Beginning Visual C# 2005

Exercise Answers

Chapter 1: Introducing C#

No Exercises.

Chapter 2: Writing a C# Program

No Exercises.

Chapter 3: Variables and Expressions

Exercise 1

Question: In the following code, how would we refer to the name `great` from code in the namespace `fabulous`?

```
namespace fabulous
{
   // code in fabulous namespace
}

namespace super
{
   namespace smashing
   {
      // great name defined
   }
}
```

Answer:

```
super.smashing.great
```

Exercise 2

Question: Which of the following is not a legal variable name:

a) `myVariableIsGood`

b) `99Flake`

c) `_floor`

d) `time2GetJiggyWidIt`

e) `wrox.com`

Answer: **b** — because it starts with a number
and
e — because it contains a full stop

Exercise 3

Question: Is the string "`supercalifragilisticexpialidocious`" too big to fit in a `string` variable? Why?

Answer: No, there is no theoretical limit to the size of a string that may be contained in a `string` variable.

Exercise 4

Question: By considering operator precedence, list the steps involved in the computation of the following expression:

```
resultVar += var1 * var2 + var3 << var4 / var5;
```

Answer: The `*` and `/` operators have the highest precedence here, followed by `+`, `<<`, and finally `+=`. The precedence in the exercise can be illustrated using parentheses as follows:

```
resultVar += (((var1 * var2) + var3) << (var4 / var5));
```

Exercise 5

Question: Write a console application that obtains four `int` values from the user and displays their product.

Answer:

```
      static void Main(string[] args)
      {
         int firstNumber, secondNumber, thirdNumber, fourthNumber;
         Console.WriteLine("Give me a number:");
         firstNumber = Convert.ToInt32(Console.ReadLine());
         Console.WriteLine("Give me another number:");
         secondNumber = Convert.ToInt32(Console.ReadLine());
         Console.WriteLine("Give me another number:");
         thirdNumber = Convert.ToInt32(Console.ReadLine());
         Console.WriteLine("Give me another number:");
         fourthNumber = Convert.ToInt32(Console.ReadLine());
         Console.WriteLine("The product of {0}, {1}, {2}, and {3} is
{4}.",
                           firstNumber, secondNumber, thirdNumber,
fourthNumber,
                           firstNumber * secondNumber * thirdNumber *
fourthNumber);
```

```
}
```

Note that `Convert.ToInt32()` is used here, which isn't covered in the chapter.

Chapter 4: Flow Control

Exercise 1

Question: If you have two integers stored in variables `var1` and `var2`, what Boolean test can you perform to see if one or the other of them (but not both) is greater than 10?

Answer:

```
(var1 > 10) ^ (var2 > 10)
```

Exercise 2

Question: Write an application that includes the logic from Exercise 1, obtains two numbers from the user and displays them, but rejects any input where both numbers are greater than 10 and asks for two new numbers.

Answer:

```
static void Main(string[] args)
{
   bool numbersOK = false;
   double var1, var2;
   var1 = 0;
   var2 = 0;
   while (!numbersOK)
   {
      Console.WriteLine("Give me a number:");
      var1 = Convert.ToDouble(Console.ReadLine());
      Console.WriteLine("Give me another number:");
      var2 = Convert.ToDouble(Console.ReadLine());
      if ((var1 > 10) ^ (var2 > 10))
      {
         numbersOK = true;
      }
      else
      {
         if ((var1 <= 10) && (var2 <= 10))
         {
            numbersOK = true;
         }
         else
         {
            Console.WriteLine("Only one number may be greater than
10.");
         }
      }
   }
```

```
         Console.WriteLine("You entered {0} and {1}.", var1, var2);
      }
```

Note that this can be performed better using different logic, for example:

```
      static void Main(string[] args)
      {
         bool numbersOK = false;
         double var1, var2;
         var1 = 0;
         var2 = 0;
         while (!numbersOK)
         {
            Console.WriteLine("Give me a number:");
            var1 = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("Give me another number:");
            var2 = Convert.ToDouble(Console.ReadLine());
            if ((var1 > 10) && (var2 > 10))
            {
               Console.WriteLine("Only one number may be greater than
10.");
            }
            else
            {
               numbersOK = true;
            }
         }
         Console.WriteLine("You entered {0} and {1}.", var1, var2);
      }
```

Exercise 3

Question: What is wrong with the following code?

```
int i;
for (i = 1; i <= 10; i++)
{
   if ((i % 2) = 0)
      continue;
   Console.WriteLine(i);
}
```

Answer: The code should read:

```
int i;
for (i = 1; i <= 10; i++)
{
   if ((i % 2) == 0)
      continue;
   Console.WriteLine(i);
}
```

Using the = assignment operator instead of the Boolean == operator is a very common mistake.

Exercise 4

Question: Modify the Mandelbrot set application to request image limits from the user and display the chosen section of the image. The current code outputs as many characters as will fit on a single line of a console application. Consider making every image chosen fit in the same amount of space to maximize the viewable area.

Answer:

```
      static void Main(string[] args)
      {
         double realCoord, imagCoord;
         double realMax = 1.77;
         double realMin = -0.6;
         double imagMax = -1.2;
         double imagMin = 1.2;
         double realStep;
         double imagStep;
         double realTemp, imagTemp, realTemp2, arg;
         int iterations;
         while (true)
         {
            realStep = (realMax - realMin) / 79;
            imagStep = (imagMax - imagMin) / 48;
            for (imagCoord = imagMin; imagCoord >= imagMax;
                 imagCoord += imagStep)
            {
               for (realCoord = realMin; realCoord <= realMax;
                    realCoord += realStep)
               {
                  iterations = 0;
                  realTemp = realCoord;
                  imagTemp = imagCoord;
                  arg = (realCoord * realCoord) + (imagCoord *
imagCoord);
                  while ((arg < 4) && (iterations < 40))
                  {
                     realTemp2 = (realTemp * realTemp) - (imagTemp *
imagTemp)
                        - realCoord;
                     imagTemp = (2 * realTemp * imagTemp) - imagCoord;
                     realTemp = realTemp2;
                     arg = (realTemp * realTemp) + (imagTemp *
imagTemp);
                     iterations += 1;
                  }
                  switch (iterations % 4)
                  {
                     case 0:
                        Console.Write(".");
```

```
                        break;
                     case 1:
                        Console.Write("o");
                        break;
                     case 2:
                        Console.Write("O");
                        break;
                     case 3:
                        Console.Write("@");
                        break;
                  }
               }
               Console.Write("\n");
            }
            Console.WriteLine("Current limits:");
            Console.WriteLine("realCoord: from {0} to {1}", realMin,
realMax);
            Console.WriteLine("imagCoord: from {0} to {1}", imagMin,
imagMax);

            Console.WriteLine("Enter new limits:");
            Console.WriteLine("realCoord: from:");
            realMin = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("realCoord: to:");
            realMax = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("imagCoord: from:");
            imagMin = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("imagCoord: to:");
            imagMax = Convert.ToDouble(Console.ReadLine());
         }
      }
```

Chapter 5: More About Variables

Exercise 1

Question: Which of the following conversions can't be performed implicitly:

a) `int` to `short`

b) `short` to `int`

c) `bool` to `string`

d) `byte` to `float`

Answer: Conversions **a** and **c** can't be performed implicitly.

Exercise 2

Question: Give the code for a `color` enumeration based on the `short` type containing the colors of the rainbow plus black and white. Can this enumeration be based on the `byte` type?

Answer:

```
enum color : short
{
   Red, Orange, Yellow, Green, Blue, Indigo, Violet, Black, White
}
```

Yes, because the `byte` type can hold numbers between 0 and 255, so `byte`-based enumerations can hold 256 entries with individual values, or more if duplicate values are used for entries.

Exercise 3

Question: Modify the Mandelbrot set generator example from the last chapter to use the following `struct` for complex numbers:

```
struct imagNum
{
   public double real, imag;
}
```

Answer:

```
      static void Main(string[] args)
      {
         imagNum coord, temp;
         double realTemp2, arg;
         int iterations;
         for (coord.imag = 1.2; coord.imag >= -1.2; coord.imag -= 0.05)
         {
            for (coord.real = -0.6; coord.real <= 1.77; coord.real +=
0.03)
            {
               iterations = 0;
               temp.real = coord.real;
               temp.imag = coord.imag;
               arg = (coord.real * coord.real) + (coord.imag *
coord.imag);
               while ((arg < 4) && (iterations < 40))
               {
                  realTemp2 = (temp.real * temp.real) - (temp.imag *
temp.imag)
                     - coord.real;
                  temp.imag = (2 * temp.real * temp.imag) - coord.imag;
                  temp.real = realTemp2;
                  arg = (temp.real * temp.real) + (temp.imag *
temp.imag);
```

```
            iterations += 1;
         }
         switch (iterations % 4)
         {
            case 0:
               Console.Write(".");
               break;
            case 1:
               Console.Write("o");
               break;
            case 2:
               Console.Write("O");
               break;
            case 3:
               Console.Write("@");
               break;
         }
      }
      Console.Write("\n");
   }
}
```

Exercise 4

Question: Will the following code compile? Why?

```
string[] blab = new string[5]
string[5] = 5th string.
```

Answer: No, for the following reasons:

1. End of statement semicolons are missing.
1. 2nd line attempts to access a non-existent 6th element of `blab`.
1. 2nd line attempts to assign a string that isn't enclosed in double quotes.

Exercise 5

Question: Write a console application that accepts a string from the user and outputs a string with the characters in reverse order.

Answer:

```
static void Main(string[] args)
{
   Console.WriteLine("Enter a string:");
   string myString = Console.ReadLine();
   string reversedString = "";
   for (int index = myString.Length - 1; index >= 0; index--)
   {
      reversedString += myString[index];
   }
   Console.WriteLine("Reversed: {0}", reversedString);
```

```
}
```

Exercise 6

Question: Write a console application that accepts a string and replaces all occurrences of the string "no" with "yes".

Answer:

```
static void Main(string[] args)
{
   Console.WriteLine("Enter a string:");
   string myString = Console.ReadLine();
   myString = myString.Replace("no", "yes");
   Console.WriteLine("Replaced \"no\" with \"yes\": {0}",
myString);
}
```

Exercise 7

Question: Write a console application that places double quotes around each word in a string.

Answer:

```
static void Main(string[] args)
{
   Console.WriteLine("Enter a string:");
   string myString = Console.ReadLine();
   myString = "\"" + myString.Replace(" ", "\" \"") + "\"";
   Console.WriteLine("Added double quotes areound words: {0}",
myString);
}
```

Or using `String.Split()`:

```
static void Main(string[] args)
{
   Console.WriteLine("Enter a string:");
   string myString = Console.ReadLine();
   string[] myWords = myString.Split(' ');
   Console.WriteLine("Adding double quotes areound words:");
   foreach (string myWord in myWords)
   {
      Console.Write("\"{0}\" ", myWord);
   }
}
```

Chapter 6: Functions

Exercise 1

Question: The following two functions have errors. What are they?

```
static bool Write()
{
   Console.WriteLine("Text output from function.");
}
```

```
static void myFunction(string label, params int[] args, bool showLabel)
{
   if (showLabel)
      Console.WriteLine(label);
   foreach (int i in args)
      Console.WriteLine("{0}", i);
}
```

Answer: The first function has a return type of `bool`, but doesn't return a `bool` value.

The second function has a `params` argument, but this argument isn't at the end of the argument list.

Exercise 2

Question: Write an application that uses two command line arguments to place values into a string and an integer variable respectively, then display these values.

Answer:

```
      static void Main(string[] args)
      {
         if (args.Length != 2)
         {
            Console.WriteLine("Two arguments required.");
            return;
         }
         string param1 = args[0];
         int param2 = Convert.ToInt32(args[1]);
         Console.WriteLine("String parameter: {0}", param1);
         Console.WriteLine("Integer parameter: {0}", param2);
      }
```

Note that this answer contains code that checks that 2 arguments have been supplied, which wasn't part of the question but seems logical in this situation.

Exercise 3

Question: Create a delegate and use it to impersonate the `Console.ReadLine()` function when asking for user input.

Answer:

```
   class Class1
   {
      delegate string ReadLineDelegate();

      static void Main(string[] args)
      {
         ReadLineDelegate readLine = new
ReadLineDelegate(Console.ReadLine);
         Console.WriteLine("Type a string:");
         string userInput = readLine();
         Console.WriteLine("You typed: {0}", userInput);
      }
   }
```

Exercise 4

Question: Modify the following `struct` to include a function that returns the total price of an order:

```
struct order
{
   public string itemName;
   public int    unitCount;
   public double unitCost;
}
```

Answer:

```
struct order
{
   public string itemName;
   public int    unitCount;
   public double unitCost;

   public double TotalCost()
   {
      return unitCount * unitCost;
   }
}
```

Exercise 5

Question: Add another function to the `order struct` that returns a formatted string as follows, where italic entries enclosed in angle brackets are replaced by appropriate values:

Order Information: <*unitCount*> <*itemName*> items at $<*unitCost*> each, total cost $<*totalCost*>.

Answer:

```
struct order
```

```
{
   public string itemName;
   public int    unitCount;
   public double unitCost;

   public double TotalCost()
   {
      return unitCount * unitCost;
   }

   public string Info()
   {
      return "Order information: " + unitCount.ToString() + " " +
itemName +
             " items at $" + unitCost.ToString() + " each, total cost $"
+
             TotalCost().ToString();
   }
}
```

Chapter 7: Debugging and Error Handling

Exercise 1

Question: "Using `Trace.WriteLine()` is preferable to using `Debug.WriteLine()` as the `Debug` version only works in debug builds." Do you agree with this statement? Why?

Answer: This statement is only true for information that you want to make available in all builds. More often, you will want debugging information to be written out only when debug builds are used. In this situation, the `Debug.WriteLine()` version is preferable.

Using the `Debug.WriteLine()` version also has the advantage that it will not be compiled into release builds, thus reducing the size of the resultant code.

Exercise 2

Question: Provide code for a simple application containing a loop that generates an error after 5000 cycles. Use a breakpoint to enter break mode just before the error is caused on the 5000th cycle. (Note: a simple way to generate an error is to attempt to access a non existent array element, such as `myArray[1000]` in an array with a hundred elements.)

