

Validating Data Annotations Programmatically Labs

Perform these labs on your own computer using Visual Studio 2022 or later to ensure you understand the lessons presented in the corresponding videos and lectures.

Lab 1a: Create a Console Application Using Visual Studio

If you are using Visual Studio 2022, follow the instructions in this lab. If you are using VS Code, please proceed to **Lab 1b**.

Open Visual Studio 2022 and select Create a new project as shown in Figure 1.

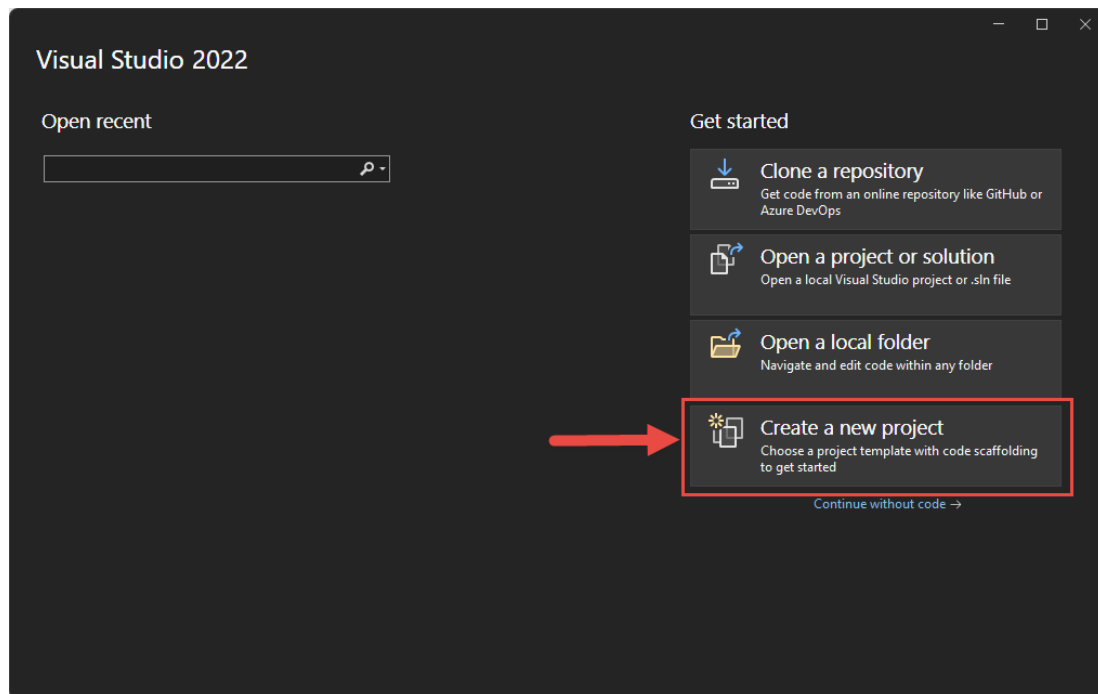


Figure 1: Select Create a new project from the Visual Studio screen.

On the Create a new project page, select a **Console App** from the list of templates (Figure 2). Be sure to choose the **C#** language. Click the **Next** button.

