Chapter 12

How to create and use classes

Objectives

Applied

1. Given the specifications for an application that uses classes with any of the members presented in this chapter, develop the application and its classes.
2. Use class diagrams, the Solution Explorer, and the Class Details window to review, delete, and start the members of the classes in a solution.

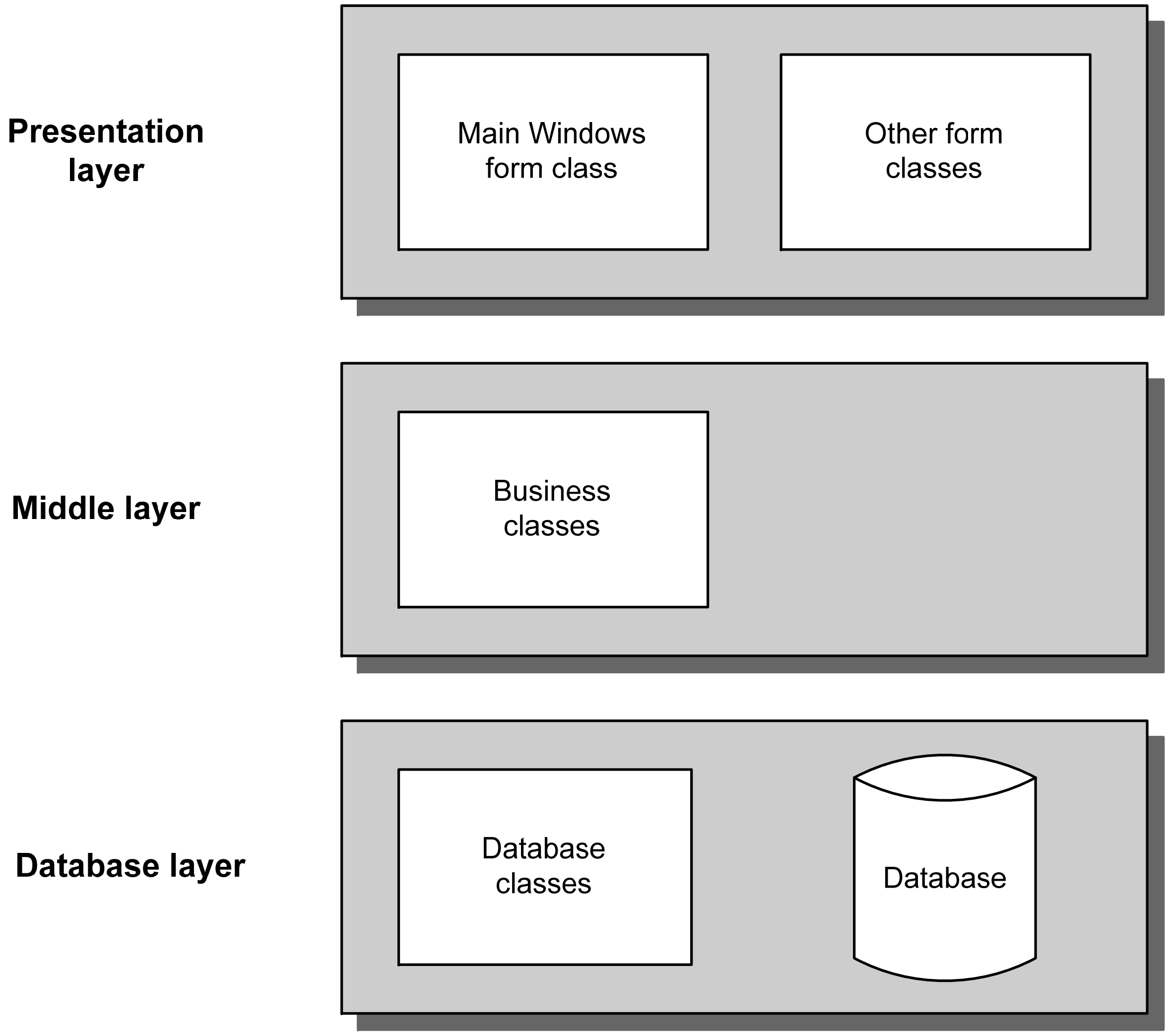
Knowledge

1. List and describe the three layers of a three-layered application.
2. Describe these members of a class: constructor, method, field, and property.
3. Describe the concept of encapsulation.
4. Explain how instantiation works.

Objectives (cont.)

5. Describe the main advantage of using object initializers.
6. Explain how auto-implemented properties work.
7. Describe the concept of overloading a method.
8. Explain what a static member is.
9. Describe the basic procedure for using the Generate From Usage feature to generate code stubs.
10. Describe the difference between a class and a structure.

The architecture of a three-layered application



The members of a Product class

Properties	Description
`Code`	A string that contains a code that uniquely identifies each product.
`Description`	A string that contains a description of the product.
`Price`	A decimal that contains the product's price.
Method	**Description**
`GetDisplayText(``sep`**`)`**	Returns a string that contains the code, description, and price in a displayable format. The *sep* parameter is a string that's used to separate the elements. It's typically set to a tab or new line character.

