بسمه تعالی

برنامه سازی پیشرفته

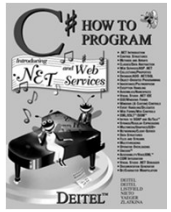
علی چوداری خسروشاهی
Akhosroshahi@iaut.ac.ir

دانشگاه آزاد اسلامی

1

مراجع

- C# How to Program
  - PAUL DEITEL
  - HARVEY DEITEL



2

فصل هشتم

برنامه نویسی مبتنی بر شئ

3

8.1 مقدمه

- کلاس ها داده ها و توابع را بسته بندی (بسته بندی با هم) می کنند.
  - encapsulation
- اشیاء می توانند پیاده سازی خود را از سایر اشیاء پنهان کنند (مخفی کردن اطلاعات)
- توابع (Methods): واحدهای برنامه نویسی
- نوع تعریف شده توسط کاربر: کلاس نوشته شده توسط برنامه نویس
- کلاسها دارند
  - اعضاء داده ای (متغیرهای عضو)
  - توابعی که اعضاء داده ای را دستکاری می کنند

4

8.2 پیاده سازی یک نوع داده انتزاعی Time با یک کلاس

- نوع داده انتزاعی – پیاده سازی را از سایر اشیاء مخفی می کند.
  - Abstract Data Type
- شروع محدوده بدنه کلاس آکولاد چپ ({) و پایان آن آکولاد سمت راست (}) می باشد.
- متغیرهای درون کلاس متغیر های عضو نامیده می شوند
- نحوه دسترسی به اعضاء
  - *public*: عضو هر کجا که یک نمونه از شئ وجود داشته باشد قابل دسترسی است
  - *Private*: عضو تنها در داخل تعریف کلاس در دسترس است

5

8.2 پیاده سازی یک نوع داده انتزاعی Time با یک کلاس

- کلاس غالباً شامل توابع دسترسی است
  - Access methods
  - خواندن و یا نمایش داده ها
- کلاس غالباً شامل توابع تست و بررسی شرایط می باشد
  - Predicate methods
  - تست صادق بودن شرایط
  - مثال IsEmpty
- سازنده (Constructor)
  - مقدار دهی اولیه اشیاء کلاس
  - می تواند آرگومان داشته باشد
  - نمی تواند مقداری برگشت دهد
  - می تواند برای یک کلاس بیش از یک سازنده داشته باشید(overloaded constructors)
- عملگر new برای ایجاد نمونه از کلاس استفاده می شود
- **Project < Add Class** را می توان برای افزودن کلاس جدید به پروژه استفاده کرد.

6

7

```
1    // Fig. 8.1: Time1.cs
2    // Class Time1 maintains time in 24-hour format.
3
4    using System;
5
6    // Time1 class definition
7    public class Time1 : Object
8    {
9       private int hour;    // 0-23
10      private int minute;  // 0-59
11      private int second;  // 0-59
12
13      // Time1 constructor initializes instance variables to
14      // zero to set default time to midnight
15      public Time1()
16      {
17         SetTime( 0, 0, 0 );
18      }
19
20      // Set new time value in 24-hour format. Perform validity
21      // checks on the data. Set invalid values to zero.
22      public void SetTime(
23         int hourValue, int minuteValue, int secondValue )
24      {
25         hour = ( hourValue >= 0 && hourValue < 24 ) ?
26            hourValue : 0;
27         minute = ( minuteValue >= 0 && minuteValue < 60 ) ?
28            minuteValue : 0;
29         second = ( secondValue >= 0 && secondValue < 60 ) ?
30            secondValue : 0;
31      }
32
```

Default constructor

nstance variables

Method SetTime

Validate arguments

Time1.cs

8

```
33      // convert time to universal-time (24 hour) format string
34      public string ToUniversalString()
35      {
36         return String.Format(
37            "{0:D2}:{1:D2}:{2:D2}", hour, minute, second );
38      }
39
40      // convert time to standard-time (12 hour) format string
41      public string ToStandardString()
42      {
43         return String.Format( "{0}:{1:D2}:{2:D2} {3}",
44            ( hour == 12 || hour == 0 ) ? 12 : hour % 12 ),
45            minute, second, ( hour < 12 ? "AM" : "PM" ) );
46      }
47
48   } // end class Time1
```

Output time in universal format

Output time in standard format

Time1.cs

9

```
1    // Fig. 8.2: TimeTest1.cs
2    // Demonstrating class Time1.
3
4    using System;
5    using System.Windows.Forms;
6
7    // TimeTest1 uses creates and uses a Time1 object
8    class TimeTest1
9    {
10      // main entry point for application
11      static void Main( string[] args )
12      {
13         Time1 time = new Time1(); // calls Time1 constructor
14         string output;
15
16         // assign string representation of time to output
17         output = "Initial universal time is: " +
18            time.ToUniversalString() +
19            "\nInitial standard time is: " +
20            time.ToStandardString();
21
22         // attempt valid time settings
23         time.SetTime( 13, 27, 6 );
24
25         // append new string representations of time to output
26         output += "\n\nUniversal time after SetTime is: " +
27            time.ToUniversalString() +
28            "\nStandard time after SetTime is: " +
29            time.ToStandardString();
30
31         // attempt invalid time settings
32         time.SetTime( 99, 99, 99 );
33
```

Call default time constructor

Call method SetTime to set the time with valid arguments

Call method SetTime with invalid arguments

TimeTest1.cs

10

```
34         output += "\n\nAfter attempting invalid settings: " +
35            "\nUniversal time: " + time.ToUniversalString() +
36            "\nStandard time: " + time.ToStandardString();
37
38         MessageBox.Show( output, "Testing Class Time1" );
39
40      } // end method Main
41
42   } // end class TimeTest1
```

TimeTest1.cs

Program Output

8.3 محدوده کلاس

- همه اعضاء درون توابع کلاس با استفاده از نام قابل دستیابی هستند
- بیرون از کلاس نمی توان با نام اعضاء به آنها دسترسی پیدا کرد، اعضاء public می توانند با عملگر نقطه قابل دسترس باشند (*referenceName.memberName*)
- متغیرهای محدوده توابع
  - فقط داخل تابعی که تعریف شده اند قابل دسترس می باشند
- به متغیر های نمونه با استفاده از کلمه کلیدی **this** و عملگر نقطه می توان دسترسی پیدا کرد (مانند ***this.****hour*)

11

8.4 کنترل دسترسی به اعضاء

- توابع Public به استفاده کننده های کلاس نمایی از *سرویسهایی* که کلاس فراهم می کند ارائه می دهد
- توابع باید فقط یک وظیفه را انجام دهند
  - اگر تابعی نیاز به انجام وظیفه دیگری داشته باشد باید از توابع کمکی استفاده کند.
  - استفاده کننده ها نباید به توابع کمکی دسترسی داشته باشند، در نتیجه آنها باید **private** اعلان شوند
- خصوصیات باید برای فراهم کردن دسترسی به داده ها بصورت سالم استفاده شوند (بخش 8.7)
  - داده های عضو باید بصورت private اعلان شوند، با خصوصیات public می توان دسترسی سالمی به آنها فراهم کرد
- خصوصیات
  - **get**: استفاده کننده ها را قادر به خواندن می کند
  - **set**: استفاده کننده ها را قادر به تغییر داده ها می کند

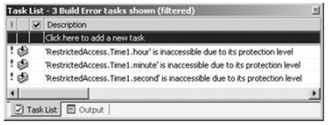
12

13

```
1  // Fig. 8.3: RestrictedAccess.cs
2  // Demonstrate compiler errors from attempt to access
3  // private class members.
4
5  class RestrictedAccess
6  {
7     // main entry point for application
8     static void Main( string[] args )
9     {
10       Time1 time = new Time1();
11
12       time.hour = 7;
13       time.minute = 15;
14       time.second = 30;
15    }
16
17 } // end class RestrictedAccess
```

Attempt to access private members

RestrictedAccess.cs



خروجی برنامه

8.5 مقدار دهی اولیه اشیاء کلاس: سازنده

- نمونه های کلاس با سازنده ها (constructor) مقدار دهی اولیه می شوند
- سازنده ها متغیرهای اشیاء را مقدار اولیه می کنند
- سازنده های سربارگذار شده راه های متفاوتی را برای مقدار دهی اولیه اشیاء کلاس فراهم می کنند.
- حتی اگر سازنده صریحاً این کار را انجام ندهد، تمام اعضای داده ای مقدار دهی اولیه خواهند شد.
  - نوع داده عددی اولیه برابر ۰ قرار داده می شود
  - نوع داده boolean برابر false قرار داده می شود
  - نوع داده ارجاعی برابر null قرار داده می شود
- اگر کلاس سازنده نداشته باشد، سازنده پیشفرض در نظر گرفته می شود
  - نه کدی خواهد داشت و نه پارامتر

14

15

```
1  // Fig. 8.4: Time2.cs
2  // Class Time2 provides overloaded constructors.
3
4  using System;
5
6  // Time2 class definition
7  public class Time2
8  {
9     private int hour;    // 0-23
10    private int minute;  // 0-59
11    private int second;  // 0-59
12
13    // Time2 constructor initializes instance variables to
14    // zero to set default time to midnight
15    public Time2()
16    {
17       SetTime( 0, 0, 0 );
18    }
19
20    // Time2 constructor: hour supplied, minute and second
21    // defaulted to 0
22    public Time2( int hour )
23    {
24       SetTime( hour, 0, 0 );
25    }
26
27    // Time2 constructor: hour and minute supplied, second
28    // defaulted to 0
29    public Time2( int hour, int minute )
30    {
31       SetTime( hour, minute, 0 );
32    }
33
```

Default constructor

Constructor which takes the hour as the input

Constructor which takes the hour and minute as input

Time2.cs

```
34    // Time2 constructor: hour, minute and second supplied
35    public Time2( int hour, int minute, int second )
36    {
37       SetTime( hour, minute, second );
38    }
39
40    // Time2 constructor: initialize using another Time2 object
41    public Time2( Time2 time )
42    {
43       SetTime( time.hour, time.minute, time.second );
44    }
45
46    // Set new time value in 24-hour format. Perform validity
47    // checks on the data. Set invalid values to zero.
48    public void SetTime(
49       int hourValue, int minuteValue, int secondValue )
50    {
51       hour = ( hourValue >= 0 && hourValue < 24 ) ?
52          hourValue : 0;
53       minute = ( minuteValue >= 0 && minuteValue < 60 ) ?
54          minuteValue : 0;
55       second = ( secondValue >= 0 && secondValue < 60 ) ?
56          secondValue : 0;
57    }
58
59    // convert time to universal-time (24 hour) format string
60    public string ToUniversalString()
61    {
62       return String.Format(
63          "{0:D2}:{1:D2}:{2:D2}", hour, minute, second );
64    }
65
```

17

```
66      // convert time to standard-time (12 hour) format string
67      public string ToStandardString()
68      {
69         return String.Format( "{0}:{1:D2}:{2:D2} {3}",
70            ( ( hour == 12 || hour == 0 ) ? 12 : hour % 12 ),
71            minute, second, ( hour < 12 ? "AM" : "PM" ) );
72      }
73
74   } // end class Time2
```

Time2.cs

18

```
1    // Fig. 8.5: TimeTest2.cs
2    // Using overloaded constructors.
3
4    using System;
5    using System.Windows.Forms;
6
7    // TimeTest2 demonstrates constructors of class Time2
8    class TimeTest2
9    {
10      // main entry point for application
11      static void Main( string[] args )
12      {
13         Time2 time1, time2, time3, time4, time5, time6;
14
15         time1 = new Time2();              // 00:00:00
16         time2 = new Time2( 2 );           // 02:00:00
17         time3 = new Time2( 21, 34 );      // 21:34:00
18         time4 = new Time2( 12, 25, 42 );  // 12:25:42
19         time5 = new Time2( 27, 74, 99 );  // 00:00:00
20         time6 = new Time2( time4 );       // 12:25:42
21
22         String output = "Constructed with: " +
23            "\ntime1: all arguments defaulted" +
24            "\n\t" + time1.ToUniversalString() +
25            "\n\t" + time1.ToStandardString();
26
27         output += "\ntime2: hour specified; minute and " +
28            "second defaulted" +
29            "\n\t" + time2.ToUniversalString() +
30            "\n\t" + time2.ToStandardString();
31
32         output += "\ntime3: hour and minute specified; " +
33            "second defaulted" +
34            "\n\t" + time3.ToUniversalString() +
35            "\n\t" + time3.ToStandardString();
```

Test the constructors

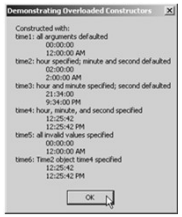
TimeTest2.cs

19

```
36
37         output += "\ntime4: hour, minute, and second specified" +
38            "\n\t" + time4.ToUniversalString() +
39            "\n\t" + time4.ToStandardString();
40
41         output += "\ntime5: all invalid values specified" +
42            "\n\t" + time5.ToUniversalString() +
43            "\n\t" + time5.ToStandardString();
44
45         output += "\ntime6: Time2 object time4 specified" +
46            "\n\t" + time6.ToUniversalString() +
47            "\n\t" + time6.ToStandardString();
48
49         MessageBox.Show( output,
50            "Demonstrating Overloaded Constructors" );
51
52      } // end method Main
53
54   } // end class TimeTest2
```

TimeTest2.cs

Program Output



8.7 خصوصیات

- خصیصه های public به استفاده کننده ها اجازه می دهد:
  - Get (به دست آوردن مقدار) داده ها private
  - Set (انتساب مقدار به) داده ها private
- Get accessor
  - قالب بندی داده ها را کنترل می کند
- Set accessor
  - اطمینان می دهد مقدار جدید مناسب عضو داده ای است

20

21

Time3.cs

```
1    // Fig. 8.6: Time3.cs
2    // Class Time2 provides overloaded constructors.
3
4    using System;
5
6    // Time3 class definition
7    public class Time3
8    {
9       private int hour;    // 0-23
10      private int minute;  // 0-59
11      private int second;  // 0-59
12
13      // Time3 constructor initializes instance variables to
14      // zero to set default time to midnight
15      public Time3()
16      {
17         SetTime( 0, 0, 0 );
18      }
19
20      // Time3 constructor: hour supplied, minute and second
21      // defaulted to 0
22      public Time3( int hour )
23      {
24         SetTime( hour, 0, 0 );
25      }
26
27      // Time3 constructor: hour and minute supplied, second
28      // defaulted to 0
29      public Time3( int hour, int minute )
30      {
31         SetTime( hour, minute, 0 );
32      }
33
```

22

Time3.cs

Constructor that takes another Time3 object as an argument. New Time3 object is initialized with the values of the argument.