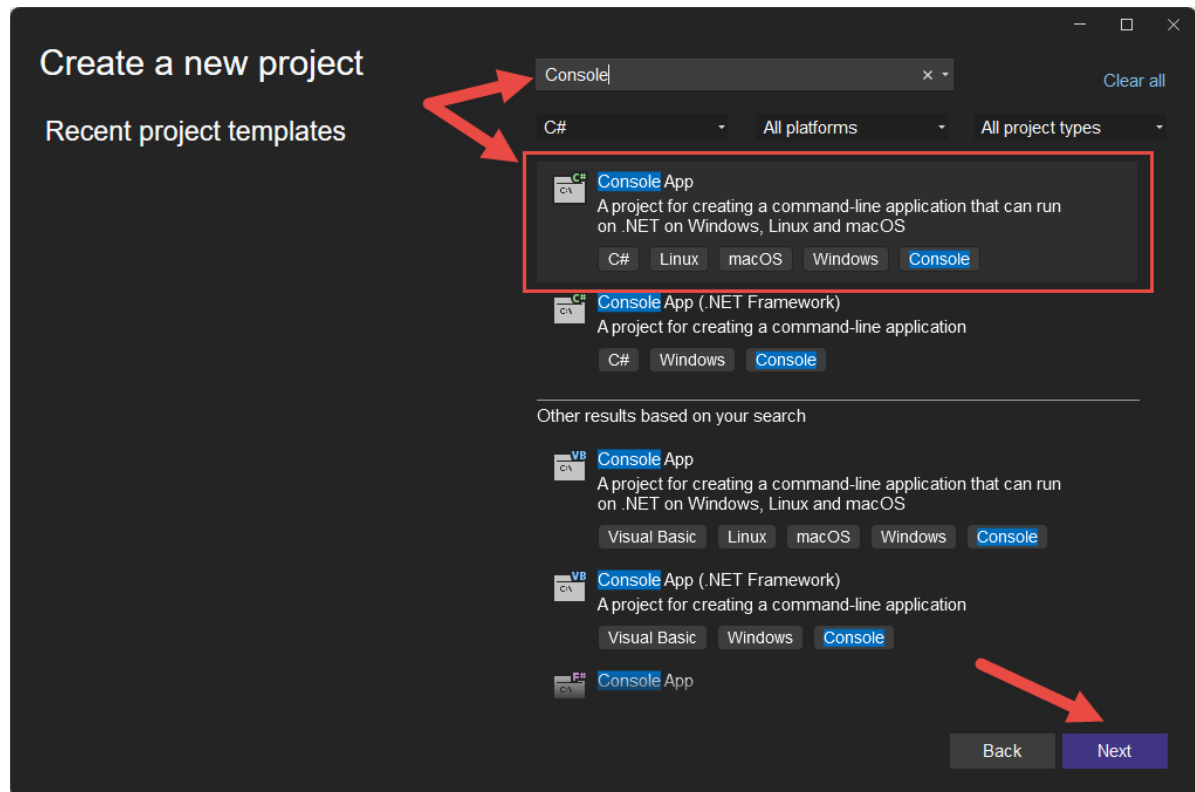


Figure 2: Select a Console App from the list of templates.

On the Configure your new project screen, set the **Project name** of the new project to **DataAnnotationsSamples** (Figure 3).

Set the **Location** to a folder on your hard drive.

Uncheck the **Place solution and project in the same directory**.

Click the **Next** button.

