COLEGIO DE
STA. TERESA DE AVILA
SCHOOL OF INFORMATION TECHNOLOGY

CPRO2
COMPUTER PROGRAMMING 2

Generic Collections, .TryParse Method and Currency Format

Image Marie Maiquez
March 2025
7th Week

Generic Collection

A **generic collection** is a collection that can store any specific data type while ensuring type safety and flexibility. It is part of the System.Collections.Generic namespace, and allows you to work with groups of objects without specifying a fixed data type at design time.

Examples of Generic Collections:

List(Of T) - A dynamic list that stores a specific type (e.g., List(Of String), List(Of Integer))

Dictionary(Of TKey, TValue) - A key-value pair collection.

Queue(Of T) - A first-in, first-out (FIFO) collection

Stack(Of T) - A last-in, first-out (LIFO) collection

HashSet(Of T) – A collection that prevents duplicate values.

Why Use Generic Collections?

Type Safety: Prevents storing incorrect data types.

```
Dim names As New List(Of String)
names.Add("John") ' Valid
names.Add(123) ' ERROR!
```

Better Performance: Avoids unnecessary type conversions (boxing/unboxing).

Flexibility & Reusability: Works with any data type.

Dynamic Resizing: Unlike arrays, generic collections grow automatically as elements are added.

When Should We Use Generic Collections?

- **When storing multiple items** of the same type dynamically
  (e.g., a list of student names, product prices, or employee records).
- **When we need fast searching**, inserting, or deleting elements
  (e.g., using a Dictionary(Of String, Integer) for quick lookups).
- **When order matter**
  (e.g., using Queue(Of String) for processing tasks in sequence).
- **When avoiding duplicate values** is important
  (e.g., using HashSet(Of Integer) to store unique IDs).

Currency Format

ToString("C") is used to format a numeric value as currency based on the system's regional currency settings.

- Converts the price (which is a Decimal or Double) into a currency-formatted string.
- Displays the value with the correct currency symbol and decimal places based on the user's system locale.

.TryParse Method

.TryParse is a method used to safely convert a string into a number (or other data types) without causing an error if the conversion fails.

Dim result As Boolean = Boolean.TryParse(input, outputVariable)

Returns: True if the conversion is successful, False if it fails.

Parameters:

input - The string to convert.

outputVariable - A variable where the converted value will be stored if successful.

.TryParse Method

```
Dim userInput As String = "123"
Dim number As Integer

If Integer.TryParse(userInput, number) Then
    MessageBox.Show("Valid number: " & number)
Else
    MessageBox.Show("Invalid input!")
End If
```

Project: PrjCurrency. Adding new price with currency format using ComboBox, TextBox, Label and Button

```
Public Class Form1
    Dim productPrices As New List(Of Decimal) ' Stores decimal values

Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
        Dim price As Decimal
        If Decimal.TryParse(txtPrice.Text, price) Then
            productPrices.Add(price)
            cboPrice.Items.Add(price.ToString("C")) ‘ ComboBox name: cboPrice
        Else
            MsgBox("Enter a valid price.“,vbInformation)
        End If
    End Sub
End Class
```

