators.
- If strings are involved and you are assigning them to a numeric variable, you must first convert them using `Val()`.

Val() Function

- Most common use for `Val()` function is in converting Text Box input (a String) into numeric data for mathematical operations.

Example: Adding two numbers

```vba
Val(txtNum1.Text) + Val(txtNum2.Text)
```

Why use `Val()`

- If you don’t use `Val()`, then `+` will act as the concatenation operator, so that
  - `lblAnswer.Text = txtNum1.Text + txtNum2.Text`
    - `= “12” + “34”` (concatenate 2 strings)
    - `= “1234”` (a string)

- Instead of
    - `= Val(“12”) + Val(“34”)` (sum of 2 numbers)
    - `= 12 + 34`
    - `= 46` (a number)

- Is there a problem here?

Convert Class

- Use the `Convert` Class to convert the data type of the expression on RHS to the data type of the variable on LHS.
- What methods are in the `Convert` Class?
- How would you find these out?
- Visit `msdn.microsoft.com` to find out what other methods can be invoked from the `Convert` Class.

Examples: Val Function

- `Val(“456”)` 456
- `Val(“24,500”)` 24
- `Val(“123x”)` 123
- `Val(“$56.88”)` 0
- `Val(“Abc”)` 0
- `Val(“25%”)` 25
Formatting output

• Suppose variable Total = 1221.67856
  – Format(Total, "Fixed") = 1221.68
  – Format(Total, "Currency") = $1,221.68
  – Format(Total, "Scientific") = 1.22E+03
  – Format(Total, "True/False") = True

• Suppose variable Total = 0
  – Format(Total, "Currency") = $0.00
  – Format(Total, "Scientific") = 0.00E+00
  – Format(Total, "Yes/No") = No

Using the Focus Method

– Focus method allows you to move the focus to a specified control while the application is running.

• Syntax
  
  `Me.object.Focus()`

  where

  object is the name of in which you want the focus, and Focus( ) is the name of the method

Random Class (for next week’s tute)

• Use the Random Class to generate random numbers
  
  – Must declare a random object first
    
    `Dim randomObject As New Random`

  – To generate a random number
    
    `randomObject.Next(1,100)`
    
    generates a random integer between 1 and 100; including 1, excluding 100

    `randomObject.NextDouble`
    
    Generates a real number between 0 and 1, including 0 but excluding 1

Questions

• Reading
  
  – Zak, Chpt 2
  
  – Unit Guide, Study Guide 2

• Stop for a moment and summarise
  
  – what you have covered
  
  – the main parts of today’s lesson
  
  – write down any questions you would like to ask

Lecture Outline

• Variables
  
  – Definition and purpose
  
  – Data Types
  
  – Naming convention
  
  – Declaration
  
  – Storing data in variables
  
  – Variable scope

• Literals and Constants

• Translating an algorithm to VB.NET code
What we need to know about variables

- **Definition** - What are they?
- **Purpose** - When do you use them?
- **Data Types** - What data can variables hold?
- **Declaration** - How and where do you declare them?
- **Naming convention** - What name to do give them?
- **Assigning values to variables**
- **Scope** - How long do they live for?

Variables in Algorithms

- **Placeholder for unknown values**
- **Used to store results of calculations**

Example:

Get distance from user
Get speed from user
Calculate \( \text{time} \leftarrow \text{distance} / \text{speed} \)
Display \( \text{time} \) to user

Variables in Programs

- **Creating a variable reserves an area of computer memory**
  - Address – where the data is physically located in memory
- **Can store values into the memory location**
  - E.g. Location in memory could be 0x0010F8A4 (a hexadecimal number)
  - Value in that location may be the number 1234.56
- **A particular memory variable can store values of a specific data type only**

VB.NET Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short, Long</td>
<td>Used to store whole numbers</td>
</tr>
<tr>
<td>Single</td>
<td>Stores floating-point numbers in 4 bytes</td>
</tr>
<tr>
<td>Double</td>
<td>Stores floating-point numbers in 8 bytes</td>
</tr>
<tr>
<td>Decimal</td>
<td>Stores numbers with a decimal point (16 b)</td>
</tr>
<tr>
<td>Boolean</td>
<td>Stores True or False</td>
</tr>
<tr>
<td>Char</td>
<td>Stores one (Unicode) character</td>
</tr>
<tr>
<td>Byte</td>
<td>Stores 8-bit of data</td>
</tr>
<tr>
<td>String</td>
<td>Stores a sequence of characters</td>
</tr>
</tbody>
</table>

