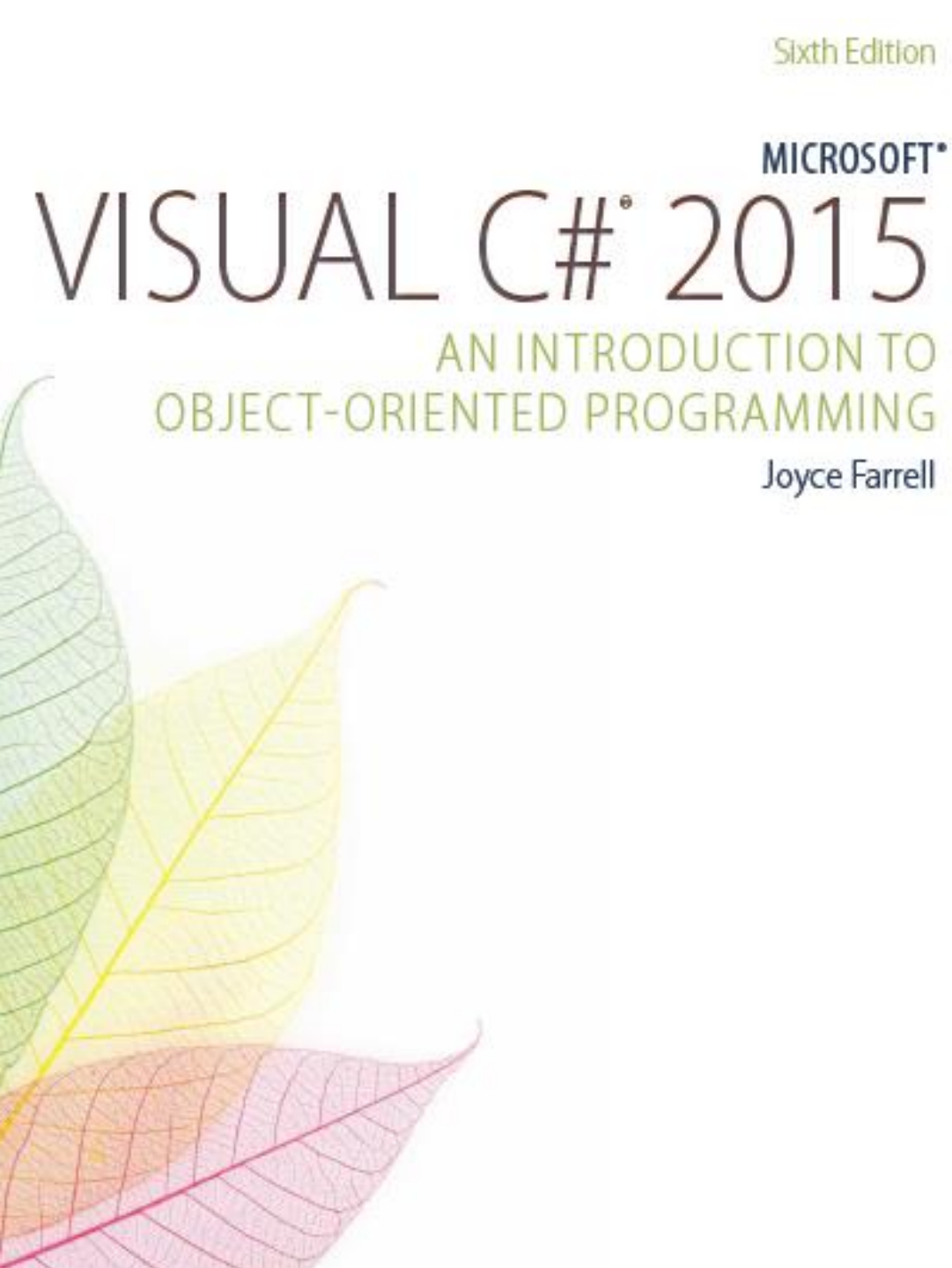

Chapter 9:

Using Classes and Objects

Objectives (cont'd.)