Answer:

```
      static void Main(string[] args)
      {
         for (int i = 1; i < 10000; i++)
         {
            Console.WriteLine("Loop cycle {0}", i);
            if (i == 5000)
            {
```

```
                Console.WriteLine(args[999]);
            }
        }
    }
```

In the preceding code a breakpoint could be placed on the following line:

```
            Console.WriteLine("Loop cycle {0}", i);
```

The properties of the breakpoint should be modified such that the hit count criterion is "break when hit count is equal to 5000".

Exercise 3

Question: "`finally` code blocks only execute if a `catch` block isn't executed." True or false?

Answer: False. Finally blocks *always* execute. This may occur after a `catch` block has been processed.

Exercise 4

Question: Given the enumeration data type `orientation` defined below, write an application that uses SEH to cast a `byte` type variable into an `orientation` type variable in a safe way. Note that you can force exceptions to be thrown using the `checked` keyword, an example of which is shown below. This code should be used in your application.

```
enum orientation : byte
{
   north = 1,
   south = 2,
   east  = 3,
   west  = 4
}
```

```
myDirection = checked((orientation)myByte);
```

Answer:

```
      static void Main(string[] args)
      {
         orientation myDirection;
         for (byte myByte = 2; myByte < 10; myByte++)
         {
            try
            {
               myDirection = checked((orientation)myByte);
               if ((myDirection < orientation.north) ||
                   (myDirection > orientation.west))
```

```
               {
                  throw new ArgumentOutOfRangeException("myByte",
myByte,
                                                "Value must be between 1
and 4");
               }
            }
            catch (ArgumentOutOfRangeException e)
            {
               // If this section is reached then myByte < 1 or myByte >
4.
               Console.WriteLine(e.Message);
               Console.WriteLine("Assigning default value,
orientation.north.");
               myDirection = orientation.north;
            }

            Console.WriteLine("myDirection = {0}", myDirection);
         }
      }
```

Note that this is a bit of a trick question. Since the enumeration is based on the `byte` type any `byte` value may be assigned to it, even if that value isn't assigned a name in the enumeration. In the preceding code you generate your own exception if necessary.

Chapter 8: Introduction to Object-Oriented Programming

Exercise 1

Question: Which of the following are real levels of accessibility in OOP?

1. Friend
1. Public
1. Secure
1. Private
1. Protected
1. Loose

Wildcard

Answer: Public, private, and protected are real levels of accessibility.

Exercise 2

Question: "You must call the destructor of an object manually, or it will waste memory." True or False?

Answer: False. You should never call the destructor of an object manually—the .NET runtime environment will do this for you during garbage collection.

Exercise 3

Question: Do you need to create an object in order to call a static method of its class?

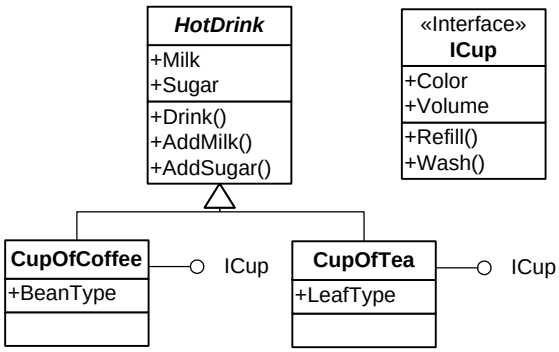
Answer: No, you can call static methods without any class instances.

Exercise 4

Question: Draw a UML diagram similar to the ones shown in this chapter for the following classes and interface:

1. An abstract class called `HotDrink` that has the methods `Drink()`, `AddMilk()`, and `AddSugar()`, and the properties `Milk` and `Sugar`.
1. An interface called `ICup` that has the methods `Refill()` and `Wash()`, and the properties `Color` and `Volume`.
1. A class called `CupOfCoffee` that derives from `HotDrink`, supports the `ICup` interface, and has the additional property `BeanType`.
1. A class called `CupOfTea` that derives from `HotDrink`, supports the `ICup` interface, and has the additional property `LeafType`.

Answer:



Exercise 5

Question: Write some code for a function that would accept either of the two cup objects in the above example as a parameter. The function should call the `AddMilk()`, `Drink()`, and `Wash()` methods for the cup object it is passed.

Answer:

```
static void ManipulateDrink(HotDrink drink)
{
   drink.AddMilk();
```

```
   drink.Drink();
   ICup cupInterface = (ICup)drink;
   cupInterface.Wash();
}
```

Note the explicit cast to `ICup`. This is necessary as `HotDrink` doesn't support the `ICup` interface, but you know that the two cup objects that might be passed to this function do. However, this is dangerous, as other classes deriving from `HotDrink` are possible, which might not support `ICup`, but could be passed to this function. To correct this you should check to see if the interface is supported:

```
static void ManipulateDrink(HotDrink drink)
{
   drink.AddMilk();
   drink.Drink();
   if (drink is ICup)
   {
      ICup cupInterface = drink as ICup;
      cupInterface.Wash();
   }
}
```

The `is` and `as` operators used here are covered in Chapter 11.

Chapter 9: Defining Classes

Exercise 1

Question: What is wrong with the following code?

```
public sealed class MyClass
{
   // class members
}

public class myDerivedClass : MyClass
{
   // class members
}
```

Answer: `myDerivedClass` derives from `MyClass`, but `MyClass` is sealed and can't be derived from.

Exercise 2

Question: How would you define a non-creatable class?

Answer: By defining all of its constructors as private.

Exercise 3

Question: Why are non-creatable classes still useful? How do you make use of their capabilities?

Answer: Non-creatable classes can be useful through the static members they possess. In fact, you can even get instances of these classes through these members, for example:

```
class CreateMe
{
   private CreateMe()
   {
   }

   static public CreateMe GetCreateMe()
   {
      return new CreateMe();
   }
}
```

The internal function has access to the private constructor.

Exercise 4

Question: Write code in a class library project called `Vehicles` that implements the `Vehicle` family of objects discussed earlier in this chapter, in the section on interfaces vs. abstract classes. There are 9 objects and 2 interfaces that require implementation.

Answer:

```
namespace Vehicles
{
   public abstract class Vehicle
   {
   }
   public abstract class Car : Vehicle
   {
   }
   public abstract class Train : Vehicle
   {
   }
   public interface IPassengerCarrier
   {
   }
   public interface IHeavyLoadCarrier
   {
   }
   public class SUV : Car, IPassengerCarrier
   {
   }
   public class Pickup : Car, IPassengerCarrier, IHeavyLoadCarrier
   {
   }
```

```
   public class Compact : Car, IPassengerCarrier
   {
   }
   public class PassengerTrain : Train, IPassengerCarrier
   {
   }
   public class FreightTrain : Train, IHeavyLoadCarrier
   {
   }
   public class T424DoubleBogey : Train, IHeavyLoadCarrier
   {
   }
}
```

Exercise 5

Question: Create a console application project, `Traffic`, that references `Vehicles.dll` (created in Question 4 above). Include a function, `AddPassenger()`, that accepts any object with the `IPassengerCarrier` interface. To prove that the code works, call this function using instances of each object that supports this interface, calling the `ToString()` method inherited from `System.Object` on each one and writing the result to the screen.

Answer:

```
using System;
using Vehicles;

namespace Traffic
{
   class Class1
   {
      static void Main(string[] args)
      {
         AddPassenger(new Compact());
         AddPassenger(new SUV());
         AddPassenger(new Pickup());
         AddPassenger(new PassengerTrain());
      }

      static void AddPassenger(IPassengerCarrier Vehicle)
      {
         Console.WriteLine(Vehicle.ToString());
      }
   }
}
```

Chapter 10: Defining Class Members

Exercise 1

Question: Write code that defines a base class, `MyClass`, with the virtual method `GetString()`. This method should return the string stored in the protected field `myString`, accessible through the write only public property `ContainedString`.

Answer:

```
class MyClass
{
   protected string myString;

   public string ContainedString
   {
      set
      {
         myString = value;
      }
   }

   public virtual string GetString()
   {
      return myString;
   }
}
```

Exercise 2

Question: Derive a class, `MyDerivedClass`, from `MyClass`. Override the `GetString()` method to return the string from the base class using the base implementation of the method, but add the text `" (output from derived class)"` to the returned string.

Answer:

```
class MyDerivedClass : MyClass
{
   public override string GetString()
   {
      return base.GetString() + " (output from derived class)";
   }
}
```

Exercise 3

Question: Write a class called `MyCopyableClass` that is capable of returning a copy of itself using the method `GetCopy()`. This method should use the `MemberwiseClone()` method inherited from `System.Object`. Add a simple property to the class, and write client code that uses the class to check that everything is working.

Answer:

```
class MyCopyableClass
{
   protected int myInt;

   public int ContainedInt
   {
      get
      {
         return myInt;
      }
      set
      {
         myInt = value;
      }
   }

    public MyCopyableClass GetCopy()
   {
      return (MyCopyableClass)MemberwiseClone();
   }
}
```

And the client code:

```
class Class1
{
   static void Main(string[] args)
   {
      MyCopyableClass obj1 = new MyCopyableClass();
      obj1.ContainedInt = 5;
      MyCopyableClass obj2 = obj1.GetCopy();
      obj1.ContainedInt = 9;
      Console.WriteLine(obj2.ContainedInt);
   }
}
```

This code displays 5, showing that the copied object has its own version of the `myInt` field.

Exercise 4

Question: Write a console client for the `Ch10CardLib` library that 'draws' 5 cards at a time from a shuffled `Deck` object. If all 5 cards are the same suit then the client should display the card names on screen along with the text `Flush!`, otherwise it should quit after 50 cards with the text `No flush`.

Answer:

```
using System;
using Ch10CardLib;
```

```
namespace Exercise_Answers
{
   class Class1
   {
      static void Main(string[] args)
      {
         while(true)
         {
            Deck playDeck = new Deck();
            playDeck.Shuffle();
            bool isFlush = false;
            int flushHandIndex = 0;
            for (int hand = 0; hand < 10; hand++)
            {
               isFlush = true;
               Suit flushSuit = playDeck.GetCard(hand * 5).suit;
               for (int card = 1; card < 5; card++)
               {
                  if (playDeck.GetCard(hand * 5 + card).suit !=
flushSuit)
                  {
                     isFlush = false;
                  }
               }
               if (isFlush)
               {
                  flushHandIndex = hand * 5;
                  break;
               }
            }
            if (isFlush)
            {
               Console.WriteLine("Flush!");
               for (int card = 0; card < 5; card++)
               {
                  Console.WriteLine(playDeck.GetCard(flushHandIndex +
card));
               }
            }
            else
            {
               Console.WriteLine("No flush.");
            }
            Console.ReadLine();
         }
      }
   }
}
```

Note: placed in a loop, as flushes are uncommon. You may need to press return several times before a flush is found in a shuffled deck. To verify that everything is working as it should, try commenting out the line that shuffles the deck.

Chapter 11: Collections, Comparisons, and Conversions

Exercise 1

Question: Create a collection class called `People` that is a collection of the `Person` class shown below. The items in the collection should be accessible via a string indexer that is the name of the person, identical to the `Person.Name` property.

```
public class Person
{
   private string name;
   private int age;

   public string Name
   {
      get
      {
         return name;
      }
      set
      {
         name = value;
      }
   }

   public int Age
   {
      get
      {
         return age;
      }
      set
      {
         age = value;
      }
   }
}
```

Answer:

```
using System;
using System.Collections;

namespace Exercise_Answers
{
   public class People : DictionaryBase
   {
      public void Add(Person newPerson)
      {
         Dictionary.Add(newPerson.Name, newPerson);
      }
```

```
      public void Remove(string name)
      {
         Dictionary.Remove(name);
      }

      public Person this[string name]
      {
         get
         {
            return (Person)Dictionary[name];
         }
         set
         {
            Dictionary[name] = value;
         }
      }
   }
}
```

Exercise 2

Question: Extend the `Person` class from the preceding exercise such that the >, <, >=, and <= operators are overloaded, and compare the `Age` properties of `Person` instances.

Answer:

```
public class Person
{
   private string name;
   private int age;

   public string Name
   {
      get
      {
         return name;
      }
      set
      {
         name = value;
      }
   }

   public int Age
   {
      get
      {
         return age;
      }
      set
      {
         age = value;
      }
   }
```

```
   public static bool operator >(Person p1, Person p2)
   {
      return p1.Age > p2.Age;
   }

   public static bool operator <(Person p1, Person p2)
   {
      return p1.Age < p2.Age;
   }

   public static bool operator >=(Person p1, Person p2)
   {
      return !(p1 < p2);
   }

   public static bool operator <=(Person p1, Person p2)
   {
      return !(p1 > p2);
   }
}
```

Exercise 3

Question: Add a `GetOldest()` method to the `People` class that returns an array of `Person` objects with the greatest `Age` property (1 or more objects, as multiple items may have the same value for this property), using the overloaded operators defined above.

Answer:

```
   public Person[] GetOldest()
   {
      Person oldestPerson = null;
      People oldestPeople = new People();
      Person currentPerson;
      foreach (DictionaryEntry p in Dictionary)
      {
         currentPerson = p.Value as Person;
         if (oldestPerson == null)
         {
            oldestPerson = currentPerson;
            oldestPeople.Add(oldestPerson);
         }
         else
         {
            if (currentPerson > oldestPerson)
            {
               oldestPeople.Clear();
               oldestPeople.Add(currentPerson);
               oldestPerson = currentPerson;
            }
            else
            {
               if (currentPerson >= oldestPerson)
```

```
            {
               oldestPeople.Add(currentPerson);
            }
         }
      }
   }
   Person[] oldestPeopleArray = new Person[oldestPeople.Count];
   int copyIndex = 0;
   foreach (DictionaryEntry p in oldestPeople)
   {
      oldestPeopleArray[copyIndex] = p.Value as Person;
      copyIndex++;
   }
   return oldestPeopleArray;
}
```

This function is made more complex by the fact that no == operator has been defined for `Person`, but the logic can still be constructed without this. In addition, returning a `People` instance would be simpler, as it is easier to manipulate this class during processing. As a compromise, a `People` instance is used throughout the function, then converted into an array of `Person` instances at the end.

Exercise 4

Question: Implement the `ICloneable` interface on the `People` class to provide deep copying capability.

Answer:

```
public class People : DictionaryBase, ICloneable
{
   public object Clone()
   {
      People clonedPeople = new People();
      Person currentPerson, newPerson;
      foreach (DictionaryEntry p in Dictionary)
      {
         currentPerson = p.Value as Person;
         newPerson = new Person();
         newPerson.Name = currentPerson.Name;
         newPerson.Age = currentPerson.Age;
         clonedPeople.Add(newPerson);
      }
      return clonedPeople;
   }

   ...
}
```

Note: this could be simplified by implementing `ICloneable` on the `Person` class.