The members of a Product class (cont.)

Constructors	Description
`()`	Creates a Product object with default values.
`(code, description, price)`	Creates a Product object using the specified code, description, and price values.

Types of class members

Class member	Description
Property	Represents a data value associated with an object instance.
Method	An operation that can be performed by an object.
Constructor	A special type of method that’s executed when an object is instantiated.
Delegate	A special type of object that’s used to wire an event to a method.
Event	A signal that notifies other objects that something noteworthy has occurred.
Field	A variable that’s declared at the class level.
Constant	A constant.

Types of class members (cont.)

Class member	Description
Indexer	A special type of property that allows individual items within the class to be accessed by index values. Used for classes that represent collections of objects.
Operator	A special type of method that's performed for a C# operator such as + or ==.
Class	A class that's defined within the class.

Class and object concepts

- An *object* is a self-contained unit that has *properties*, *methods*, and other *members*. A *class* contains the code that defines the members of an object.
- An object is an *instance* of a class, and the process of creating an object is called *instantiation*.
- *Encapsulation* is one of the fundamental concepts of object-oriented programming. It lets you control the data and operations within a class that are exposed to other classes.
- The data of a class is typically encapsulated within a class using *data hiding*. In addition, the code that performs operations within the class is encapsulated so it can be changed without changing the way other classes use it.
- Although a class can have many different types of members, most of the classes you create will have just properties, methods, and constructors.

The Product class: Fields and constructors

```
using System;

namespace ProductMaint
{
    public class Product
    {
        private string code;
        private string description;
        private decimal price;

        public Product(){}

        public Product(string code, string description,
            decimal price)
        {
            this.Code = code;
            this.Description = description;
            this.Price = price;
        }
```

The Product class: The Code property

```
public string Code
{
    get
    {
        return code;
    }
    set
    {
        code = value;
    }
}
```

The Product class: The Description property

```
public string Description
{
    get
    {
        return description;
    }
    set
    {
        description = value;
    }
}
```

The Product class: The Price property

```
public decimal Price
{
    get
    {
        return price;
    }
    set
    {
        price = value;
    }
}
```

The Product class: The GetDisplayText method

```
        public string GetDisplayText(string sep)
        {
            return code + sep + price.ToString("c") +
                sep + description;
        }
    }
}
```

Two Product objects that have been instantiated from the Product class

product1	**product2**
Code=CS12	Code=VB12
Description=Murach’s C# 2012	Description=Murach’s Visual Basic 2012
Price=54.50	Price=54.50

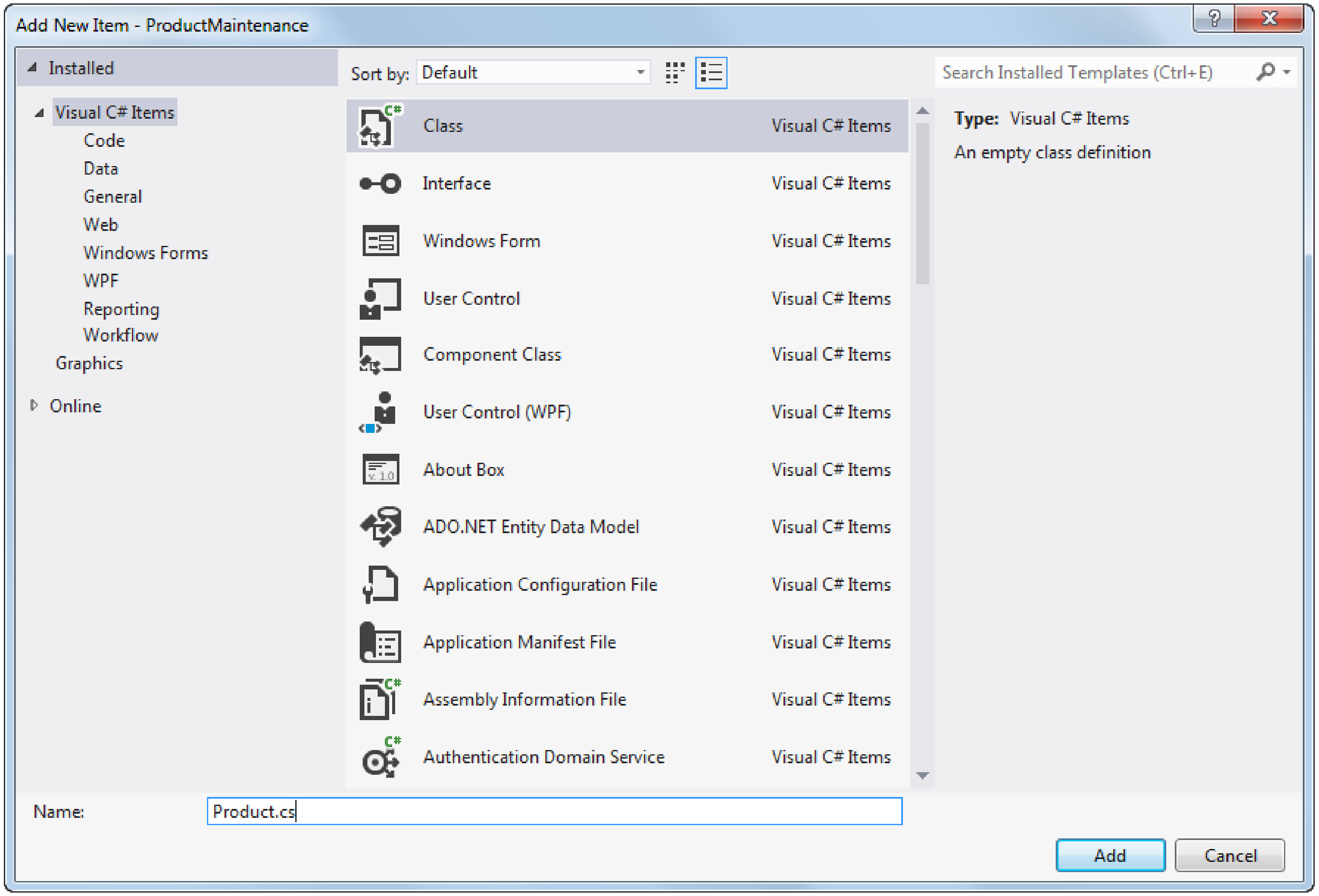
Code that creates these two object instances

```
Product product1, product2;
product1 = new Product("CS12", "Murach's C# 2012", 54.50m);
product2 = new Product("VB12", "Murach's Visual Basic 2012",
                       54.50m);
```