- Write and use constructors
- Use object initializers
- Overload operators
- Declare object arrays and use methods with them
- Write destructors
- Understand GUI application objects

Understanding Constructors

- **Constructor**
  - A method that instantiates an object
- **Default constructor**
  - An automatically supplied constructor without parameters
- **Default value of the object**
  - The value of an object initialized with a default constructor

Passing Parameters to Constructors

- **Parameterless constructor**
  - A constructor that takes no arguments
  - You can also create a constructor that receives argument(s)

```
class Employee
{
   private int idNumber;
   private string name;
   public Employee()
   {
      PayRate = 9.99;
   }
   public double PayRate {get; set;}
   //  Other class members can go here
}
```

Figure 9-22 Employee class with a parameterless constructor

```
public Employee(double rate)
{
   PayRate = rate;
}
```

Figure 9-23 Employee constructor with parameter

Overloading Constructors

- C# automatically provides a default constructor until you provide your own constructor
- Constructors can be overloaded
  - You can write as many constructors as you want, as long as their argument lists do not cause ambiguity

Overloading Constructors (Cont'd.)

```
class Employee
{
   public int IdNumber {get; set;}

   public double Salary {get; set;}
   public Employee()
   {
      IdNumber = 999;
      Salary = 0;
   }
   public Employee(int empId)
   {
      IdNumber = empId;
      Salary = 0;
   }
   public Employee(int empId, double sal)
   {
      IdNumber = empId;
      Salary = sal;
   }
   public Employee(char code)
   {
      IdNumber = 111;
      Salary = 100000;
   }
}
```

This parameterless constructor is the class's default constructor.

Figure 9-24 Employee class with four constructors

Overloading Constructors (cont'd.)

```
using static System.Console;
class CreateSomeEmployees
{
   static void Main()
   {
      Employee aWorker = new Employee();
      Employee anotherWorker = new Employee(234);
      Employee theBoss = new Employee('A');
      WriteLine("{0, 4}{1, 14}", aWorker.IdNumber,
         aWorker.Salary.ToString("C"));
      WriteLine("{0, 4}{1, 14}",
         anotherWorker.IdNumber,
         anotherWorker.Salary.ToString("C"));
      WriteLine("{0, 4}{1, 14}", theBoss.IdNumber,
         theBoss.Salary.ToString("C"));
   }
}
```

Figure 9-25 The CreateSomeEmployees program

```
C:\C#\Chapter.09>CreateSomeEmployees
 999          $0.00
 234          $0.00
 111    $100,000.00

C:\C#\Chapter.09>
```

Figure 9-26 Output of the CreateSomeEmployees program

Using Constructor Initializers

- **Constructor initializer**
  - A clause that indicates another instance of a class constructor should be executed before any statements in the current constructor body

Using Constructor Initializers (cont'd.)

```
class Employee
{
   public int IdNumber {get; set;}
   public double Salary {get; set;}
   public Employee() : this(999, 0)
   {
   }
   public Employee(int empId) : this(empId, 0)
   {
   }
   public Employee(int empId, double sal)
   {
      IdNumber = empId;
      Salary = sal;
   }
   public Employee(char code) : this(111, 100000)
   {
   }
}
```

Figure 9-27 Employee class with constructor initializers

Using the `readonly` Modifier in a Constructor

- `readonly` modifiers similar to `const`
  - They are assigned a value that cannot be changed
  - Their value can be assigned at run time rather than at compile time
  - They can get their value from user input or the operating system

Using the `readonly` Modifier in a Constructor (Cont'd.)

```
class EmployeesReadOnlyDemo
{
   static void Main()
   {
       Employee myAssistant = new Employee(1234);
       Employee myDriver = new Employee(2345);
       myAssistant.IdNumber = 3456;
   }
}
class Employee
{
   private readonly int idNumber;
   public Employee(int id)
   {
       idNumber = id;
   }
   public int IdNumber
   {
      get
      {
         return idNumber;
      }
      set
      {
         idNumber = value;
      }
   }
}
```

Don't Do It

The `idNumber` field cannot be assigned a value after construction. This statement generates an error.

