# Module 3

Basic types and constructs of Visual C#

# Module Overview

- Implementing Structs and Enums
- Organizing Data into Collections
- Handling Events

# Lesson 1: Implementing Structs and Enums

- Creating and Using Enums
- Creating and Using Structs
- Initializing Structs
- Creating Properties
- Creating Indexers
- Demonstration: Creating and Using a Struct

# Creating and Using Enums

- Create variables with a fixed set of possible values

```
enum Day { Sunday, Monday, Tuesday, Wednesday, … };
```

- Set instance to the member you want to use

```
Day favoriteDay = Day.Friday;
```

- Set enum variables by name or by value

```
Day day1 = Day.Friday;
// is equivalent to
Day day1 = (Day)4;
```

# Flags Enums

- Enums that can have multiple values

- Decorate with Flags attribute
- Provide power-of-2 numerical value

```
[Flags]
enum Weather {
    Warm =1,
    Cold = 2,
    Sunny = 4,
    // …
    Windy = 128
}
```

- Combine with a “pipe” (logical “or”) character

```
Weather brussels = Weather.Cold | Weather.Windy;
```

# Creating and Using Structs

- Use structs to create simple custom types:
  - Represent related data items as a single logical entity
  - Add fields, properties, methods, and events

- Use the **struct** keyword to create a struct

```
public struct Coffee { ... }
```

- Use the **new** keyword to instantiate a struct

```
Coffee coffee1 = new Coffee();
```

# Initializing Structs

- Use constructors to initialize a struct

```
public struct Coffee
{
   public Coffee(int strength, string bean, string origin)
   { ... }
}
```

- Provide arguments when you instantiate the struct

```
Coffee coffee1 = new Coffee(4, "Arabica", "Columbia");
```

- Add multiple constructors with different combinations of parameters

# Creating Properties

- Properties use get and set accessors to control access to private fields

```
private int strength;
public int Strength
{
   get { return strength; }
   set { strength = value; }
}
```

- Properties enable you to:
  - Control access to private fields
  - Change accessor implementations without affecting clients
  - Data-bind controls to property values

# Creating Indexers

- Use the **this** keyword to declare an indexer
- Use **get** and **set** accessors to provide access to the collection

```
public int this[int index]
{
   get { return this.beverages[index]; }
   set { this.beverages[index] = value; }
}
```

- Use the instance name to interact with the indexer

```
Menu myMenu = new Menu();
string firstDrink = myMenu[0];
```

# Demonstration: Creating and Using a Struct

In this demonstration, you will learn how to:

- Create a custom struct
- Add properties to a custom struct
- Use a custom struct in the same way that you would use a standard .NET Framework type

# Lesson 2: Organizing Data into Collections

- Choosing Collections
- Standard Collection Classes
- Specialized Collection Classes
- Using List Collections
- Using Dictionary Collections
- Querying a Collection

# Choosing Collections

- *List* classes store linear collections of items
- *Dictionary* classes store collections of key/value pairs
- *Queue* classes store items in a first in, first out collection
- *Stack* classes store items in a last in, first out collection

# Standard Collection Classes

| Class | Description |
| --- | --- |
| ArrayList | • General-purpose list collection<br>• Linear collection of objects |
| BitArray | • Collection of Boolean values<br>• Useful for bitwise operations and Boolean arithmetic (for example, AND, NOT, and XOR) |
| Hashtable | • General-purpose dictionary collection<br>• Stores key/value object pairs |
| Queue | • First in, first out collection |
| SortedList | • Dictionary collection sorted by key<br>• Retrieve items by index as well as by key |
| Stack | • Last in, first out collection |