Another way to create the object instances

```
product1 = new Product { Code = "CS12",
    Description = "Murach's C# 2012", Price = 54.50m };
product2 = new Product { Code = "VB12",
    Description = "Murach's Visual Basic 2012", Price = 54.50m };
```

The dialog box for adding a class



Class	Visual C# Items
Interface	Visual C# Items
Windows Form	Visual C# Items
User Control	Visual C# Items
Component Class	Visual C# Items
User Control (WPF)	Visual C# Items
About Box	Visual C# Items
ADO.NET Entity Data Model	Visual C# Items
Application Configuration File	Visual C# Items
Application Manifest File	Visual C# Items
Assembly Information File	Visual C# Items
Authentication Domain Service	Visual C# Items

Type: Visual C# Items

An empty class definition

Name: Product.cs

The starting code for the new class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProductMaintenance
{
    class Product
    {
    }
}
```

Examples of field declarations

```
private int quantity;              // A private field.
public decimal Price;              // A public field.
public readonly int Limit = 90;    // A public read-only field.
```

A Product class that uses public fields

```
public class Product
{
    // Public fields
    public string Code;
    public string Description;
    public decimal Price;

    public Product()
    {
    }

    public Product(string code, string description, decimal price)
    {
        this.Code = code;
        this.Description = description;
        this.Price = price;
    }

    public string GetDisplayText(string sep)
    {
        return Code + sep + Price.ToString("c") + sep + Description;
   }
}
```

The syntax for coding a public property

```
public type PropertyName
{
    [get { get accessor code }]
    [set { set accessor code }]
}
```

A read/write property

```
public string Code
{
    get { return code; }
    set { code = value; }
}
```

A read-only property

```
public decimal DiscountAmount
{
    get
    {
        discountAmount = subtotal * discountPercent;
        return discountAmount;
    }
}
```

An auto-implemented property

```
public string Code { get; set; }
```

A statement that sets a property value

```
product.Code = txtProductCode.Text;
```

A statement that gets a property value

```
string code = product.Code;
```

The syntax for coding a public method

```
public returnType MethodName([parameterList])
{
   statements
}
```

A method that accepts parameters

```
public string GetDisplayText(string sep)
{
    return code + sep + price.ToString("c") + sep
        + description;
}
```

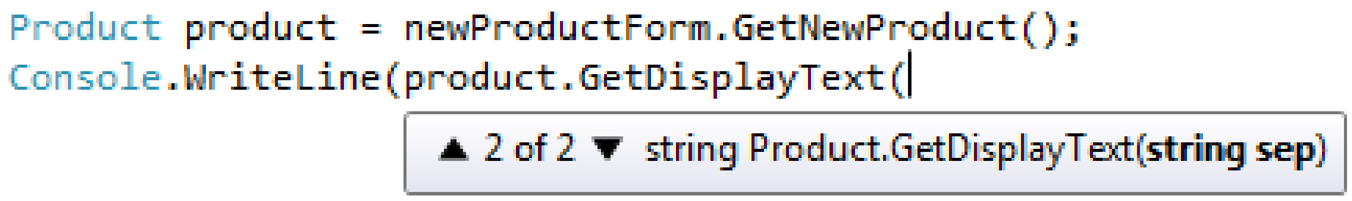
An overloaded version of the method