COLEGIO DE
STA. TERESA DE AVILA

SCHOOL OF INFORMATION TECHNOLOGY

CPRO2
COMPUTER PROGRAMMING 2

VB.Net Array

Image Marie Maiquez
March 2025
8th Week

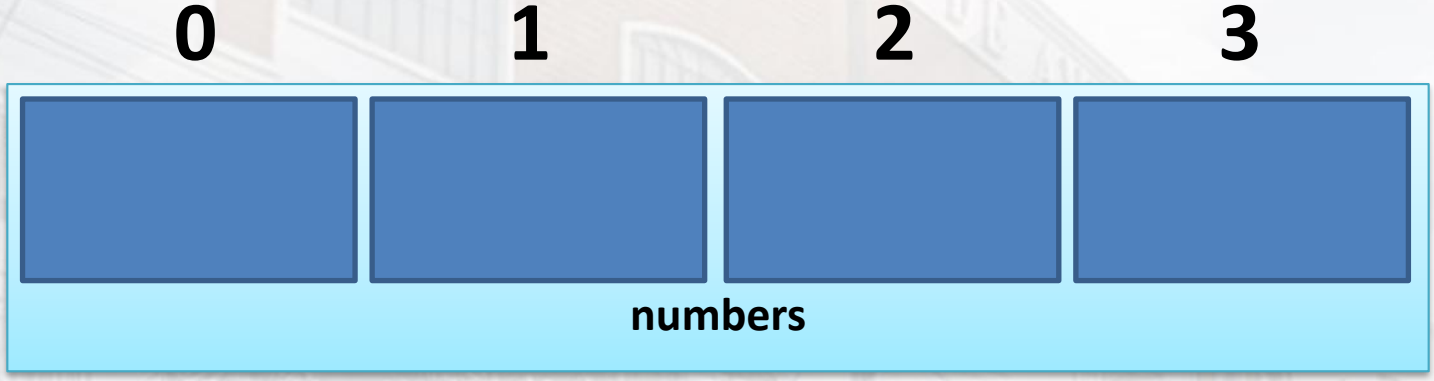
Array

An **array** is a collection of elements of the same data type stored in contiguous memory locations. Arrays allow you to store and manipulate multiple values using a **single variable name**.

Declaring and Initializing Arrays

```
Dim numbers(3) As Integer  ' Declares an array with 4 elements (0 to 3)
```

Array Indexing: Arrays in VB.NET are zero-based.
(i.e., first element is at index 0).



Initializing an Array with String Values

Dim colors() As String = {“Red”, "Green", "Blue"}

Output:
colors(0) - "Red“
colors(1) - "Green“
colors(2) - "Blue"

Looping Through a String Values in an Array

```
Dim names() As String = {“Harold", “Valeree", “James"}

For Each name In names
    Console.WriteLine(name)
Next
```

Multi-dimensional Arrays

A multi-dimensional array allows the storage of tabular or grid-like data.

Declaring a 2D Array:

```
Dim matrix(2, 2) As Integer
'This creates a 3×3 matrix (0-based index).
```

Initializing a 2D Array:

```
Dim grid(,) As Integer = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
}
```

Multi-dimensional Arrays

Accessing Elements:

```
Console.WriteLine(grid(1,2)) ' Output: 6
```

Looping through a 2D Array:

```
For i As Integer = 0 To 2
    For j As Integer = 0 To 2
        Console.Write(grid(i, j) & " ")
    Next
        Console.WriteLine()
Next
```

Array.Sort Method

In VB.NET, you can sort an array using the built-in Array.Sort method. This method sorts the elements in ascending order by default.

Sorting a Numeric Array:

```
Dim numbers() As Integer = {5, 2, 8, 1, 3}
Array.Sort(numbers) ' Sorts in ascending order

' Display sorted array
For Each num As Integer In numbers
    Console.WriteLine(num)
Next
```

Control Arrays

A Control Array is a group of controls that share the same name and can be managed collectively.

Button Control Array:

```
Dim buttons(3) As Button
For i As Integer = 0 To 2
    buttons(i) = New Button()
    buttons(i).Text = "Button " & (i + 1)
    buttons(i).Location = New Point(20, i * 30 + 10)
    Me.Controls.Add(buttons(i))
Next
```

Referencing and Passing Arrays to Procedures

In VB.NET, arrays can be passed to functions and subroutines as parameters.

Passing an Array to a Procedure (By Reference):

```
Sub PrintArray(ByVal arr() As Integer)
    For Each num In arr
        Console.WriteLine(num)
    Next
End Sub

Dim numbers() As Integer = {1, 2, 3, 4, 5}
PrintArray(numbers) ' Call the function and pass the array
```

Passing an Array to a Function and Returning a Value

```
Function GetSum(ByVal arr() As Integer) As Integer
    Dim sum As Integer = 0
    For Each num In arr
        sum += num
    Next
    Return sum
End Function

Dim values() As Integer = {10, 20, 30}
Dim total As Integer = GetSum(values)
Console.WriteLine("Total Sum: " & total) ' Output: Total Sum: 60
```