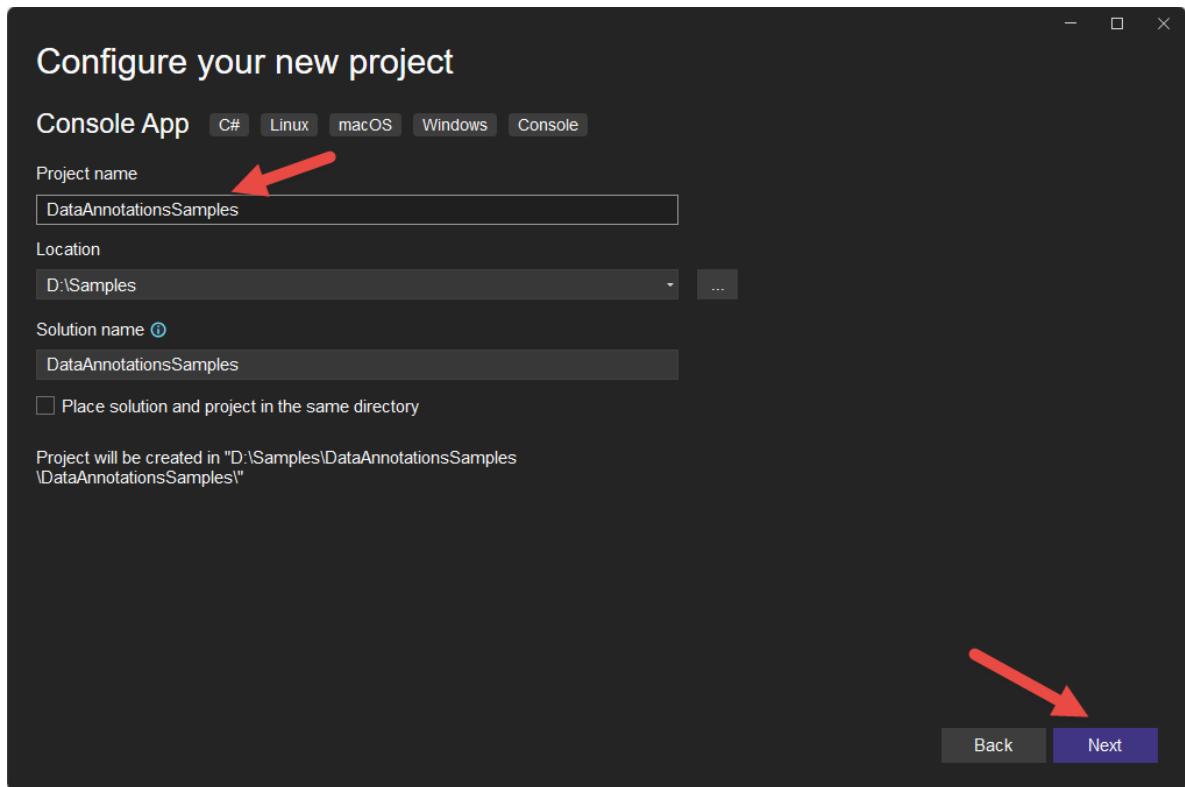


Figure 3: Set the project name and location.

On the Additional information screen, choose **.NET 8** or later (Figure 4).

Ensure the other two check boxes are unchecked.

Click on the **Create** button.

