

# Corrections for *Murach's C#*

These are the corrections for the significant errors in each printing of this book. In addition to the corrections listed here, you may find some trivial typos and formatting errors. All types of corrections will be made in the next printing of the book.

## **How to tell which printing your book is in**

Below the copyright notation on the back of the title page (page ii), you'll find a series of numbers like this:

10 9 8 7 6 5 4 3 2

The number on the right of this sequence tells which printing your book is. In this example, it's the second printing, which is the current printing for this book (September 2005).

## Corrections to the second printing

### Chapter 4, page 94

The third paragraph on this page describes the division and modulus operators incorrectly. It should say that the division operator returns an integer value that represents the number of times the right operand goes into the left operand, not the number of times the left operand goes into the right operand. Similarly, the modulus operator returns an integer value that represents the remainder after dividing the left operand by the right operand, not the right operand by the left operand.

### Chapter 4, page 113

The last bullet should indicate that you can use the Format method of the String class to format two or more values, not up to three values.

### Chapter 4, page 117

The syntax for declaring an enumeration is incorrect. The bracket after ConstantName2 should be moved after [= value] like this:

```
enum EnumerationName [: type]
{
    ConstantName1 [= value][,
    ConstantName2 [= value]]...
}
```

### Chapter 7, page 185

In the third example on this page, the code in the try block attempts to convert the value in a text box named txtSubtotal to a decimal. However, if the conversion causes an error, the code in the catch block moves the focus to a text box named textBox. The name of this text box should be the same as the name of the text box in the try block.

### Chapter 8, page 223

In the first example on this page, the GetRateArray method creates an array of decimals that can contain one less element than the value of the elementCount argument that's passed to the method. This is incorrect. It should create an array with the exact number of elements specified by the elementCount argument like this:

```
decimal[] rates = new decimal[elementCount];
```

### Chapter 8, page 229

In the foreach statement in the last example on this page, the ToString method is used to convert the Value property of the EmployeeSalesEntry variable to a string. Because this value is concatenated with other strings, however, it isn't necessary to use the ToString method.

### Chapter 10, page 274

In the first paragraph, the last sentence should begin, "Then, in chapter 23" instead of chapter 24.

### Chapter 22, page 692

The third and fourth paragraphs on this page refer to the foreach loop in the code example on page 693. However, this example uses a for loop, not a foreach loop.

**Appendix A, pages 732 and 733**

The third paragraph on page 732 and the bullet under the heading “How to prepare your system for doing the exercises” on page 733 indicate that you should copy the directory that contains the files and directories for the exercises to the root directory of your C drive before you use them. However, we have modified the file that you download from our web site so that when you install the download, this copy is done for you by default. Unless you change the default, then, you won’t need to do this copy.

## Corrections to the first printing

### Chapter 5, page 143

The code example in this figure that uses a continue statement within a for loop doesn't produce the indicated results. To produce these results, you can code the loop like this:

```
string numbers = null;
for (int i = 1; i < 6; i++)
{
    numbers += i;
    if (i < 4)
    {
        numbers += "\n";
        continue;
    }
    numbers += " Big\n";
}
```

However, this can be coded more easily with an if/else statement:

```
string numbers = null;
for (int i = 1; i < 6; i++)
{
    numbers += i;
    if (i < 4)
        numbers += "\n";
    else
        numbers += " Big\n";
}
```

A better illustration of the continue statement would be this code:

```
string numbers = null;
for (int i = 1; i < 6; i++)
{
    numbers += i;
    numbers += "\n";
    if (i < 4)
        continue;
    numbers += "Big\n";
}
```

which produces this output:

```
1
2
3
4
Big
5
Big
```

## **Chapter 10, pages 290 through 297**

If invalid data is entered on the Payment form described on these pages, control returns to the Customer form after the error message is displayed. To correct this problem, the DialogResult property of the OK button should not be set to OK as shown on page 292. Instead, it should be left at its default of None. Then, this statement should be added to the end of the Click event procedure for the OK button shown on page 296 if the data is valid:

```
    this.DialogResult = DialogResult.OK
```

That way, when the user clicks the OK button, control is returned to the Customer form only if the data on the Payment form is valid.

## **Chapter 15, page 441**

The first statement in the Clone method of the Product class shown on this page is incorrect. It should be

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