Property Hour

```
34      // Time3 constructor: hour, min
35      public Time3( int hour, int min
36      {
37         SetTime( hour, minute, secon
38      }
39
40      // Time3 constructor: initialize using another Time3 object
41      public Time3( Time3 time )
42      {
43         SetTime( time.Hour, time.Minute, time.Second );
44      }
45
46      // Set new time value in 24-hour format. Perform validity
47      // checks on the data. Set invalid values to zero.
48      public void SetTime(
49         int hourValue, int minuteValue, int secondValue )
50      {
51         Hour = hourValue;
52         Minute = minuteValue;
53         Second = secondValue;
54      }
55
56      // property Hour
57      public int Hour
58      {
59         get
60         {
61            return hour;
62         }
63
64         set
65         {
66            hour = ( ( value >= 0 && value < 24 ) ? value : 0 );
67         }
68
```

23

Time3.cs

Property Second

Property Minute

```
69      } // end property Hour
70
71      // property Minute
72      public int Minute
73      {
74         get
75         {
76            return minute;
77         }
78
79         set
80         {
81            minute = ( ( value >= 0 && value < 60 ) ? value : 0 );
82         }
83
84      } // end property Minute
85
86      // property Second
87      public int Second
88      {
89         get
90         {
91            return second;
92         }
93
94         set
95         {
96            second = ( ( value >= 0 && value < 60 ) ? value : 0 );
97         }
98
99      } // end property Second
100
```

24

Time3.cs

```
101     // convert time to universal-time (24 hour) format string
102     public string ToUniversalString()
103     {
104        return String.Format(
105           "{0:D2}:{1:D2}:{2:D2}", Hour, Minute, Second );
106     }
107
108     // convert time to standard-time (12 hour) format string
109     public string ToStandardString()
110     {
111        return String.Format( "{0}:{1:D2}:{2:D2} {3}",
112           ( ( Hour == 12 || Hour == 0 ) ? 12 : Hour % 12 ),
113           Minute, Second, ( Hour < 12 ? "AM" : "PM" ) );
114     }
115
116  } // end class Time3
```

25

TimeTest3.cs

```
1    // Fig. 8.7: TimeTest3.cs
2    // Demonstrating Time3 properties Hour, Minute and Second.
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10
11   // TimeTest3 class definition
12   public class TimeTest3 : System.Windows.Forms.Form
13   {
14      private System.Windows.Forms.Label hourLabel;
15      private System.Windows.Forms.TextBox hourTextBox;
16      private System.Windows.Forms.Button hourButton;
17
18      private System.Windows.Forms.Label minuteLabel;
19      private System.Windows.Forms.TextBox minuteTextBox;
20      private System.Windows.Forms.Button minuteButton;
21
22      private System.Windows.Forms.Label secondLabel;
23      private System.Windows.Forms.TextBox secondTextBox;
24      private System.Windows.Forms.Button secondButton;
25
26      private System.Windows.Forms.Button addButton;
27
28      private System.Windows.Forms.Label displayLabel1;
29      private System.Windows.Forms.Label displayLabel2;
30
31      // required designer variable
32      private System.ComponentModel.Container components = null;
33
34      private Time3 time;
35
```

26

TimeTest3.cs

```
36      public TimeTest3()
37      {
38         // Required for Windows Form Designer support
39         InitializeComponent();
40
41         time = new Time3();
42         UpdateDisplay();
43      }
44
45      // Visual Studio .NET generated code
46
47      // main entry point for application
48      [STAThread]
49      static void Main()
50      {
51         Application.Run( new TimeTest3() );
52      }
53
54      // update display labels
55      public void UpdateDisplay()
56      {
57         displayLabel1.Text = "Hour: " + time.Hour +
58            "; Minute: " + time.Minute +
59            "; Second: " + time.Second;
60         displayLabel2.Text = "Standard time: " +
61            time.ToStandardString() + "\nUniversal time: " +
62            time.ToUniversalString();
63      }
64
```

27

TimeTest3.cs

```
65      // set Hour property when hourButton pressed
66      private void hourButton_Click(
67         object sender, System.EventArgs e )
68      {
69         time.Hour = Int32.Parse( hourTextBox.Text );
70         hourTextBox.Text = "";
71         UpdateDisplay();
72      }
73
74      // set Minute property when minuteButton pressed
75      private void minuteButton_Click(
76         object sender, System.EventArgs e )
77      {
78         time.Minute = Int32.Parse( minuteTextBox.Text );
79         minuteTextBox.Text = "";
80         UpdateDisplay();
81      }
82
83      // set Second property when secondButton pressed
84      private void secondButton_Click(
85         object sender, System.EventArgs e )
86      {
87         time.Second = Int32.Parse( secondTextBox.Text );
88         secondTextBox.Text = "";
89         UpdateDisplay();
90      }
91
92      // add one to Second when addButton pressed
93      private void addButton_Click(
94         object sender, System.EventArgs e )
95      {
96         time.Second = ( time.Second + 1 ) % 60;
97
```

Set Minute property of Time3 object

Set Second property of Time3 object

Set Hour property of Time3 object

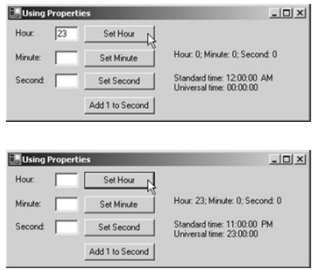
Add 1 second to Time3 object

28

TimeTest3.cs

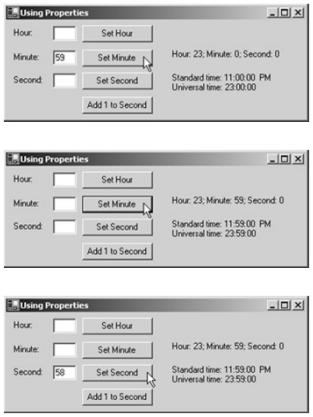
```
98         if ( time.Second == 0 )
99         {
100           time.Minute = ( time.Minute + 1 ) % 60;
101
102           if ( time.Minute == 0 )
103              time.Hour = ( time.Hour + 1 ) % 24;
104        }
105
106        UpdateDisplay();
107     }
108
109  } // end class TimeTest3
```

خروجی برنامه



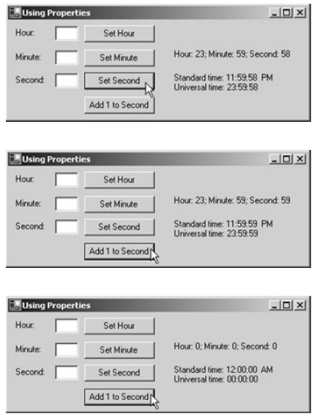
29

TimeTest3.cs
خروجی برنامه



30

TimeTest3.cs
خروجی برنامه



8.8 ترکیب: ارجاء اشیاء به عنوان متغیرهای دیگر کلاس ها

- استفاده مجدد نرم افزار – استفاده از اشیاء موجود نسبت به نوشتن مجدد کدهای اشیاء برای کلاسهای جدید راحت تر می باشد
- نوع های تعریف شده توسط کاربر به عنوان متغیر های نمونه استفاده می شود
- به عنوان مثال برای پیاده سازی Alarm-Clock به جای پیاده سازی Time می توان از کلاس Time استفاده کرد

31

32

سازنده ای که ماه، روز و سال را به عنوان آرگومان دریافت می کند. آرگومانها اعتبار سنجی می شوند، اگر معتبر نباشند برابر مقدار پیشفرض قرار داده می شوند.

Date.cs

```
1   // Fig. 8.8: Date.cs
2   // Date class definition
3
4   using System;
5
6   // Date class definition
7   public class Date
8   {
9      private int month;  // 1-12
10     private int day;    // 1-31 based on month
11     private int year;   // any year
12
13     // constructor confirms proper value for month;
14     // call method CheckDay to confirm proper
15     // value for day
16     public Date( int theMonth, int theDay, int theYear )
17     {
18        // validate month
19        if ( theMonth > 0 && theMonth <= 12 )
20           month = theMonth;
21
22        else
23        {
24           month = 1;
25           Console.WriteLine(
26              "Month {0} invalid. Set to month 1.", theMonth );
27        }
28
29        year = theYear;                // could validate year
30        day = CheckDay( theDay );      // validate day
31     }
32
```

33

```
33      // utility method confirms proper day value
34      // based on month and year
35      private int CheckDay( int testDay )
36      {
37         int[] daysPerMonth =
38            { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
39
40         // check if day in range for month
41         if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
42            return testDay;
43
44         // check for leap year
45         if ( month == 2 && testDay == 29 &&
46              ( year % 400 == 0 ||
47                ( year % 4 == 0 && year % 100 != 0 ) ) )
48            return testDay;
49
50         Console.WriteLine(
51            "Day {0} invalid. Set to day 1.", testDay );
52
53         return 1;  // leave object in consistent state
54      }
55
56      // return date string as month/day/year
57      public string ToDateString()
58      {
59         return month + "/" + day + "/" + year;
60      }
61
62   }  // end class Date
```

Date.cs

اعتبار سنجی می کند که یک ماه مشخص شده می تواند تعداد روزهای مشخص شده را داشته باشد.

34

```
1    // Fig. 8.9: Employee.cs
2    // Employee class definition encapsulat
3    // last name, birth date and hire date.
4
5    using System;
6
7    // Employee class definition
8    public class Employee
9    {
10      private string firstName;
11      private string lastName;
12      private Date birthDate;
13      private Date hireDate;
14
15      // constructor initializes name, birth date and hire date
16      public Employee( string first, string last,
17         int birthMonth, int birthDay, int birthYear,
18         int hireMonth, int hireDay, int hireYear )
19      {
20         firstName = first;
21         lastName = last;
22
23         // create new Date for Employee birth day
24         birthDate = new Date( birthMonth, birthDay, birthYear );
25         hireDate = new Date( hireMonth, hireDay, hireYear );
26      }
27
```

Employee.cs

دو شئ Date عضو کلاس Employee هستند

سازنده ای که ناماً تاریخ تولد و تاریخ استخدام کارمندها را مقدار دهی اولیه می کند.

35

```
28      // convert Employee to String format
29      public string ToEmployeeString()
30      {
31         return lastName + ", " + firstName +
32            "  Hired: " + hireDate.ToDateString() +
33            "  Birthday: " + birthDate.ToDateString();
34      }
35
36   }  // end class Employee
```

Employee.cs

36

```
1    // Fig. 8.10: CompositionTest.cs
2    // Demonstrate an object with member object reference.
3
4    using System;
5    using System.Windows.Forms;
6
7    // Composition class definition
8    class CompositionTest
9    {
10      // main entry point for application
11      static void Main( string[] args )
12      {
13         Employee e =
14            new Employee( "Bob", "Jones", 7, 24, 1949, 3, 12, 1988 );
15
16         MessageBox.Show( e.ToEmployeeString(),
17            "Testing Class Employee" );
18
19      } // end method Main
20
21   } // end class CompositionTest
```

CompositionTest.cs

خروجی برنامه

8.9 استفاده از ارجاء this

- هر شئ می تواند به خودش با استفاده از **this** ارجاء داشته باشد
- بعضی مواقع برای تمایز مابین متغیرهای توابع و متغیر های شئ نمونه استفاده می شود

Time4.cs

```
1   // Fig. 8.11: Time4.cs
2   // Class Time2 provides overloaded constructors.
3
4   using System;
5
6   // Time4 class definition
7   public class Time4
8   {
9      private int hour;    // 0-23
10     private int minute;  // 0-59
11     private int second;  // 0-59
12
13     // constructor
14     public Time4( int hour, int minute, int second )
15     {
16        this.hour = hour;
17        this.minute = minute;
18        this.second = second;
19     }
20
21     // create string using this and implicit references
22     public string BuildString()
23     {
24        return "this.ToStandardString(): " +
25           this.ToStandardString() +
26           "\nToStandardString(): " + ToStandardString();
27     }
28
```

The this reference is used to set the class member variables to the constructor arguments

The this reference is used to refer to an instance method

Time4.cs

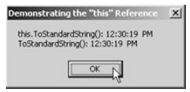
```
29     // convert time to standard-time (12 hour) format string
30     public string ToStandardString()
31     {
32        return String.Format( "{0}:{1:D2}:{2:D2} {3}",
33           ( ( this.hour == 12 || this.hour == 0 ) ? 12 :
34           this.hour % 12 ), this.minute, this.second,
35           ( this.hour < 12 ? "AM" : "PM" ) );
36     }
37
38  } // end class Time4
```

The this reference is used to access member variables

ThisTest.cs

```
1   // Fig. 8.12: ThisTest.cs
2   // Using the this reference.
3
4   using System;
5   using System.Windows.Forms;
6
7   // ThisTest class definition
8   class Class1
9   {
10     // main entry point for application
11     static void Main( string[] args )
12     {
13        Time4 time = new Time4( 12, 30, 19 );
14
15        MessageBox.Show( time.BuildString(),
16           "Demonstrating the \"this\" Reference" );
17     }
18  }
```

خروجی برنامه



8.10 Garbage Collection

- عملگر new حافظه اختصاص می دهد
- حافظه و منابع اختصاص یافته باید به سیستم برگشت داده شود
- چهارچوب .net عمل garbage collection حافظه را برای برگشت دادن حافظه ای که دیگر مورد نیاز نیست به سیستم انجام می دهد.
- Garbage collection از هدر رفتن حافظه جلوگیری می کند
- اختصاص و آزاد سازی سایر منابع (اتصال پایگاه داده، دسترسی فایل و غیره) باید صریحاً توسط برنامه مدیریت شود

8.10 Garbage Collection

- *Finalizer*s در ارتباط با garbage collector برای آزاد سازی منابع و حافظه استفاده می شود
- هر کلاس فقط می تواند یک پایان دهنده داشته باشد (همچنین مخرب نیز نامیده می شود)
  - destructor
- نام تابع مخرب همان نام کلاس می باشد که با ~ شروع می شود
- مخرب نمی تواند پارامتر ورودی داشته باشد.