Figure 9-28 Employee class with a `readonly` field

Using Object Initializers

- **Object initializer**
  - Allows you to assign values to any accessible members or properties of a class at the time of instantiation without calling a constructor with parameters
- For you to use object initializers, a class must have a default constructor:

```
Employee aWorker = new Employee{IdNumber = 101};
Employee aWorker = new Employee(){IdNumber = 101};
```

Using Object Initializers (cont'd.)

```
C:\C#\Chapter.09>DemoObjectInitializer
Employee #0 created. Salary is 99.99.
Employee #101 exists. Salary is 99.99.

C:\C#\Chapter.09>
```

Figure 9-31 Output of the DemoObjectInitializer program

```
using static System.Console;
class DemoObjectInitializer
{
   static void Main()
   {
      Employee aWorker = new Employee {IdNumber = 101};
      WriteLine("Employee #{0} exists. Salary is {1}.",
         aWorker.IdNumber, aWorker.Salary);
   }
}
class Employee
{
   public int IdNumber {get; set;}
   public double Salary {get; set;}
   public Employee()
   {
      Salary = 99.99;
      WriteLine("Employee #{0} created. Salary is {1}.",
         IdNumber, Salary);
   }
}
```

Figure 9-30 The DemoObjectInitializer program

Using Object Initializers (cont'd.)

- Using object initializers allows you to:
  - Create multiple objects with different initial assignments without having to provide multiple constructors to cover every possible situation
  - Create objects with different starting values for different properties of the same data type
  - The Box class (right) contains multiple properties of the same type -- The constructor sets the Height, Width, and Depth properties to 1

```
class Box
{
   public int Height {get; set;}
   public int Width {get; set;}
   public int Depth {get; set;}
   public Box()
   {
      Height = 1;
      Width = 1;
      Depth = 1;
   }
}
```

Figure 9-32 The Box class

Using Object Initializers (cont'd.)

```
using static System.Console;
class DemoObjectInitializer2
{
   static void Main()
   {
      Box box1 = new Box {Height = 3};
      Box box2 = new Box {Width = 15};
      Box box3 = new Box {Depth = 268};
      DisplayDimensions(1, box1);
      DisplayDimensions(2, box2);
      DisplayDimensions(3, box3);
   }
   static void DisplayDimensions(int num, Box box)
   {
      WriteLine("Box {0}: Height: {1} Width: {2} Depth: {3}",
         num, box.Height, box.Width, box.Depth);
   }
}
```

Figure 9-33 The DemoObjectInitializer2 program

```
C:\C#\Chapter.09>DemoObjectInitializer2
Box 1: Height: 3 Width: 1 Depth: 1
Box 2: Height: 1 Width: 15 Depth: 1
Box 3: Height: 1 Width: 1 Depth: 268

C:\C#\Chapter.09>
```

Figure 9-34 Output of the DemoObjectInitializer2 program

Overloading Operators

- Overloading operators
  - Enables you to use arithmetic symbols with your own objects
- Overloadable unary operators:

```
+ - ! ~ ++ -- true false
```

- Overloadable binary operators:

```
+ - * / % & | ^ == != > < >= <=
```

- You cannot overload the following operators:

```
= && || ?? ?: checked unchecked new typeof
as is
```

- You cannot overload an operator for a built-in data type

Overloading Operators (cont'd.)