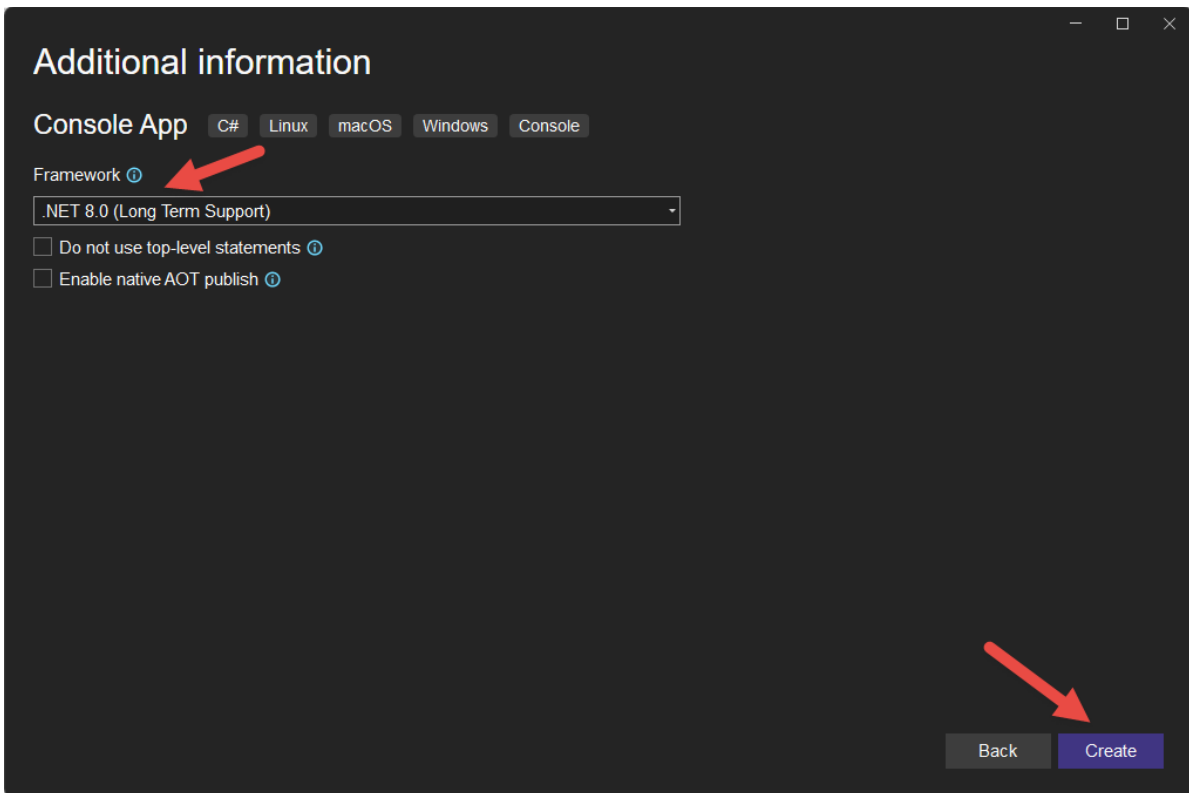


Figure 4: Choose .NET 8 or later.

At this point, please proceed to **Lab 2**.

Lab 1b: Create a Console Application Using VS Code

If you are using VS code, follow the instructions in this lab.

Open VS Code and open a Terminal Window by clicking on the **Terminal | New Terminal** menu.

Navigate to the folder where you like to create your projects. On my machine I use my D drive and the Samples folder as shown in Figure 5.