8.11 اعضاء static کلاس

- هر شئ از کلاس کپی خاص خود را از متغیر های کلاس خواهد داشت
- برخی مواقع ممکن است بخواهید تمام نمونه های کلاس کپی یکسانی از یک متغیر را داشته باشید
  - متغیر عضو مشترک
- تعریف متغیر با کلمه کلیدی **static** موجب ایجاد یک کپی از یک متغیر در زمان می شود (مشترک مابین تمام اشیاء یک کلاس)
- حوزه ممکن است برای متغیرهای static تعریف شود(**public**، **private** و غیره)

Employee.cs

```
1   // Fig. 8.13: Employee.cs
2   // Employee class contains static data and a static method.
3
4   using System;
5
6   // Employee class definition
7   public class Employee
8   {
9      private string firstName;
10     private string lastName;
11     private static int count; // Emp
12
13     // constructor increments static Employee count
14     public Employee( string fName, string lName )
15     {
16        firstName = fName;
17        lastName = lName;
18
19        ++count;
20
21        Console.WriteLine( "Employee object constructor: " +
22           firstName + " " + lastName + "; count = " + Count );
23     }
24
25     // destructor decrements static Employee count
26     ~Employee()
27     {
28        --count;
29
30        Console.WriteLine( "Employee object destructor: " +
31           firstName + " " + lastName + "; count = " + Count );
32     }
33
```

مخرب Employee

تعداد کارمند ها را به روز می کند.

تعداد عضو استاتیک را کاهش می دهد، تا مشخص کند که یک کارمند کمتر شده.

45

```
34      // FirstName property
35      public string FirstName
36      {
37         get
38         {
39            return firstName;
40         }
41      }
42
43      // LastName property
44      public string LastName
45      {
46         get
47         {
48            return lastName;
49         }
50      }
51
52      // static Count property
53      public static int Count
54      {
55         get
56         {
57            return count;
58         }
59      }
60
61   } // end class Employee
```

Employee.cs

46

```
1    // Fig. 8.14: StaticTest.cs
2    // D                                    embers.
3
4    using System;
5
6    // StaticTest class definition
7    class StaticTest
8    {
9       // main entry point for application
10      static void Main( string[] args )
11      {
12         Console.WriteLine( "Employees before instantiation: " +
13            Employee.Count + "\n" );
14
15         // create two Employees
16         Employee employee1 = new Employee( "Susan", "Baker" );
17         Employee employee2 = new Employee( "Bob", "Jones" );
18
19         Console.WriteLine( "\nEmployees after instantiation: " +
20            "Employee.Count = " + Employee.Count + "\n" );
21
22         // display the Employees
23         Console.WriteLine( "Employee 1: " +
24            employee1.FirstName + " " + employee1.LastName +
25            "\nEmployee 2: " + employee2.FirstName +
26            " " + employee2.LastName + "\n" );
27
28         // mark employee1 and employee1 objects for
29         // garbage collection
30         employee1 = null;
31         employee2 = null;
32
33         // force garbage collection
34         System.GC.Collect();
35
```

دو شئ Employee ایجاد می کند.

شئ Employee را برابر null قرار می دهد

مجبور کردن به garbage collection

StaticTest.cs

47

```
36         Console.WriteLine(
37            "\nEmployees after garbage collection: " +
38            Employee.Count );
39      }
40   }
```

StaticTest.cs

خروجی برنامه

```
Employees before instantiation: 0

Employee object constructor: Susan Baker; count = 1
Employee object constructor: Bob Jones; count = 2

Employees after instantiation: Employee.Count = 2

Employee 1: Susan Baker
Employee 2: Bob Jones

Employee object destructor: Bob Jones; count = 1
Employee object destructor: Susan Baker; count = 0

Employees after garbage collection: 2
```

8.12 اعضاء ثابت و فقط خواندنی

- اعضاء ثابت (اعضایی که مقدار آنها هرگز تغییر نخواهد کرد) با کلمه کلیدی **const** اعلان می شوند
- اعضای **const** به طور ضمنی **static** هستند
- اعضاء **const** باید هنگام اعلان مقدار دهی اولیه شوند.
- استفاده از کلمه کلیدی readonly برای اعلان اعضای موجب خواهد شد که در سازنده مقدار دهی اولیه شود اما پس از آن تغییر نکند.

48

49

```
1   // Fig. 8.15: UsingConstAndReadOnly.cs
2   // Demonstrating constant values with
3
4   using System;
5   using System.Windows.Forms;
6
7   // Constants class definition
8   public class Constants
9   {
10     // PI is constant variable
11     public const double PI = 3.14159;
12
13     // radius is a constant variable
14     // that is uninitialized
15     public readonly int radius;
16
17     public Constants( int radiusValue )
18     {
19        radius = radiusValue;
20     }
21
22  } // end class Constants
23
24  // UsingConstAndReadOnly class definition
25  public class UsingConstAndReadOnly
26  {
27     // method Main creates Constants
28     // object and displays it's values
29     static void Main( string[] args )
30     {
31        Random random = new Random();
32
33        Constants constantValues =
34           new Constants( random.Next( 1, 20 ) );
35
```

Readonly variable radius; must be initialized in constructor

Initialize readonly member radius

UsingConstAndReadOnly.cs

50

```
36        MessageBox.Show( "Radius = " + constantValues.radius +
37           "\nCircumference = " +
38           2 * Constants.PI * constantValues.radius,
39           "Circumference" );
40
41     } // end method Main
42
43  } // end class UsingConstAndReadOnly
```

UsingConstAndReadOnly.cs

خروجی برنامه

8.13 اندیس سازها

- بعضی مواقع کلاسها داده ها را داده هایی را بسته بندی می کنند که شبیه لیستی از عناصر است.
- اندیس ساز خصوصیت ویژه ای است که دسترسی آرایه ای را به داده ها در کلاس اجازه می دهد.
- اندیس سازها می توانند برای پذیرفتن زیر نویسهای عددی و غیر عددی تعریف شوند.
- استفاده از کلمه کلیدی **this** تعریف شده
- هنگام استفاده از اندیس ساز برنامه نویسها از نماد براکت ([]) همانند آرایه ها برای دسترسی **get** و **set** استفاده می کنند.

51

52

```
1   // Fig. 8.10: IndexerTest.cs
2   // Indexers provide access to an object's members via a
3   // subscript operator.
4
5   using System;
6   using System.Drawing;
7   using System.Collections;
8   using System.ComponentModel;
9   using System.Windows.Forms;
10  using System.Data;
11
12  // Box class definition represents a box with length,
13  // width and height dimensions
14  public class Box
15  {
16     private string[] names = { "length", "width", "height" };
17     private double[] dimensions = new double[ 3 ];
18
19     // constructor
20     public Box( double length, double width, double height )
21     {
22        dimensions[ 0 ] = length;
23        dimensions[ 1 ] = width;
24        dimensions[ 2 ] = height;
25     }
26
27     // access dimensions by index number
28     public double this[ int index ]
29     {
30        get
31        {
32           return ( index < 0 || index > dimensions.Length ) ?
33              -1 : dimensions[ index ];
34        }
35
```

تعریف اندیس ساز: اندیس ساز یک عدد صحیح را برای مشخص کردن بعد مورد نظر دریافت می کند.

مربوط به دسترسی دهنده اندیس

اگر اندیس درخواست شده خارج از محدوده باشد ۱- را بر می گرداند در غیر این صورت عنصر مورد نظر را بر می گرداند.

IndexerTest.cs

53

IndexerTest.cs

```
36          set
37          {
38             if ( index >= 0 && index <
39                dimensions[ index ] = va
40          }
41
42       } // end numeric indexer
43
44       // access dimensions by their names
45       public double this[ string name ]
46       {
47          get
48          {
49             // locate element to get
50             int i = 0;
51
52             while ( i < names.Length &&
53                   name.ToLower() != names[ i ] )
54                i++;
55
56             return ( i == names.Length ) ? -1 : dimensions[ i ];
57          }
58
59          set
60          {
61             // locate element to set
62             int i = 0;
63
64             while ( i < names.Length &&
65                   name.ToLower() != names[ i ] )
66                i++;
67
```

اندیس سازی که نام بعد را به عنوان آرگومان دریافت می کند.

Set مربوط به دسترسی دهنده اندیس

اعتبار سنجی می کند که اندیسی که کاربر می خواهد دسترسی داشته باشد معتبر است سپس آنرا set کند.

54

IndexerTest.cs

```
68             if ( i != names.Length )
69                dimensions[ i ] = value;
70          }
71
72       } // end indexer
73
74    } // end class Box
75
76    // Class IndexerTest
77    public class IndexerTest : System.Windows.Forms.Form
78    {
79       private System.Windows.Forms.Label indexLabel;
80       private System.Windows.Forms.Label nameLabel;
81
82       private System.Windows.Forms.TextBox indexTextBox;
83       private System.Windows.Forms.TextBox valueTextBox;
84
85       private System.Windows.Forms.Button nameSetButton;
86       private System.Windows.Forms.Button nameGetButton;
87
88       private System.Windows.Forms.Button intSetButton;
89       private System.Windows.Forms.Button intGetButton;
90
91       private System.Windows.Forms.TextBox resultTextBox;
92
93       // required designer variable
94       private System.ComponentModel.Container components = null;
95
96       private Box box;
97
98       // constructor
99       public IndexerTest()
100      {
101         // required for Windows Form Designer support
102         InitializeComponent();
```

55

IndexerTest.cs

```
103
104         // create block
105         box = new Box( 0.0, 0.0, 0.0 );
106      }
107
108      // Visual Studio .NET generated code
109
110      // main entry point for application
111      [STAThread]
112      static void Main()
113      {
114         Application.Run( new IndexerTest() );
115      }
116
117      // display value at specified index number
118      private void ShowValueAtIndex( string prefix, int index )
119      {
120         resultTextBox.Text =
121            prefix + "box[ " + index + " ] = " + box[ index ];
122      }
123
124      // display value with specified name
125      private void ShowValueAtIndex( string prefix, string name )
126      {
127         resultTextBox.Text =
128            prefix + "box[ " + name + " ] = " + box[ name ];
129      }
130
131      // clear indexTextBox and valueTextBox
132      private void ClearTextBoxes()
133      {
134         indexTextBox.Text = "";
135         valueTextBox.Text = "";
136      }
137
```

استفاده از دسترسی دهنده اندیس ساز get

استفاده از دسترسی دهنده اندیس ساز set

56

IndexerTest.cs

```
138      // get value at specified index
139      private void intGetButton_Click(
140         object sender, System.EventArgs e )
141      {
142         ShowValueAtIndex(
143            "get: ", Int32.Parse( indexTextBox.Text ) );
144         ClearTextBoxes();
145      }
146
147      // set value at specified index
148      private void intSetButton_Click(
149         object sender, System.EventArgs e )
150      {
151         int index = Int32.Parse( indexTextBox.Te
152         box[ index ] = Double.Parse( valueTextBox.Text );
153
154         ShowValueAtIndex( "set: ", index );
155         ClearTextBoxes();
156      }
157
158      // get value with specified name
159      private void nameGetButton_Click(
160         object sender, System.EventArgs e )
161      {
162         ShowValueAtIndex( "get: ", indexTextBox.Text );
163         ClearTextBoxes();
164      }
165
```

استفاده از اندیس ساز صحیص برای ست کردن مقدار

استفاده از اندیس ساز صحیص برای گرفتن مقدار

استفاده از اندیس ساز رشته برای گرفتن مقدار

57

```
166    // set value with specified name
167    private void nameSetButton_Click(
168       object sender, System.EventArgs e )
169    {
170       box[ indexTextBox.Text ] =
171          Double.Parse( valueTextBox.Text );
172
173       ShowValueAtIndex( "set: ", indexTextBox.Text );
174       ClearTextBoxes();
175    }
176
177 } // end class IndexerTest
```

Use string indexer to set value

IndexerTest.cs

خروجی برنامه

Before setting value by index number

After setting value by index number

58

Before getting value by dimension name

IndexerTest.cs

خروجی برنامه

After getting value by dimension name

Before setting value by dimension name

59

After setting value by dimension name

IndexerTest.cs

خروجی برنامه

Before getting value by index number

After getting value by index number

8.14 داده های انتزاعی و مخفی سازی اطلاعات

- کلاس ها باید جزئیات پیاده سازی را مخفی کند
- پشته
  - Last-in, first-out (LIFO)
    - آیتم ها بر روی بالای پشته قرار (push) می گیرند
    - آیتم ها را از بالای پشته برداشته (pop) می شوند
- صف
  - شبیه به خط انتظار می باشد
  - First-in, first-out (FIFO)
  - Items are enqueued (added to the end)
  - Items are dequeued (taken off the front)

60

8.15 قابلیت استفاده مجدد نرم افزار

- کتابخانه کلاس فریم ورک شامل هزاران کلاس از پیش تعریف شده می باشد.
  - Framework Class Library (FCL)
  - یک Library بسیار بزرگ است شامل تعداد زیادی کلاس که این کلاس ها در بخش های مختلفی که NameSpace نامیده می شوند دسته بندی شده اند و به طور کلی در برنامه نویسی ویندوز میتوان از این کلاس ها استفاده کنید.
- کلاسهای FCL باید هر موقع که ممکن باشد استفاده شود
  - No bugs
  - Optimized
  - Well-Documented
- استفاده مجدد از کد توسعه سریع نرم افزار را تسهیل می کند
  - Rapid Application Development (RAD)

