Introduction to VB.NET

Agenda

- Why VB.NET
- What is new in VB.NET
- Update to VB.NET?
- VB.NET Language Essential

Why VB.NET (from technical standpoint)

- The world of applications is changing:
  - The move to Web
  - The need for reusability, centralization and scalability
  - MTS, COM+, and Component Services cannot be fully taken advantage of by VB.
  - SOAP: features can be implemented more completely with .NET.

Why VB.NET (cont.)

- To get the benefit of .NET framework and its core execution engine: CLR.
  - Garbage collection
  - OO mechanism
  - Standard security services
  - Integrated debugging tools

Why VB.NET (cont.)

- Why not *C#*
  - VB.NET----"The most productive tool for building .NET-connected applications. "---- *Microsoft Corporation*
  - *Root in Basic, the most pure-flavor language product from MS.*
  - *Easier for VB programmers: a number of unique features.*
    - *E.g.: Only VB.NET has background compilation, dropdown list of the code window.*

What is New in VB.NET ----For Experienced VB Programmers

- IDE changes
- Project Changes
- Web Changes
- WebClass Changes
- Data Changes
- Component Authoring Changes
- UserControl Changes
- Forms Changes
- Debugging Changes
- Setup and Deployment Changes
- International Changes
- Windows API Changes
- Registry Access Changes
- Constant Changes
- Namespace Changes
- Run-Time Changes

Overview of Big Changes in VB.Net

- Everything is object-oriented: abstraction, inheritance, overloading, encapsulation and polymorphism.(Note: no multiple inheritance, but interfaces supported.)
- Multithreaded applications are possible.
- Language syntax changes

……

Changes in VB Language

- All data are objects, based on the class: ***System.Object***.
  - E.g. class supports Windows forms: ***System.Windows.Forms.Form***.
- The built-in VB functionality is encapsulated in a namespace called ***System***.
  - E.g ***Collection*** has be replaced by ***System.Collections***.
- Old control are gone, and new ones have appeared.

Changes in VB Language (cont.)

- Many keywords are renamed or gone, while some new added.
  - E.g. ***Gosub*** removed
- Strict data typing is now enforced
  - Variable must be declared before used by default.
  - Cannot assign one data type to another, but can use ***Ctype*** to convert between types.
  - The same as in VC++ and C#.
- Structured exception handling: ***Try…Catch…Finally***.

Changes in VB Language (cont.)

- When calling procedures, must use parentheses.
- Parameters are by default passed by value, instead of by reference.
- Supports constructors and destructors for use when initializing an object of a class.
- ***If…Then*** statements are now short-circuited.

Changes in VB Language (cont.)

- A number of new compound operators
  - E.g. x+=2
- The ***And***, ***Or***, ***Not*** and ***Xor*** operators have changed from bitwise to boolean operators. Meanwhile, the bitwise versions are ***BitAnd***, ***BitOr***, ***BitNot***, and ***BitXor***.
- No default property supported
  - E.g. VB6: TextBox1="Hello"
    VB.Net: TextBox1.Text="Hello"

Changes in VB Language (cont.)

- Three new data types
  - ***Char***: unsigned 16-bit
  - ***Short***: signed 16-bit
  - ***Decimal***: signed 96-bit (replaces ***Variant***)

Integer Type	VB 6.0	VB.NET
8 bit	Byte	Byte
16 bit	Integer	Short
32 bit	Long	Integer
64 bit	Not Applicable	Long

Changes in Data Handling

- A new data-handling model: ADO.NET.
  - Facilitates Web application.
  - Uses XML to exchange data.
- COM/DCOM technologies have been replaced by .NET framework.
- Datasets (not record sets now) are based on XML schema, so they are strongly typed.
- Many new tools are provided to handle data.
- But can still work with ADO using ***COM interoperability*** in the .NET framework.