- When a binary operator is overloaded and has a corresponding assignment operator, it is also overloaded
- Some operators must be overloaded in pairs:

  == with !=, and < with >
- Syntax to overload unary operators:

  type `operator` *overloadable-operator* (*type identifier*)
- Syntax to overload binary operators:

  type `operator` *overloadable-operator* (*type identifier, type operand*)

Overloading Operators (cont'd.)

```
class Book
{
   public Book(string title, int pages, double price)
   {
      Title = title;
      NumPages = pages;
      Price = price;
   }
   public static Book operator+(Book first, Book second)
   {
      const double EXTRA = 10.00;
      string newTitle = first.Title + " and " + second.Title;
      int newPages = first.NumPages + second.NumPages;
      double newPrice;
      if(first.Price > second.Price)
         newPrice = first.Price + EXTRA;
      else
         newPrice = second.Price + EXTRA;
      return(new Book(newTitle, newPages, newPrice));
   }
   public string Title {get; set;}
   public int NumPages {get; set;}
   public double Price {get; set;}
}
```

Figure 9-35 Book class with overloaded + operator

Overloading Operators (cont'd.)

```
C:\C#\Chapter.09>AddBooks
The new book is "Silas Marner and Moby Dick"
It has 600 pages and costs $26.00

C:\C#\Chapter.09>
```

Figure 9-37 Output of the AddBooks program

```
using static System.Console;
class AddBooks
{
   static void Main()
   {
      Book book1 = new Book("Silas Marner", 350, 15.95);
      Book book2 = new Book("Moby Dick", 250, 16.00);
      Book book3;
      book3 = book1 + book2;
      WriteLine("The new book is \"{0}\"", book3.Title);
      WriteLine("It has {0} pages and costs {1}",
         book3.NumPages, book3.Price.ToString("C"));
   }
}
```

Figure 9-36 The AddBooks program

Overloading Operators (cont'd.)

- Overloaded unary operators take a single argument

```
public static Book operator-(Book aBook)
{
    aBook.Price = -aBookPrice;
    return aBook;
}
```

Figure 9-38 An operator-() method for a Book

Declaring an Array of Objects

- You can declare arrays that hold elements of any type, including objects
- Example:

```
Employee[] empArray = new Employee[7];

for(int x = 0; x < empArray.Length; ++x)
   empArray[x] = new Employee();
```

Using the `Sort()` and `BinarySearch()` Methods with Arrays of Objects

- The **`Sort()` method** accepts an array parameter and arranges its elements in descending order
- The **`BinarySearch()` method** accepts a sorted array and a value that it attempts to match in the array

Using the `Sort()` and `BinarySearch()` Methods with Arrays of Objects (cont'd.)

- **Interface**
  - A collection of empty abstract methods that can be used by any class as long as the class provides a definition to override the interface's do-nothing, or abstract, method definitions
  - An **abstract method** has no method statements
- When a method **overrides** another, it takes precedence, hiding the original version

Using the `Sort()` and `BinarySearch()` Methods with Arrays of Objects (cont'd.)

- **`IComparable` interface**
  - Contains the definition for the `CompareTo()` method
  - Compares one object to another and returns an integer

```
interface IComparable
{
    int CompareTo(Object o);
}
```

Figure 9-39 The `IComparable` interface

Using the `Sort()` and `BinarySearch()` Methods with Arrays of Objects (cont'd.)

Return Value	Meaning
`Negative`	This instance is less than the compared object.
`Zero`	This instance is equal to the compared object.
`Positive`	This instance is greater than the compared object.

Table 9-2 Return values of `IComparable.CompareTo()` method

Using the `Sort()` and `BinarySearch()` Methods with Arrays of Objects (cont'd.)

```
class Employee : IComparable
{
    public int IdNumber {get; set;}
    public double Salary {get; set;}
    int IComparable.CompareTo(Object o)
    {
        int returnVal;
        Employee temp = (Employee)o;
        if(this.IdNumber > temp.IdNumber)
            returnVal = 1;
        else
            if(this.IdNumber < temp.IdNumber)
                returnVal = -1;
            else
                returnVal = 0;
        return returnVal;
    }
}
```