61

62

TimeLibrary.cs

```
1   // Fig. 8.17: TimeLibrary.cs
2   // Placing class Time3 in an assembly for reuse.
3
4   using System;
5
6   namespace TimeLibrary
7   {
8      // Time3 class definition
9      public class Time3
10     {
11        private int hour;    // 0-23
12        private int minute;  // 0-59
13        private int second;  // 0-59
14
15        // Time3 constructor initializes instance variables to
16        // zero to set default time to midnight
17        public Time3()
18        {
19           SetTime( 0, 0, 0 );
20        }
21
22        // Time3 constructor: hour supplied, minute and second
23        // defaulted to 0
24        public Time3( int hour )
25        {
26           SetTime( hour, 0, 0 );
27        }
28
29        // Time3 constructor: hour and minute supplied, second
30        // defaulted to 0
31        public Time3( int hour, int minute )
32        {
33           SetTime( hour, minute, 0 );
34        }
35
```

63

TimeLibrary.cs

```
36        // Time3 constructor: hour, minute and second supplied
37        public Time3( int hour, int minute, int second )
38        {
39           SetTime( hour, minute, second );
40        }
41
42        // Time3 constructor: initialize using another Time3 object
43        public Time3( Time3 time )
44        {
45           SetTime( time.Hour, time.Minute, time.Second );
46        }
47
48        // Set new time value in 24-hour format. Perform validity
49        // checks on the data. Set invalid values to zero.
50        public void SetTime(
51           int hourValue, int minuteValue, int secondValue )
52        {
53           Hour = hourValue;
54           Minute = minuteValue;
55           Second = secondValue;
56        }
57
58        // property Hour
59        public int Hour
60        {
61           get
62           {
63              return hour;
64           }
65
66           set
67           {
68              hour = ( ( value >= 0 && value < 24 ) ? value : 0 );
69           }
70
```

64

TimeLibrary.cs

```
71        } // end property Hour
72
73        // property Minute
74        public int Minute
75        {
76           get
77           {
78              return minute;
79           }
80
81           set
82           {
83              minute = ( ( value >= 0 && value < 60 ) ? value : 0 );
84           }
85
86        } // end property Minute
87
88        // property Second
89        public int Second
90        {
91           get
92           {
93              return second;
94           }
95
96           set
97           {
98              second = ( ( value >= 0 && value < 60 ) ? value : 0 );
99           }
100
101       } // end property Second
102
```

65

```
103        // convert time to universal-time (24 hour) format string
104        public string ToUniversalString()
105        {
106           return String.Format(
107              "{0:D2}:{1:D2}:{2:D2}", Hour, Minute, Second );
108        }
109
110        // convert time to standard-time (12 hour) format string
111        public string ToStandardString()
112        {
113           return String.Format( "{0}:{1:D2}:{2:D2} {3}",
114              ( ( Hour == 12 || Hour == 0 ) ? 12 : Hour % 12 ),
115              Minute, Second, ( Hour < 12 ? "AM" : "PM" ) );
116        }
117
118     } // end class Time3
119  }
```

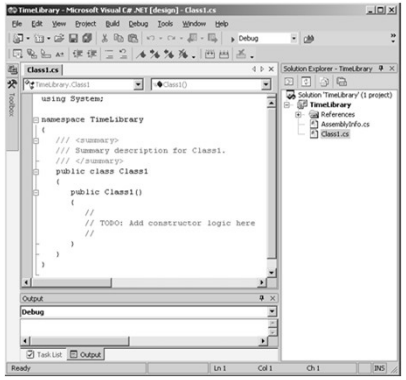
TimeLibrary.cs

8.16 Namespaces

- اجزاء نرم افزار باید قابل استفاده مجدد باشند
- Namespaces گروه بندی منطقی از کلاسها را فراهم می کند
- نباید دوکلاس در یک فضای نامی، نام یکسانی را داشته باشند.
- دوکلاس در دو فضای نامی متفاوت، می توانند نام یکسانی را داشته باشند.
- فایل های DLL به کلاسها اجازه می دهند که برای استفاده مجدد بسته بندی شوند.
- Dynamic Link Libraries (DLL)

66

8.16 Namespaces



67

Fig. 8.18 *Simple Class Library.*

68

```
1   // Fig. 8.19: AssemblyTest.cs
2   // Using class Time3 from assembly TimeLibrary.
3
4   using System;
5   using TimeLibrary;
6
7   // AssemblyTest class definition
8   class AssemblyTest
9   {
10     // main entry point for application
11     static void Main( string[] args )
12     {
13        Time3 time = new Time3( 13, 27, 6 );
14
15        Console.WriteLine(
16           "Standard time: {0}\nUniversal time: {1}\n",
17           time.ToStandardString(), time.ToUniversalString() );
18     }
19  }
```

Use Time3 as usual

Reference the TimeLibrary namespace

AssemblyTest.cs

```
Standard time: 1:27:06 PM
Universal time: 13:27:06
```

Program Output

68

سربار گذاری توابع

- Method Overloading
  - توابع با نامهای یکسان تعریف می شوند
    - می توانند نامهای یکسانی داشته باشند اما باید پارامتر های متفاوتی را داشته باشند
      - متغیر های ارسال شده باید متفاوت باشند
        - چه در نوع و چه در ترتیب
    - معمولا عمل یکسانی را انجام می دهند
      - روی نوع های داده ای متفاوت

69

70

MethodOverload.cs

```
1   // Fig. 6.18: MethodOverload.cs
2   // Using overloaded methods.
3
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
10
11  public class MethodOverload : System.Windows.Forms.Form
12  {
13     private System.ComponentModel.Container components = null;
14
15     private System.Windows.Forms.Label outputLabel;
16
17     public MethodOverload()
18     {
19        InitializeComponent();
20
21        // call both versions of Square
22        outputLabel.Text =
23           "The square of integer 7 is " + Square( 7 ) +
24           "\nThe square of double 7.5 is " + Square ( 7.5 );
25     }
26
27     // Visual Studio .NET-generated code
28
```

Two versions of the square method are called

71

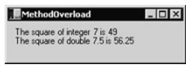
MethodOverload.cs

```
29     // first version, takes one integer
30     public int Square ( int x )
31     {
32        return x * x;
33     }
34
35     // second version, takes one double
36     public double Square ( double y )
37     {
38        return y * y;
39     }
40
41     [STAThread]
42     static void Main()
43     {
44        Application.Run( new MethodOverload() );
45     }
46
47  } // end of class MethodOverload
```

One method takes an **int** as parameters

The other version of the method uses a **double** instead of an integer

Program Output



72

MethodOverload2.cs

```
1   // Fig. 6.19: MethodOverload2.cs
2   // Overloaded methods with identical signatures and
3   // different return types.
4
5   using System;
6
7   class MethodOverload2
8   {
9      public int Square( double
10     {
11        return x * x;
12     }
13
14     // second Square method takes same number,
15     // order and type of arguments, error
16     public double Square( double y )
17     {
18        return y * y;
19     }
20
21     // main entry point for application
22     static void Main()
23     {
24        int squareValue = 2;
25        Square( squareValue );
26     }
27
28  } // end of class MethodOverload2
```

This method returns an integer

This method returns a double number

Since the compiler cannot tell which method to use based on passed values an error is generated

Program Output

ساختار تکرار foreach

- ساختار تکرار حلقه foreach پیمایش مقادیر در ساختمانهای داده ای مانند آرایه استفاده می شود.
- شمارنده ای وجود ندارد
- یک متغییر برای نمایش مقدار هر عنصر استفاده می شود.


```
1   // Fig. 7.16: ForEach.cs
2   // Demonstrating for/each structure
3   using System;
4
5   class ForEach
6   {
7      // main entry point for the application
8      static void Main( string[] args )
9      {
10        int[,] gradeArray = { { 77, 68, 86, 73 },
11           { 98, 87, 89, 81 }, { 70, 90, 86, 81 } };
12
13        int lowGrade = 100;
14
15        foreach ( int grade in gradeArray )
16        {
17           if ( grade < lowGrade )
18              lowGrade = grade;
19        }
20
21        Console.WriteLine( "The minimum grade is: " + lowGrade );
22     }
23  }
```

Use the foreach loop to examine each element in the array

If the current array element is smaller then lowGrade, set lowGrade to contain the value of the current element

ForEach.cs

```
The minimum grade is: 68
```

8.17 Class View and Object Browser

- **Class View** and **Object Browser** are features of Visual Studio that facilitate the design of object-oriented applications
- **Class View**
  - Displays variables and methods for all classes in a project
  - Displays as treeview hierarchical structure
  - + at nodes allows nodes to be expanded
  - - at nodes allows nodes to be collapsed
  - Can be seen by selecting **View < Class View**

8.17 Class View and Object Browser

- **Object Browser**
  - Lists all classes in a library
  - Helps developers learn about the functionality of a specific class
  - To view the **Object Browser** select any .NET FCL method and select **Go To Definition**

8.17 Class View and Object Browser

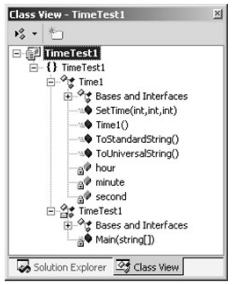


***Fig. 8.20 Class View** of class `Time1` (Fig. 8.1) and class `TimeTest` (Fig. 8.2).*

8.17 Class View and Object Browser

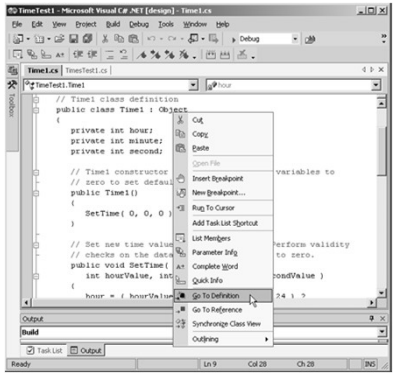


***Fig. 8.21 Object Browser** when user selects `Object` from `Time1.cs`. (part 1)*

8.17 Class View and Object Browser

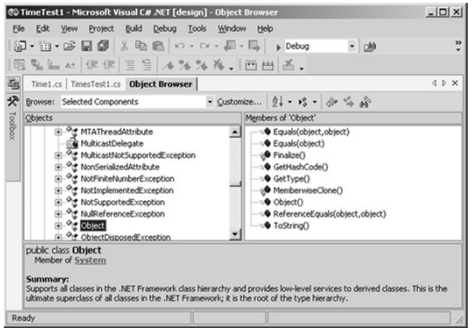


***Fig. 8.21 Object Browser** when user selects `Object` from `Time1.cs`. (part 1)*

فصل نهم

برنامه نویسی شئ گرا

ارث بری (Inheritance)

9.1 مقدمه

- ارث بری:
  - کلاس ها با جذب توابع و متغیرهای یک کلاس موجود ایجاد می شوند
  - سپس توابع خاص خود را به منظور افزایش قابلیت به آن اضافه می کند.
  - این کلاس یک کلاس مشتق شده نامیده می شود زیرا توابع و متغیرها را از یک کلاس پایه به ارث می برد
  - اشیاء کلاس مشتق شده اشیاء کلاس پایه هستند، اما نه بالعکس
  - ارتباط "is a": شی کلاس مشتق شده می تواند به عنوان شی کلاس پایه تلقی شوند.
  - ارتباط "has a": شی کلاس دارای ارجاعات شی به عنوان اعضای می باشد
  - کلاس مشتق شده می تواند تنها به اعضای غیر خصوصی کلاس پایه دسترسی داشته باشید مگر اینکه توابع دسترسی را به ارث ببرد.

81

9.2 کلاسهای پایه و کلاسهای مشتق

- یک شی اغلب یک شی از کلاس دیگری است
- یک کلاس مشتق شئ از کلاس پایه اش است
- ارث بری یک سلسله مراتب درخت مانند را شکل می دهد.
- برای مشخص کردن اینکه کلاس one از کلاس two مشتق شده
  - class one : two
- ترکیب:
  - با ارتباط "has a" شکل داده می شود.
- سازنده ها به ارث برده نمی شوند.

82

9.2 کلاسهای پایه و کلاسهای مشتق

Base class	Derived classes
`Student`	`GraduateStudent` `UndergraduateStudent`

`Shape`	`Circle`
	`Triangle`
	`Rectangle`

`Loan`	`CarLoan`
	`HomeImprovementLoan`
	`MortgageLoan`

`Employee`	`FacultyMember` `StaffMember`
`Account`	`CheckingAccount` `SavingsAccount`

Fig. 9.1 Inheritance examples.

83

9.2 کلاسهای پایه و کلاسهای مشتق

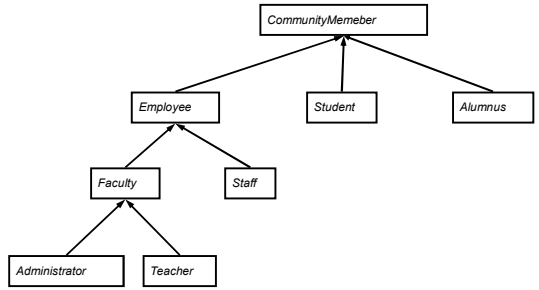


Fig. 9.2 *Inheritance hierarchy for university* `CommunityMembers`.