Changes in Web Development

- Two major types of Web application:
  - Web forms: web-based applications with GUI.
    - Based on ASP.NET
    - Can use standard HTML control, or new Server control handled by the Web server.
    - Controls can be bound on a Web form by setting the codes in the properties.
  - Web services: to process data using HTTP and XML files on the Internet.

Update to VB.NET ?

- "Visual Basic .NET represents a major departure form previous versions of Visual Basic in several ways."

  ----*Microsoft Corporation*
- Plenty changes in VB.NET will take lots of effort of even the experienced VB developers.
- Old but running fine systems, fund, experienced developers…

Update to VB.NET ? (cont.)

- Consideration
  - Unsupported features
    - OLE Container Control
    - Dynamic Data Exchange
    - DAO or RDO Data Binding
    - VB5 Controls
    - DHTML Applications
    - ActiveX Documents
    - Property Pages

Update to VB.NET ? (cont.)

- Carefully reworked
  - Single-tier Database Applications
  - VB Add-ins
  - Games
  - Graphics
  - Drag and Drop Functionality
  - Variants
  - Windows APIs

Update to VB.NET ? (cont.)

- Visual Basic Upgrade Wizard
  - Automatically invoked when open a VB6 project.
  - Results are not satisfactory due to the big different.
- Recoding by hand.

VB.NET Language Essential ---- For Non-VB Programmers

- Projects Types
  - Three most commonly used:
    - Windows Forms
    - Web Forms
    - Console Applications
- Statements

Statement: If…Else

```
Module Module1
  Sub Main()
     Dim intInput As Integer
     System.Console.WriteLine("Enter an interger…")
     intInput=Val(System.Console.ReadLine())
     If intInput=1 Then
        System.Console.WriteLine("Thank you!")
     ElseIf intInput=2 Then
        System.Console.WriteLine("That's good!")
     Else
        System.Console.WriteLine("Not a right number!")
     End If
  End Sub
End Module
```

Statement: Select Case

```
Module Module1
 Sub Main()
    Dim intInput As Integer
    System.Console.WriteLine("Enter an interger...")
    intInput=Val(System.Console.ReadLine())
    Select Case intInput
        Case 1
            System.Console.WriteLine("Thank you!")
        Case 2
            System.Console.WriteLine("That's good!")
        Case 3 To 7
            System.Console.WriteLine("OK")
        Case Is> 7
            System.Console.WriteLine("Too Big")
        Case Else
            System.Console.WriteLine("Not a right number!")
     End Select
 End Sub
End Module
```

Functions: Switch and Choose

- Switch Function
  - Syntax
    - Switch(expr1, value1[, expr2, value2…[,exprn, valuen]])
  - E.g.
    - intAbsValue=Switch(intValue<0, -1 * intValue, intValue>=0, intValue)
- Choose Function
  - Syntax
    - Choose(index, choice1[, choice2,…[,choicen]])
    - Note: unlike array index, choose index from 1 to n
  - E.g.
    - Str=Choose(intValue, "Thank you!", "That is good!")

Loop Statement: Do

- Syntax:

```
Do [While|Until] condition]
    [statements]
    [Exit Do]
    [statements]
Loop
```

- E.g.

```
Module Module1
  Sub Main()
     Dim strInput As String
     Do Until Ucase(strInput)="Stop"
        System.Console.WriteLine("What should I do?")
        strInput=System.Console.ReadLine()
     Loop
  End Sub
End Module
```

Loop Statement: For

- Syntax:

```
For index=start To end [Step step]
    [statements]
    [Exit For]
    [statements]
Next [index]
```

- E.g.

```
Module Module1
  Sub Main()
    Dim loopIndex As Integer
    For loopIndex=0 to 3
      System.Console.WriteLine("Hello!")
    Next loopIndex
  End Sub
End Module
```