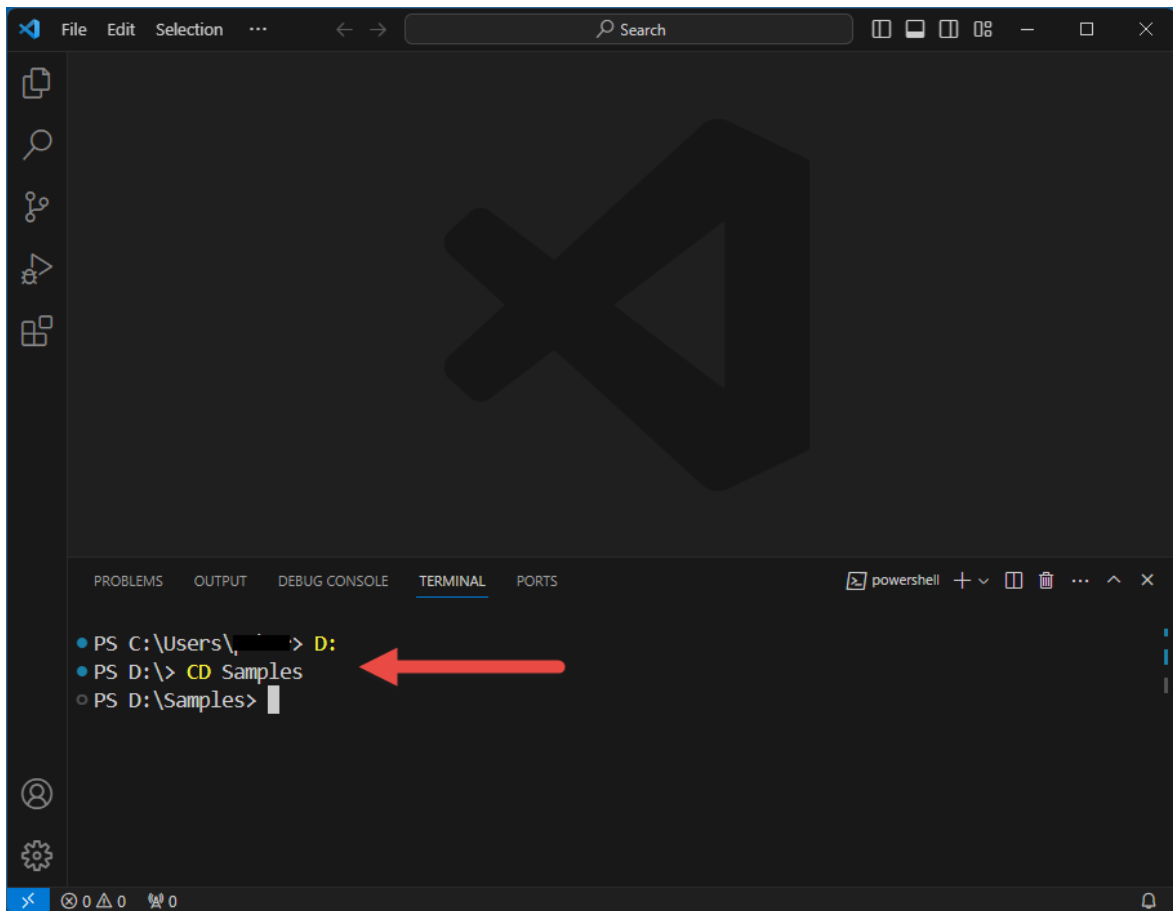


Figure 5: Navigate to where you wish to create your project using the Terminal.

Create a new folder by typing into the terminal window

```
MKDIR DataAnnotationsSamples
```

Navigate to this new folder.

```
CD DataAnnotationsSamples
```

Create a new .NET 8 console application.

```
dotnet new console --framework net8.0
```

Open this new folder by selecting **File | Open Folder...** from the menu. Select the **D:\Samples\DataAnnotationsSamples** folder (or wherever the folder is you created).

If you are prompted if you wish to trust the authors of this folder, select the **Yes, I trust the authors** button.

Click on the **Program.cs** file in the Explorer window.

Lab 2: Create Validation Helper Class

Add a new folder to your project named **ValidationClasses**.

Right mouse-click on the **ValidationClasses** folder and add a new class named **ValidationHelper**.

Replace the entire contents of this new file with the following code.

```
using System.ComponentModel.DataAnnotations;

namespace Common.Library;

public static class ValidationHelper
{
    #region Validate<T> Method
    /// <summary>
    /// Call this method to validate any entity class with
    Data Annotations
    /// </summary>
    /// <param name="entity">The entity to
    validate</param>
    /// <returns>Zero or more ValidationResult
    objects</returns>
    public static List<ValidationResult> Validate<T>(T
entity) where T : class
    {
        List<ValidationResult> ret = new();

        // Create instance of ValidationContext object
        ValidationContext context = new(entity);

        // Call TryValidateObject() method
        Validator.TryValidateObject(entity, context, ret,
true);

        return ret;
    }
    #endregion
}
```

Lab 3: Add and Validate a Product Class

Add a new folder to your project named **EntityClasses**.

Right mouse-click on the **EntityClasses** folder and add a new class named **Product**.

Replace the entire contents of this new file with the following code.

```
namespace DataAnnotationsSamples;

public partial class Product {
    public int ProductID { get; set; }
    public string Name { get; set; } = string.Empty;
    public string ProductNumber { get; set; } =
string.Empty;
    public string Color { get; set; } = string.Empty;
    public decimal StandardCost { get; set; }
    public decimal ListPrice { get; set; }
    public DateTime SellStartDate { get; set; }
    public DateTime? SellEndDate { get; set; }
    public DateTime? DiscontinuedDate { get; set; }

    public override string ToString() {
        return $"{Name} ({ProductID})";
    }
}
```