84

9.2 کلاسهای پایه و کلاسهای مشتق

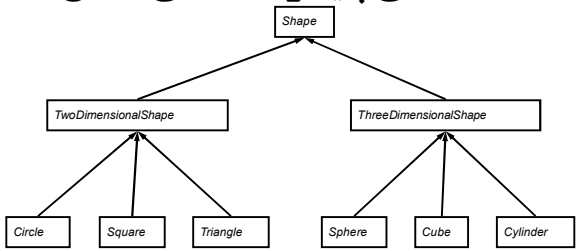


Fig. 9.3 *Portion of a* **`Shape`** *class hierarchy.*

85

9.3 اعضاء `protected` و `internal`

- اعضاء `protected`
  - می توان از کلاس پایه و یا هر کلاس مشتق شده از کلاس پایه به دسترسی پیدا کرد
- اعضاء `internal`
  - تنها می تواند بوسیله کلاسهایی که در یک اسمبلی تعریف شده اند دسترسی پیدا کرد
- Overridden base class members can be accessed:
  - base.member

86

9.4 ارتباط مابین کلاس پایه و کلاس مشتق

- از یک سلسله مراتب نقطه دایره برای نشان رابطه بین پایه و کلاس های مشتق شده استفاده می شود.
- اولین چیزی که یک کلاس مشتق شده می کند این است سازنده کلاس پایه اش را فراخوانی می کند، به طور صریح یا ضمنی
- کلمه کلیدی `override` مورد نیاز است اگر یک تابع کلاس مشتق شده یک تابع کلاس پایه را لغو کند
- اگر یک تابع کلاس پایه باید باطل شود آن را باید به صورت `virtual` اعلان نماییم

87

88

`Point.cs`

```
1   // Fig. 9.4: Point.cs
2   // Point class represents an x-y coordinate pair.
3
4   using System;
5
6   // Point class definition implicitly inherits from Object
7   public class Point
8   {
9      // point coordinates
10     private int x, y;
11
12     // default (no-argument) constructor
13     public Point()
14     {
15        // implicit call to Object constructor occurs here
16     }
17
18     // constructor
19     public Point( int xValue, int yValue )
20     {
21        // implicit call to Object constructor occurs here
22        X = xValue;
23        Y = yValue;
24     }
25
26     // property X
27     public int X
28     {
29        get
30        {
31           return x;
32        }
33
```

X and Y coordinates, declared private so other classes cannot directly access them

Default point constructor with implicit call to Object constructor

Constructor to set coordinates to parameters, also has implicit call to Object constructor

89

```
34         set
35         {
36            x = value; // no need for validation
37         }
38
39      } // end property X
40
41      // property Y
42      public int Y
43      {
44         get
45         {
46            return y;
47         }
48
49         set
50         {
51            y = value; // no need for validation
52         }
53
54      } // end property Y
55
56      // return string representation of Point
57      public override string ToString()
58      {
59         return "[" + x + ", " + y + "]";
60      }
61
62   } // end class Point
```

Definition of overridden method ToString

Point.cs

Program Output

90

```
1    // Fig. 9.5: PointTest.cs
2    // Testing class Point.
3
4    using System;
5    using System.Windows.Forms;
6
7    // PointTest class definition
8    class PointTest
9    {
10      // main entry point for application
11      static void Main( string[] args )
12      {
13         // instantiate Point object
14         Point point = new Point( 72, 115 );
15
16         // display point coordinates via X and Y properties
17         string output = "X coordinate is " + point.X +
18            "\n" + "Y coordinate is " + point.Y;
19
20         point.X = 10; // set x-coordinate via X property
21         point.Y = 10; // set y-coordinate via Y property
22
23         // display new point value
24         output += "\n\nThe new location of point is " + point;
25
26         MessageBox.Show( output, "Demonstrating Class Point" );
27
28      } // end method Main
29
30   } // end class PointTest
```

Create a Point object

Calls the ToString method of class Point implicitly

Change coordinates of Point object

PointTest.cs

91

```
1    // Fig. 9.6: Circle.cs
2    // Circle class contains x-y coordinate pair and radius.
3
4    using System;
5
6    // Circle class definition implicitly inherits from Object
7    public class Circle
8    {
9       private int x, y;    // coordinates of Circle's ce
10      private double radius; // Circle's radius
11
12      // default constructor
13      public Circle()
14      {
15         // implicit call to Object constructor occurs here
16      }
17
18      // constructor
19      public Circle( int xValue, int yValue, double radiusValue )
20      {
21         // implicit call to Object constructor occurs here
22         x = xValue;
23         y = yValue;
24         Radius = radiusValue;
25      }
26
27      // property X
28      public int X
29      {
30         get
31         {
32            return x;
33         }
34
```

Declare coordinates and radius of circle as private

Circle constructors

Circle.cs

92

```
35         set
36         {
37            x = value;  // no need for validation
38         }
39
40      } // end property X
41
42      // property Y
43      public int Y
44      {
45         get
46         {
47            return y;
48         }
49
50         set
51         {
52            y = value;  // no need for validation
53         }
54
55      } // end property Y
56
57      // property Radius
58      public double Radius
59      {
60         get
61         {
62            return radius;
63         }
64
65         set
66         {
67            if ( value >= 0 )   // validation needed
68               radius = value;
69         }
```

Circle.cs

93

Circle.cs

```
70
71      } // end property Radius
72
73      // calculate Circle diameter
74      public double Diameter()
75      {
76         return radius * 2;
77      }
78
79      // calculate Circle circumference
80      public double Circumference()
81      {
82         return Math.PI * Diameter();
83      }
84
85      // calculate Circle area
86      public double Area()
87      {
88         return Math.PI * Math.Pow( radius, 2 );
89      }
90
91      // return string representation of Circle
92      public override string ToString()
93      {
94         return "Center = [" + x + ", " + y + "]" +
95            "; Radius = " + radius;
96      }
97
98     } // end class Circle
```

Definition of overridden method ToString

94

CircleTest.cs

```
1   // Fig. 9.7: CircleTest.cs
2   // Testing class Circle.
3
4   using System;
5   using System.Windows.Forms;
6
7   // CircleTest class definition
8   class CircleTest
9   {
10     // main entry point for application.
11     static void Main( string[] args )
12     {
13        // instantiate Circle
14        Circle circle = new Circle( 37, 43, 2.5 );
15
16        // get Circle's initial x-y coordinates and radius
17        string output = "X coordinate is " + circle.X +
18           "\nY coordinate is " + circle.Y + "\nRadius is " +
19           circle.Radius;
20
21        // set Circle's x-y coordinates and radius to new values
22        circle.X = 2;
23        circle.Y = 2;
24        circle.Radius = 4.25;
25
26        // display Circle's string representation
27        output += "\n\nThe new location and radius of " +
28           "circle are \n" + circle + "\n";
29
30        // display Circle's diameter
31        output += "Diameter is " +
32           String.Format( "{0:F}", circle.Diameter() ) + "\n";
33
```

Create a Circle object

Change coordinates and radius of Circle object

Implicit call to circle's ToString method

95

```
34        // display Circle's circumference
35        output += "Circumference is " +
36           String.Format( "{0:F}", circle.Circumference() ) + "\n";
37
38        // display Circle's area
39        output += "Area is " +
40           String.Format( "{0:F}", circle.Area() );
41
42        MessageBox.Show( output, "Demonstrating Class Circle" );
43
44     } // end method Main
45
46  } // end class CircleTest
```

Call Circle's Circumference and Area methods for output

96

Circle2.cs

```
1   // Fig. 9.8: Circle2.cs
2   // Circle2 class that inherits from cl
3
4   using System;
5
6   // Circle2 class definition inherits from Point
7   class Circle2 : Point
8   {
9      private double radius; // Circle2's radius
10
11     // default constructor
12     public Circle2()
13     {
14        // implicit call to Point constructor occurs here
15     }
16
17     // constructor
18     public Circle2( int xValue, int yValue, double radiusValue )
19     {
20        // implicit call to Point constructor occurs here
21        x = xValue;
22        y = yValue;
23        Radius = radiusValue;
24     }
25
26     // property Radius
27     public double Radius
28     {
29        get
30        {
31           return radius;
32        }
33
```

Declare class Circle to derive from class Point

Declare radius as private

Implicit calls to base class constructor

Attempt to directly change private base class methods results in an error

97

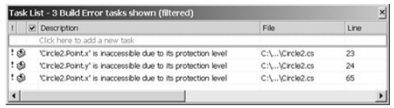
```
34         set
35         {
36            if ( value >= 0 )
37               radius = value;
38         }
39
40      } // end property Radius
41
42      // calculate Circle diameter
43      public double Diameter()
44      {
45         return radius * 2;
46      }
47
48      // calculate Circle circumference
49      public double Circumference()
50      {
51         return Math.PI * Diameter();
52      }
53
54      // calculate Circle area
55      public virtual double area()
56      {
57         return Math.PI * Math.Pow( radius, 2 );
58      }
59
60      // return string representation Circle
61      public override string ToString()
62      {
63         return "Center = [" + x + ", " + y + "]" +
64            "; Radius = " + radius;
65      }
66
67   } // end class Circle2
```

Circle2.cs

Attempt to directly access private base class members results in an error

98

Circle2.cs
program output



99

```
1    // Fig. 9.9: Point2.cs
2    // Point2 class contains an x-y coordinate pair as protected data.
3
4    using System;
5
6    // Point2 class definition implicitly inherits from Object
7    public class Point2
8    {
9       // point coordinate
10      protected int x, y;
11
12      // default constructor
13      public Point2()
14      {
15         // implicit call to Object constructor occurs here
16      }
17
18      // constructor
19      public Point2( int xValue, int yValue )
20      {
21         // implicit call to Object constructor occurs here
22         X = xValue;
23         Y = yValue;
24      }
25
26      // property X
27      public int X
28      {
29         get
30         {
31            return x;
32         }
33
```

Point2.cs

Declare coordinates as protected so derived classes can directly access them

100

```
34         set
35         {
36            x = value; // no need for validation
37         }
38
39      } // end property X
40
41      // property Y
42      public int Y
43      {
44         get
45         {
46            return y;
47         }
48
49         set
50         {
51            y = value; // no need for validation
52         }
53
54      } // end property Y
55
56      // return string representation of Point2
57      public override string ToString()
58      {
59         return "[" + x + ", " + y + "]";
60      }
61
62   } // end class Point2
```

Point2.cs

101

Circle3.cs

```
1    // Fig. 9.10: Circle3.cs
2    // Circle2 class that inherits from class Point2.
3
4    using System;
5
6    // Circle3 class definition inherits from Point2
7    public class Circle3 : Point2
8    {
9       private double radius; // Circle's radius
10
11      // default constructor
12      public Circle3()
13      {
14         // implicit call to Point constructor occurs here
15      }
16
17      // constructor
18      public Circle3(
19         int xValue, int yValue, double radiusValue )
20      {
21         // implicit call to Point constructor occurs here
22         x = xValue;
23         y = yValue;
24         Radius = radiusValue;
25      }
26
27      // property Radius
28      public double Radius
29      {
30         get
31         {
32            return radius;
33         }
34
```

Class Circle3 inherits from Point2

Directly changing protected base class members does not result in error

102

Circle3.cs

```
35         set
36         {
37            if ( value >= 0 )
38               radius = value;
39         }
40
41      } // end property Radius
42
43      // calculate Circle diameter
44      public double Diameter()
45      {
46         return radius * 2;
47      }
48
49      // calculate circumference
50      public double Circumference()
51      {
52         return Math.PI * Diameter();
53      }
54
55      // calculate Circle area
56      public virtual double Area()
57      {
58         return Math.PI * Math.Pow( radius, 2 );
59      }
60
61      // return string representation of Circle3
62      public override string ToString()
63      {
64         return "Center = [" + x + ", " + y + "]" +
65            "; Radius = " + radius;
66      }
67
68   } // end class Circle3
```

Directly accessing protected members does not result in error

103

CircleTest3.cs

```
1    / Fig. 9.11: CircleTest3.cs
2    // Testing class Circle3.
3
4    using System;
5    using System.Windows.Forms;
6
7    // CircleTest3 class definition
8    class CircleTest3
9    {
10      // main entry point for application
11      static void Main( string[] args )
12      {
13         // instantiate Circle3
14         Circle3 circle = new Circle3( 37, 43, 2.5 );
15
16         // get Circle3's initial x-y coordinates and radius
17         string output = "X coordinate is " + circle.X + "\n" +
18            "Y coordinate is " + circle.Y + "\nRadius is " +
19            circle.Radius;
20
21         // set Circle3's x-y coordinates and radius to new values
22         circle.X = 2;
23         circle.Y = 2;
24         circle.Radius = 4.25;
25
26         // display Circle3's string representation
27         output += "\n\n" +
28            "The new location and radius of circle are " +
29            "\n" + circle + "\n";
30
31         // display Circle3's Diameter
32         output += "Diameter is " +
33            String.Format( "{0:F}", circle.Diameter() ) + "\n";
34
```

Change coordinates and radius of Circle3 object

Create new Circle3 object

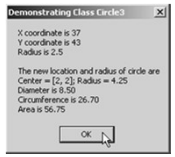
Implicit call to Circle3's ToString method

104

cs

```
35         // display Circle3's Circumference
36         output += "Circumference is " +
37            String.Format( "{0:F}", circle.Circumference() ) + "\n";
38
39         // display Circle3's Area
40         output += "Area is " +
41            String.Format( "{0:F}", circle.Area() );
42
43         MessageBox.Show( output, "Demonstrating Class Circle3" );
44
45      } // end method Main
46
47   } // end class CircleTest3
```

Call Circle's Circumference and Area methods for output



105

Point3.cs

```
1    // Fig. 9.12: Point3.cs
2    // Point3 class represents an x-y coordinate pair.
3
4    using System;
5
6    // Point3 class definition implicitly inherits from Object
7    public class Point3
8    {
9       // point coordinate
10      private int x, y;
11
12      // default constructor
13      public Point3()
14      {
15         // implicit call to Object constructor occurs here
16      }
17
18      // constructor
19      public Point3( int xValue, int yValue )
20      {
21         // implicit call to Object constructor occurs here
22         X = xValue;   // use property X
23         Y = yValue;   // use property Y
24      }
25
26      // property X
27      public int X
28      {
29         get
30         {
31            return x;
32         }
33
```

Declare coordinates as private

106

Point3.cs

```
34         set
35         {
36            x = value; // no need for validation
37         }
38
39      } // end property X
40
41      // property Y
42      public int Y
43      {
44         get
45         {
46            return y;
47         }
48
49         set
50         {
51            y = value; // no need for validation
52         }
53
54      } // end property Y
55
56      // return string representation of Point3
57      public override string ToString()
58      {
59         return "[" + X + ", " + Y + "]";
60      }
61
62   } // end class Point3
```