Loop Statement: While

- Syntax:

```
While condition
    [statements]
End While
```

- E.g.

```
Sub CheckWhile()
    Dim intCounter As Integer =0
    Dim intNumber As Integer =10
    While intNumer>6
        intNumber-=1
        intCounter+=1
    End While
    MsgBox("The loop ran " & intCounter & " times.")
End Sub
```

Loop Statement: For Each…Next

- Syntax:

```
For Each element In group
    [statements]
    [Exit For]
    [statements]
Next element
```

- E.g.

```
Sub Main()
    Dim intArray(2), intItem As Integer
    intArray(0)=0
    intArray(1)=1
    intArray(2)=2
    For Each intItem In intArray
        System.Console.WriteLine(intArray)
    Next intItem
End Sub
```

Like a Loop: With

- Syntax:

  With *object*

  [statements]

  End With

- E.g.

```
With TextBox1
    ,Height = 1000
    .Width = 3000
    .Text = "Welcome, World!"
End With
```

Thank you!

Introducing the Microsoft .NET Framework and Visual Basic .NET

Objectives

- Explore the Microsoft .NET Framework
- Write a Visual Basic .NET module definition
- Define Visual Basic .NET variables and data types
- Write basic computational statements
- Read input from the keyboard

Exploring the Microsoft .NET Framework

- .NET Framework key parts:
  - Compilers for:
    - VB .NET
    - Other supported .NET languages
  - Common Language Runtime (CLR)
  - Framework Class Library (FCL)