```
public string GetDisplayText()
{
    return code + ", " + price.ToString("c") + ", "
        + description;
}
```

Statements that call the GetDisplayText method

```
lblProduct.Text = product.GetDisplayText("\t");
lblProduct.Text = product.GetDisplayText();
```

How the IntelliSense feature lists overloaded methods



```
Product product = newProductForm.GetNewProduct();
Console.WriteLine(product.GetDisplayText(
```

```
▲ 2 of 2 ▼  string Product.GetDisplayText(string sep)
```

A constructor with no parameters

```
public Product()
{
}
```

A constructor with three parameters

```
public Product(string code, string description,
    decimal price)
{
    this.Code = code;
    this.Description = description;
    this.Price = price;
}
```

A constructor with one parameter

```
public Product(string code)
{
    Product p = ProductDB.GetProduct(code);
    this.Code = p.Code;
    this.Description = p.Description;
    this.Price = p.Price;
}
```

Statements that call the Product constructors

```
Product product1 = new Product();

Product product2 = new Product("CS12",
    "Murach's C# 2012", 54.50m);

Product product3 = new Product(txtCode.Text);
```

Default values for instance variables

Data type	Default value
All numeric types	zero (0)
Boolean	false
Char	binary 0 (null)
Object	null (no value)
Date	12:00 a.m. on January 1, 0001

A class that contains static members

```
public static class Validator
{
    private static string title = "Entry Error";

    public static string Title
    {
        get
        {
            return title;
        }
        set
        {
            title = value;
        }
    }
```

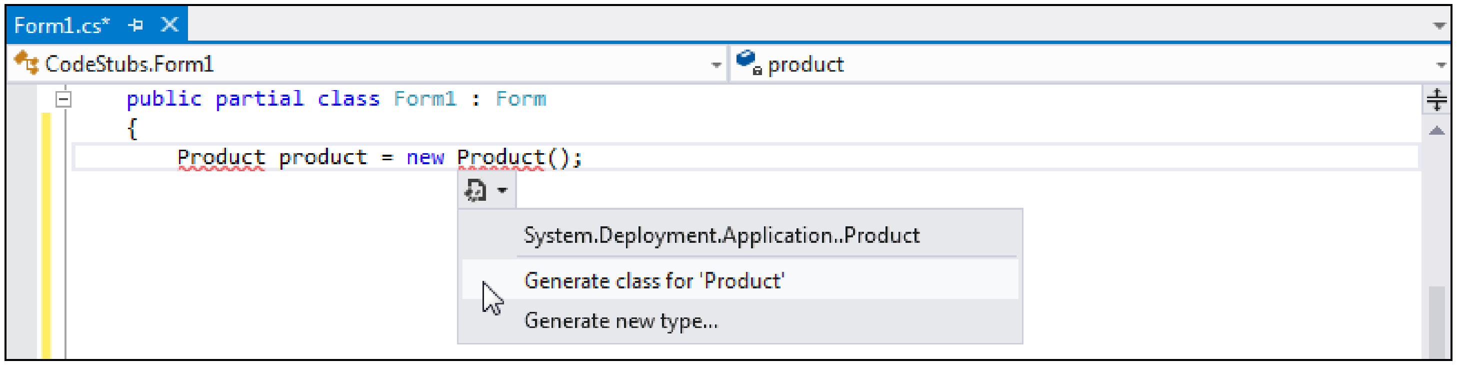
A class that contains static members (cont.)

```
public static bool IsPresent(TextBox textBox)
{
    if (textBox.Text == "")
    {
        MessageBox.Show(textBox.Tag
            + " is a required field.", Title);
        textBox.Focus();
        return false;
    }
    return true;
}
```

Code that uses static members

```
if (  Validator.IsPresent(txtCode) &&
      Validator.IsPresent(txtDescription) &&
      Validator.IsPresent(txtPrice) )
    isValidData = true;
else
    isValidData = false;
```

Right-clicking on an undefined name to generate a class



Form1.cs*

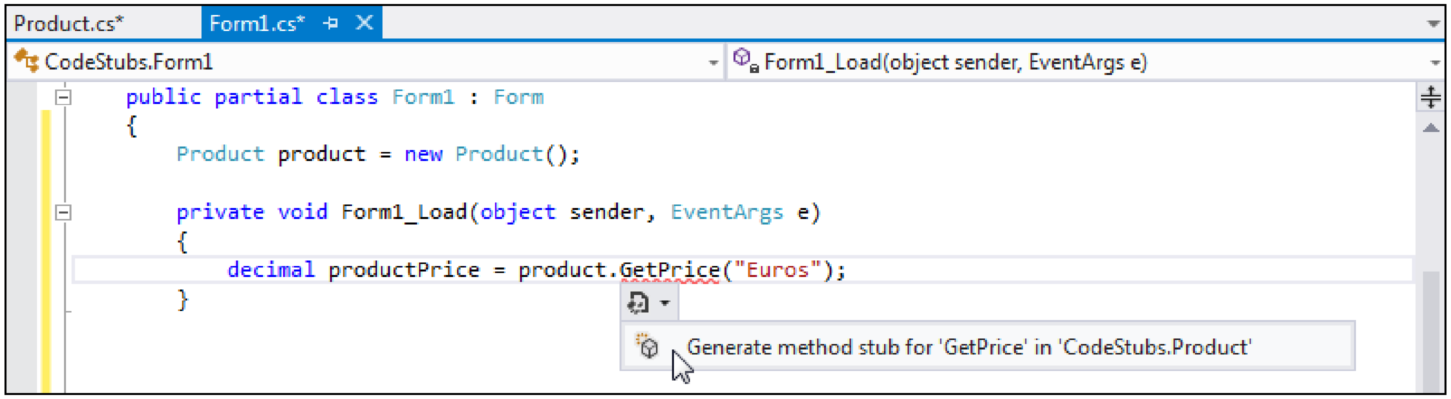
CodeStubs.Form1

product