Methods to set x and y coordinates

Overridden ToString method

107

Circle4.cs

```
1    // Fig. 9.13: Circle4.cs
2    // Circle4 class that inherits from class Point3.
3
4    using System;
5
6    // Circle4 class definition inherits from Point3
7    public class Circle4 : Point3
8    {
9       private double radius;
10
11      // default constructor
12      public Circle4()
13      {
14         // implicit call to Point constructor occurs here
15      }
16
17      // constructor
18      public Circle4( int xValue, int yValue, double radiusValue )
19         : base( xValue, yValue )
20      {
21         Radius = radiusValue;
22      }
23
24      // property Radius
25      public double Radius
26      {
27         get
28         {
29            return radius;
30         }
31
32         set
33         {
34            if ( value >= 0 )   // validation needed
35               radius = value;
```

Constructor with implicit call to base class constructor

Constructor with explicit call to base class constructor

Explicit call to base class constructor

108

Circle4.cs

```
36         }
37
38      } // end property Radius
39
40      // calculate Circle diameter
41      public double Diameter()
42      {
43         return Radius * 2;     // use property Radius
44      }
45
46      // calculate Circle circumference
47      public double Circumference()
48      {
49         return Math.PI * Diameter();
50      }
51
52      // calculate Circle area
53      public virtual double Area()
54      {
55         return Math.PI * Math.Pow( Radius, 2 );  // use propert
56      }
57
58      // return string representation of Circle4
59      public override string ToString()
60      {
61         // use base reference to return Point string representation
62         return "Center= " + base.ToString() +
63            "; Radius = " + Radius;  // use property Radius
64      }
65
66   } // end class Circle4
```

Method area declared virtual so it can be overridden

Circle4's ToString method overrides Point3's ToString method

Call Point3's ToString method to display coordinates

CircleTest4.cs

```
1   // Fig. 9.14: CircleTest4.cs
2   // Testing class Circle4.
3
4   using System;
5   using System.Windows.Forms;
6
7   // CircleTest4 class definition
8   class CircleTest4
9   {
10     // main entry point for application
11     static void Main( string[] args )
12     {
13        // instantiate Circle4
14        Circle4 circle = new Circle4( 37, 43, 2.5 );
15
16        // get Circle4's initial x-y coordinates and radius
17        string output = "X coordinate is " + circle.X + "\n" +
18           "Y coordinate is " + circle.Y + "\n" +
19           "Radius is " + circle.Radius;
20
21        // set Circle4's x-y coordinates and radius to new values
22        circle.X = 2;
23        circle.Y = 2;
24        circle.Radius = 4.25;
25
26        // display Circle4's string representation
27        output += "\n\n" +
28           "The new location and radius of circle are " +
29           "\n" + circle + "\n";
30
31        // display Circle4's Diameter
32        output += "Diameter is " +
33           String.Format( "{0:F}", circle.Diameter() ) + "\n";
34
```

Change coordinates and radius of Circle4 object

Create new Circle4 object

Implicit call to Circle4's ToString method

CS

```
35        // display Circle4's Circumference
36        output += "Circumference is " +
37           String.Format( "{0:F}", circle.Circumference() ) + "\n";
38
39        // display Circle4's Area
40        output += "Area is " +
41           String.Format( "{0:F}", circle.Area() );
42
43        MessageBox.Show( output, "Demonstrating Class Circle4" );
44
45     } // end method Main
46
47  } // end class CircleTest4
```

Call Circle's Circumference and Area methods for output

9.5 مطالعه موردی: سلسله مراتب ارث بری سه سطحی

- مثال ارث بری سه سطحی
  - کلاس **Cylinder** از کلاس **Circle4** ارث می برد
  - کلاس **Circle4** از کلاس **Point3** ارث می برد

Cylinder.cs

```
1   // Fig. 9.15: Cylinder.cs
2   // Cylinder class inherits from class Circle4.
3
4   using System;
5
6   // Cylinder class definition inherits from Circle4
7   public class Cylinder : Circle4
8   {
9      private double height;
10
11     // default constructor
12     public Cylinder()
13     {
14        // implicit call to Circle4 constructor occurs here
15     }
16
17     // four-argument constructor
18     public Cylinder( int xValue, int yValue, double radius,
19        double heightValue ) : base( xValue, yValue, radiusValue )
20     {
21        Height = heightValue; // set Cylinder height
22     }
23
24     // property Height
25     public double Height
26     {
27        get
28        {
29           return height;
30        }
31
32        set
33        {
34           if ( value >= 0 ) // validate height
35              height = value;
```

Class Cylinder inherits from class Circle4

Declare variable height as private

Constructor that implicitly calls base class constructor

Constructor that explicitly calls base class constructor

113

Cylinder.cs

```
36         }
37
38      } // end property Height
39
40      // override Circle4 method Area to calculate Cylinder area
41      public override double Area()
42      {
43         return 2 * base.Area() + base.Circumference() * Height;
44      }
45
46      // calculate Cylinder volume
47      public double Volume()
48      {
49         return base.Area() * Height;
50      }
51
52      // convert Cylinder to string
53      public override string ToString()
54      {
55         return base.ToString() + "; Height = " + Height;
56      }
57
58   } // end class Cylinder
```

Method Area overrides Circle4's Area method

Calculate volume of cylinder

Overridden ToString method

Call Circle4's ToString method to get its output

114

CylinderTest.cs

```
1    // Fig. 9.16: CylinderTest.cs
2    // Tests class Cylinder.
3
4    using System;
5    using System.Windows.Forms;
6
7    // CylinderTest class definition
8    class CylinderTest
9    {
10      // main entry point for application
11      static void Main( string[] args )
12      {
13         // instantiate object of class Cylinder
14         Cylinder cylinder = new Cylinder(12, 23, 2.5, 5.7);
15
16         // properties get initial x-y coordinate, radius and height
17         string output = "X coordinate is " + cylinder.X + "\n" +
18            "Y coordinate is " + cylinder.Y + "\nRadius is " +
19            cylinder.Radius + "\n" + "Height is " + cylinder.Height;
20
21         // properties set new x-y coordinate, radius and height
22         cylinder.X = 2;
23         cylinder.Y = 2;
24         cylinder.Radius = 4.25;
25         cylinder.Height = 10;
26
27         // get new x-y coordinate and radius
28         output += "\n\nThe new location, radius and height of " +
29            "cylinder are\n" + cylinder + "\n\n";
30
31         // display Cylinder's Diameter
32         output += "Diameter is " +
33            String.Format( "{0:F}", cylinder.Diameter() ) + "\n";
34
```

Create new cylinder

Change coordinates, radius and height

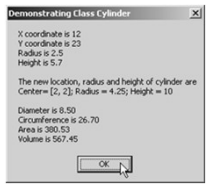
Implicit call to ToString

115

CylinderTest.cs

```
35         // display Cylinder's Circumference
36         output += "Circumference is " +
37            String.Format( "{0:F}", cylinder.Circumference() ) + "\n";
38
39         // display Cylinder's Area
40         output += "Area is " +
41            String.Format( "{0:F}", cylinder.Area() ) + "\n";
42
43         // display Cylinder's Volume
44         output += "Volume is " +
45            String.Format( "{0:F}", cylinder.Volume() );
46
47         MessageBox.Show( output, "Demonstrating Class Cylinder" );
48
49      } // end method Main
50
51   } // end class CylinderTest
```

Call methods Circumference, Area and Volume



9.6 سازنده های و مخرب های کلاس پایه

- نمونه سازی یک کلاس مشتق شده، باعث می شود سازنده کلاس پایه به صورت صریح یا ضمنی فراخوانی شود
- موجب اعمال زنجیره ای می شود اگر کلاس پایه خود یک کلاس مشتق باشد.
- زمانی که مخرب فراخوانی می شود، ابتدا کار خود را انجام داده سپس مخرب کلاس پایه فراخوانی می شود.

116

117

Point4.cs

```
1    // Fig. 9.17: Point4.cs
2    // Point4 class represents an x-y coordinate pair.
3
4    using System;
5
6    // Point4 class definition
7    public class Point4
8    {
9       // point coordinate
10      private int x, y;
11
12      // default constructor
13      public Point4()
14      {
15         // implicit call to Object constructor occurs here
16         Console.WriteLine( "Point4 constructor: {0}", this );
17      }
18
19      // constructor
20      public Point4( int xValue, int yValue )
21      {
22         // implicit call to Object constructor occurs here
23         X = xValue;
24         Y = yValue;
25         Console.WriteLine( "Point4 constructor: {0}", this );
26      }
27
28      // destructor
29      ~Point4()
30      {
31         Console.WriteLine( "Point4 destructor: {0}", this );
32      }
33
34      // property X
35      public int X
```

Constructors with output messages and implicit calls to base class constructor

Output statements use reference this to implicitly call ToString method

Destructor with output message

118

Point4.cs

```
36      {
37         get
38         {
39            return x;
40         }
41
42         set
43         {
44            x = value; // no need for validation
45         }
46
47      } // end property X
48
49      // property Y
50      public int Y
51      {
52         get
53         {
54            return y;
55         }
56
57         set
58         {
59            y = value; // no need for validation
60         }
61
62      } // end property Y
63
64      // return string representation of Point4
65      public override string ToString()
66      {
67         return "[" + x + ", " + y + "]";
68      }
69
70   } // end class Point4
```

119

Circle5.cs

```
1    // Fig. 9.18: Circle5.cs
2    // Circle5 class that inherits from class Point4.
3
4    using System;
5
6    // Circle5 class definition inherits from Point4
7    public class Circle5 : Point4
8    {
9       private double radius;
10
11      // default constructor
12      public Circle5()
13      {
14         // implicit call to Point3 constructor occurs here
15         Console.WriteLine( "Circle5 constructor: {0}", this );
16      }
17
18      // constructor
19      public Circle5( int xValue, int yValue, double radiusValue )
20         : base( xValue, yValue )
21      {
22         Radius = radiusValue;
23         Console.WriteLine( "Circle5 constructor: {0}", this );
24      }
25
26      // destructor overrides version in class Point4
27      ~Circle5()
28      {
29         Console.WriteLine( "Circle5 destructor: {0}", this );
30      }
31
32      // property Radius
33      public double Radius
34      {
```

Constructors with calls to base class and output statements

Output statements use reference this to implicitly call ToString method

Destructor with output message

120

Circle5.cs

```
35         get
36         {
37            return radius;
38         }
39
40         set
41         {
42            if ( value >= 0 )
43               radius = value;
44         }
45
46      } // end property Radius
47
48      // calculate Circle5 diameter
49      public double Diameter()
50      {
51         return Radius * 2;
52      }
53
54      // calculate Circle5 circumference
55      public double Circumference()
56      {
57         return Math.PI * Diameter();
58      }
59
60      // calculate Circle5 area
61      public virtual double Area()
62      {
63         return Math.PI * Math.Pow( Radius, 2 );
64      }
65
66      // return string representation of Circle5
67      public override string ToString()
68      {
```

121

```
69          // use base reference to return Point3 string
70          return "Center = " + base.ToString() +
71             "; Radius = " + Radius;
72       }
73
74    } // end class Circle5
```

Circle5.cs

122

```
1    // Fig. 9.19: ConstructorAndDestructor.cs
2    // Display order in which base-class and derived-class constructors
3    // and destructors are called.
4
5    using System;
6
7    // ConstructorAndFinalizer class definition
8    class ConstructorAndFinalizer
9    {
10      // main entry point for application.
11      static void Main( string[] args )
12      {
13         Circle5 circle1, circle2;
14
15         // instantiate objects
16         circle1 = new Circle5( 72, 29, 4.5 );
17         circle2 = new Circle5( 5, 5, 10 );
18
19         Console.WriteLine();
20
21         // mark objects for garbage collection
22         circle1 = null;
23         circle2 = null;
24
25         // inform garbage collector to execute
26         System.GC.Collect();
27
28      } // end method Main
29
30   } // end class ConstructorAndDestructor
```

Create two objects of type Circle5

Remove references to Circle5 objects

Run the garbage collector

ConstructorAndDestructor.cs

123

```
Point4 constructor: Center = [72, 29]; Radius = 0
Circle5 constructor: Center = [72, 29]; Radius = 4.5
Point4 constructor: Center = [5, 5]; Radius = 0
Circle5 constructor: Center = [5, 5]; Radius = 10

Circle5 destructor: Center = [5, 5]; Radius = 10
Point4 destructor: Center = [5, 5]; Radius = 10
Circle5 destructor: Center = [72, 29]; Radius = 4.5
Point4 destructor: Center = [72, 29]; Radius = 4.5
```

ConstructorAndDestructor.cs program output

فصل دهم

برنامه نویسی شئ گرا

Polymorphism

124

10.1 مقدمه

- Polymorphism اجازه می دهد به برنامه نویس برای:
  - برنامه هایی که با تنوع گسترده ای از کلاس های مرتبط به شیوه ای عمومی به کار می برد.
  - سیستم هایی که به راحتی قابل توسعه اند

10.2 تبدیل شئ کلاس مشتق به شئ کلاس پایه

- سلسله مراتب کلاسها
  - می توان شی های کلاس مشتق را به ارجاع های کلاس پایه انتساب داد.
  - می توان به صراحت ما بین سلسله مراتب کلاسها تبدیل انجام داد
- یک شئ کلاس مشتق می تواند به عنوان شئ کلاس پایه تلقی شود
  - آرایه ای از ارجاع های کلاس پایه که به اشیایی از نوع های زیادی از کلاس مشتق ارجاع می کند.
  - شی کلاس پایه یک شی از کلاس مشتق نیست