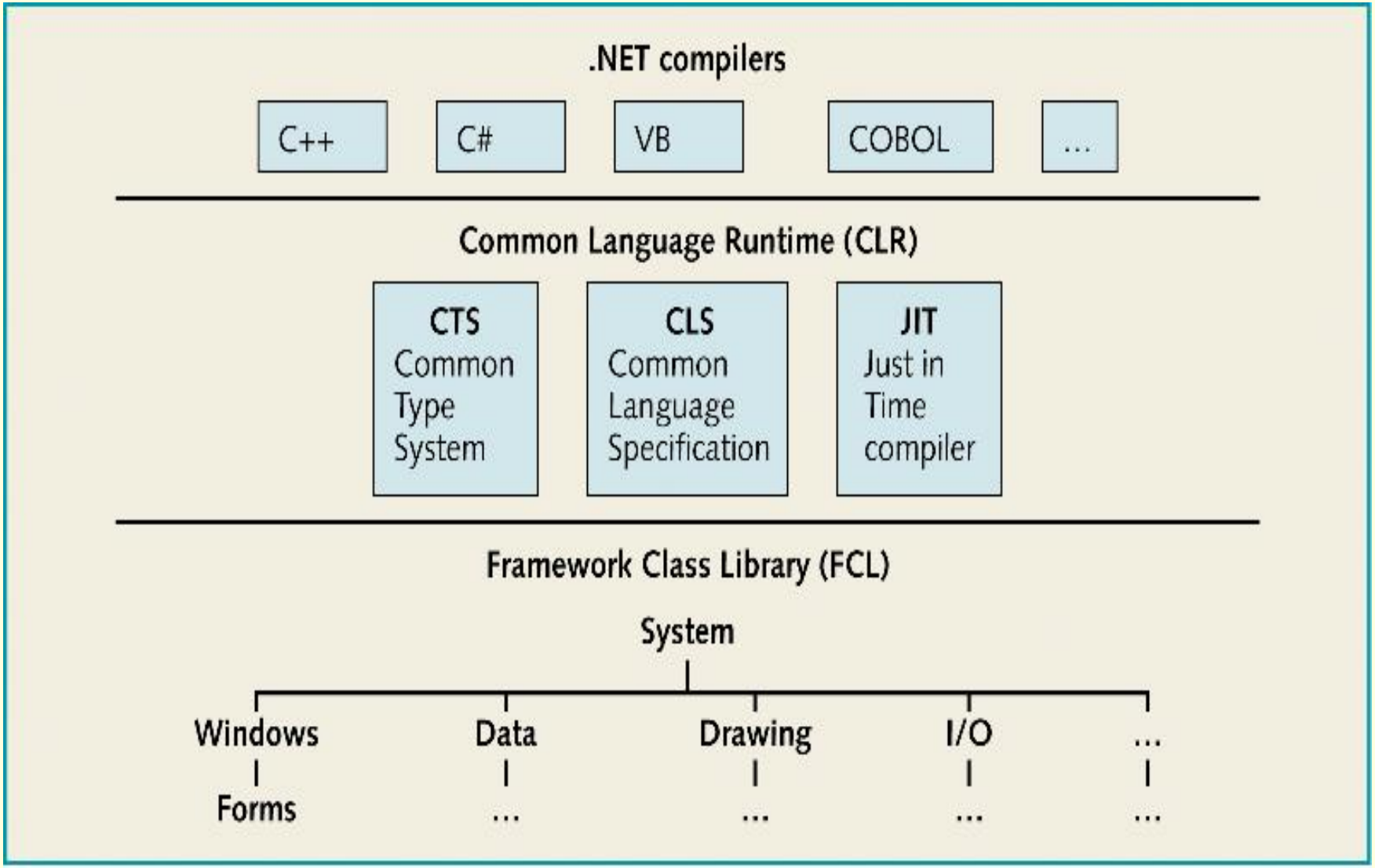

Figure 3-1 The .NET Framework

The Microsoft .NET Compilers

- Includes compilers for:
  - VB
  - C++
  - C#
  - J#
  - COBOL

The Microsoft .NET Compilers (continued)

- Compiler has two primary purposes:
  - Check source code for valid syntax
  - Translate it into executable form
- Compilers translate source code into language called Microsoft Intermediate Language (MSIL)
  - Language used by CLR
  - CLR translates IL into executable code

The Common Language Runtime

- Responsibility:
  - Connect IL files coming from various .NET compilers
  - Translate these into executable files
  - Manage execution of code in file