```
public partial class Form1 : Form
{
    Product product = new Product();
```

System.Deployment.Application..Product
Generate class for 'Product'
Generate new type...

Using a smart tag menu to generate a method stub



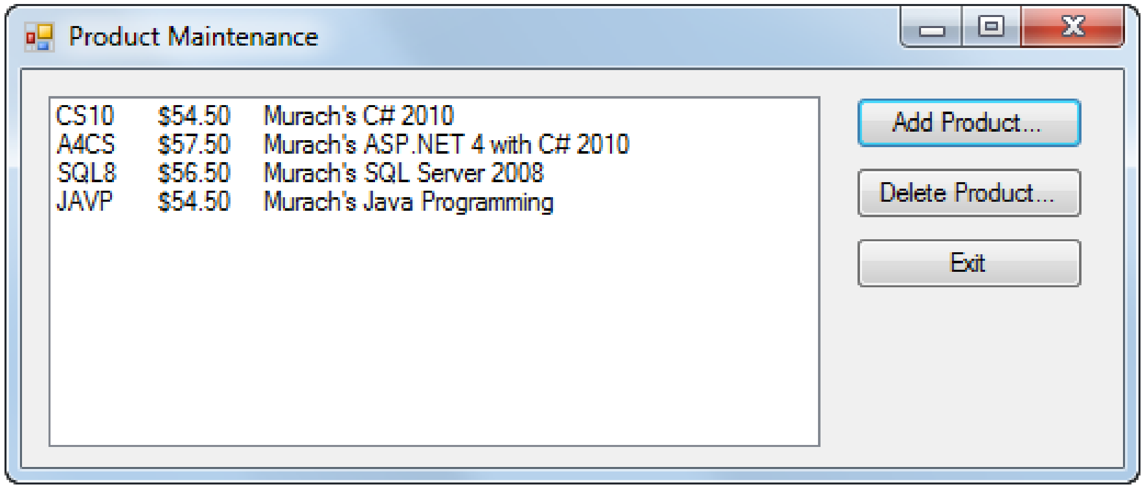
```
public partial class Form1 : Form
{
    Product product = new Product();

    private void Form1_Load(object sender, EventArgs e)
    {
        decimal productPrice = product.GetPrice("Euros");
    }
```

Generate method stub for 'GetPrice' in 'CodeStubs.Product'

The generated code

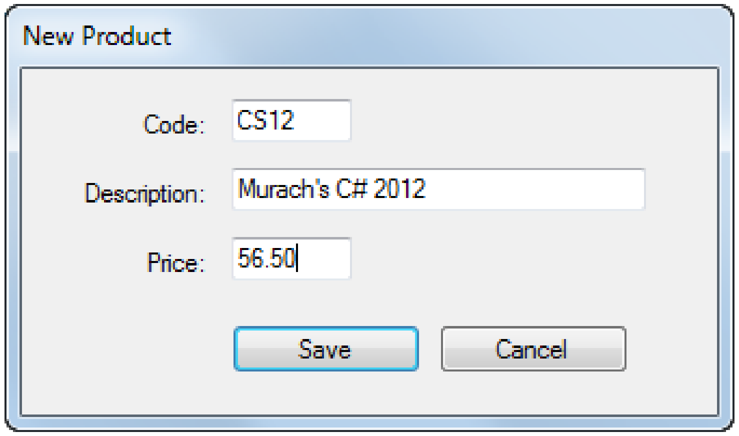
```
class Product
{
    internal decimal GetPrice(string p)
    {
        throw new NotImplementedException();
    }
}
```

The Product Maintenance form



CS10	$54.50	Murach's C# 2010
A4CS	$57.50	Murach's ASP.NET 4 with C# 2010
SQL8	$56.50	Murach's SQL Server 2008
JAVP	$54.50	Murach's Java Programming

The New Product form



Code: CS12

Description: Murach's C# 2012

Price: 56.50

The Tag property settings for the text boxes on the New Product form

Control	Tag property setting
txtCode	Code
txtDescription	Description
txtPrice	Price

The Product class

Property	Description
`Code`	A string that contains a code that uniquely identifies the product.
`Description`	A string that contains a description of the product.
`Price`	A decimal that contains the product’s price.
Method	**Description**
`GetDisplayText(``sep`**`)`**	Returns a string that contains the code, description, and price separated by the sep string.

The Product class (cont.)

Constructor	Description
`()`	Creates a Product object with default values.
`(code, description, price)`	Creates a Product object using the specified values.

The ProductDB class

Method	Description
`GetProducts()`	A static method that returns a List<> of Product objects from the Products file.
`SaveProducts(list)`	A static method that writes the products in the specified List<> of Product objects to the Products file.

Note

- You don’t need to know how the ProductDB class works. You just need to know about its methods so you can use them in your code.

The Validator class

Property	Description
`Title`	A static string that contains the text that's displayed in the title bar of a dialog box that's displayed for an error message.
Static method	**Returns a Boolean value that indicates whether…**
`IsPresent(textBox)`	Data was entered into the text box.
`IsInt32(textBox)`	An integer was entered into the text box.
`IsDecimal(textBox)`	A decimal was entered into the text box.
`IsWithinRange(textBox, min, max)`	The value entered into the text box is within the specified range.

The code for the Product Maintenance form