Point.cs

Definition of class Point

```
1   // Fig. 10.1: Point.cs
2   // Point class represents an x-y coordinat
3
4   using System;
5
6   // Point class definition implicitly inherits from Object
7   public class Point
8   {
9      // point coordinate
10     private int x, y;
11
12     // default constructor
13     public Point()
14     {
15        // implicit call to Object constructor occurs here
16     }
17
18     // constructor
19     public Point( int xValue, int yValue )
20     {
21        // implicit call to Object constructor occurs here
22        X = xValue;
23        Y = yValue;
24     }
25
26     // property X
27     public int X
28     {
29        get
30        {
31           return x;
32        }
33
```


Point.cs

```
34        set
35        {
36           x = value;  // no need for validation
37        }
38
39     } // end property X
40
41     // property Y
42     public int Y
43     {
44        get
45        {
46           return y;
47        }
48
49        set
50        {
51           y = value;  // no need for validation
52        }
53
54     } // end property Y
55
56     // return string representation of Point
57     public override string ToString()
58     {
59        return "[" + X + ", " + Y + "]";
60     }
61
62  } // end class Point
```

129

```
1    // Fig. 10.2: Circle.cs
2    // Circle class that inherits from Point
3
4    using System;
5
6    // Circle class definition inherits from Point
7    public class Circle : Point
8    {
9       private double radius; // circle's radius
10
11      // default constructor
12      public Circle()
13      {
14         // implicit call to Point constructor occurs here
15      }
16
17      // constructor
18      public Circle( int xValue, int yValue, double radiusValue )
19         : base( xValue, yValue )
20      {
21         Radius = radiusValue;
22      }
23
24      // property Radius
25      public double Radius
26      {
27         get
28         {
29            return radius;
30         }
31
```

Definition of class Circle which inherits from class Point

Circle.cs

130

```
32         set
33         {
34            if ( value >= 0 ) // validate radius
35               radius = value;
36         }
37
38      } // end property Radius
39
40      // calculate Circle diameter
41      public double Diameter()
42      {
43         return Radius * 2;
44      }
45
46      // calculate Circle circumference
47      public double Circumference()
48      {
49         return Math.PI * Diameter();
50      }
51
52      // calculate Circle area
53      public virtual double Area()
54      {
55         return Math.PI * Math.Pow( Radius, 2 );
56      }
57
58      // return string representation of Circle
59      public override string ToString()
60      {
61         return "Center = " + base.ToString() +
62            "; Radius = " + Radius;
63      }
64
65   } // end class Circle
```

Circle.cs

131

```
1    // Fig. 10.3: PointCircleTest.cs
2    // Demonstrating inheritance and polymorphism.
3
4    using System;
5    using System.Windows.Forms;
6
7    // PointCircleTest class definition
8    class PointCircleTest
9    {
10      // main entry point for application.
11      static void Main( string[] args )
12      {
13         Point point1 = new Point( 30, 50 );
14         Circle circle1 = new Circle( 120, 89, 2.7 );
15
16         string output = "Point point1: " + point1.ToString() +
17            "\nCircle circle1: " + circle1.ToString();
18
19         // use 'is a' relationship to assign
20         // Circle circle1 to Point reference
21         Point point2 = circle1;
22
23         output += "\n\nCCircle circle1 (via point2): " +
24            point2.ToString();
25
26         // downcast (cast base-class reference to derived-class
27         // data type) point2 to Circle circle2
28         Circle circle2 = ( Circle ) point2;
29
30         output += "\n\nCircle circle1 (via circle2): " +
31            circle2.ToString();
32
33         output += "\nArea of circle1 (via circle2): " +
34            circle2.Area().ToString( "F" );
35
```

Create a Point object

Assign a Point reference to reference a Circle object

Assign a Circle reference to reference a Point object (downcast)

Use base-class reference to access a derived-class object

Use derived-class reference to access a base-class object

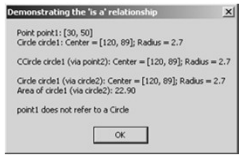
PointCircleTest.cs

132

```
36         // attempt to assign point1 object to Circle reference
37         if ( point1 is Circle )
38         {
39            circle2 = ( Circle ) point1;
40            output += "\n\ncast successful";
41         }
42         else
43         {
44            output += "\n\npoint1 does not refer to a Circle";
45         }
46
47         MessageBox.Show( output,
48            "Demonstrating the 'is a' relationship" );
49
50      } // end method Main
51
52   } // end class PointCircleTest
```

Test if point1 references a Circle object – it cannot, because of the downcasting on line 28

PointCircleTest.cs



Program Output

10.3 فیلدهای نوع و دستور switch

- استفاده از **switch** برای تایین نوع شئ
  - تمایز بین شی، انجام اقدام مناسب بسته به نوع شی.
- مشکلات بالقوه با استفاده از **switch**
  - برنامه نویس ممکن است آزمون نوع را فراموش کند.
  - برنامه نویس ممکن است تست کردن تمامی حالتهای ممکن را فراموش کند.
  - زمانی که نوع جدیدی افزوده می شود، برنامه نویس ممکن است اصلاح تمام ساختارهای **switch** مرتبط را فراموش کند.
  - هر افزودن یا حذف کردن کلاس اصلاح هر دستور switch در سیستم را نیاز دارد؛ این رد یابی زمان بر بوده و مستعد خطا می باشد.

133

10.4 مثال Polymorphism

- **Quadrilateral** base-class (چهار ضلعی)
  - **Rectangle** derived-class (مستطیل)
  - **Square** derived-class (مربع)
  - **Parallelogram** derived-class (متوازی الاضلاع)
  - **Trapezoid** derived-class (ذوزنقه)
- تابع محیط (perimeter) ممکن است در تمام کلاسها به کار برده شود.
- با ارجاع **Quadrilateral**، C# بصورت polymorphical تابع صحیح را در کلاس مشتق از اشیای نمونه سازی شده انتخاب کند.

134

10.4 مثال Polymorphism

- **SpaceObject** base-class – contains method **DrawYourself**
  - **Martian** derived-class (implements **DrawYourself**)
  - **Venutian** derived-class (implements **DrawYourself**)
  - **Plutonian** derived-class (implements **DrawYourself**)
  - **SpaceShip** derived-class (implements **DrawYourself**)
- برنامه مدیریت صفحه نمایش ممکن است شامل آرایه **SpaceObject** باشد که ارجاع به اشیای متنوعی می باشد که از **SpaceObject** مشتق شده اند
- برای بازسازی صفحه نمایش، مدیریت صفحه نمایش تابع **DrawYourself** هر شی در آرایه را فراخوانی می کند.
- برنامه بصورت polymorphical نسخه مناسب از **DrawYourself** را در هر شی، بر اساس نوع آن شی فراخوانی می کند.

135

10.5 Abstract Classes and Methods

- Abstract classes
  - نمی تواند نمونه سازی شود.
  - به عنوان کلاس پایه استفاده می شود.
  - تعریف کلاس کامل نیست – کلاس مشتق باید بخشهای ناقص را تعریف کند.
  - می تواند شامل abstract method ها و/یا abstract property باشد
    - پیاده سازی ندارند.
    - کلاس مشتق باید abstract method ها و abstract property ارث بری شده را برای نمونه سازی شدن لغو (override) کند.

136

10.5 Abstract Classes and Methods

- Abstract class ها برای فراهم کردن کلاس های پایه مناسب از کلاسهای دیگری که ممکن است ارث بری شوند (concrete classes) استفاده می شوند.
- کلاس های پایه Abstract بیش از حد برای تعریف اشیای حقیقی عمومی می باشند.
- برای تعریف abstract class، هنگام اعلان از کلمه کلیدی **abstract** استفاده می شود.
- برای اعلان تابع یا خصوصیت abstract، کلمه کلیدی **abstract** در اعلان استفاده می شود؛ تابع یا خصوصیت abstract پیاده سازی ندارند.

10.5 Abstract Classes and Methods

- Concrete class ها کلمه کلیدی **override** را برای فراهم کردن پیاده سازی از تمام توابع یا خصوصیات abstract از کلاس پایه استفاده می کنند.
- هر کلاس با یک تابع یا خصوصیت abstract باید **abstract** اعلان شوند.
- با این حال که abstract classes نمی توانند نمونه سازی شوند، ما می توانیم ارجاع abstract classes را برای رجوع به نمونه های هر concrete class مشتق شده از abstract class استفاده کنیم.

10.6 مطالعه موردی: ارث بری رابط و پیاده سازی

- Abstract base class **Shape**
  - Concrete **virtual** method **Area** (default return value is 0)
  - Concrete **virtual** method **Volume** (default return value is 0)
  - **Abstract** read-only property **Name**
- Class **Point2** inherits from **Shape**
  - Overrides property **Name** (required)
  - Does NOT override methods **Area** and **Volume**
- Class **Circle2** inherits from **Point2**
  - Overrides property **Name**
  - Overrides method **Area**, but not **Volume**
- Class **Cylinder2** inherits from **Circle2**
  - Overrides property **Name**
  - Overrides methods **Area** and **Volume**

Shape.cs

```
1   // Fig. 10.4: Shape.cs
2   // Demonstrate a shape hierarchy using an abstract base class.
3   using System;
4
5   public abstract class Shape
6   {
7      // return Shape's area
8      public virtual double Area()
9      {
10        return 0;
11     }
12
13     // return Shape's volume
14     public virtual double Volume()
15     {
16        return 0;
17     }
18
19     // return Shape's name
20     public abstract string Name
21     {
22        get;
23     }
24  }
```

Declaration of virtual methods Area and Volume with default implementations

Declaration of read-only abstract property Name; implementing classes will have to provide an implementation for this property

Declaration of abstract class Shape

141

Point2.cs

```
1    // Fig. 10.5: Point2.cs
2    // Point2 inherits from abstract class Shape and represents
3    // an x-y coordinate pair.
4    using System;
5
6    // Point2 inherits from abstract class Shape
7    public class Point2 : Shape
8    {
9       private int x, y; // Point2 coordinates
10
11      // default constructor
12      public Point2()
13      {
14         // implicit call to Object constructor occurs here
15      }
16
17      // constructor
18      public Point2( int xValue, int yValue )
19      {
20         X = xValue;
21         Y = yValue;
22      }
23
24      // property X
25      public int X
26      {
27         get
28         {
29            return x;
30         }
31
32         set
33         {
34            x = value; // no validation needed
35         }
```

Class Point2 inherits from class Shape

142

Point2.cs

```
36      }
37
38      // property Y
39      public int Y
40      {
41         get
42         {
43            return y;
44         }
45
46         set
47         {
48            y = value; // no validation needed
49         }
50      }
51
52      // return string representation of Point2 object
53      public override string ToString()
54      {
55         return "[" + X + ", " + Y + "]";
56      }
57
58      // implement abstract property Name of class Shape
59      public override string Name
60      {
61         get
62         {
63            return "Point2";
64         }
65      }
66
67   } // end class Point2
```

Point2's implementation of the read-only Name property

143

Circle2.cs

```
1    // Fig. 10.6: Circle2.cs
2    // Circle2 inherits from class Point2 and overrides key members.
3    using System;
4
5    // Circle2 inherits from class Point2
6    public class Circle2 : Point2
7    {
8       private double radius; // Circle2 radius
9
10      // default constructor
11      public Circle2()
12      {
13         // implicit call to Point2 constructor occurs here
14      }
15
16      // constructor
17      public Circle2( int xValue, int yValue, double radiusValue )
18         : base( xValue, yValue )
19      {
20         Radius = radiusValue;
21      }
22
23      // property Radius
24      public double Radius
25      {
26         get
27         {
28            return radius;
29         }
30
```

Definition of class Circle2 which inherits from class Point2

144

Circle2.cs

```
31         set
32         {
33            // ensure non-negative radius value
34            if ( value >= 0 )
35               radius = value;
36         }
37      }
38
39      // calculate Circle2 diameter
40      public double Diameter()
41      {
42         return Radius * 2;
43      }
44
45      // calculate Circle2 circumference
46      public double Circumference()
47      {
48         return Math.PI * Diameter();
49      }
50
51      // calculate Circle2 area
52      public override double Area()
53      {
54         return Math.PI * Math.Pow( Radius, 2 );
55      }
56
57      // return string representation of Circle2 object
58      public override string ToString()
59      {
60         return "Center = " + base.ToString() +
61            "; Radius = " + Radius;
62      }
```

Override the Area method (defined in class Shape)

145

Circle2.cs

```
63
64      // override property Name from class Point2
65      public override string Name
66      {
67         get
68         {
69            return "Circle2";
70         }
71      }
72
73   } // end class Circle2
```

Override the read-only Name property

146

Cylinder2.cs

```
1    // Fig. 10.7: Cylinder2.cs
2    // Cylinder2 inherits from class Circle2 and overrides key members.
3    using System;
4
5    // Cylinder2 inherits from class Circle2
6    public class Cylinder2 : Circle2
7    {
8       private double height; // Cylinder2 height
9
10      // default constructor
11      public Cylinder2()
12      {
13         // implicit call to Circle2 constructor occurs here
14      }
15
16      // constructor
17      public Cylinder2( int xValue, int yValue, double radiusValue,
18         double heightValue ) : base( xValue, yValue, radiusValue )
19      {
20         Height = heightValue;
21      }
22
23      // property Height
24      public double Height
25      {
26         get
27         {
28            return height;
29         }
30
31         set
32         {
33            // ensure non-negative height value
34            if ( value >= 0 )
35               height = value;
```

Class Cylinder2 derives from Circle2

147

Cylinder2.cs

```
36          }
37       }
38
39       // calculate Cylinder2 area
40       public override double Area()
41       {
42          return 2 * base.Area() + base.Circumference() * Height;
43       }
44
45       // calculate Cylinder2 volume
46       public override double Volume()
47       {
48          return base.Area() * Height;
49       }
50
51       // return string representation of Circle2 object
52       public override string ToString()
53       {
54          return base.ToString() + "; Height = " + Height;
55       }
56
57       // override property Name from class Circle2
58       public override string Name
59       {
60          get
61          {
62             return "Cylinder2";
63          }
64       }
65
66    } // end class Cylinder2
```