Exercise 5

Question: Add an iterator to the `People` class that enables you to get the ages of all members in a `foreach` loop as follows:

```
foreach (int age in myPeople.Ages)
{
   // Display ages.
}
```

Answer:

```
   public IEnumerable Ages
   {
      get
      {
         foreach (object person in Dictionary.Values)
            yield return (person as Person).Age;
      }
   }
```

Chapter 12: Generics

Exercise 1

Question: Which of the following can be generic?

a) classes

b) methods

c) properties

d) operator overloads

e) structs

f) enumerations

Answer: **a**, **b**, & **e**: yes
c & **d**: no, although they can use generic type parameters supplied by the class containing them
f: no

Exercise 2

Question: Extend the `Vector` class in Ch12Ex01 such that the `*` operator returns the dot product of two vectors.

The dot product of two vectors is defined as the product of their magnitudes multiplied by the cosine of the angle between them.

Answer:

```
      public static double? operator *(Vector op1, Vector op2)
      {
         try
         {
            double angleDiff = (double)(op2.ThetaRadians.Value –
               op1.ThetaRadians.Value);
            return op1.R.Value * op2.R.Value * Math.Cos(angleDiff);
         }
         catch
         {
            return null;
         }
      }
```

Exercise 3

Question: What is wrong with the following code? Fix it.

```
public class Instantiator<T>
{
   public T instance;

   public Instantiator()
   {
      instance = new T();
   }
}
```

Answer: You can't instantiate `T` without enforcing the `new()` constraint on it, which ensures that a public default constructor is available:

```
public class Instantiator<T>
   where T : new()
{
   public T instance;

   public Instantiator()
   {
      instance = new T();
   }
}
```

Exercise 4

Question: What is wrong with the following code? Fix it.

```
public class StringGetter<T>
{
   public string GetString<T>(T item)
   {
      return item.ToString();
   }
}
```

Answer: The same generic type parameter, `T`, is used on both the generic class and generic method. You need to rename one or both, for example:

```
public class StringGetter<U>
{
   public string GetString<T>(T item)
   {
      return item.ToString();
   }
}
```

Exercise 5

Question: Create a generic class called `ShortCollection<T>` that implements `IList<T>` and consists of a collection of items with a maximum size. This maximum size should be an integer that can be supplied to the constructor of `ShortCollection<T>`, or defaults to 10. The constructor should also be able to take an initial list of items via a `List<T>` parameter. The class should function exactly like `Collection<T>`, but throw an exception of type `IndexOutOfRangeException` if an attempt is made to add too many items to the collection, or if the `List<T>` passed to the constructor contains too many items.

Answer: One way of doing this is as follows:

```
public class ShortCollection<T> : IList<T>
{
   protected Collection<T> innerCollection;
   protected int maxSize = 10;

   public ShortCollection() : this(10)
   {
   }

   public ShortCollection(int size)
   {
      maxSize = size;
      innerCollection = new Collection<T>();
   }

   public ShortCollection(List<T> list) : this(10, list)
   {
   }

   public ShortCollection(int size, List<T> list)
   {
      maxSize = size;
      if (list.Count <= maxSize)
      {
         innerCollection = new Collection<T>(list);
      }
      else
      {
```

```
         ThrowTooManyItemsException();
      }
   }

   protected void ThrowTooManyItemsException()
   {
      throw new IndexOutOfRangeException(
         "Unable to add any more items, maximum size is " +
maxSize.ToString()
         + " items.");
   }

   #region IList<T> Members

   public int IndexOf(T item)
   {
      return (innerCollection as IList<T>).IndexOf(item);
   }

   public void Insert(int index, T item)
   {
      if (Count < maxSize)
      {
         (innerCollection as IList<T>).Insert(index, item);
      }
      else
      {
         ThrowTooManyItemsException();
      }
   }

   public void RemoveAt(int index)
   {
      (innerCollection as IList<T>).RemoveAt(index);
   }

   public T this[int index]
   {
      get
      {
         return (innerCollection as IList<T>)[index];
      }

      set
      {
         (innerCollection as IList<T>)[index] = value;
      }
   }

   #endregion

   #region ICollection<T> Members

   public void Add(T item)
```

```
{
   if (Count < maxSize)
   {
      (innerCollection as ICollection<T>).Add(item);
   }
   else
   {
      ThrowTooManyItemsException();
   }
}

public void Clear()
{
   (innerCollection as ICollection<T>).Clear();
}

public bool Contains(T item)
{
   return (innerCollection as ICollection<T>).Contains(item);
}

public void CopyTo(T[] array, int arrayIndex)
{
   (innerCollection as ICollection<T>).CopyTo(array, arrayIndex);
}

public int Count
{
   get
   {
      return (innerCollection as ICollection<T>).Count;
   }
}

public bool IsReadOnly
{
   get
   {
      return (innerCollection as ICollection<T>).IsReadOnly;
   }
}

public bool Remove(T item)
{
   return (innerCollection as ICollection<T>).Remove(item);
}

#endregion

#region IEnumerable<T> Members

public IEnumerator<T> GetEnumerator()
{
   return (innerCollection as IEnumerable<T>).GetEnumerator();
```

```
   }

   #endregion
}
```

Chapter 13: Additional OOP Techniques

Exercise 1

Question: Show the code for an event handler that uses the general purpose `(object sender, EventArgs e)` syntax that will accept either the `Timer.Elapsed` event or the `Connection.MessageArrived` event from the code earlier in this chapter. The handler should output a string specifying which type of event has been received, along with the `Message` property of the `MessageArrivedEventArgs` parameter or the `SignalTime` property of the `ElapsedEventArgs` parameter, depending on which event occurs.

Answer:

```
   public void ProcessEvent(object source, EventArgs e)
   {
      if (e is MessageArrivedEventArgs)
      {
         Console.WriteLine("Connection.MessageArrived event received.");
         Console.WriteLine("Message: {0}",
                           (e as MessageArrivedEventArgs).Message);
      }
      if (e is ElapsedEventArgs)
      {
         Console.WriteLine("Timer.Elapsed event received.");
         Console.WriteLine("SignalTime: {0}",
                            (e as ElapsedEventArgs ).SignalTime);
      }
   }

   public void ProcessElapsedEvent(object source, ElapsedEventArgs e)
   {
      ProcessEvent(source, e);
   }
```

Note that you need this extra `ProcessElapsedEvent()` method, as the `ElapsedEventHandler` delegate can't be cast to an `EventHandler` delegate. You don't need to do this for your `MessageHandler` delegate since it has a syntax identical to `EventHandler`:

```
   public delegate void MessageHandler(object source, EventArgs e);
```

Exercise 2

Question: Modify the card game example to check for the more interesting winning condition of the popular card game rummy. This means that a player wind the game if

their hand contains two 'sets' of cards, one of which consists of three cards, and one of which consists of four cards. A set is defined as either a sequence of cards of the same suit (such as 3H, 4H, 5H, 6H) or several cards of the same rank (such as 2H, 2D, 2S).

Answer: Modify `Player.cs` as follows (1 modified method, 2 new ones):

```
public bool HasWon()
{
   // get temporary copy of hand, which may get modified.
   Cards tempHand = (Cards)hand.Clone();

   // find three and four of a kind sets
   bool fourOfAKind = false;
   bool threeOfAKind = false;
   int fourRank = -1;
   int threeRank = -1;

   int cardsOfRank;
   for (int matchRank = 0; matchRank < 13; matchRank++)
   {
      cardsOfRank = 0;
      foreach (Card c in tempHand)
      {
         if (c.rank == (Rank)matchRank)
         {
            cardsOfRank++;
         }
      }
      if (cardsOfRank == 4)
      {
         // mark set of four
         fourRank = matchRank;
         fourOfAKind = true;
      }
      if (cardsOfRank == 3)
      {
         // two threes means no win possible
         // (threeOfAKind will only be true if this code
         // has already executed)
         if (threeOfAKind == true)
         {
            return false;
         }
         // mark set of three
         threeRank = matchRank;
         threeOfAKind = true;
      }
   }

   // check simple win condition
   if (threeOfAKind && fourOfAKind)
   {
      return true;
```

```
   }

   // simplify hand if three or four of a kind is found, by removing
used cards
   if (fourOfAKind || threeOfAKind)
   {
      for (int cardIndex = tempHand.Count - 1; cardIndex >= 0;
cardIndex--)
      {
         if ((tempHand[cardIndex].rank == (Rank)fourRank)
             || (tempHand[cardIndex].rank == (Rank)threeRank))
         {
            tempHand.RemoveAt(cardIndex);
         }
      }
   }

   // at this point the function may have returned, because:
   // - a set of four and a set of three has been found, winning.
   // - two sets of three have been found, losing.
   // if the function hasn't returned then either:
   // - no sets have been found, and tempHand contains 7 cards.
   // - a set of three has been found, and tempHand contains 4 cards.
   // - a set of four has been found, and tempHand contains 3 cards.

   // find run of four sets, start by looking for cards of same suit in
the same
   // way as before
   bool fourOfASuit = false;
   bool threeOfASuit = false;
   int fourSuit = -1;
   int threeSuit = -1;

   int cardsOfSuit;
   for (int matchSuit = 0; matchSuit < 4; matchSuit++)
   {
      cardsOfSuit = 0;
      foreach (Card c in tempHand)
      {
         if (c.suit == (Suit)matchSuit)
         {
            cardsOfSuit++;
         }
      }
      if (cardsOfSuit == 7)
      {
         // if all cards are the same suit then two runs
         // are possible, but not definite.
         threeOfASuit = true;
         threeSuit = matchSuit;
         fourOfASuit = true;
         fourSuit = matchSuit;
      }
      if (cardsOfSuit == 4)
```

```
      {
         // mark four card suit.
         fourOfASuit = true;
         fourSuit = matchSuit;
      }
      if (cardsOfSuit == 3)
      {
         // mark three card suit.
         threeOfASuit = true;
         threeSuit = matchSuit;
      }
   }

   if (!(threeOfASuit || fourOfASuit))
   {
      // need at least one run possibility to continue.
      return false;
   }

   if (tempHand.Count == 7)
   {
      if (!(threeOfASuit && fourOfASuit))
      {
         // need a three and a four card suit.
         return false;
      }

      // create two temporary sets for checking.
      Cards set1 = new Cards();
      Cards set2 = new Cards();

      // if all 7 cards are the same suit...
      if (threeSuit == fourSuit)
      {
         // get min and max cards
         int maxVal, minVal;
         GetLimits(tempHand, out maxVal, out minVal);
         for (int cardIndex = tempHand.Count - 1; cardIndex >= 0;
cardIndex--)
         {
            if (((int)tempHand[cardIndex].rank < (minVal + 3))
                || ((int)tempHand[cardIndex].rank > (maxVal - 3)))
            {
               // remove all cards in a three card set that
               // starts at minVal or ends at maxVal.
               tempHand.RemoveAt(cardIndex);
            }
         }
         if (tempHand.Count != 1)
         {
            // if more then one card is left then there aren't two runs.
            return false;
         }
         if ((tempHand[0].rank == (Rank)(minVal + 3))
```

```
             || (tempHand[0].rank == (Rank)(maxVal - 3)))
         {
            // if spare card can make one of the three card sets into a
            // four card set then there are two sets.
            return true;
         }
         else
         {
            // if spare card doesn't fit then there are two sets of
three
            // cards but no set of four cards.
            return false;
         }
      }

      // if three card and four card suits are different...
      foreach (Card card in tempHand)
      {
         // split cards into sets.
         if (card.suit == (Suit)threeSuit)
         {
            set1.Add(card);
         }
         else
         {
            set2.Add(card);
         }
      }

      // check if sets are sequential.
      if (isSequential(set1) && isSequential(set2))
      {
         return true;
      }
      else
      {
         return false;
      }
   }

   // if four cards remain (three of a kind found)
   if (tempHand.Count == 4)
   {
      // if four cards remain then they must be the same suit.
      if (!fourOfASuit)
      {
         return false;
      }
      // won if cards are sequential.
      if (isSequential(tempHand))
      {
         return true;
      }
   }
```

```
   // if three cards remain (four of a kind found)
   if (tempHand.Count == 3)
   {
      // if three cards remain then they must be the same suit.
      if (!threeOfASuit)
      {
         return false;
      }
      // won if cards are sequential.
      if (isSequential(tempHand))
      {
         return true;
      }
   }

   // return false if two valid sets don't exist.
   return false;
}

// utility function to get max and min ranks of cards
// (same suit assumed)
private void GetLimits(Cards cards, out int maxVal, out int minVal)
{
   maxVal = 0;
   minVal = 14;
   foreach (Card card in cards)
   {
      if ((int)card.rank > maxVal)
      {
         maxVal = (int)card.rank;
      }
      if ((int)card.rank < minVal)
      {
         minVal = (int)card.rank;
      }
   }
}

// utility function to see if cards are in a run
// (same suit assumed)
private bool isSequential(Cards cards)
{
   int maxVal, minVal;
   GetLimits(cards, out maxVal, out minVal);
   if ((maxVal - minVal) == (cards.Count - 1))
   {
      return true;
   }
   else
   {
      return false;
   }
}
```

Chapter 14: Basic Windows Programming

Exercise 1

Question: In previous versions of Visual Studio it was quite difficult to get your own applications to display their controls in the new Windows XP style—that changed in this version of Visual Studio. For this exercise, locate where in a Windows Forms Application Windows XP styles are enabled in a new Windows Forms project. Experiment with enabling and disabling the styles and see how it affects the controls on the forms.

Answer: The file Program.cs in a Windows Forms project contains the Main() method of the application. By default this method looks similar to this:

```
    [STAThread]
    static void Main()
    {
      Application.EnableVisualStyles();
      Application.Run(new Form1());
    }
```

The line

```
Application.EnableVisualStyles();
```

Controls the visual style of the windows forms.

Please note this line does nothing on Windows 2000.

Exercise 2

Question: Modify the TabControl example by adding a couple of tab pages and display a message box with the text “You changed the current tab to <Text of the current tab> from <Text of the tab that was just left>."

Answer: The TabControl includes an event called `SelectedIndexChanged` that can be used to execute code when the user moves to another tab page.