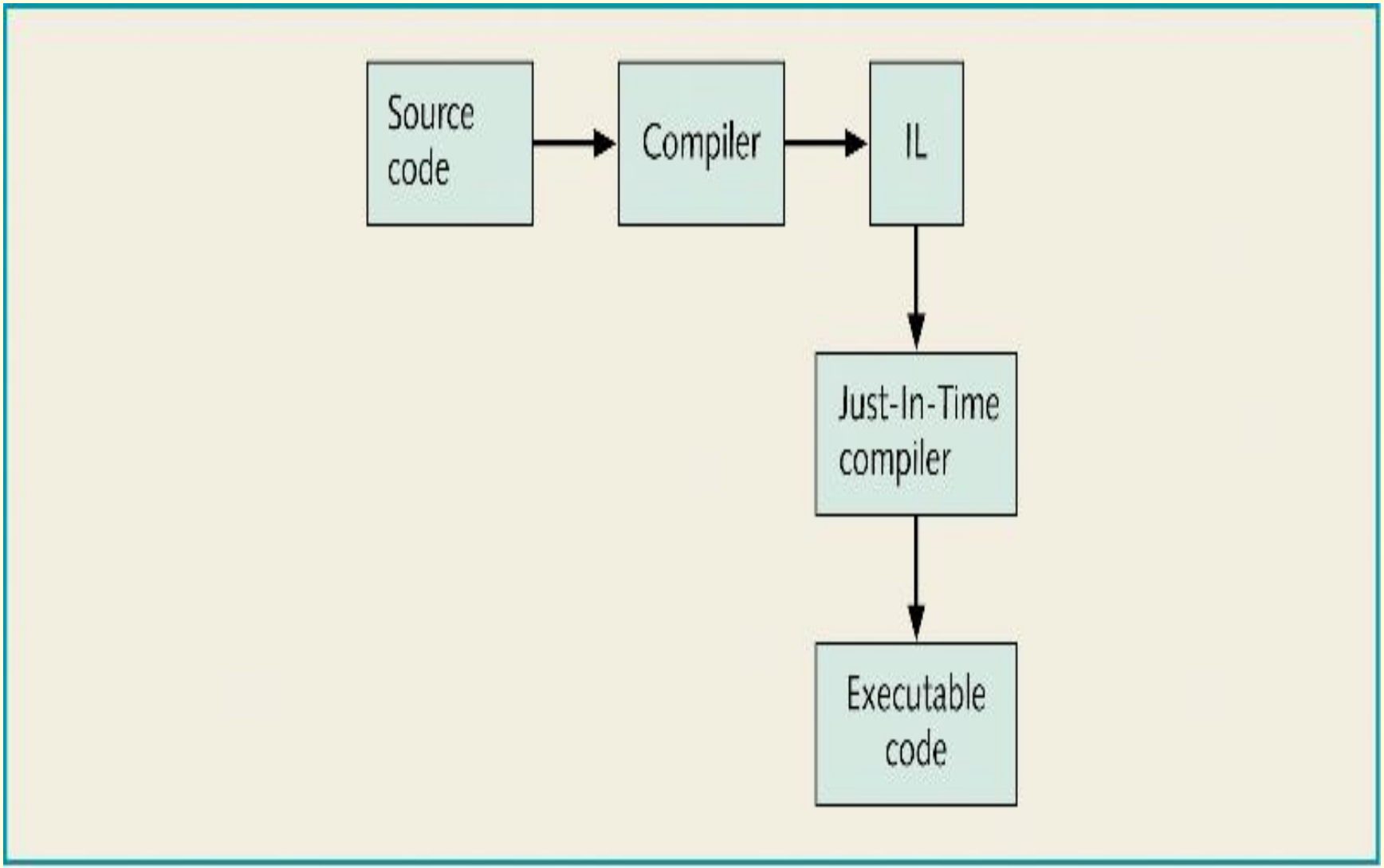

Figure 3-2 Compiling and executing

The Common Language Runtime (continued)

- CLR consists of
  - Common Type System (CTS)
  - Common Language Specification (CLS)
  - Just-In-Time (JIT) compiler
- Allocates and reclaims memory while application running

The Framework Class Library

- Assembly
  - File containing IL
  - Each contains one or more classes
- FLC
  - Consists of approximately 100 assemblies
  - Have suffix of .dll
- Members
  - Methods and attributes in .NET classes

The Framework Class Library (continued)

- Namespaces
  - Organize classes
  - Can contain both classes and other namespaces
  - Compilers do not automatically search all namespaces for classes used by code
    - Must use keyword Imports
    - Tell compiler specific namespaces to access