```
public partial class frmProductMain : Form
{
    public frmProductMain()
    {
        InitializeComponent();
    }

    private List<Product> products = null;

    private void frmProductMain_Load(object sender,
        System.EventArgs e)
    {
        products = ProductDB.GetProducts();
        FillProductListBox();
    }

    private void FillProductListBox()
    {
        lstProducts.Items.Clear();
        foreach (Product p in products)
        {
            lstProducts.Items.Add(p.GetDisplayText("\t"));
        }
    }
```

The code for the Product Maintenance form (cont.)

```
private void btnAdd_Click(object sender, System.EventArgs e)
{
    frmNewProduct newProductForm = new frmNewProduct();
    Product product = newProductForm.GetNewProduct();
    if (product != null)
    {
        products.Add(product);
        ProductDB.SaveProducts(products);
        FillProductListBox();
    }
}

private void btnDelete_Click(object sender, System.EventArgs e)
{
    int i = lstProducts.SelectedIndex;
    if (i != -1)
    {
        Product product = products[i];
        string message = "Are you sure you want to delete "
            + product.Description + "?";
        DialogResult button =
            MessageBox.Show(message, "Confirm Delete",
            MessageBoxButtons.YesNo);
```

The code for the Product Maintenance form (cont.)

```
            if (button == DialogResult.Yes)
            {
                products.Remove(product);
                ProductDB.SaveProducts(products);
                FillProductListBox();
            }
        }
    }

    private void btnExit_Click(object sender, EventArgs e)
    {
        this.Close();
    }
}
```

The code for the New Product form

```
public partial class frmNewProduct : Form
{
    public frmNewProduct()
    {
        InitializeComponent();
    }

    private Product product = null;

    public Product GetNewProduct()
    {
        this.ShowDialog();
        return product;
    }

    private void btnSave_Click(object sender, System.EventArgs e)
    {
        if (IsValidData())
        {
            product = new Product(txtCode.Text, txtDescription.Text,
                Convert.ToDecimal(txtPrice.Text));
            this.Close();
        }
    }
```

The code for the New Product form (cont.)

```
    private bool IsValidData()
    {
        return Validator.IsPresent(txtCode)        &&
               Validator.IsPresent(txtDescription) &&
               Validator.IsPresent(txtPrice)       &&
               Validator.IsDecimal(txtPrice);
    }

    private void btnCancel_Click(object sender, System.EventArgs e)
    {
        this.Close();
    }
}
```

The code for the Validator class

```
public static class Validator
{
    private static string title = "Entry Error";