1. In the Windows Form designer, select the TabControl and add two tabs to the control.
2. Name the tabs Tab Three and Tab Four
3. With the TabControl selected, add the event SelectedIndexChanged and go to the code window.
4. Enter the following code

```
    private void tabControl1_SelectedIndexChanged(object sender,
EventArgs e)

{
```

```
        string message = "You changed the current tab to '" +
tabControl1.SelectedTab.Text + "' from '" +
tabControl1.TabPages[mCurrentTabIndex].Text + "'";
        mCurrentTabIndex = tabControl1.SelectedIndex;
        MessageBox.Show(message);
    }
```

5. Add the private field mCurrentTabIndex to the top of the class as such:

```
partial class Form1 : Form
{
  private int mCurrentTabIndex = 0;
```

6. Run the application.

How It Works: By default the first tab that is displayed in a TabControl has index 0. You use this by setting the private field `mCurrentTabIndex` to zero.

In the `SelectedIndexChanged` method you build the message to display. This is done by using the property `SelectedTab` to get the Text property of the tab that was just selected and the `TabPages` collection to get the Text property of the tab pages specified by the `mCurrentTabIndex` field.

After the message is build the `mCurrentTabIndex` field is changed to point to the newly selected tab.

Exercise 3

Question: In the ListView example you used the `tag` property to save the fully qualified path to the folders and files in the ListView. Change this behavior by creating a new class that is derived from `ListViewItem` and use instances of this new class as the items in the ListView. Store the information about the files and folders in the new class using a property named `FullyQualifiedPath`.

Answer: By creating a class that is derived from the `ListViewItem` class you create something that can be used in place of the “intended” `ListViewItem` class. This means that, even though the ListView itself doesn’t know about the extra information on the class, you are able to store additional information on the items displayed in the ListView directly on the items.

1. Start by creating a new class and nam it `FQListViewItem`. Below is a full listing of the class:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;

namespace ListView
```

```
{
  class FQListViewItem : ListViewItem
  {
    private string mFullyQualifiedPath;

    public string FullyQualifiedPath
    {
      get { return mFullyQualifiedPath; }
      set { mFullyQualifiedPath = value; }
    }
  }
}
```

2. Find and change `ListViewItem` types to FQListViewItem types in the Form.cs file.
3. Find and change any reference to `.Tag` to `.FullyQualifiedPath`

In the `listViewFilesAndFoldes_ItemActivate` method, you will need to cast the selected item in the second line to an `FQListViewItem` item as such:

```
string filename =
((FQListViewItem)lw.SelectedItems[0]).FullyQualifiedPath;
```

Chapter 15: Advanced Windows Forms Features

Exercise 1

Question: Using the `LabelTextbox` example as the base, create a new property called `MaxLength` that stores the maximum number of characters that can be entered into the textbox. Then create two new events called `MaxLengthChanged` and `MaxLengthReached`. The `MaxLengthChanged` event should be raised when the `MaxLength` property is changed and `MaxLengthReached` should be raised when the user enters a character making the length of the text in the text box equal to the value of `MaxLength`.

Answer: In order to accomplish the task at hand, you are going to make one new property and two events.

Start by creating the property:

```
private int mMaxLength = 32767;
```

Create the property like this:

```
public int MaxLength
{
  get { return mMaxLength; }
  set
  {
    if (value >= 0 && value <= 32767)
    {
      mMaxLength = value;
```

```
        if (MaxLengthChanged != null)
          MaxLengthChanged(this, new EventArgs());
        txtLabelText.MaxLength = value;
      }
    }
  }
```

Next create the two new events:

```
public event System.EventHandler MaxLengthChanged;
public event System.EventHandler MaxLengthReached;
```

In the Form designer, select the TextBox and add an event handler for the TextChanged event. Add the following code:

```
private void txtLabelText_TextChanged(object sender, EventArgs e)
{
  if (txtLabelText.Text.Length >= mMaxLength)
  {
    if (MaxLengthReached != null)
      MaxLengthReached(this, new EventArgs());
  }
}
```

How It Works: The maximum length of the text in a normal TextBox is the size of an `System.Int32` type, but the default is 32767 characters which normally is well beyond what is needed. In the property in step 2 above, you check to see if the value is negative or above 32767 and ignore the change request if it is. If the value is found to be acceptable, the `MaxLength` property of the TextBox is set and the event `MaxLengthChanged` is raised.

The event handler `txtLabelText_TextChanged` checks if the maximum number of characters in the TextBox is equal to or above the number specified in `mMaxLength` and raise the `MaxLengthReached` events if it is.

Chapter 16: Using Common Dialogs

Exercise 1

Question: Because the `FontDialog` and the `ColorDialog` work in a similar way to the other dialogs discussed in this chapter, it's an easy job to add these dialogs to the Simple Editor application from Chapter 16.

Let the user change the font of the text box. To make this possible add a new menu entry to the main menu: F&ormat, and a submenu for Format: &Font.... Add a handler to this menu item. Add a `FontDialog` to the application with the help of the Windows Forms designer. Display this dialog in the menu handler, and set the ~~Font~~ property of the text box to the selected font.

Answer: This is the code for the `Click` event handler of the Format | Font menu:

```
private void OnFormatFont(object sender, EventArgs e)
{
   if (dlgSelectFont.ShowDialog() == DialogResult.OK)
   {
      textBoxEdit.Font = dlgSelectFont.Font;
   }
}
```

You also have to change the implementation of the `OnPrintPage()` method to use the selected font for a printout. In the previous implementation you created a new `Font` object in the `DrawString()` method of the `Graphics` object. Now, use the font of the `textBoxEdit` object by accessing the ~~`Font`~~ property instead. You also have to be aware of a font location problem if the user chooses a big font. To avoid one line partly overwriting the one above/below, change the fixed value you used to change the vertical position of the lines. A better way to do this would be to use the size of the font to change the vertical increment: use the ~~`Height`~~ property of the ~~`Font`~~ class.

The code changes for `OnPrintPage()` are shown here:

```
private void OnPrintPage(object sender,
      System.Drawing.Printing.PrintPageEventArgs e)
{
   int x = e.MarginBounds.Left;
   int y = e.MarginBounds.Top;

   while (linesPrinted < lines.Length)
   {
      e.Graphics.DrawString(lines[linesPrinted++],
            dlgSelectFont.Font,
            Brushes.Black, x, y);

      y += (dlgSelectFont.Font.Height + 3);

      if (y > e.MarginBounds.Bottom)
      {
         e.HasMorePages = true;
         return;
      }
   }

   linesPrinted = 0;
   e.HasMorePages = false;
}
```

Exercise 2

Question: Another great extension to the Simple Editor application would be to change the font color. Add a second submenu to the Format menu entry: Color…. Add a handler to this menu entry where you open up a `ColorDialog`. If the user presses the OK button, set the selected color of the `ColorDialog` to the `ForeColor` property of the text box.

In the `OnPrintPage()` method make sure that the chosen color is used only if the printer supports colors. You can check the color support of the printer with the `PageSettings.Color` property of the `PrintPageEventArgs` argument. You can create a brush object with the color of the text box with this code:

```
Brush brush = new SolidBrush(textBoxEdit.ForeColor);
```

This brush can then be used as an argument in the `DrawString()` method instead of the black brush you used in the example before.

Answer: The `Click` event handler of the Format | Color menu is implemented in the method `OnFormatColor`:

```
      private void OnFormatColor(object sender, EventArgs e)
      {
         if (dlgColor.ShowDialog() == DialogResult.OK)
         {
            textBoxEdit.ForeColor = dlgColor.Color;
         }
      }
```

The `OnBeginPrint()` method is changed to set the `pintBrush` field to a brush that is defined by `textboxEdit`—if the printer supports it:

```
      private void OnBeginPrint(object sender, PrintEventArgs e)
      {
         char[] param = { '\n' };

         if (dlgPrint.PrinterSettings.PrintRange ==
PrintRange.Selection)
         {
            lines = textBoxEdit.SelectedText.Split(param);
         }
         else
         {
            lines = textBoxEdit.Text.Split(param);
         }

         int i = 0;
         char[] trimParam = { '\r' };
         foreach (string s in lines)
         {
            lines[i++] = s.TrimEnd(trimParam);
         }

         if (dlgPrint.PrinterSettings.SupportsColor)
            printBrush = new SolidBrush(textBoxEdit.ForeColor);
         else
            printBrush = Brushes.Black;
      }
```

The `OnPrintPage()` method is changed to use the brush:

```
private void OnPrintPage(object sender,
      System.Drawing.Printing.PrintPageEventArgs e)
{
   int x = e.MarginBounds.Left;
   int y = e.MarginBounds.Top;

   while (linesPrinted < lines.Length)
   {
      e.Graphics.DrawString(lines[linesPrinted++],
            dlgSelectFont.Font,
            printBrush, x, y);

      y += (dlgSelectFont.Font.Height + 3);

      if (y > e.MarginBounds.Bottom)
      {
         e.HasMorePages = true;
         return;
      }
   }


   linesPrinted = 0;
   e.HasMorePages = false;
}
```

Chapter 17: Deploying Windows Applications

Exercise 1

Question: What are advantages of ClickOnce deployment?

Answer: ClickOnce deployment has the advantage that the user installing the application doesn't need administrator privileges. The application can be automatically installed by clicking on a hyperlink. Also, you can configure that new versions of the application can be installed automatically.

Exercise 2

Question: How are the required permissions defined with ClickOnce deployment?

Answer: The required permissions can be found with the Calculate Permission feature in the project settings of Visual Studio.

Exercise 3

Question: What is defined with a ClickOnce manifest?

Answer: The application manifest describes the application and required permissions, the deployment manifest describes deployment configuration such as update policies.

Exercise 4

Question: When is it necessary to use the Windows Installer?

Answer: If administrator permissions are required by the installation program, the Windows Installer is needed instead of ClickOnce deployment.

Exercise 5

Question: What different editors can you use to create a Windows installer package using Visual Studio?

Answer: File System Editor, Registry Editor, File Types Editor, User Interface Editor, Custom Actions Editor, Launch Condition Editor.

Chapter 18: Basic Web Programming

Exercise 1

Question: Add the username to the top of the ASP.NET pages you created in this chapter. You can use the LoginName control for this task.

Answer: The LoginName control can be prefixed with a "Hello, " as shown:

```
    <form id="form1" runat="server">
    <div>
        Hello,<asp:LoginName ID="LoginName1" runat="server" />
```

Exercise 2

Question: Add a page for password recovery. Add a link to the Web page `login.aspx`, so that the user can recover the password if the login fails.

Answer: Similar to the previously created page CreateUser.aspx, the password recovery page must be placed into a directory where a username is not required, e.g. in the Intro directory. The page just needs the PasswordRecovery control included:

```
        <asp:PasswordRecovery ID="PasswordRecovery1" runat="server">
        </asp:PasswordRecovery>
```

Exercise 3

Question: Change the data source of the page `Default.aspx` page so that it uses the Events database for displaying the events.

Answer: Add a `SqlDataSource` control to the page `Default.aspx`, and configure it to access the Events database:

```
     <asp:SqlDataSource ID="SqlDataSource1" runat="server"
        ConnectionString="<%$ ConnectionStrings:EventsConnectionString
%>"
```

```
        SelectCommand="SELECT [Id], [Title] FROM [Events] ORDER BY
[Title]"></asp:SqlDataSource>
```

With the drop down list of the page, set the `DataSourceId` to the id of the data source, the `DataTextField` to Title, and the `DataValueField` to Id.

Chapter 19: Advanced Web Programming

Exercise 1

Question: Create a new portal-style personal Web application

Answer: Create a new Web application by selecting the Visual Studio 2005 menu File | New | Web Site…

Exercise 2

Question: Create a user control to display your resume.

Answer: Select the template Web User Control from the Add New Item dialog. The name of the user control must be set, e.g. `Resume.ascx`. Add text and HTML code for your resume.

Exercise 3

Question: Create a user control to display a list of links to your preferred Web sites.

Answer: Use the same dialog and template as with exercise 2 to create a user control. With this control add HTML code for a list and links to Websites as shown:

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="Links.ascx.cs" Inherits="Links" %>
<ul>
  <li><a href="http://www.microsoft.com">Microsoft</a></li>
  <li><a href="http://www.thinktecture.com">Thinktecture</a></li>
</ul>
```

Instead of using the <a> and <li> elements, you can also use LinkButton controls and a HTML table.

Exercise 4

Question: Define Forms authentication as the authentication mechanism for the Web site.

Answer: Forms authentication is defined by changing the authentication type with the ASP.NET configuration tool. For Forms authentication this configuration must be added to the file web.config:

```
    <system.web>
        <authentication mode="Forms" />
    </system.web>
```

Exercise 5

Question: Create a user control where a user can register to the Web site.

Answer: Select the template Web User Control from the Add New Item dialog. Add a Register.ascx control that includes the CreateUserWizard control.

Exercise 6

Question: Create a master page that defines a common layout with top and bottom sections.

Answer: Select the Master Page template with the Add New Item dialog. Add a table consisting of three rows and one column.

Exercise 7

Question: Define a sitemap to create links to all the Web pages with this Web site.

Answer: Select the Site map template with the Add New Item dialog. The file Web.sitemap must be changed to include all Web pages in their hierarchy:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="default.aspx" title="Home"  description="">
        <siteMapNode url="Demo.aspx" title="Demo"  description="" />
    </siteMapNode>
</siteMap>
```

With the top of the master page, you can now add a SiteMapDataSource control and a Menu control to display the site map.

Exercise 8

Question: Create a Web page that's using the Web Parts framework where the user can define the layout of the page.

Answer: Create a new Web page, and add a WebPartManager control. Add a table to define the layout of the page. Within every table cell where the user should add and configure WebParts, add a WebPartZone control. Add EditorZone and CatalogZone as appropriate.

For the user to allow changing the layout of Web Parts, the WebPartManager must be set to the display mode, e.g.:

```
WebPartManager1.DisplayMode = WebPartManager.DesignDisplayMode.
```

Chapter 20: Web Services

Exercise 1

Question: Create a new Web service named CinemaReservation.

Answer: Create a new Web service by selecting the Visual Studio 2005 menu File | New | Web Site…, and select the template ASP.NET Web Service.

Exercise 2

Question: The `ReserveSeatRequest` and `ReserveSeatResponse` classes are needed to define the data sent to and from the Web service. The `ReserveSeatRequest` class needs a member `Name` of type `string` to send a name and two members of type `int` to send a request for a seat defined with `Row` and `Seat`. The class `ReserveSeatResponse` defines the data to be sent back to the client—that is, the name for the reservation and the seat that is really reserved.

Answer: The classes should look similar to this code segment:

```
public class ReserveSeatRequest
{
    private string name;

    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    private int row;

    public int Row
    {
        get { return row; }
        set { row = value; }
    }

    private int seat;

    public int Seat
    {
        get { return seat; }
        set { seat = value; }
    }

}

public class ReserveSeatResponse
{
    private string reservationName;

    public string ReservationName
```

```
    {
        get { return reservationName; }
        set { reservationName = value; }
    }

    private int row;

    public int Row
    {
        get { return row; }
        set { row = value; }
    }

    private int seat;

    public int Seat
    {
        get { return seat; }
        set { seat = value; }
    }
}
```

Exercise 3

Question: Create a Web method `ReserveSeat` that requires a `ReserveSeatRequest` as a parameter and returns a `ReserveSeatResponse`. Within the implementation of the Web service you can use a `Cache` object (see Chapter 18) to remember the seats that already have been reserved. If the seat that is requested is available, return the seat and reserve it in the `Cache` object. If it is not available, take the next free seat. For the seats in the `Cache` object you can use a two-dimensional array, as was shown in Chapter 5.

Answer: For all the seats an array `reservedSeats` should be declared, so you can remember reserved seats.