Open the **Program.cs** file and replace the entire contents of this new file with the following code.

```
using Common.Library;
using DataAnnotationsSamples;
using System.ComponentModel.DataAnnotations;

List<ValidationResult> msgs;

Product entity = new() {
    ProductID = 1,
    Name = "",
    ProductNumber = "",
    Color = "",
    StandardCost = 0,
    ListPrice = 0,
    SellStartDate = Convert.ToDateTime("1/1/1999"),
    SellEndDate = Convert.ToDateTime("1/1/2031"),
    DiscontinuedDate = DateTime.Now
};

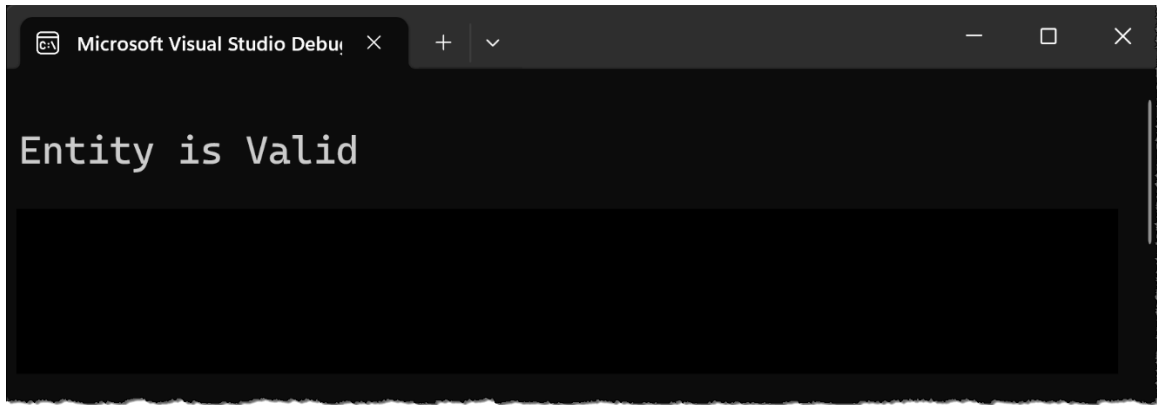
msgs = ValidationHelper.Validate(entity);

if (msgs.Count > 0) {
    // Display Failed Validation Messages
    foreach (var item in msgs) {
        Console.WriteLine(item);
    }

    // Display Total Count
    Console.WriteLine();
    Console.WriteLine($"Total Validations Failed:
{msgs.Count}");
}
else {
    Console.WriteLine();
    Console.WriteLine("Entity is Valid");
}
```

Try It Out

Run the application and view the results. You should see a message appear as shown in the figure below.



Lab 4: Using the [Required] Attribute

Open the **Product.cs** file and add the **[Required]** attribute above the **Name** property declaration as shown below.

```
[Required]
public string Name { get; set; } = string.Empty;
```

Add the **[Required]** attribute above the **ProductNumber** property declaration as shown below.

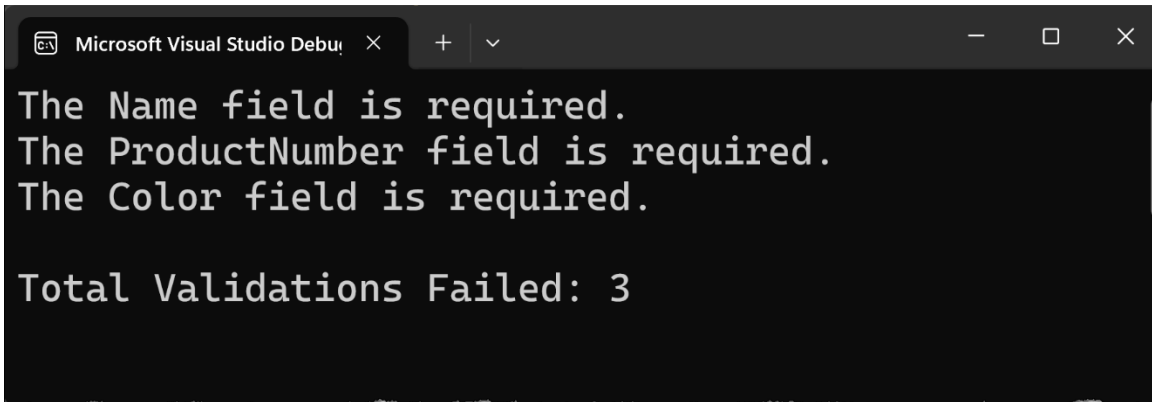
```
[Required]
public string ProductNumber { get; set; } =
string.Empty;
```