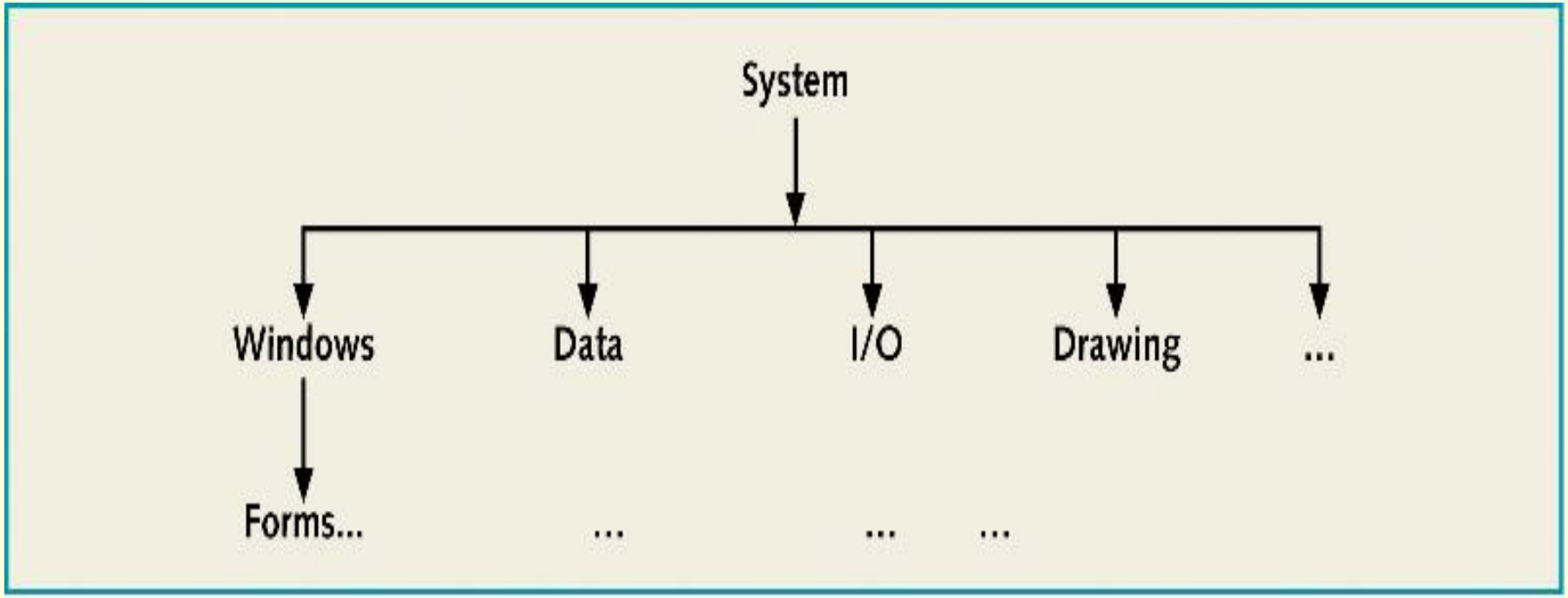

Figure 3-3 The .NET FCL namespaces

Table 3-1 Selected FCL namespaces

Namespace	Selected Classes
System	Array Console Convert DateTime Exception TimeSpan String Math
System.Collections	ArrayList
System.IO	StreamReader StreamWriter
System.Data	DataRow DataTable DataSet
System.Data.OleDb	OleDbCommand OleDbConnection OleDbDataAdapter OleDbParameter
System.Windows.Forms	Button CheckBox Form Label Menu MenuItem RadioButton TextBox

Writing a Visual Basic .NET Module Definition

- Module definition
  - Begins with keyword Module
  - Ends with keyword End Module
- Statements contain:
  - Keywords
  - Identifiers

Writing a Visual Basic .NET Module Definition (continued)

- Identifier
  - Name assigned to things such as:
    - Modules
    - Procedures
    - Variables

Writing a Visual Basic .NET Module Definition (continued)

- Identifier naming rules:
  - Can be up to 1023 characters long
  - Can include any:
    - Letter
    - Number
    - Underscore character
    - No spaces
  - Cannot begin with a number
  - Cannot be a keyword

Writing a Visual Basic .NET Module Definition (continued)

- Code not case sensitive
- Comment lines
  - Add explanations to code
  - Ignored by compiler
- Module header
  - Names module
  - Syntax:
    - Module modulename

Writing a Visual Basic .NET Module Definition (continued)

- Procedure:
  - Contains statements that perform processing
  - Types:
    - Sub
    - Function
  - Begin with header
- Procedure Main invoked automatically

Writing a Visual Basic .NET Module Definition (continued)

- Argument
  - Information contained in parentheses when calling procedure
  - Passed to procedure
- Literal
  - Value defined within a statement

Defining Visual Basic .NET Variables And Data Types

- Variable
  - Memory location that contains data
  - Characteristics:
    - Name
    - Data type
    - Value

Understanding VB .NET Data Types

- Each variable has a data type
- Can be:
  - Primitive
  - Complex
- Unicode character set
  - Allocates two bytes for each character
  - Accommodates all characters of major international languages