Figure 9-40 Employee class using `IComparable` interface

Using the `Sort()` and `BinarySearch()` Methods with Arrays of Objects (cont'd.)

```
using System;
using static System.Console;
class ComparableEmployeeArray
{
   static void Main()
   {
      Employee[] empArray = new Employee[5];
      int x;
      for(x = 0; x < empArray.Length; ++x)
         empArray[x] = new Employee();
      empArray[0].IdNumber = 333;
      empArray[1].IdNumber = 444;
      empArray[2].IdNumber = 555;
      empArray[3].IdNumber = 111;
      empArray[4].IdNumber = 222;
      Employee seekEmp = new Employee();
      seekEmp.IdNumber = 222;
      Array.Sort(empArray);
      WriteLine("Sorted employees:");
      for(x = 0; x < empArray.Length; ++x)
         WriteLine("Employee #{0}: {1} {2}", x,
            empArray[x].IdNumber, empArray[x].Salary.ToString("C"));
      x = Array.BinarySearch(empArray, seekEmp);
      WriteLine("Employee #{0} was found at position {1}",
         seekEmp.IdNumber, x);
   }
}
```

Figure 9-41 The ComparableEmployeeArray program

Using the `Sort()` and `BinarySearch()` Methods with Arrays of Objects (cont'd.)

```
C:\C#\Chapter.09>ComparableEmployeeArray
Sorted employees:
Employee #0: 111 $0.00
Employee #1: 222 $0.00
Employee #2: 333 $0.00
Employee #3: 444 $0.00
Employee #4: 555 $0.00
Employee #222 was found at position 1

C:\C#\Chapter.09>
```

Figure 9-42 Output of the `ComparableEmployeeArray` program

Understanding Destructors

- **Destructor**
  - Contains the actions you require when an instance of a class is destroyed
- Most often, an instance of a class is destroyed when it goes out of scope
- Explicitly declare a destructor
  - The identifier consists of a tilde (~) followed by the class name

Understanding Destructors (cont'd.)

```
class Employee
{
   public int idNumber {get; set;}
   public Employee(int empID)
   {
      IdNumber = empID;
      WriteLine("Employee object {0} created", IdNumber);
   }
   ~Employee()
   {
      WriteLine("Employee object {0} destroyed!", IdNumber);
   }
}
```

Figure 9-44 Employee class with destructor

```
using static System.Console;
class DemoEmployeeDestructor
{
   static void Main()
   {
      Employee aWorker = new Employee(101);
      Employee anotherWorker = new Employee(202);
   }
}
```

Figure 9-45 The DemoEmployeeDestructor program

```
C:\C#\Chapter.09>DemoEmployeeDestructor
Employee object 101 created
Employee object 202 created
Employee object 202 destroyed!
Employee object 101 destroyed!

C:\C#\Chapter.09>
```

Figure 9-46 Output of DemoEmployeeDestructor program

Understanding GUI Application Objects

- Objects you have been using in GUI applications are just like other objects
  - They encapsulate properties and methods

Understanding GUI Application Objects (cont'd.)