# Specialized Collection Classes

| Class | Description |
| --- | --- |
| ListDictionary | • Dictionary collection<br>• Optimized for small collections (<10) |
| HybridDictionary | • Dictionary collection<br>• Implemented as ListDictionary when small, changes to Hashtable as collection grows larger |
| OrderedDictionary | • Unsorted dictionary collection<br>• Retrieve items by index as well as by key |
| NameValueCollection | • Dictionary collection in which both keys and values are strings<br>• Retrieve items by index as well as by key |
| StringCollection | • List collection in which all items are strings |
| StringDictionary | • Dictionary collection in which both keys and values are strings |
| BitVector32 | • Fixed size 32-bit structure<br>• Represent values as Booleans or integers |

# Using List Collections

- Add objects of any type

```
Coffee coffee1 = new Coffee(4, "Arabica", "Columbia");
ArrayList beverages = new ArrayList();
beverages.Add(coffee1);
```

- Retrieve items by index

```
Coffee firstCoffee = (Coffee)beverages[0];
```

- Use a foreach loop to iterate over the collection

```
foreach(Coffee c in beverages)
{
   // Console.WriteLine(c.CountryOfOrigin);
}
```

# Using Dictionary Collections

- Specify both a key and a value when you add an item

```
Hashtable ingredients = new Hashtable();
ingredients.Add("Café Mocha", "Coffee, Milk, Chocolate");
```

- Retrieve items by key

```
string recipeMocha = ingredients["Café Mocha"];
```

- Iterate over key collection or value collection

```
foreach(string key in ingredients.Keys)
{
   Console.WriteLine(ingredients[key]);
}
```

# Querying a Collection

- Use LINQ expressions to query collections

```
var drinks =
   from string drink in prices.Keys
   orderby prices[drink] ascending
   select drink;
```

- Use extensions methods to retrieve specific items from results

```
decimal lowestPrice = drinks.FirstOrDefault();
decimal highestPrice = drinks.Last();
```

# Lesson 3: Handling Events

- Creating Events and Delegates
- Raising Events
- Subscribing to Events
- Demonstration: Working with Events in XAML
- Demonstration: Writing Code for the Grades Prototype Application Lab

# Subscribing to Events

- Create a method that matches the delegate signature

```
public void HandleOutOfBeans(Coffee c, EventArgs e)
{
   // Do something useful here.
}
```

- Subscribe to the event

```
coffee1.OutOfBeans += HandleOutOfBeans;
```

- Unsubscribe from the event

```
coffee1.OutOfBeans -= HandleOutOfBeans;
```

# Demonstration: Writing Code for the Grades Prototype Application Lab

In this demonstration, you will learn how to:

- Create an event handler for a button click event in XAML
- Use the event handler to set the contents of a label

# Creating Events and Delegates

- Create a delegate for the event

```
public delegate void OutOfBeansHandler(Coffee coffee,
EventArgs args);
```

- Create the event and specify the delegate

```
public event OutOfBeansHandler OutOfBeans;
```

# Raising Events

- Check whether the event is null
- Raise the event by using method syntax

```
if (OutOfBeans != null)
{
   OutOfBeans(this, e);
}
```

# Demonstration: Writing Code for the Grades Prototype Application Lab

In this demonstration, you will learn about the tasks that you will perform in the lab for this module

# Lab: Writing the Code for the Grades Prototype Application

- Exercise 1: Adding Navigation Logic to the Grades Prototype Application
- Exercise 2: Creating Data Types to Store User and Grade Information
- Exercise 3: Displaying User and Grade Information

Estimated Time: 90 minutes

# Lab Scenario

The School of Fine Arts has decided that they want to extend their basic class enrollment application to enable teachers to record the grades that students in their class have achieved for each subject, and to allow students to view their own grades. This functionality necessitates implementing application log on functionality to authenticate the user and to determine whether the user is a teacher or a student.

You decide to start by developing parts of a prototype application to test proof of concept and to obtain client feedback before embarking on the final application. The prototype application will use basic WPF views rather than separate forms for the user interface. These views have already been designed and you must add the code to navigate among them.

You also decide to begin by storing the user and grade information in basic structs, and to use a dummy data source in the application to test your log on functionality.

# Module Review and Takeaways

- Review Questions