Table 3-3 VB .NET primitive data types

	Type	Range of Values	Size
Numeric with no decimals	1. `Byte`	0 to 255	8 bits
	2. `Short`	-32,768 to 32,767	16 bits
	3. `Integer`	-2,147,483,648 to 2,147,483,647	32 bits
	4. `Long`	±9,223,372,036,854,775,807	64 bits
Numeric with decimals	5. `Single`	±1.5E-45 to ±3.4E+38; up to 6 decimal positions	32 bits
	6. `Double`	±5.0E-324 to ±1.7E+308; up to 14 decimal positions	64 bits
	7. `Decimal`	1.0E-28 to 7.9E+28; up to 28 decimal positions	128 bits
Other	8. `Boolean`	`True` or `False`	16 bits
	9. `Char`	Any Unicode character	16 bits

Declaring and Populating Variables

- Declaration statements
  - Define variables
- Syntax:
  - Dim variablename As datatype
- Assignment operator
  - =
  - Assigns value on right side to variable named on left side

Example 3-2: Declaring Variables

Dim myInteger **As Integer**

Dim myDouble **As Double**

Dim myBoolean **As Boolean**

Example 3-4: Populating Variables

myInteger = 1

myDouble = 2.5

Defining Constants

- Constant
  - Variable with a value that does not change
  - Contain values such as:
    - Company name
    - Tax identification number
  - Syntax:
    - Const constantname As datatype
  - Must be initialized in the same statement that declares them

Defining Constants (continued)

- Naming convention:
  - Capitalize constant names
  - If name consists of more than one word
    - Separate words with underscore character (_)
  - Example:
    - TAX_ID

Converting Data Types

- Numeric data types have different capacities:
  - Byte variable can hold maximum value of 255
  - Integer variable has maximum value of 2.1 billion
- Implicit type conversion
  - Use assignment operator to assign contents of variable to a variable with different data type

Example 3-7: Implicit Type Conversion

```
Dim myInteger As Integer = 1
Dim myDouble As Double = 2.5
myDouble = myInteger
```

- Assign Integer value to Double variable
  - Data type Double has greater capacity than Integer
  - No potential loss of data

Example 3-8: Loss of Precision

- Loss of precision
  - Computing error that can occur when decimal positions are dropped

```
Dim myInteger As Integer = 1

Dim myDouble As Double = 2.5

myInteger = myDouble
```

- VB .NET will automatically round decimal values before truncating

Example 3-8: Loss of Precision (continued)

- Option Strict
  - Prevent unintentional loss of precision when mixing data types in assignment statements
  - Compiler detects potential loss of precision
    - Displays error message
- Explicit type conversion
  - Invoke Convert method to convert data types

Table 3-4 Methods in the Convert class

Method	Description
`ToInt16(x)`	Converts the argument to `Short`
`ToInt32(x)`	Converts the argument to `Integer`
`ToInt64(x)`	Converts the argument to `Long`
`ToSingle(x)`	Converts the argument to `Single`
`ToDouble(x)`	Converts the argument to `Double`
`ToString(x)`	Converts the argument to `String`

Converting Data Types (continued)

- Option Explicit
  - Must define variable before using it in a statement
  - Otherwise
    - Compiler generates error message
  - Generally set On

Using Reference Variables

- Uses class name as data type
- For example:
  - String
- Variable refers to or points to instance of class
  - Does not actually contain data
  - Contains memory address of instance of class that contains data