```
    private const int maxRows = 12;
    private const int maxSeats = 16;
    private bool[,] reservedSeats = new bool[maxRows, maxSeats];
```

The implementation of the Web service method can look similar to the code shown. If the requested seat is free, the seat is reserved and returned from the Web service. If the seat is not free, the next free seat is returned.

```
   [WebMethod]
    public ReserveSeatResponse ReserveSeat(ReserveSeatRequest req) {
        ReserveSeatResponse resp = new ReserveSeatResponse();
        resp.ReservationName = req.Name;

        object o = HttpContext.Current.Cache["Cinema"];
        if (o == null)
        {
            // fill seats with data from the database or a file...
            HttpContext.Current.Cache["Cinema"] = reservedSeats;
```

```
        }
        else
        {
            reservedSeats = (bool[,])o;
        }
        if (reservedSeats[req.Row, req.Seat] == false)
        {
            reservedSeats[req.Row, req.Seat] = true;
            resp.Row = req.Row;
            resp.Seat = req.Seat;
        }
        else
        {
            int row;
            int seat;
            GetNextFreeSeat(out row, out seat);
            resp.Row = row;
            resp.Seat = seat;
        }
        return resp;
    }
```

Exercise 4

Question: Create a Windows client application that uses the Web service to reserve a seat in the cinema.

Answer: Create a new Windows application and add a reference to the Web service. The call to the Web service is shown below.

```
        private void OnRequestSeat(object sender, EventArgs e)
        {
            CinemaService.ReserveSeatRequest req =
               new CinemaClient.CinemaService.ReserveSeatRequest();
            req.Name = textName.Text;
            req.Seat = int.Parse(textSeat.Text);
            req.Row = int.Parse(textRow.Text);
            CinemaService.Service ws = new
CinemaClient.CinemaService.Service();
            CinemaService.ReserveSeatResponse resp =
ws.ReserveSeat(req);

            MessageBox.Show("Reserved seat " + resp.Row + " " +
resp.Seat);
        }
```

Chapter 21: Deploying Web Applications

Exercise 1

Question: What is the difference between copying and publishing a Web application? When should you use what?

Answer: Copying the Website copies all files required to run the Web application. Visual Studio 2005 has a dialog for a bi-directional copy. Newer files from the target server can be copied locally. If the source code should not be copied to the target Web server, precompiling allows creating assemblies, and then you can copy just the assemblies to the target Web server.

Exercise 2

Question: When is using a Setup program preferred compared copying a site?

Answer: Copying the site requires that the virtual directory on the target server is already created. With a setup program it is possible to create the virtual directory on IIS during setup. Also, if assemblies should be installed in the GAC, or COM objects are needed is not solved by copying files. Here it is required to create a setup program.

Exercise 3

Question: What must be done before a Website can be published?

Answer: A virtual directory must be created.

Exercise 4

Question: Publish the Web service from Chapter 20 to a virtual directory that you define with IIS.

Answer: First use the IIS Admin tool to create a virtual directory. Then use Visual Studio to copy the Web service to the server.

Chapter 22: File System Data

Exercise 1

Question: What is the namespace that must be imported to allow an application to work with files?

Answer: `System.IO`

Exercise 2

Question: When would you use a `FileStream` object to write to a file instead of using a `StreamWriter` object?

Answer: When you need random access to files, or when you are not dealing with string data.

Exercise 3

Question: What methods of the `StreamReader` class allow you to read data from files and what does each one do?

Answer:

1. `Peek()` – gets the value of the next character in the file, but does not advance the file position.
1. `Read()` – gets the value of the next character in the file and advances the file position.
1. `Read(char[] buffer, int index, int count)` – reads count characters into buffer, starting at `buffer[index]`.
1. `ReadLine()` – gets a line of text
1. `ReadToEnd()` – gets all text in a file

Exercise 4

Question: What class would you use to compress a stream using the Deflate algorithm?

Answer: `DeflateStream`

Exercise 5

Question: How would you prevent a class you have created from being serialized?

Answer: Ensure that it doesn't possess the `Serializable` attribute.

Exercise 6

Question: What events does the `FileSystemWatcher` class expose and what are they for?

Answer:

1. `Changed` – occurs when a file is modified
1. `Created` – occurs when a file in created
1. `Deleted` – occurs when a file is deleted
1. `Renamed` – occurs when a file is renamed

Exercise 7

Question: Modify the `FileWatch` application you built in this chapter. Add the ability to turn the file system monitoring on and off without exiting the application.

Answer: Add a button that toggles the value of the `watcher.EnableRaisingEvents` property.

Chapter 23: XML

Exercise 1

Question: Change the Insert example in the Creating Nodes Try It Out section to insert an attribute called `Pages` with the value `1000` on the book node.

Answer: In order to create an attribute on the Book element you use the Attributes collection of the XmlElement and the CreateAttribute method of the XmlDocument class.

In the `buttonCreateNode_Click` insert the following code:

```
    private void buttonCreateNode_Click(object sender, EventArgs e)
    {
      // Load the XML document
      XmlDocument document = new XmlDocument();
      document.Load(@"C:\Beginning Visual C#\Chapter 26\Books.xml");

      // Get the root element
      XmlElement root = document.DocumentElement;

      // Create the new nodes
      XmlElement newBook = document.CreateElement("book");
      XmlAttribute newPages = document.CreateAttribute("Pages");
      newPages.Value = "1000";
      XmlElement newTitle = document.CreateElement("title");
      XmlElement newAuthor = document.CreateElement("author");
      XmlElement newCode = document.CreateElement("code");
      XmlText title = document.CreateTextNode("Beginning Visual C# 3rd
Edition");
      XmlText author = document.CreateTextNode("Karli Watson et al");
      XmlText code = document.CreateTextNode("1234567890");
      XmlComment comment = document.CreateComment("This book is the book
you are reading");

      // Insert the elements
      newBook.AppendChild(comment);
      newBook.AppendChild(newTitle);
      newBook.AppendChild(newAuthor);
      newBook.AppendChild(newCode);
      newBook.Attributes.Append(newPages);
      newTitle.AppendChild(title);
      newAuthor.AppendChild(author);
      newCode.AppendChild(code);
      root.InsertAfter(newBook, root.FirstChild);

      document.Save(@"C:\Beginning Visual C#\Chapter 26\Books.xml");
    }
```

Exercise 2

Question: Currently, the last example about XPath doesn't have built-in capability to search for an attribute. Add a radio button and make a query for all books where the `pages` attribute of the book node equals `1000`.

Answer: Follow these steps to complete the exercise:

1. Insert a new RadioButton on the form and name it `radioButtonSearchAttribute`
2. Change the text property to "Select by attribute"
3. Add the `CheckedChanged` event to the code and insert the following code:

```
      XmlNodeList nodeList =
mCurrentNode.SelectNodes("//books/book[@pages=1000]");
      ClearListBox();
      DisplayList(nodeList);
```

How It Works: The XPath query "`//books/book[@pages=1000]`" selects all book elements that have an attribute named pages with the value 1000.

Exercise 3

Question: Change the methods that write to the listbox (`RecurseXmlDocument` and `RecurseXmlDocumentNoSiblings`) in the XPath example so that they're able to display attributes.

Answer: Change the code in the two methods as follows:

```
    private void RecurseXmlDocumentNoSiblings(XmlNode root, int indent)
    {
      // Make sure we don't do anything if the root is null
      if (root == null)
        return;

      if (root is XmlElement) // Root is an XmlElement type
      {
        if (root.Attributes.Count > 0)
        {
          string itemValue = root.Name.PadLeft(root.Name.Length +
indent) + " [";
          foreach (XmlAttribute attribute in root.Attributes)
          {
            itemValue += " " + attribute.Name + " = \"" +
attribute.Value.ToString() + "\"";
          }
          itemValue += "]";
          listBoxResult.Items.Add(itemValue);
        }
        else
        {
          // first, print the name
```

```
          listBoxResult.Items.Add(root.Name.PadLeft(root.Name.Length +
indent));
        }
        // Then check if there are any child nodes and if there are,
call this
        // method again to print them
        if (root.HasChildNodes)
          RecurseXmlDocument(root.FirstChild, indent + 2);
      }
      else if (root is XmlText)
      {
        // Print the text
        string text = ((XmlText)root).Value;
        listBoxResult.Items.Add(text.PadLeft(text.Length + indent));
      }
      else if (root is XmlComment)
      {
        // Print text
        string text = root.Value;
        listBoxResult.Items.Add(text.PadLeft(text.Length + indent));

        // Then check if there are any child nodes and if there are,
call this
        // method again to print them
        if (root.HasChildNodes)
          RecurseXmlDocument(root.FirstChild, indent + 2);
      }
    }

    private void RecurseXmlDocument(XmlNode root, int indent)
    {
      // Make sure we don't do anything if the root is null
      if (root == null)
        return;

      if (root is XmlElement) // Root is an XmlElement type
      {
        if (root.Attributes.Count > 0)
        {
          string itemValue = root.Name.PadLeft(root.Name.Length +
indent) + " [";
          foreach (XmlAttribute attribute in root.Attributes)
          {
            itemValue += " " + attribute.Name + " = \"" +
attribute.Value.ToString() + "\"";
          }
          itemValue += "]";
          listBoxResult.Items.Add(itemValue);
        }
        else
        {
          // first, print the name
          listBoxResult.Items.Add(root.Name.PadLeft(root.Name.Length +
indent));
```

```
        }
        // Then check if there are any child nodes and if there are,
call this
        // method again to print them
        if (root.HasChildNodes)
          RecurseXmlDocument(root.FirstChild, indent + 2);

        // Finally check to see if there are any siblings and if there
are
        // call this method again to have them printed
        if (root.NextSibling != null)
          RecurseXmlDocument(root.NextSibling, indent);
      }
      else if (root is XmlText)
      {
        // Print the text
        string text = ((XmlText)root).Value;
        listBoxResult.Items.Add(text.PadLeft(text.Length + indent));
      }
      else if (root is XmlComment)
      {
        // Print text
        string text = root.Value;
        listBoxResult.Items.Add(text.PadLeft(text.Length + indent));

        // Then check if there are any child nodes and if there are,
call this
        // method again to print them
        if (root.HasChildNodes)
          RecurseXmlDocument(root.FirstChild, indent + 2);

        // Finally check to see if there are any siblings and if there
are
        // call this method again to have them printed
        if (root.NextSibling != null)
          RecurseXmlDocument(root.NextSibling, indent);
      }
    }
```

How It Works: First the `Count` property of the `Attributes` collection on the root node is examined. If there are attributes on the node, a string containing the name of the current node. Next a `foreach` loop is used to add the names and values of all the attributes in the `Attributes` collection.

Chapter 24: Databases and ADO.NET

Exercise 1

Question: Modify the program given in the first sample to use the `Employees` table in the `Northwind` database. Retrieve the `EmployeeID` and `LastName` columns.

Answer:

```
// Chapter 24_Exercise_1
using System;
using System.Data;  // use ADO.NET namespace
using System.Data.SqlClient; // use namespace for SQL Server .NET Data
Provider

namespace DataReading
{
 class Program
 {
  static void Main(string[] args)
  {
   // Specify SQL Server-specific connection string
   SqlConnection thisConnection = new SqlConnection(
    @"Server=(local)\sqlexpress;Integrated Security=True;" +
    "Database=northwind");


   // Open connection
   thisConnection.Open();

   // Create command for this connection
   SqlCommand thisCommand = thisConnection.CreateCommand();

   // Specify SQL query for this command
   thisCommand.CommandText =
                "SELECT EmployeeID, LastName FROM Employees";

   // Execute DataReader for specified command
   SqlDataReader thisReader = thisCommand.ExecuteReader();

   // While there are rows to read
   while (thisReader.Read())
   {
    // Output ID and name columns
    Console.WriteLine("\t{0}\t{1}",
    thisReader["EmployeeID"], thisReader["LastName"]);
   }

   // Close reader
   thisReader.Close();

   // Close connection
   thisConnection.Close();

   Console.Write("Program finished, press Enter/Return to continue:");
   Console.ReadLine();

  }
 }
}
```

Exercise 2

Question: Modify the first program showing the `Update()` method to change the company name back to Bottom-Dollar Markets.

Answer:

```
// Chapter 24_Exercise_2
using System;
using System.Data;            // Use ADO.NET namespace
using System.Data.SqlClient;  // Use SQL Server data provider namespace

namespace UpdatingData
{
 class Program
 {
  static void Main(string[] args)
  {
   // Specify SQL Server-specific connection string
   SqlConnection thisConnection = new SqlConnection(
      @"Server=(local)\sqlexpress;Integrated Security=True;" +
     "Database=northwind");
   // Create DataAdapter object for update and other operations
   SqlDataAdapter thisAdapter = new SqlDataAdapter(
     "SELECT CustomerID, CompanyName FROM Customers", thisConnection);

   // Create CommandBuilder object to build SQL commands
   SqlCommandBuilder thisBuilder = new SqlCommandBuilder(thisAdapter);
   // Create DataSet to contain related data tables, rows, and columns
   DataSet thisDataSet = new DataSet();

   // Fill DataSet using query defined previously for DataAdapter
   thisAdapter.Fill(thisDataSet, "Customers");