Data Types

- **A Data Type is distinguished by**
  - the operations which can be applied
  - the way it is stored in memory (number of bytes, order of bits)
  - the values it can take
- **Categories of Data Types:**
  - Numeric
  - String / Character-based
  - Classes and Structures
  - Logical Value (True/False)
- **Failure to understand their importance and use them correctly is the largest cause of errors in student programming!**

Data Types

- **Character**
  - Digit, 0 1 2 3 4 5 6 7 8 9
  - Or other symbol, e.g.: ! @ # $ % ^ & * " { \\ ( / )
  - Some non-printable (control codes)

- **String**
  - Values take the form of a sequence of characters
  - Use double-quotes around the string, e.g. "Hello"
  - Requires 1 byte per character
  - Operations include: + &
Example – Integer variables

Calculate the sum and average of three integers (whole numbers)

CalcAvg
sum ← 0
Read num1
  sum ← sum + num1
Read num2
  sum ← sum + num2
Read num3
  sum ← sum + num3
average ← sum / 3
END

Sum ← 0

Read num1 (num1 is 10)

Read num2 (num2 is 6)
Representing Values in VB.NET

- There are 3 main ways to represent values in code:
  - Variables
  - Literals
  - Constants

Variables in VB.NET

- Variables must be declared before being used
  - To specify the type of the variable
    - Integer, Single, Double, String, Boolean
  - reserves memory space for the value to be stored
  - gives a name to variable
- Values can then be assigned to or stored in the variable

Declaring Variables

- Need to specify the data type of a variable
  - So computer knows how to organise memory
- The Dim statement does this
- The effect is to allocate space in memory for the variable and initialize it to a default value
- The syntax is
  ```vbscript
  Dim variableName As Type = initialValue
  ```
Choosing a Name for Variables

- You should assign a descriptive name to each variable used in an application.
- Name must not match a VB.NET keyword.
  - E.g., `Val` or `Print` are not allowable variable names.
- The name should help you remember the variable’s data type and purpose.
  - Use the standard three-letter prefix to indicate data type.
  - Capitalize each word comprising the name.

Prefixes for variable names

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>bln</td>
</tr>
<tr>
<td>Byte</td>
<td>byt</td>
</tr>
<tr>
<td>Char</td>
<td>cha</td>
</tr>
<tr>
<td>Date</td>
<td>dtm</td>
</tr>
<tr>
<td>Decimal</td>
<td>dec</td>
</tr>
<tr>
<td>Double</td>
<td>dbl</td>
</tr>
<tr>
<td>Integer</td>
<td>int</td>
</tr>
<tr>
<td>Long</td>
<td>lng</td>
</tr>
<tr>
<td>Object</td>
<td>obj</td>
</tr>
<tr>
<td>Short</td>
<td>shr</td>
</tr>
<tr>
<td>Single</td>
<td>sing</td>
</tr>
<tr>
<td>String</td>
<td>str</td>
</tr>
</tbody>
</table>

Dim statement examples

```
Dim shrNumberOfDoors As Short
Dim intTotalEmployees As Integer
Dim sngTotalPrice As Single
Dim strEmployeeName As String
Dim blnValidData As Boolean
```

Assigning Values

- Giving a data value to a variable is called assignment.
- In an algorithm (pseudocode) use `=` or `:=`
- In VB use an assignment statement.
- The syntax for an assignment statement is `variableName = expression`
  - Where expression is any literal value, calculation or other operation that results in a value.
  - The data type of the `expression` on RHS must match (or be included by) the data type of the variable on LHS.

Assigning Values

- Example Assignment Statements:
  - `strEmployerName = “Churchill Software Developers”`
- Can also assign an initial value when declaring variable:
  - `Dim strAddress As String = “Princes Highway”`
- By default, strings are empty, and numeric variables are 0.

Option Explicit and Option Strict

- Option Explicit
  - Turning Option Explicit On ensures that you always declare a variable before you use it.
- Option Strict
  - Turning Option Strict On enforces explicit use of type casting when variables are assigned to other variables of a different type.