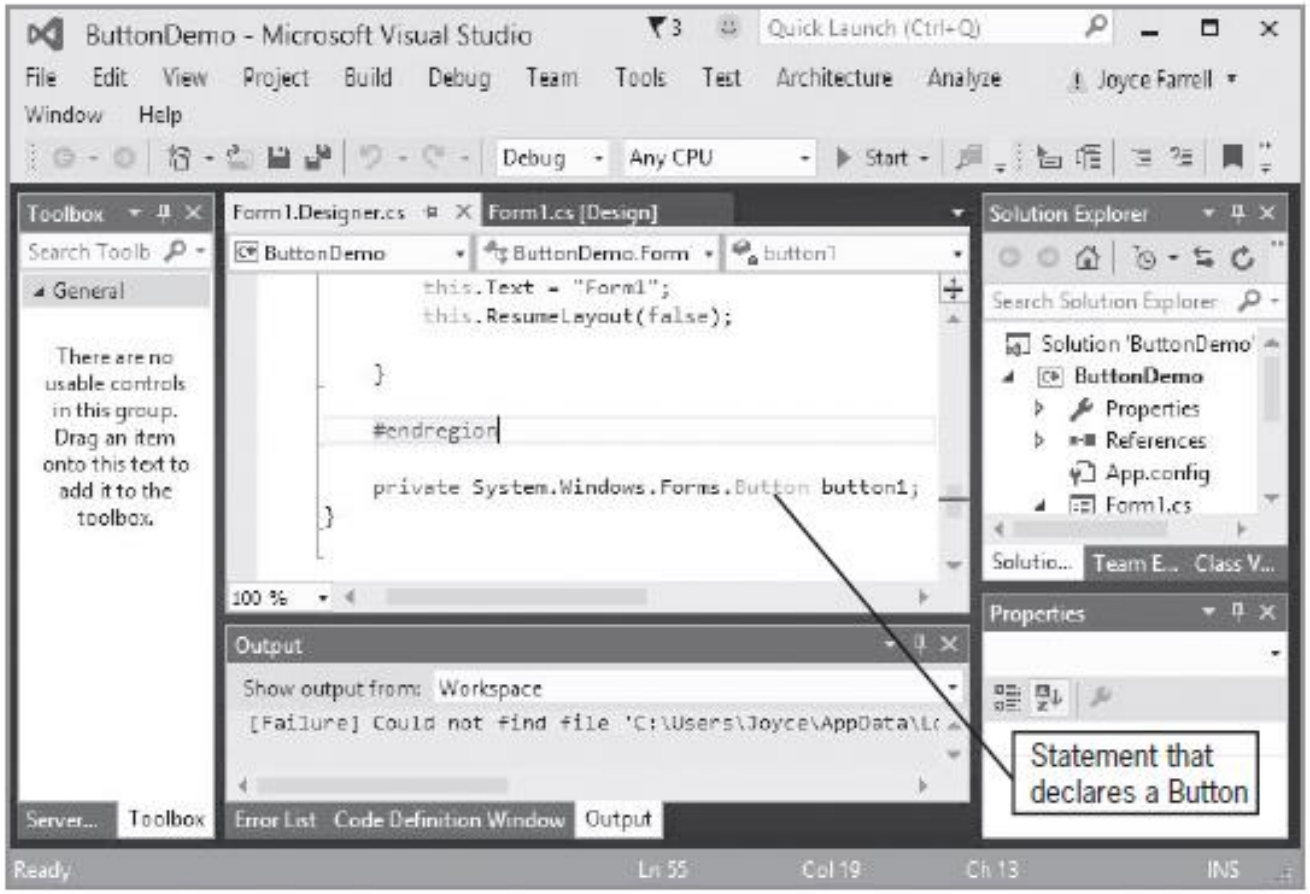

Figure 9-47 A Button on a Form

Understanding GUI Application Objects (cont'd.)