   // Show data before change
   Console.WriteLine("name before change: {0}",
     thisDataSet.Tables["Customers"].Rows[9]["CompanyName"]);

   // Change data in Customers table, row 9, CompanyName column
            thisDataSet.Tables["Customers"].Rows[9]["CompanyName"]
                    = "Bottom-Dollar Markets";

   // Call Update command to mark change in table
   thisAdapter.Update(thisDataSet, "Customers");

   Console.WriteLine("name after change: {0}",
     thisDataSet.Tables["Customers"].Rows[9]["CompanyName"]);

   thisConnection.Close();
   Console.Write("Program finished, press Enter/Return to continue:");
   Console.ReadLine();
  }
 }
}
```

Exercise 3

Question: Write a program that asks the user for a Customer ID.

Answer:

```
using System;
// use ADO.NET namespace
using System.Data;
// use SQL .NET Data Provider
using System.Data.SqlClient;

class Chapter_24_Exercise_3
{
 public static void Main()
 {
  // Specify SQL Server-specific connection string
  SqlConnection thisConnection = new SqlConnection(
  @"Server=(local)\sqlexpress;Integrated Security=True;" +
   "Database=northwind");
  // open connection
  thisConnection.Open();
  // create DataAdapter object for update and other operations
   SqlDataAdapter thisAdapter = new  SqlDataAdapter(
   "SELECT CustomerID, CompanyName FROM Customers", thisConnection);
  // create CommandBuilder object to build SQL commands
   SqlCommandBuilder thisBuilder = new  SqlCommandBuilder(thisAdapter);
  // create DataSet to contain related data tables, rows, and columns
  DataSet thisDataSet = new DataSet();
  // fill DataSet using query defined previously for DataAdapter
  thisAdapter.Fill(thisDataSet, "Customers");


  Console.WriteLine("# rows before change: {0}",
   thisDataSet.Tables["Customers"].Rows.Count);

  // set up keys object for defining primary key
  DataColumn[] keys = new DataColumn[1];
  keys[0] = thisDataSet.Tables["Customers"].Columns["CustomerID"];
  thisDataSet.Tables["Customers"].PrimaryKey = keys;

  Console.WriteLine("Enter a 5-character Customer ID to Search for:");
  // More validation logic could be added to the following, but this is
the quickest solution
  String customerID = Console.ReadLine();

  DataRow findRow =
thisDataSet.Tables["Customers"].Rows.Find(customerID);

  if (findRow == null)
  {
   Console.WriteLine("Customer ID {0} not found", customerID);
```

```
  }
  else
  {
   Console.WriteLine("Customer ID {0} already present in database",
customerID);
  }

  thisConnection.Close();
 }
}
```

Exercise 4

Question: Write a program to create some orders for the customer "Zachary Zithers"; use the sample programs to view the orders.

Answer:

```
using System;
// use ADO.NET namespace
using System.Data;
// use SQL Server .NET Data Provider
using System.Data.SqlClient;

class Chapter_24_Exercise_4
{
 public static void Main()
 {
  // connect to local Northwind database with SQL Server Express
   SqlConnection thisConnection = new  SqlConnection(
   @"Server=(local)\sqlexpress;Integrated
Security=True;Database=northwind");
  // open connection
  thisConnection.Open();
  // create DataSet to contain related data tables, rows, and columns
  DataSet thisDataSet = new DataSet();
  // create DataAdapter object for customers
   SqlDataAdapter custAdapter = new  SqlDataAdapter(
   "SELECT CustomerID, CompanyName FROM Customers", thisConnection);
  // create CommandBuilder object to build SQL commands for customers
   SqlCommandBuilder custBuilder = new  SqlCommandBuilder(custAdapter);
  // fill DataSet using query defined previously for customers
  custAdapter.Fill(thisDataSet, "Customers");

  // create DataAdapter object for orders
   SqlDataAdapter orderAdapter = new  SqlDataAdapter(
   "SELECT OrderID, CustomerID, Freight FROM Orders", thisConnection);
  // create CommandBuilder object to build SQL commands for customers
   SqlCommandBuilder orderBuilder = new
SqlCommandBuilder(orderAdapter);
  // fill DataSet using query defined previously for customers
  orderAdapter.Fill(thisDataSet, "Orders");
```

```
  Console.WriteLine("# rows in Customers, Orders before change: {0},
{1}",
   thisDataSet.Tables["Customers"].Rows.Count,
   thisDataSet.Tables["Orders"].Rows.Count);

  // set up keys object for defining primary key for customers
  DataColumn[] cust_keys = new DataColumn[1];
  cust_keys[0] = thisDataSet.Tables["Customers"].Columns["CustomerID"];
  thisDataSet.Tables["Customers"].PrimaryKey = cust_keys;

  // set up keys object for defining primary key for orders
  DataColumn[] ord_keys = new DataColumn[1];
  ord_keys[0] = thisDataSet.Tables["Orders"].Columns["OrderID"];
  thisDataSet.Tables["Orders"].PrimaryKey = ord_keys;


  // find ZACZI in customer table
  if (thisDataSet.Tables["Customers"].Rows.Find("ZACZI") == null)
  {
   Console.WriteLine("ZACZI not found, will add to Customers table");

   DataRow custRow = thisDataSet.Tables["Customers"].NewRow();
   custRow["CustomerID"] = "ZACZI";
   custRow["CompanyName"] = "Zachary Zithers Ltd.";
   thisDataSet.Tables["Customers"].Rows.Add(custRow);
   Console.WriteLine("ZACZI successfully added to Customers table");

   // find 12000 in order table
   if (thisDataSet.Tables["Orders"].Rows.Find(12000) == null)
   {
     DataRow orderRow = thisDataSet.Tables["Orders"].NewRow();
     orderRow["OrderID"] = 12000;
     orderRow["CustomerID"] = "ZACZI";
     orderRow["Freight"] = 29.25;
     thisDataSet.Tables["Orders"].Rows.Add(orderRow);
     Console.WriteLine("Order # 12000 for Zachary Zithers successfully
added to Orders table");
   }
   // find 12001 in order table
   if (thisDataSet.Tables["Orders"].Rows.Find(12001) == null)
   {
     DataRow orderRow = thisDataSet.Tables["Orders"].NewRow();
     orderRow["OrderID"] = 12001;
     orderRow["CustomerID"] = "ZACZI";
     orderRow["Freight"] = 40.21;
     thisDataSet.Tables["Orders"].Rows.Add(orderRow);
     Console.WriteLine("Order # 12001 for Zachary Zithers successfully
added to Orders table");
   }

  }
  else
  {
   Console.WriteLine("ZACZI already present in database");
```

```
  }

  custAdapter.Update(thisDataSet, "Customers");
  orderAdapter.Update(thisDataSet, "Orders");
  Console.WriteLine("# rows in Customers, Orders after change: {0},
{1}",
   thisDataSet.Tables["Customers"].Rows.Count,
   thisDataSet.Tables["Orders"].Rows.Count);
  thisConnection.Close();
 }
}
```

Exercise 5

Question: Write a program to display a different part of the relationship hierarchies in the `Northwind` database than the one used in this chapter; for example, `Products`, `Suppliers`, and `Categories`.

Answer:

```
using System;
// use ADO.NET namespace
using System.Data;
// use SQL Server .NET Data Provider
using System.Data.SqlClient;

class Chapter24_Exercise_5
{
 public static void Main()
 {
  // connect to local Northwind database with SQL Server Express
   SqlConnection thisConnection = new  SqlConnection(
   @"Server=(local)\sqlexpress;Integrated
Security=True;Database=northwind");
  // open connection
  thisConnection.Open();

  DataSet thisDataSet = new DataSet();

   SqlDataAdapter prodAdapter = new  SqlDataAdapter(
   "SELECT * FROM Products", thisConnection);
  prodAdapter.Fill(thisDataSet, "Products");

   SqlDataAdapter suppAdapter = new  SqlDataAdapter(
   "SELECT * FROM Suppliers", thisConnection);
  suppAdapter.Fill(thisDataSet, "Suppliers");

   SqlDataAdapter catAdapter = new  SqlDataAdapter(
   "SELECT * FROM Categories", thisConnection);
  catAdapter.Fill(thisDataSet, "Categories");

 // Products is a child of Suppliers and Categories
  DataRelation prodSuppRel = thisDataSet.Relations.Add("ProdSupp",
   thisDataSet.Tables["Suppliers"].Columns["SupplierID"],
```

```
   thisDataSet.Tables["Products"].Columns["SupplierID"]
   );
  DataRelation prodCatRel = thisDataSet.Relations.Add("ProdCat",
   thisDataSet.Tables["Categories"].Columns["CategoryID"],
   thisDataSet.Tables["Products"].Columns["CategoryID"]
   );

  foreach (DataRow prodRow in thisDataSet.Tables["Products"].Rows)
  {
   Console.WriteLine("Product ID: " + prodRow["ProductID"]);
   Console.WriteLine("Product Name: " + prodRow["ProductName"]);

   DataRow suppRow = prodRow.GetParentRow(prodSuppRel);

   Console.WriteLine("\tSupplier ID: " + suppRow["SupplierID"]);
   Console.WriteLine("\tSupplier Name: " + suppRow["CompanyName"]);

   DataRow catRow = prodRow.GetParentRow(prodCatRel);

   Console.WriteLine("\tCategory ID: " + catRow["CategoryID"]);
   Console.WriteLine("\tCategory Name: " + catRow["CategoryName"]);
  }

  thisConnection.Close();
 }
}
```

Exercise 6

Question: Write out the data generated in the previous exercise as an XML document.

Answer:

```
using System;
// use ADO.NET namespace
using System.Data;
// use SQL Server .NET Data Provider
using System.Data.SqlClient;

class Chapter24_Exercise_6
{
 public static void Main()
 {
  // connect to local Northwind database with SQL Server Express
  SqlConnection thisConnection = new SqlConnection(
   @"Server=(local)\sqlexpress;Integrated
Security=True;Database=northwind");
  // open connection
  thisConnection.Open();

  DataSet thisDataSet = new DataSet();

  SqlDataAdapter prodAdapter = new SqlDataAdapter(
   "SELECT * FROM Products", thisConnection);
```

```
  prodAdapter.Fill(thisDataSet, "Products");

  SqlDataAdapter suppAdapter = new SqlDataAdapter(
   "SELECT * FROM Suppliers", thisConnection);
  suppAdapter.Fill(thisDataSet, "Suppliers");

  SqlDataAdapter catAdapter = new SqlDataAdapter(
   "SELECT * FROM Categories", thisConnection);
  catAdapter.Fill(thisDataSet, "Categories");

  // see diagram at beginning of "More Tables" section of Chapter 24;
  // Products is a child of Suppliers and Categories
  DataRelation prodSuppRel = thisDataSet.Relations.Add("ProdSupp",
   thisDataSet.Tables["Suppliers"].Columns["SupplierID"],
   thisDataSet.Tables["Products"].Columns["SupplierID"]
   );
  prodSuppRel.Nested = true;

  DataRelation prodCatRel = thisDataSet.Relations.Add("ProdCat",
   thisDataSet.Tables["Categories"].Columns["CategoryID"],
   thisDataSet.Tables["Products"].Columns["CategoryID"]
   );
  // only one of supplier/category can be nested as both are parents
  // prodCatRel.Nested=true;

  // write data to XML file)
  thisDataSet.WriteXml(@"c:\tmp\nwindprod.xml");

  thisConnection.Close();
 }
}
```

Exercise 7

Question: Change the program used to print all customer order details and products to use a `WHERE` clause in the `SELECT` statement for `Customers` limiting the customers processed.

Answer:

```
using System;
// use ADO.NET namespace
using System.Data;
// use SQL Server .NET Data Provider
using System.Data.SqlClient;

class Chapter24_Exercise_7
{
 public static void Main()
 {
  // connect to local Northwind database with SQL Server Express
  SqlConnection thisConnection = new SqlConnection(
   @"Server=(local)\sqlexpress;Integrated
Security=True;Database=northwind");
```

```
  // open connection
  thisConnection.Open();
  // create DataSet to contain related data tables, rows, and columns
  DataSet thisDataSet = new DataSet();
  SqlDataAdapter custAdapter = new SqlDataAdapter(
   "SELECT * FROM Customers WHERE CustomerID = 'ALFKI'",
thisConnection);
  custAdapter.Fill(thisDataSet, "Customers");
  SqlDataAdapter orderAdapter = new SqlDataAdapter(
   "SELECT * FROM Orders WHERE CustomerID = 'ALFKI'", thisConnection);
  orderAdapter.Fill(thisDataSet, "Orders");
  SqlDataAdapter detailAdapter = new SqlDataAdapter(
   "SELECT * FROM [Order Details] WHERE OrderID IN (SELECT OrderID FROM
Orders WHERE CustomerID = 'ALFKI')", thisConnection);
  detailAdapter.Fill(thisDataSet, "Order Details");

  SqlDataAdapter prodAdapter = new SqlDataAdapter(
   "SELECT * FROM Products", thisConnection);
  prodAdapter.Fill(thisDataSet, "Products");

  DataRelation custOrderRel = thisDataSet.Relations.Add("CustOrders",
   thisDataSet.Tables["Customers"].Columns["CustomerID"],
   thisDataSet.Tables["Orders"].Columns["CustomerID"]);

  DataRelation orderDetailRel = thisDataSet.Relations.Add("OrderDetail",
   thisDataSet.Tables["Orders"].Columns["OrderID"],
   thisDataSet.Tables["Order Details"].Columns["OrderID"]);

  DataRelation orderProductRel =
thisDataSet.Relations.Add("OrderProducts",
   thisDataSet.Tables["Products"].Columns["ProductID"],
   thisDataSet.Tables["Order Details"].Columns["ProductID"]);

  foreach (DataRow custRow in thisDataSet.Tables["Customers"].Rows)
  {
   Console.WriteLine("Customer ID: " + custRow["CustomerID"]);
   foreach (DataRow orderRow in custRow.GetChildRows(custOrderRel))
   {
    Console.WriteLine("\tOrder ID: " + orderRow["OrderID"]);
    Console.WriteLine("\t\tOrder Date: " + orderRow["OrderDate"]);
    foreach (DataRow detailRow in orderRow.GetChildRows(orderDetailRel))
    {
     Console.WriteLine("\t\tProduct: " +
      detailRow.GetParentRow(orderProductRel)["ProductName"]);
     Console.WriteLine("\t\tQuantity: " + detailRow["Quantity"]);
    }
   }
  }

  thisConnection.Close();
 }
}
```

Exercise 8

Question: Modify the program shown to print out `UPDATE`, `INSERT`, and `DELETE` statements to use `"SELECT * FROM Customers"` as the SQL `SELECT` command. Note the complexity of the generated statements.

Answer:

```
using System;
using System.Data;
using System.Data.SqlClient;

class Chapter24_Exercise_8
{
 public static void Main()
 {
  SqlConnection thisConnection = new SqlConnection(
   @"Server=(local)\sqlexpress;Integrated
Security=True;Database=northwind");
  thisConnection.Open();

  SqlDataAdapter thisAdapter = new
   SqlDataAdapter("SELECT * from Customers", thisConnection);

  SqlCommandBuilder thisBuilder = new SqlCommandBuilder(thisAdapter);

  Console.WriteLine("SQL SELECT Command is:\n{0}\n",
   thisAdapter.SelectCommand.CommandText);

  SqlCommand updateCommand = thisBuilder.GetUpdateCommand();
  Console.WriteLine("SQL UPDATE Command is:\n{0}\n",
   updateCommand.CommandText);

  SqlCommand insertCommand = thisBuilder.GetInsertCommand();
  Console.WriteLine("SQL INSERT Command is:\n{0}\n",
   insertCommand.CommandText);

  SqlCommand deleteCommand = thisBuilder.GetDeleteCommand();
  Console.WriteLine("SQL DELETE Command is:\n{0}",
   deleteCommand.CommandText);

  thisConnection.Close();
 }
}
```

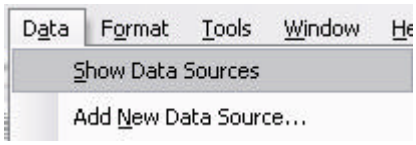
Chapter 25: Data Binding

Exercise 1

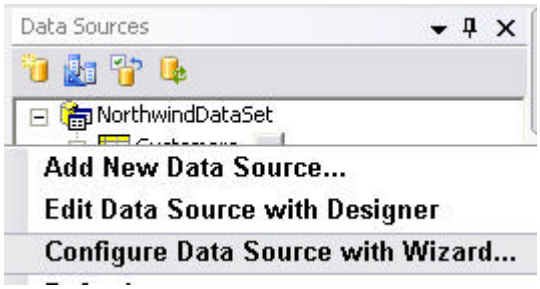
Question: Modify the first example by adding the column `ContactTitle` to the data source `NorthwindDataSet`.

Answer:

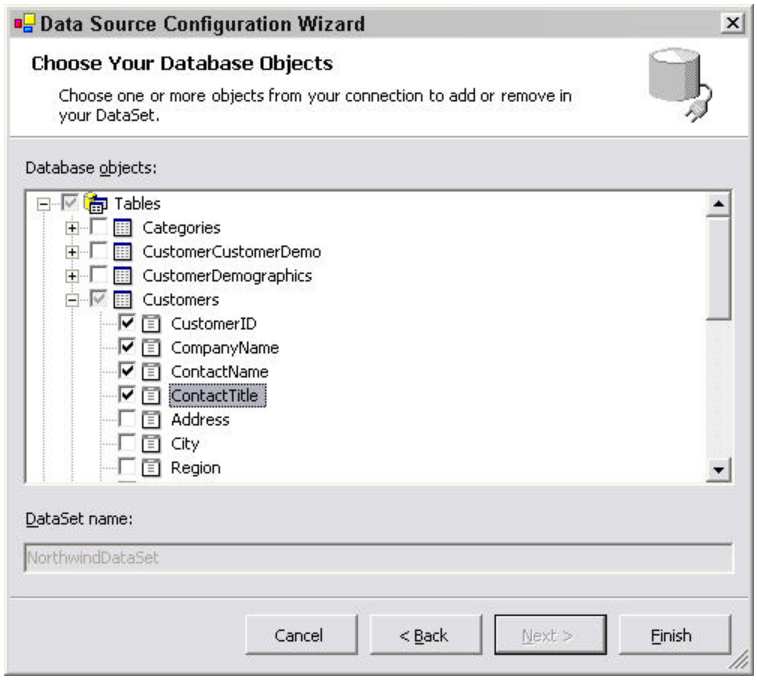
1. Open the GettingData project – Select Data ? Show Data Sources:



2. In the Data Sources window, right-click on `NorthwindDataSet` then select Configure Data Source with Wizard:



3. In the Choose Database Objects dialog of the wizard, expand Tables then Customers, and check the box beside ContactTitle:



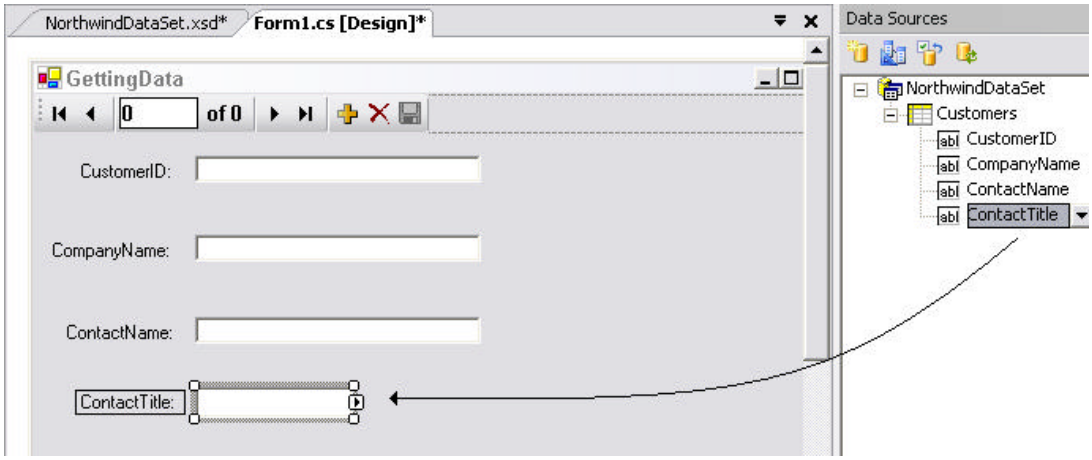
4. Press Finish to complete the operation.

Exercise 2

Question: Add a text box to the form that will display the `ContactTitle` information.

Answer:

1. Assuming you have completed Exercise 1, simply drag the ContactTitle field from the DataSource window on the Form1.cs design window:



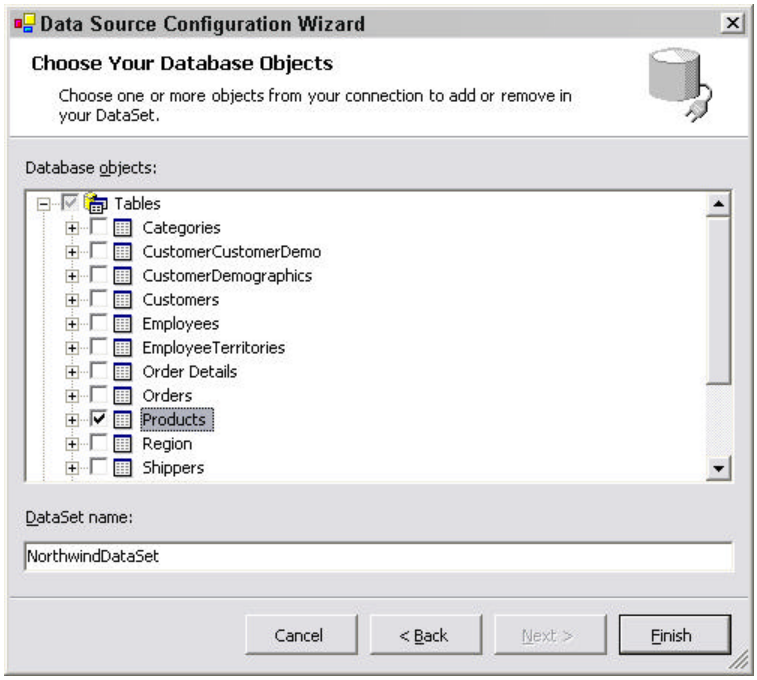
2. Press F5 to build the project and run in the debugger. The contact title field will automatically be filled.

Exercise 3

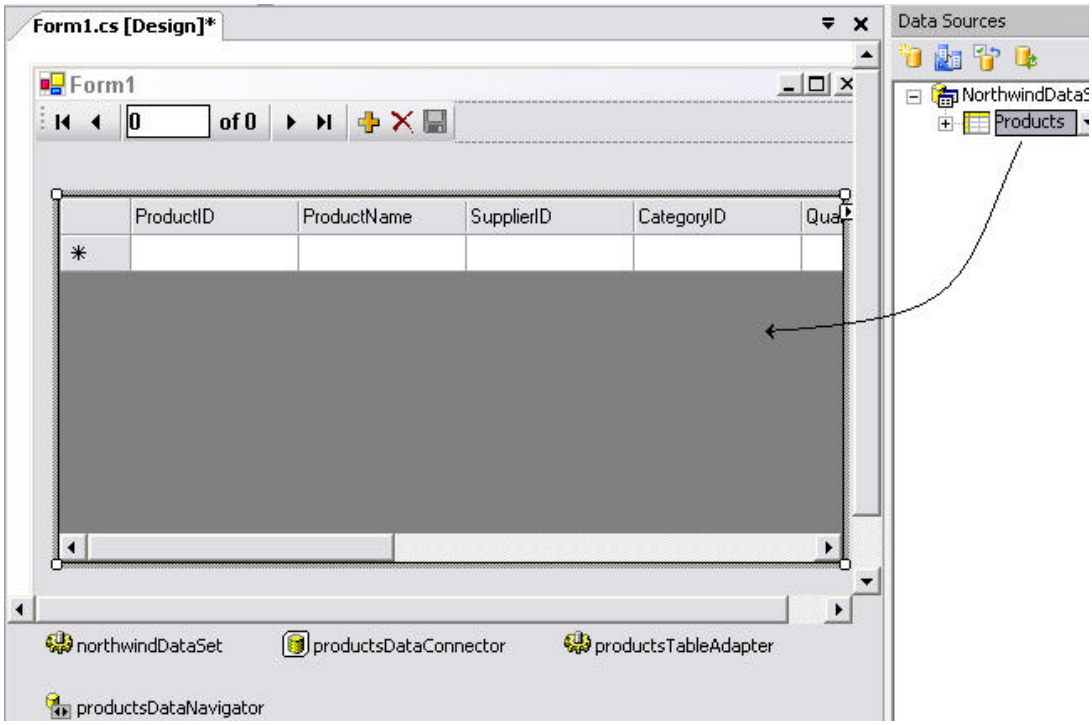
Question: Create a new application that displays data from the `Products` table of the `Northwind` database.

Answer:

1. Create a new Windows Application project called GettingProducts.
2. Select Data ? Add New Data Source
3. In the Data Source Configuration Wizard, press Next.
4. In the Choose a Data Source Type screen, choose Database.
5. In the Choose Your Data Connection screen, select the existing Northwind connection (created with the GettingData project).
6. In the Choose Database Objects screen, select the Products table, then press Finish:



7. Select individual fields from the Products table in the Northwind data source just created, or simply drag the Products table onto the form to create a DataGridView:



8. Press F5 to build and run in the debugger. The DataGridView is fully functional for data display and navigation.

Exercise 4

Question: Modify the code for the *Updating the Database* example to handle changes to all the data fields, not just the `ContactName`.

Answer:

NOTE: The `dataNavigator()`class was replaced with the `bindingNavigator()`class in the final release of VS2005.

1. Open the project. In the design window for `Form1.cs`, double-click on the text box fields for `CustomerID` and `CompanyName` to create empty event handler routines in `Form1.cs` (`companyNameTextBox_TextChanged` and `companyNameTextBox_TextChanged`).
2. Add the line to enable the Save Item button to both of these routines:

```
private void contactNameTextBox_TextChanged(object sender, EventArgs
e)
{
      this.bindingNavigatorSaveItem.Enabled = true;
```

```
}

private void companyNameTextBox_TextChanged(object sender, EventArgs
e)
{
      this.bindingNavigatorSaveItem.Enabled = true;
}

private void customerIDTextBox_TextChanged(object sender, EventArgs e)
{
      this.bindingNavigatorSaveItem.Enabled = true;
}

private void bindingNavigatorSaveItem_Click(object sender, EventArgs
e)
{
      int rowsUpdated =
this.customersTableAdapter.Update(northwindDataSet);
      MessageBox.Show(rowsUpdated.ToString() +
               ((rowsUpdated == 1) ? " row" : " rows") + " updated.");
            }
```

3. Press F5 to build and run in the debugger. The previous code calling `Update()` in `bindingNavigatorSaveItem_Click()` handles the update. That's all there is to it!

Chapter 26: .NET Assemblies

Exercise 1

Question: Change the version number and other assembly attributes in `AssemblyInfo.cs` or `Shapes.cs`, and view the resulting changes in `Shapes.dll` using `Ildasm`.

Answer: Make these changes to `AssemblyInfo.cs`, then view in `ildasm`.

```
using System.Reflection;
using System.Runtime.CompilerServices;

//
// General Information about an assembly is controlled through the
following
// set of attributes. Change these attribute values to modify the
information
// associated with an assembly.
//
[assembly: AssemblyTitle("Shapes")]
[assembly: AssemblyDescription("Shapes example")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("")]
[assembly: AssemblyCopyright("")]
[assembly: AssemblyTrademark("")]
```

```
[assembly: AssemblyCulture("")]

//
// Version information for an assembly consists of the following four
values:
//
// . . .
[assembly: AssemblyVersion("1.0.0.1")]
```

Exercise 2

Question: Make a new `MoreShapes` assembly with a `Square` class in addition to `Circle` and `Triangle`, and then examine its properties with `Ildasm`.

Answer: Make a new `MoreShapes` project with the following source in the `moreshapes.cs` file; compile and view `moreshapes.dll` in `ildasm`.