    public static string Title
    {
        get
        {
            return title;
        }
        set
        {
            title = value;
        }
    }
```

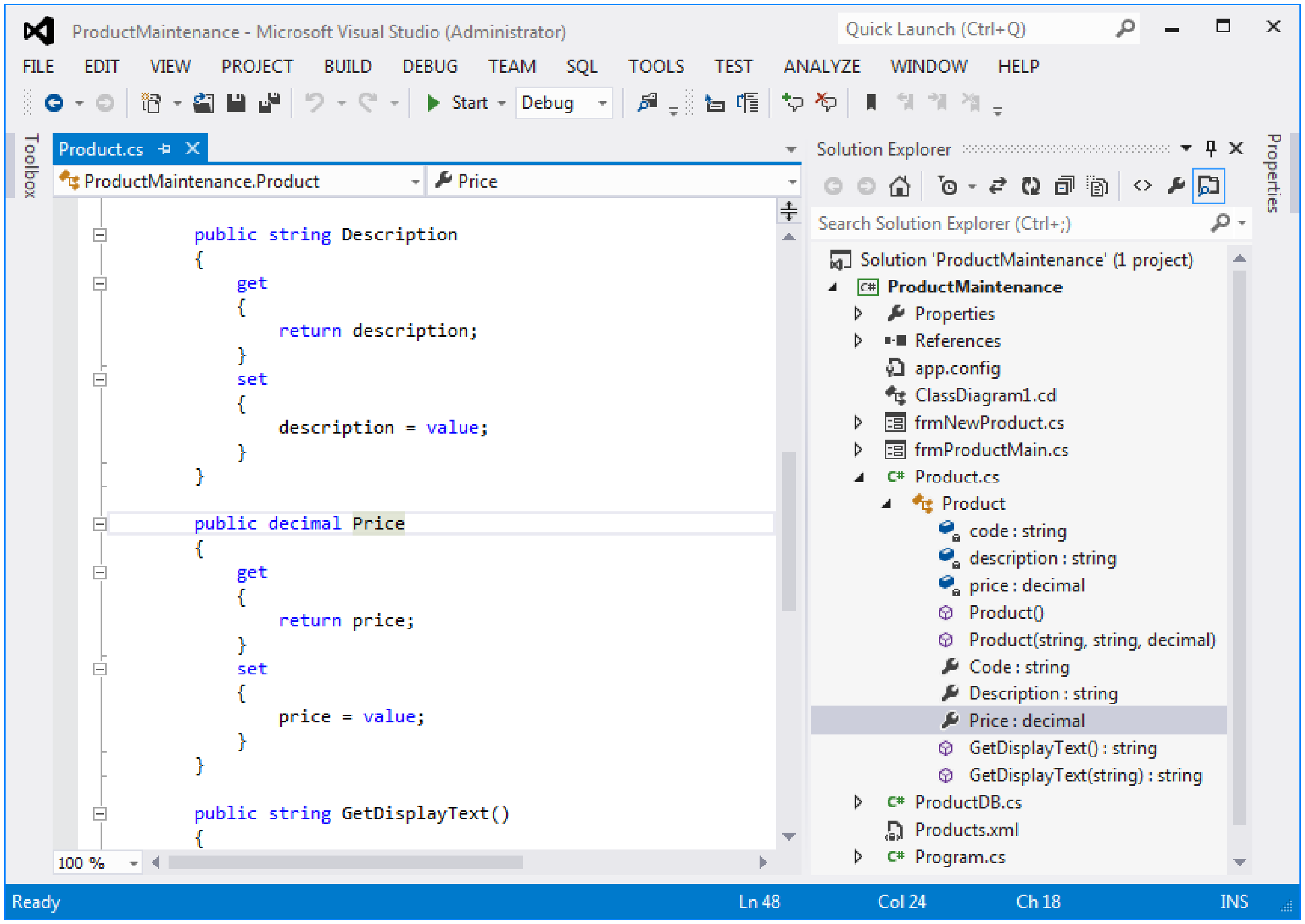
The code for the Validator class (cont.)

```
public static bool IsPresent(TextBox textBox)
{
    if (textBox.Text == "")
    {
        MessageBox.Show(textBox.Tag + " is a required field.",
            Title);
        textBox.Focus();
        return false;
    }
    return true;
}
```

The code for the Validator class (cont.)