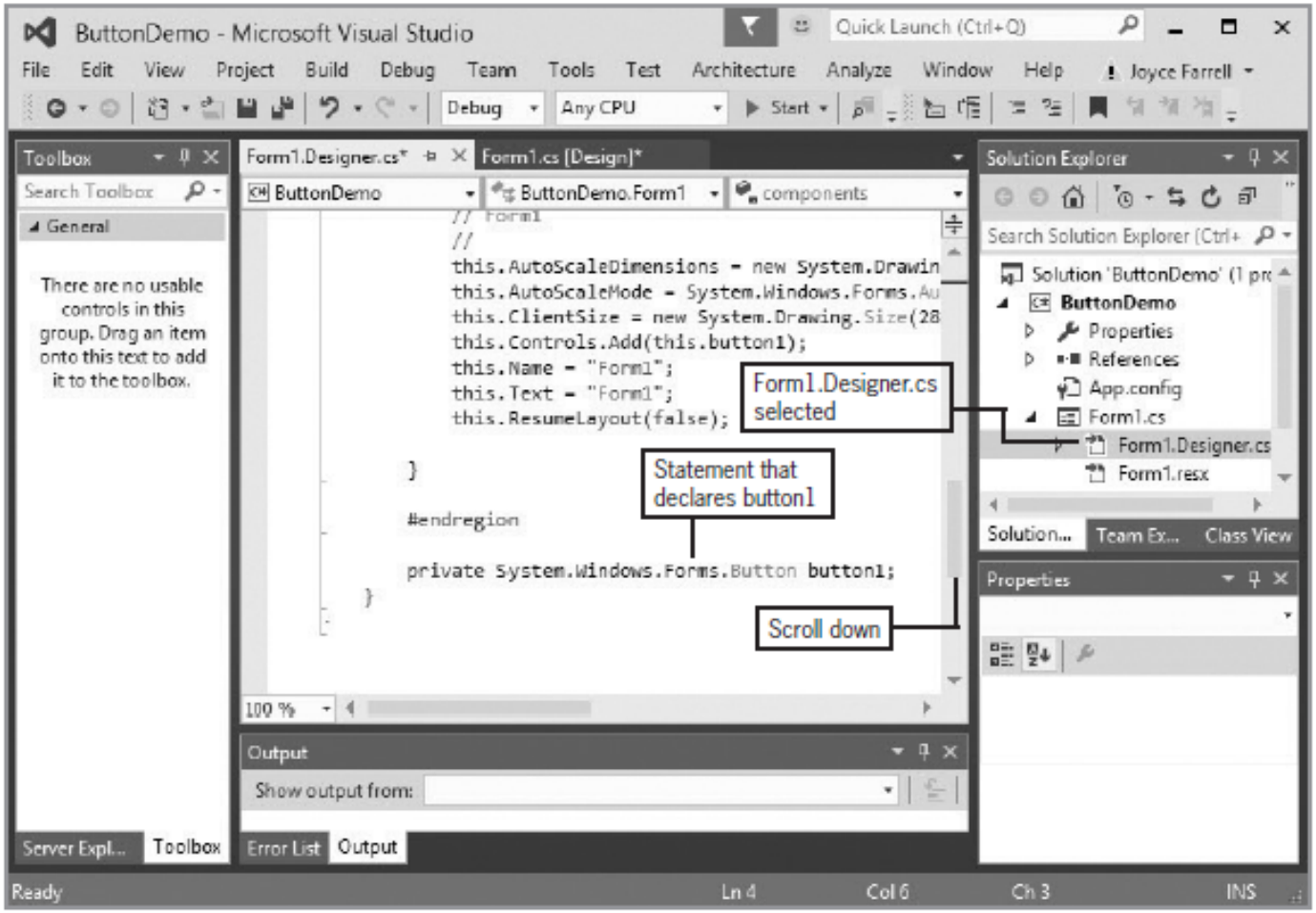

Figure 9-48 The automatically generated statement that declares `button1`

Understanding GUI Application Objects (cont'd.)