```
namespace MoreShapes
{
   public class Square
   {
      double SideLength;

      public Square()
      {
          SideLength = 0;
      }

      public Square(double givenSideLength)
      {
         SideLength = givenSideLength;
      }

      public double Area()
      {
         return SideLength * SideLength;  // area = sidelength squared
      }
   }
  public class Circle
   {
      double Radius;

      public Circle()
      {
          Radius = 0;
      }

      public Circle(double givenRadius)
      {
         Radius = givenRadius;
      }

      public double Area()
      {
```

```
            return System.Math.PI * (Radius * Radius);  // area = pi r
squared
         }
      }

      public class Triangle
      {
         double Base;      // capital B to avoid conflict with keyword
base
         double Height;

         public Triangle()
         {
            Base = 0;
            Height = 0;
         }

         public Triangle(double givenBase, double givenHeight)
           {
              Base = givenBase;
              Height = givenHeight;
           }

         public double Area()
         {
            return 0.5F * Base * Height;  // area = 1/2 base * height
         }
      }
   }
```

Exercise 3

Question: Change the assembly reference in `ShapeUser.exe` to use `MoreShapes.dll`, and then view the changed properties in `Ildasm`.

Answer: No source change is associated with this exercise. Open the `shapeuser` project, then go to Project | Add Reference. Delete the reference to `shapes.dll`, then click on the Browse button and browse to your `moreshapes\bin\debug` directory to add a reference to `moreshapes.dll`.

Exercise 4

Question: Make your own client for `MoreShapes` that uses the `Square` and `Triangle` objects as well as `Circle`. Examine the assembly for this client with `Ildasm`.

Answer: Create a project `MyShapeUser` and enter the following as the source for `MyShapeUser.cs`. Build the project, then view `myshapeuser.exe` in `ildasm`.

```
   namespace MyShapeUser
   {
      using System;
      using Shapes;

      public class MyShapeUser
```

```
    {
       public static void Main()
       {
          Circle c = new Circle(2.0);

          Console.WriteLine("Area of Circle(2.0) is {0}", c.Area());

             Triangle t = new Triangle(2.0, 2.0);

          Console.WriteLine("Area of Triangle(2.0, 2.0) is {0}",
t.Area());

             Square s = new Square(2.0);

          Console.WriteLine("Area of Square(2.0) is {0}", s.Area());
          Console.ReadKey();
       }
    }
}
```

Exercise 5

Question: Display the command options for `Gacutil` with the `/?` flag, then use its options to list the properties of all the global assemblies on your system.

Answer: Use `gacutil` at the Visual Studio.2005 Command Prompt as shown:

```
C:\>gacutil

Microsoft (R) .NET Global Assembly Cache Utility.  Version 2.0.40607.16
Copyright (C) Microsoft Corporation. All rights reserved.

Usage: Gacutil <command> [ <options> ]
Commands:
  /i <assembly_path> [ /r <...> ] [ /f ]
    Installs an assembly to the global assembly cache.

  /il <assembly_path_list_file> [ /r <...> ] [ /f ]
    Installs one or more assemblies to the global assembly cache.

  /u <assembly_display_name> [ /r <...> ]
    Uninstalls an assembly from the global assembly cache.

  /ul <assembly_display_name_list_file> [ /r <...> ]
    Uninstalls one or more assemblies from the global assembly cache.

  /l [ <assembly_name> ]
    List the global assembly cache filtered by <assembly_name>

  /lr [ <assembly_name> ]
    List the global assembly cache with all traced references.

  /cdl
    Deletes the contents of the download cache
```

```
  /ldl
    Lists the contents of the download cache

  /?
    Displays a detailed help screen

 Options:
  /r <reference_scheme> <reference_id> <description>
    Specifies a traced reference to install (/i, /il) or uninstall (/u, /ul).

  /f
    Forces reinstall of an assembly.

  /nologo
    Suppresses display of the logo banner

  /silent
    Suppresses display of all output
```

```
C:\>gacutil /L
Microsoft (R) .NET Global Assembly Cache Utility.  Version 2.0.40607.16
Copyright (C) Microsoft Corporation. All rights reserved.

The Global Assembly Cache contains the following assemblies:
  CustomMarshalers, Version=2.0.3600.0, Culture=neutral,
PublicKeyToken=b03f5f7f
11d50a3a, processorArchitecture=x86
  ISymWrapper, Version=2.0.3600.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50
a3a, processorArchitecture=x86
  MFCMIFC80, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a,
processorArchitecture=x86
  Microsoft.SqlServer.BatchParser, Version=9.0.242.0, Culture=neutral,
PublicKey
Token=89845dcd8080cc91, processorArchitecture=x86
  Microsoft.SqlServer.Replication, Version=9.0.242.0, Culture=neutral, PublicKey
Token=89845dcd8080cc91, processorArchitecture=x86
  Microsoft.VisualC, Version=8.0.1200.0, Culture=neutral,
PublicKeyToken=b03f5f7
f11d50a3a, processorArchitecture=x86
  Microsoft.VisualC.VSCodeParser, Version=8.0.1200.0, Culture=neutral,
PublicKey
Token=b03f5f7f11d50a3a, processorArchitecture=x86
  mscorlib, Version=2.0.3600.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089
, ProcessorArchitecture=x86
  System.Data, Version=2.0.3600.0, Culture=neutral,
PublicKeyToken=b77a5c561934e
```

```
089, processorArchitecture=x86
  System.Data.OracleClient, Version=2.0.3600.0, Culture=neutral,
PublicKeyToken=
b77a5c561934e089, processorArchitecture=x86
      . . .

Number of items = 332
```

Exercise 6

Question: Create a strong name for `MoreShapes` and sign the assembly. View the results with `Ildasm`. Try executing your client program, then recompile to reference the signed assembly.

Answer: Start the Visual Studio.NET Command Line in the Visual Studio.NET Tools menu. Go to the `Moreshapes\obj\debug` directory. Use `sn.exe` as follows to make a strong name:

```
C:\Moreshapes\obj\debug > sn -k moreshapes.snk

Microsoft (R) .NET Framework Strong Name Utility  Version 2.0.40607.16
Copyright (C) Microsoft Corporation. All rights reserved.

Key pair written to moreshapes.snk
```

In the `AssemblyInfo.cs` file for the `moreshapes` project, make the following change:

```
[assembly: AssemblyKeyFile("moreshapes.snk")]
```

Build the project. Replace the `moreshapes.dll` in the `MyShapesUser` project directory with the new signed version of `moreshapes.dll`. Try to run the `MyShapesUser.exe` program that used `moreshapes.dll`. Note the error message because the private assembly reference is trying to use a signed shared assembly. Now remove the signed `moreshapes.dll` assembly, redo the reference in `MyShapesUser`, and try again.

Exercise 7

Question: Install `moreshapes.dll` in the global assembly cache using `gacutil`, and experiment with `shapeuser` and your own client by running with and without the `moreshapes` assembly being present in the local directory and/or global cache.

There is no new source code to create for this exercise; go to the `Moreshapes\bin\debug` directory. run `gacutil` with the `/I` option as follows:

```
E:\jon\wrox\shapes\bin\Debug>gacutil /i moreshapes.dll

Microsoft (R) .NET Global Assembly Cache Utility.  Version
2.0.40607.16
Copyright (C) Microsoft Corp.  All rights reserved.
```

```
   Assembly successfully added to the cache
```

Run `shapeuser.exe` and `MyShapeUser.exe` as instructed. The remove `moreshapes` from the cache with

```
   gacutil /u moreshapes
```

and experiment with running `shapeuser.exe` and `MyShapeUser.exe` again.

Chapter 27: Attributes

No Exercises.

Chapter 28: XML Documentation

Exercise 1

Question: Which of the following elements can you use XML documentation:

a) `<summary>`

b) `<usage>`

c) `<member>`

d) `<seealso>`

e) `<typeparamref>`

Answer: Technically, you can use whatever elements you like—the vocabulary is extensible to meet your own needs. However, only **a**, **c**, and **d** are among the recommended attributes for XML documentation.

Exercise 2

Question: Which top-level XML documentation tags would you use to document a property?

Answer: You would use `<summary>` and `<value>`, and perhaps `<remarks>`, `<permission>`, and `<seealso>` for additional information, or `<include>` if the XML was external.

Exercise 3

Question: XML documentation is contained in C# source files. True or false?

Answer: False—since you can use `<include>` to reference external XML.

Exercise 4

Question: What is the major disadvantage of adding XML documentation using a class diagram?

Answer: Markup is escaped.

Exercise 5

Question: What do you need to do to ensure that other projects can make use of XML documentation in your class libraries?

Answer: Compile with the "XML Documentation File" build option, and ensure that VS can find the XML documentation file relating to your assembly, either by leaving it in the same directory as the assembly, or placing it in a well known location such as the GAC.

Chapter 29: Networking

Exercise 1

Question: Extend the FTP client application that makes use of the `FtpWebRequest` and `FtpWebResponse` classes to not only download files from the FTP server, but also to allow uploading files to the server. Add one more button to the form with the Text property set to Upload File, use the `OpenFileDialog` to ask the user for a file to upload, and then send a request to the server. For uploading files the `WebRequestMethods` class offers the `Ftp.UploadFile` member.

Answer: Uploading files to the FTP server is very similar to downloading files as you've already done in the method `OnDownloadFile()`. The differences are that the FTP method `Ftp.UploadFile` must be set and the file stream must be passed with the FTP request. You can see this code part following `Stream requestStream = request.GetRequestStream()`. With the answer from the server, the response stream is not needed, instead the status information can be shown with `response.StatusDescription`.

```
                if (openFileDialog.ShowDialog() == DialogResult.OK)
                {
                    FtpWebResponse response = null;
                    Stream inStream = null;
                    Stream outStream = null;

                    Uri baseUri = new Uri(textServer.Text);

                    string filename =
listFiles.SelectedValue.ToString().Trim();
                    string fullFilename = serverDirectory + @"/" +
filename;

                    Uri uri = new Uri(baseUri, fullFilename);

                    FtpWebRequest request =
                       (FtpWebRequest)WebRequest.Create(uri);
                    request.Credentials =
                          new NetworkCredential(textUsername.Text,
                          textPassword.Text);
```

```
                    request.Method = WebRequestMethods.Ftp.UploadFile;
                    request.UseBinary = checkBoxBinary.Checked;

                    Stream requestStream = request.GetRequestStream();
                    FileStream fs = File.Open(openFileDialog.FileName,
                          FileMode.Open);
                    byte[] buffer = new byte[8192];
                    int i;
                    while ((i = fs.Read(buffer, 0, buffer.Length)) > 0)
                    {
                        requestStream.Write(buffer, 0, i);
                    }
                    requestStream.Close();

                    response = (FtpWebResponse)request.GetResponse();

                    MessageBox.Show(response.StatusDescription);
                }
```

Exercise 2

Question: Modify the Picture server and client applications that use the `TcpListener` and `TcpClient` classes, so that it is possible to upload picture files from the client to the server.

Answer: With the TcpListener and TcpClient applications that have been created in the chapter, a file stream was sent from the server to the client. Now it is the other way around.

The client application needs an upload button where the `Click` event handler opens an `OpenFileDialog` dialog to ask the user about what file to open.

```
      private void OnUploadFile(object sender, EventArgs e)
        {
            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {
```

Similar to the `OnDownloadPicture` event handler, a connection must be opened to the server. Sending a file to the server, a new request command must be defined. Here the request command `UPLOAD` is used, where the filename and file content follow. The file is read with help of `File` and `FileStream` classes. The data that is read is written to the network stream to send it to the server.

```
                const int bufferSize = 8192;

                TcpClient client = new TcpClient();
                IPHostEntry host =
Dns.GetHostEntry(Properties.Settings.Default.Server);
                client.Connect(host.AddressList[0],
Properties.Settings.Default.ServerPort);

                NetworkStream clientStream = client.GetStream();
```

```
                FileInfo file = new FileInfo(openFileDialog.FileName);

                string request = "UPLOAD:" + file.Name + ":";
                byte[] requestBuffer = Encoding.ASCII.GetBytes(request);
                clientStream.Write(requestBuffer, 0,
requestBuffer.Length);

                FileStream fileStream = file.OpenRead();

                byte[] buffer = new byte[bufferSize];
                int bytesRead = 0;
                do
                {
                    bytesRead = fileStream.Read(buffer, 0, bufferSize);
                    clientStream.Write(buffer, 0, bytesRead);

                } while (bytesRead > 0);
                fileStream.Close();

                clientStream.Close();
                client.Close();
            }
        }
```

The server application must be modified to accept larger streams from the client. Instead of reading the data just once, a do/while loop is added to read the data into a `MemoryStream` object.

```
   static void Main(string[] args)
   {
      System.Net.Sockets.TcpListener listener =
            new System.Net.Sockets.TcpListener(IPAddress.Any,
            Properties.Settings.Default.Port);
      listener.Start();

      while (true)
      {
         const int bufferSize = 512;

         TcpClient client = listener.AcceptTcpClient();
         NetworkStream clientStream = client.GetStream();
         MemoryStream memStream = new MemoryStream();

         byte[] buffer = new byte[bufferSize];
         int readBytes = 0;
         int offset = 0;
         do
         {
            readBytes = clientStream.Read(buffer, 0, bufferSize);
            memStream.Write(buffer, offset, readBytes);
            offset += readBytes;
         } while (readBytes > 0);
```

With the `if/else` sequence, a check for the `UPLOAD` command must be added. As defined with the request that is sent from the client, the filename follows the `UPLOAD` command. After the filename, the byte stream contains the content of the file. After separating the content of the request, the file is written to a directory that is defined with the application settings `UploadDirectory`.

```
            else if (request.StartsWith("UPLOAD"))
            {
                // get the filename
                string[] requestMessage = request.Split(':');
                string filename = requestMessage[1];

                // get index of second ':'
                int startStreamPosition = request.IndexOf(':', 7);
                string uploadDirectory =
                      Properties.Settings.Default.UploadDirectory;
                FileStream stream =
File.OpenWrite(Path.Combine(uploadDirectory,
                      filename));
                byte[] data = memStream.ToArray();
                stream.Write(data, startStreamPosition + 1, data.Length
-
                      startStreamPosition - 1);
                stream.Close();

            }
```

Chapter 30: Introduction to GDI+

Exercise 1

Question: Write a small application that attempts to inappropriately dispose of a `Pen` object that was obtained from the `Pens` class. Note the error message that result.

Answer: See source code file 578472_Ch30_DisposalErrors.zip

Exercise 2

Question: Create an application that displays a three-color traffic light. Have the traffic light change color when the user clicks on it.

Answer: See source code file 578472_ch30_TrafficLight.zip

Exercise 3

Question: Write an application that inputs RGB values. Display the color in a rectangle. Display the HSB values of the color.

Answer: See source code file 578472_Ch30_ColorConverter.zip