```
    public static bool IsDecimal(TextBox textBox)
    {
        decimal number = 0m;
        if (Decimal.TryParse(textBox.Text, out number))
        {
            return true;
        }
        else
        {
            MessageBox.Show(textBox.Tag +
                " must be a decimal value.", Title);
            textBox.Focus();
            return false;
        }
    }
}
```

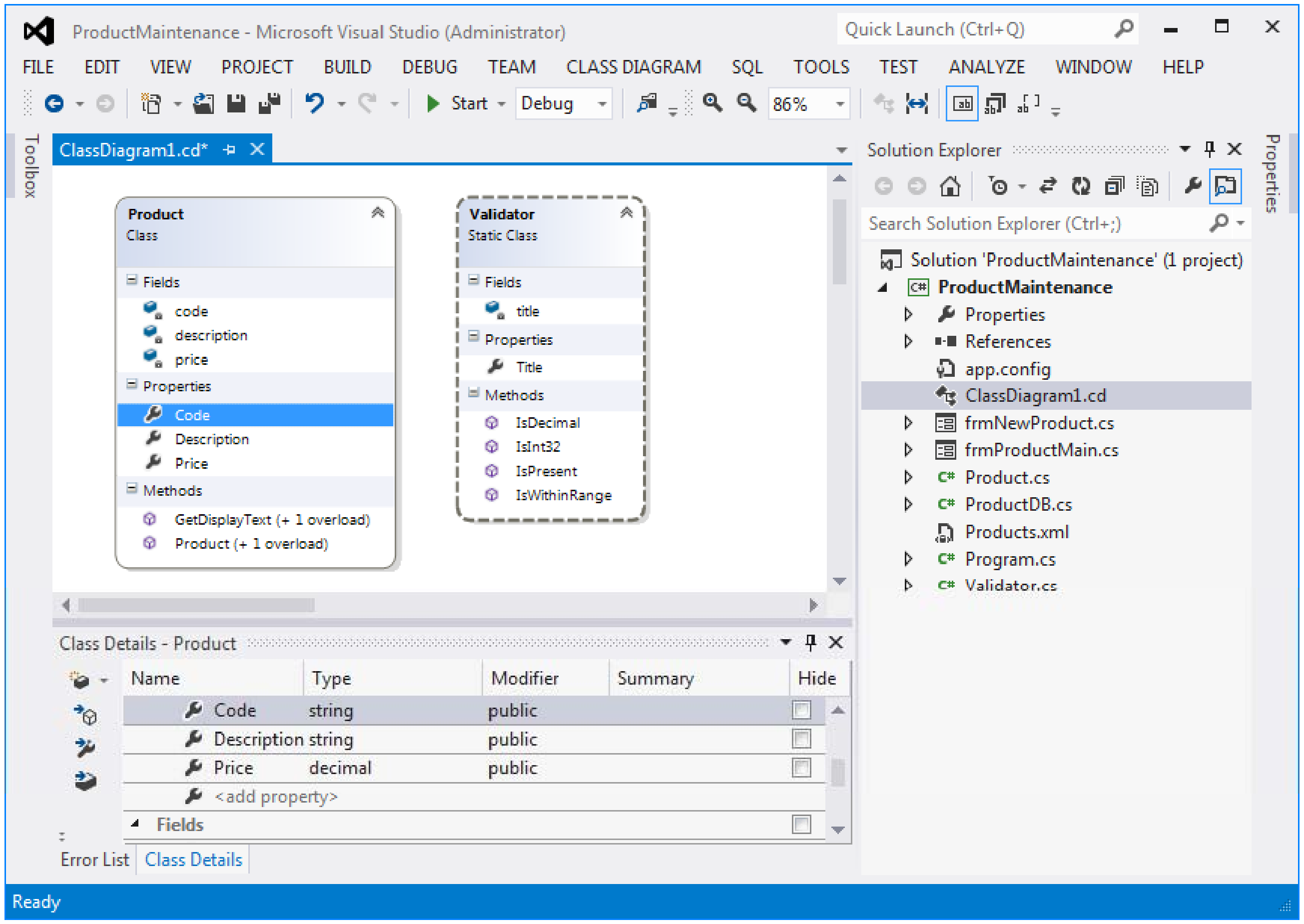
The Solution Explorer with the members of the Product class displayed


ProductMaintenance - Microsoft Visual Studio (Administrator)
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ANALYZE WINDOW HELP
Start Debug
Toolbox
Product.cs
ProductMaintenance.Product
Price
100 %
Solution Explorer
Search Solution Explorer (Ctrl+;)
Properties
Ready Ln 48 Col 24 Ch 18 INS

```
        public string Description
        {
            get
            {
                return description;
            }
            set
            {
                description = value;
            }
        }

        public decimal Price
        {
            get
            {
                return price;
            }
            set
            {
                price = value;
            }
        }

        public string GetDisplayText()
        {
```

```
Solution 'ProductMaintenance' (1 project)
  ProductMaintenance
    Properties
    References
    app.config
    ClassDiagram1.cd
    frmNewProduct.cs
    frmProductMain.cs
    Product.cs
      Product
        code : string
        description : string
        price : decimal
        Product()
        Product(string, string, decimal)
        Code : string
        Description : string
        Price : decimal
        GetDisplayText() : string
        GetDisplayText(string) : string
    ProductDB.cs
    Products.xml
    Program.cs
```

A class diagram that shows two of the classes*



Name	Type	Modifier	Summary	Hide
Code	string	public		☐
Description	string	public		☐
Price	decimal	public		☐
<add property>				
Fields				☐

*Express Edition doesn’t have class diagrams or the Class Details window

The syntax for creating a structure

```
public struct StructureName
{
    structure members...
}
```

A Product structure

```
public struct Product
{
    private string code;
    private string description;
    private decimal price;

    public Product(string code, string description, decimal price)
    {
        this.code = code;
        this.description = description;
        this.price = price;
    }
    public string Code
    {
        get
        {
            return code;
        }
        set
        {
            code = value;
        }
    }
```

A Product structure (cont.)

```
    .
    .
    public string GetDisplayText(string sep)
    {
        return Code + sep + Price.ToString("c") + sep + Description;
    }
}
```

Code that declares a variable as a structure type and assigns values to it

```
// Create an instance of the Product structure
Product p;

// Assign values to each instance variable
p.Code = "CS12";
p.Description = "Murach's C# 2012";
p.Price = 54.50m;

// Call a method
string msg = p.GetDisplayText("\n");
```

Code that uses the default constructor to initialize the instance variables

```
Product p = new Product();
```

Code that uses the constructor that accepts parameters to initialize the instance variables

```
Product p = new Product("CS12", "Murach's C# 2012",
    54.50m);
```