Throughout this unit you are required to turn Option Strict On and Option Explicit On in all your programs.
The Concept of Scope

- Programs are very structured
- All VB.NET Forms are actually a class
- Inside the Form class's code there are:
  - Event procedures
  - Other procedures and functions (cover later)
  - Variables and Constants
  - User-defined data types (cover later)
- Each of the above elements have names so we can refer to them
- Scope = those parts of a program a particular element is defined, i.e. where the name of the element is understood

The Scope of a Variable

- A variable’s scope indicates which procedures in an application can use the variable
- The scope is determined by where the Dim statement appears in the code window
- When you declare a variable in a procedure, the variable is called a local variable (or procedure-level variable) and is said to have procedure scope because only that procedure can use the variable
- When you declare a variable in the form’s Declarations section, the variable is called a module-level variable and is said to have module scope

Example – Procedure Scope

```vbnet
Public Class Form1 Inherits System.Windows.Forms.Form
    Windows Form Generated Code
    Private Sub Button1_Click(...) Handles Button1.Click
        Dim strName As String
        strName = NameTextBox.Text
    End Sub
    Private Sub Button2_Click(...) Handles Button2.Click
        Dim intQuantity As Integer
        intQuantity = Val(QuantityTextBox.Text)
    End Sub
End Class
```

strName only exists inside the Button1_Click procedure.
Button2_Click doesn’t know what strName means.

Example – Module Scope

```vbnet
Public Class Form1 Inherits System.Windows.Forms.Form
    Windows Form Generated Code
    Dim strName As String
    Dim intQuantity As Integer
    Private Sub Button1_Click(...) Handles Button1.Click
        strName = NameTextBox.Text
    End Sub
    Private Sub Button2_Click(...) Handles Button2.Click
        intQuantity = Val(QuantityTextBox.Text)
    End Sub
End Class
```

Both strName and intQuantity are known to both event procedures.

Be Careful of Shadow Variables

- What will the following code display in the textbox when each button is clicked?

```vbnet
Public Class Form1 Inherits System.Windows.Forms.Form
    Windows Form Generated Code
    Dim strName As String = "Adam"
    Private Sub Button1_Click(...) Handles Button1.Click
        Dim strName As String = "Eve"
        txtName.Text = strName
    End Sub
    Private Sub Button2_Click(...) Handles Button2.Click
        txtName.Text = strName
    End Sub
End Class
```

"Eve" is displayed in the textbox.

Literals

- An item of data whose value cannot change while the program is running
- Hard-coded, by the programmer
- Examples:
  - 0.6
  - 7
  - "Welcome"
- Do all 7’s throughout the program mean the same thing?
Named Constants (or Symbolic constants)

- Similar to a variable
  - Identifier for a memory location holding a value needed by the application
  - Can be used in expressions anywhere variables are allowed
- Value assigned at design (coding) time
  - Cannot be changed by the program once created in memory
- Values may be numeric or string
- The syntax is `Const constantName [As Type] = expression`
- The data type is optional (it is implicit from `expression`)

Example Declarations:
```
const sngDiscountRate As Single = 0.05
const strCompanyName As String = "Hi there"
```

Example usage of a constant:
```
sngFinalPrice = sngOrderPrice * sngDiscountPrice
CompanyNameTextbox.Text = strCompanyName
```

Constants

Why use constants?
- Only need to change value in one location in program
  - E.g. If `tax_rate` is 10% today (such as for GST), and changes next year, easier to change the constant’s value in one place than scattered through-out your program

Summary

Literals:
- Are values hard-coded in the program

Variables and Constants:
- Represent values
- Have a name
- Use memory space
- Have a specific data type
- Variables may change (vary), constants may not change, during the life of the program.

Exercise: Translate an Algorithm into VB.NET code

```
CalcAvg
sum ← 0
num1 ← 10
num2 ← 6
num3 ← 11
sum ← sum + num1
sum ← sum + num2
sum ← sum + num3
average ← sum / 3
END
```

Documentation

- Internal Documentation:
  - Used by programmers
  - Good practice to leave messages as reminders in the Code window
  - Place apostrophe (') before the comment
  - Comments are ignored by VB.NET program
- User Documentation:
  - Used by users of the system
  - Is what is valid and invalid input
  - How to achieve a complex task that requires many steps
  - Can be online help or a physical document
Questions/Reading

- Stop for a moment and summarise
  - what you have covered
  - the main parts of today’s lesson
  - questions to ask
- Reading Zak, Chpt 3
  p72-116
- Unit Guide, SG 3