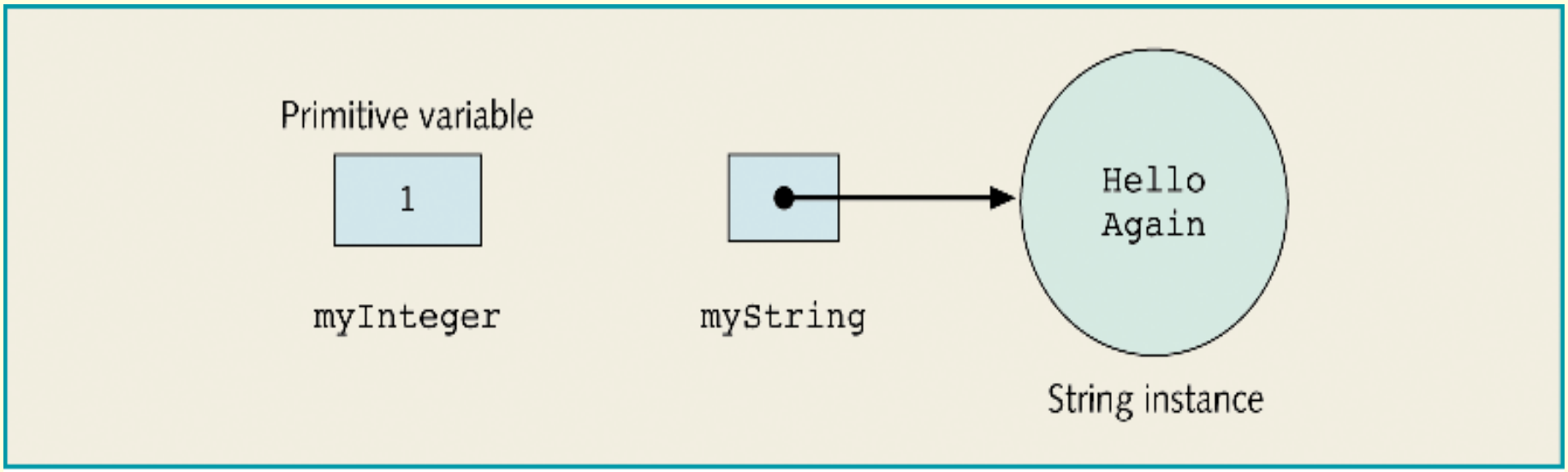

Figure 3-5 Contrasting primitive and reference variables

Writing Basic Computational Statements

- Concatenate operator
  - &
  - Joins two Strings
- Arithmetic operators
  - For multiplication, division, addition, and subtraction
  - *, /, +, -

Using the Arithmetic Operators

- Evaluated in predetermined order called precedence
  - Standard algebraic rules of precedence apply
- Other operators:
  - Exponentiation
  - Integer division
  - Remainder computation

Example 3-15: Integer Division (\)

```
Dim firstInt As Integer = 11
Dim secondInt As Integer = 2
Dim integerResult As Integer = 0
integerResult = firstInt \ secondInt
Console.WriteLine("integerResult = firstInt \
    secondInt: " &  integerResult)
```

- Sample Run:
  - integerResult = firstInt \ secondInt: 5

Table 3-5: VB .NET arithmetic operators

Operator	Description	Example	Result
^	Exponentiation	11 ^ 2	121
*	Multiplication	11 * 2	22
/	Division	11 / 2	5.5
\	Integer division	11 \ 2	5
Mod	Remainder	11 Mod 2	1
+	Addition	11 + 2	13
-	Subtraction	11 - 2	9

Using the Arithmetic Operators (continued)

- Assignment operators:
  - Formed by combining arithmetic operator with assignment operator
  - Example:
    - i += 1

Invoking Methods in the Math Class

- System namespace includes Math class
  - Contains methods to accomplish
    - Exponentiation
    - Rounding
    - Trigonometric calculations
- Use .NET Help facility to explore methods
- Invoke method:
  - Math.Pow(firstInt, secondInt)

Invoking Methods in the Math Class (continued)

- Math class constants:
  - PI
  - E
  - To access:
    - Math.E

Reading Input From the Keyboard

- Use Console class
  - ReadLine method
    - Read one or more characters from keyboard
    - Convert any numeric data to desired data type
- Prompt
  - Message displayed to user asking for input