Add the **[Required]** attribute above the **Color** property declaration as shown below.

```
[Required]
public string Color { get; set; } = string.Empty;
```

Try It Out

Run the application and view the results. You should see the messages appear as shown in the figure below.



```
Microsoft Visual Studio Debug Console
The Name field is required.
The ProductNumber field is required.
The Color field is required.

Total Validations Failed: 3
```

Lab 5: Add Other Methods to Validation Helper

Right mouse-click on the **ValidationClasses** folder and add a new class named **ValidationMessage**. Replace the entire contents of this new file with the following code.

```
namespace Common.Library;

public class ValidationMessage
{
    public string PropertyName { get; set; } =
    string.Empty;

    public string ErrorMessage { get; set; } =
    string.Empty;

    public override string ToString()
    {
        return $"{ErrorMessage}";
    }
}
```

Open the **ValidationHelper.cs** file and add the following methods to this class.

```

#region ValidateToValidationMessage Method
/// <summary>
/// Convert a List<ValidationResult> objects into a
List<ValidateMessage>. This is typically used when you
with to perform data binding on both the property name
in error and the validation message.
/// </summary>
/// <param name="results">A collection of
ValidationResult objects</param>
/// <returns>A Dictionary with the validation
errors</returns>
public static List<ValidationMessage>
ValidateToValidationMessage<T>(T entity) where T : class
{
    List<ValidationMessage> ret = new();

    // Validate Entity
    List<ValidationResult> results = Validate(entity);

    // See if any validation results returned
    if (results.Count != 0) {
        // Iterate over validation results
        foreach (ValidationResult item in results) {
            string propName = "UnknownProperty";
            // See if there is a property name
            if (item.MemberNames.Any()) {
                propName = item.MemberNames.ToArray()[0];
            }
            // Create a new ValidationMessage object and add
            to return results
            ret.Add(new() {
                PropertyName = propName,
                ErrorMessage = item.ErrorMessage ?? "Unknown
Validation Error"
            });
        }

        return ret;
    }
}
#endregion

```

Add another method for Dictionary

```
#region ValidateToDictionary Method
/// <summary>
/// Convert a List<ValidationResult> objects into a
dictionary. This is typically used in Web API
applications where you need to return the collection of
validation errors in a 400 Bad Request.
/// </summary>
/// <param name="results">A collection of
ValidationResult objects</param>
/// <returns>A Dictionary with the validation
errors</returns>
public static Dictionary<string, string[]>
ValidateToDictionary<T>(T entity) where T : class
{
    Dictionary<string, string[]> ret = new();

    // Validate Entity
    List<ValidationResult> results = Validate(entity);

    // See if any validation results returned
    if (results.Count != 0) {
        // Iterate over validation results
        foreach (ValidationResult item in results) {
            string propName = "UnknownProperty";
            // See if there is a property name
            if (item.MemberNames.Any()) {
                propName = item.MemberNames.ToArray()[0];
            }
            // Create a new dictionary object and add to
            return results
            ret.Add(propName, new string[] {
                item.ErrorMessage ?? "Unknown Validation
Error"
            });
        }
    }

    return ret;
}
#endregion
```

You are not going to need these methods for this course, but they will be useful to you in the real world. The `ValidateToValidationMessage()` method is useful when you want to bind a collection of validation messages to a list box or other type of list control in any type of application. The `ValidationMessage` class is much simpler

than either the dictionary or the `ValidationResult` class as it only has two properties as opposed to a single property and a string array property.

The `ValidateToDictionary()` method is useful when you need to return validation errors back from a call to a Web API. You wrap this within a status code of 400 and it produces a standard JSON object to describe each property that has validation errors.