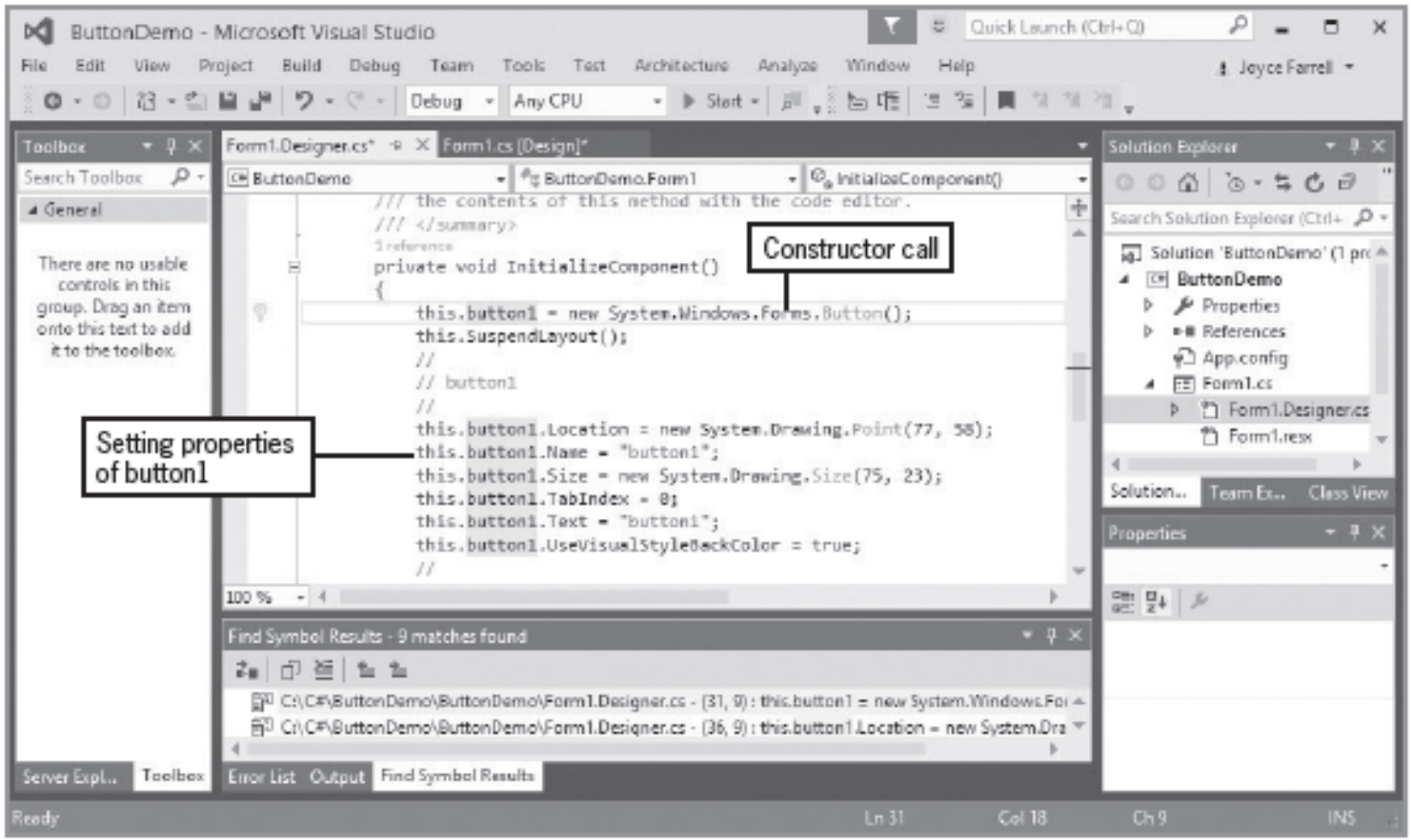

Figure 9-49 Some automatically generated `button1` references in the IDE

Summary

- You can create classes that are only programs with a `Main()` method, and classes from which you instantiate objects
- When creating a class:
  - You must assign a name to it, and determine what data and methods will be part of the class
  - You usually declare instance variables to be `private` and instance methods to be `public`
- When creating an object, supply a type and an identifier, and allocate computer memory for that object

Summary (cont'd.)

- A property is a member of a class that provides access to a field of a class
- A property is a member of a class that provides access to a field. Properties have set accessors for setting an object's fields and get accessors for retrieving the stored values
- In most classes, fields are private and methods are public
- The `this` reference is passed to every instance method and property accessor in a class

Summary (cont'd.)

- A constructor is a method that instantiates an object. With no constructor, each class is automatically supplied with a public constructor with no parameters
- An object initializer allows you to assign values to any accessible members or properties of a class at the time of instantiation without calling a constructor with parameters

Summary (cont'd.)

- You can overload an operator by writing a method whose identifier is operator, followed by the operator being overloaded—for example, operator+() or operator*()
- You can declare arrays that hold elements of any type, including objects
- A destructor contains the actions you require when an instance of a class is destroyed