Override read-only property Name

Override Area implementation of class Circle2

Override Volume implementation of class Shape

148

AbstractShapesTest.cs

```
1    // Fig. 10.8: AbstractShapesTest.cs
2    // Demonstrates polymorphism in Point-Circle-Cylinder hierarchy.
3    using System;
4    using System.Windows.Forms;
5
6    public class AbstractShapesTest
7    {
8       public static void Main( string[] args )
9       {
10         // instantiate Point2, Circle2 and Cyli
11         Point2 point = new Point2( 7, 11 );
12         Circle2 circle = new Circle2( 22, 8, 3.5 );
13         Cylinder2 cylinder = new Cylinder2( 10, 10, 3.3, 10 );
14
15         // create empty array of Shape base-class references
16         Shape[] arrayOfShapes = new Shape[ 3 ];
17
18         // arrayOfShapes[ 0 ] refers to Point2 object
19         arrayOfShapes[ 0 ] = point;
20
21         // arrayOfShapes[ 1 ] refers to Circle2 object
22         arrayOfShapes[ 1 ] = circle;
23
24         // arrayOfShapes[ 1 ] refers to Cylinder2 object
25         arrayOfShapes[ 2 ] = cylinder;
26
27         string output = point.Name + ": " + point + "\n" +
28            circle.Name + ": " + circle + "\n" +
29            cylinder.Name + ": " + cylinder;
30
```

Assign a Shape reference to reference a Point2 object

Assign a Shape reference to reference a Cylinder2 object

reference a Circle2 object

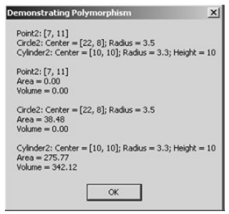
149

```
31          // display Name, Area and Volume for each object
32          // in arrayOfShapes polymorphically
33          foreach( Shape shape in arrayOfShapes )
34          {
35             output += "\n\n" + shape.Name + ": " + shape +
36                "\nArea = " + shape.Area().ToString( "F" ) +
37                "\nVolume = " + shape.Volume().ToString( "F" );
38          }
39
40          MessageBox.Show( output, "Demonstrating Polymorphism" );
41       }
42    }
```

AbstractShapesTest.cs

Use a foreach loop to access every element of the array

Rely on polymorphism to call appropriate version of methods



Program Output

10.7 توابع و کلاسهای sealed

- **sealed** یک کلمه کلیدی در C# می باشد
- توابع **sealed** نمی توانند در کلاس مشتق لغو شوند.
  - توابعی که بصورت **static** یا **private** تعریف می شوند، ذاتاً sealed هستند.
- کلاسهای **sealed** نمی توانند هیچ کلاس مشتقی داشته باشند.
- ایجاد کلاس **sealed** می تواند به برخی بهینه سازی های زمان اجرا (runtime optimizations) اجازه دهد.
  - به عنوان مثال، فراخوانی تابع **virtual** می تواند به فراخوانی تابع غیر **virtual**تبدیل شود.

150

10.8 مطالعه موردی: سیستم حقوقی استفاده کننده از Polymorphism

- Base-class **Employee**
  - **abstract**
  - **abstract** method **Earnings** (درآمد)
- کلاسهایی که از کلاس Employee مشتق می شوند
  - **Boss**
  - **CommissionWorker**
  - **PieceWorker**
  - **HourlyWorker**
- تمام کلاسهای مشتق تابع **Earnings** را پیاده سازی می کنند
- برنامه مدیریت کننده ارجاع های Employee را برای ارجاع به نمونه های کلاس مشتق استفاده می کند.
- Polymorphism نسخه صحیحی از **Earnings** را فراخوانی می کند.

151

152

```
1   // Fig. 10.9: Employee.cs
2   // Abstract base class for company employees.
3   using System;
4
5   public abstract class Employee
6   {
7      private string firstName;
8      private string lastName;
9
10     // constructor
11     public Employee( string firstNameValue,
12        string lastNameValue )
13     {
14        FirstName = firstNameValue;
15        LastName = lastNameValue;
16     }
17
18     // property FirstName
19     public string FirstName
20     {
21        get
22        {
23           return firstName;
24        }
25
26        set
27        {
28           firstName = value;
29        }
30     }
31
```

Employee.cs

Definition of abstract class Employee

153

```
32      // property LastName
33      public string LastName
34      {
35         get
36         {
37            return lastName;
38         }
39
40         set
41         {
42            lastName = value;
43         }
44      }
45
46      // return string representation of Employee
47      public override string ToString()
48      {
49         return FirstName + " " + LastName;
50      }
51
52      // abstract method that must be implemented for each derived
53      // class of Employee to calculate specific earnings
54      public abstract decimal Earnings();
55
56   } // end class Employee
```

Declaration of abstract class Earnings – implementation must be provided by all derived classes

Employee.cs

154

```
1    // Fig. 10.10: Boss.cs
2    // Boss class derived from Employee.
3    using System;
4
5    public class Boss : Employee
6    {
7       private decimal salary; // Boss's salary
8
9       // constructor
10      public Boss( string firstNameValue, string lastNameValue,
11         decimal salaryValue)
12         : base( firstNameValue, lastNameValue )
13      {
14         WeeklySalary = salaryValue;
15      }
16
17      // property WeeklySalary
18      public decimal WeeklySalary
19      {
20         get
21         {
22            return salary;
23         }
24
25         set
26         {
27            // ensure positive salary value
28            if ( value > 0 )
29               salary = value;
30         }
31      }
32
```

Definition of class Boss – derives from Employee

Boss.cs

155

```
33      // override base-class method to calculate Boss's earnings
34      public override decimal Earnings()
35      {
36         return WeeklySalary;
37      }
38
39      // return string representation of Boss
40      public override string ToString()
41      {
42         return "Boss: " + base.ToString();
43      }
44   }
```

Implementation of method Earnings (required by classes deriving from Employee)

Boss.cs

156

```
1    // Fig. 10.11: CommisionWorker.cs
2    // CommissionWorker class derived from Employee
3    using System;
4
5    public class CommissionWorker : Employee
6    {
7       private decimal salary;     // base weekly salary
8       private decimal commission; // amount paid per item sold
9       private int quantity;       // total items sold
10
11      // constructor
12      public CommissionWorker( string firstNameValue,
13         string lastNameValue, decimal salaryValue,
14         decimal commissionValue, int quantityValue )
15         : base( firstNameValue, lastNameValue )
16      {
17         WeeklySalary = salaryValue;
18         Commission = commissionValue;
19         Quantity = quantityValue;
20      }
21
22      // property WeeklySalary
23      public decimal WeeklySalary
24      {
25         get
26         {
27            return salary;
28         }
29
```

Definition of class CommissionWorker – derives from Employee

CommisionWorker.cs

157

```
30         set
31         {
32            // ensure non-negative salary value
33            if ( value > 0 )
34               salary = value;
35         }
36      }
37
38      // property Commission
39      public decimal Commission
40      {
41         get
42         {
43            return commission;
44         }
45
46         set
47         {
48            // ensure non-negative commission value
49            if ( value > 0 )
50               commission = value;
51         }
52      }
53
54      // property Quantity
55      public int Quantity
56      {
57         get
58         {
59            return quantity;
60         }
61
```

CommisionWorker.cs

158

```
62         set
63         {
64            // ensure non-negative quantity value
65            if ( value > 0 )
66               quantity = value;
67         }
68      }
69
70      // override base-class method to calculate
71      // CommissionWorker's earnings
72      public override decimal Earnings()
73      {
74         return WeeklySalary + Commission * Quantity;
75      }
76
77      // return string representation of CommissionWorker
78      public override string ToString()
79      {
80         return "CommissionWorker: " + base.ToString();
81      }
82
83   } // end class CommissionWor
```

CommisionWorker.cs

Implementation of method Earnings (required by classes deriving from Employee)

159

```
1    // Fig. 10.12: PieceWorker.cs
2    // PieceWorker class derived from Employee.
3    using System;
4
5    public class PieceWorker : Employee
6    {
7       private decimal wagePerPiece; // wage per piece produced
8       private int quantity;         // quantity of pieces produced
9
10      // constructor
11      public PieceWorker( string firstNameValue,
12         string lastNameValue, decimal wagePerPieceValue,
13         int quantityValue )
14         : base( firstNameValue, lastNameValue )
15      {
16         WagePerPiece = wagePerPieceValue;
17         Quantity = quantityValue;
18      }
19
20      // property WagePerPiece
21      public decimal WagePerPiece
22      {
23         get
24         {
25            return wagePerPiece;
26         }
27
28         set
29         {
30            if ( value > 0 )
31               wagePerPiece = value;
32         }
33      }
34
```

PieceWorker.cs

Definition of class PieceWorker – derives from Employee

160

```
35      // property Quantity
36      public int Quantity
37      {
38         get
39         {
40            return quantity;
41         }
42
43         set
44         {
45            if ( value > 0 )
46               quantity = value;
47         }
48      }
49
50      // override base-class method to calculate
51      // PieceWorker's earnings
52      public override decimal Earnings()
53      {
54         return Quantity * WagePerPiece;
55      }
56
57      // return string representation of PieceWorker
58      public override string ToString()
59      {
60         return "PieceWorker: " + base.ToString();
61      }
62   }
```

PieceWorker.cs

Implementation of method Earnings (required by classes deriving from Employee)

HourlyWorker.cs

```
1    // Fig. 10.13: HourlyWorker.cs
2    // HourlyWorker class derived from Employee.
3    using System;
4
5    public class HourlyWorker : Employee
6    {
7       private decimal wage;       // wage per hour of work
8       private double hoursWorked; // hours worked during week
9
10      // constructor
11      public HourlyWorker( string firstNameValue, string LastNameValue,
12         decimal wageValue, double hoursWorkedValue )
13         : base( firstNameValue, LastNameValue )
14      {
15         Wage = wageValue;
16         HoursWorked = hoursWorkedValue;
17      }
18
19      // property Wage
20      public decimal Wage
21      {
22         get
23         {
24            return wage;
25         }
26
27         set
28         {
29            // ensure non-negative wage value
30            if ( value > 0 )
31               wage = value;
32         }
33      }
34
```

Definition of class HourlyWorker – derives from Employee

HourlyWorker.cs

```
35      // property HoursWorked
36      public double HoursWorked
37      {
38         get
39         {
40            return hoursWorked;
41         }
42
43         set
44         {
45            // ensure non-negative hoursWorked value
46            if ( value > 0 )
47               hoursWorked = value;
48         }
49      }
50
51      // override base-class method to calculate
52      // HourlyWorker earnings
53      public override decimal Earnings()
54      {
55         // compensate for overtime (paid "time-and-a-half")
56         if ( HoursWorked <= 40 )
57         {
58            return Wage * Convert.ToDecimal( HoursWorked );
59         }
60
61         else
62         {
63            // calculate base and overtime pay
64            decimal basePay = Wage * Convert.ToDecimal( 40 );
65            decimal overtimePay = Wage * 1.5M *
66               Convert.ToDecimal( HoursWorked - 40 );
67
```

Implementation of method Earnings (required by classes deriving from Employee)

HourlyWorker.cs

```
68            return basePay + overtimePay;
69         }
70      }
71
72      // return string representation of HourlyWorker
73      public override string ToString()
74      {
75         return "HourlyWorker: " + base.ToString();
76      }
77   }
```


EmployeesTest.cs

```
1    // Fig. 10.14: EmployeesTest.cs
2    // Demonstrates polymorphism by displaying earnings
3    // for various Employee types.
4    using System;
5    using System.Windows.Forms;
6
7    public class EmployeesTest
8    {
9       public static void Main( string[] args )
10      {
11         Boss boss = new Boss( "John", "Smith",
12
13         CommissionWorker commissionWorker =
14            new CommissionWorker( "Sue", "Jones", 400, 3, 150 );
15
16         PieceWorker pieceWorker = new PieceWorker( "Bob", "Lewis",
17            Convert.ToDecimal( 2.5 ), 200 );
18
19         HourlyWorker hourlyWorker = new HourlyWorker( "Karen",
20            "Price", Convert.ToDecimal( 13.75 ), 50 );
21
22         Employee employee = boss;
23
24         string output = GetString( employee ) + boss + " earned " +
25               boss.Earnings().ToString( "C" ) + "\n\n";
26
27         employee = commissionWorker;
28
29         output += GetString( employee ) + commissionWorker +
30            " earned " +
31            commissionWorker.Earnings().ToString( "C" ) + "\n\n";
32
33         employee = pieceWorker;
34
```

Assign Employee reference to reference a Boss object

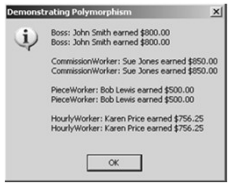
Use method GetString to polymorphically obtain salary information. Then use the original Boss reference to obtain the information

```
35          output += GetString( employee ) + pieceWorker +
36             " earned " + pieceWorker.Earnings().ToString( "C" ) +
37             "\n\n";
38
39          employee = hourlyWorker;
40
41          output += GetString( employee ) + hourlyWorker +
42             " earned " + hourlyWorker.Earnings().ToString( "C" ) +
43             "\n\n";
44
45          MessageBox.Show( output, "De
46             MessageBoxButtons.OK, Mes
47
48       } // end method Main
49
50       // return string that contains Employee information
51       public static string GetString( Employee worker )
52       {
53          return worker.ToString() + " earned " +
54             worker.Earnings().ToString( "C" ) + "\n";
55       }
56    }
```

Definition of method GetString, which takes as an argument an Employee object.

Polymorphically call the method of the appropriate derived class

EmployeesTest.cs



Program Output

10.9 مطالعه موردی: ایجاد و استفاده رابط

- رابط ها سرویسهای عمومی (تابع ها و خصوصیات) را مشخص می کنند که کلاسها باید پیاده سازی کنند.
- رابط ها پیاده سازی پیشفرضی را ارائه نمی دهند در مقابل کلاسهای abstract که ممکن است برخی پیاده سازی های پیشفرضی را ارائه کنند.
- رابط ها برای "گرد هم آوردن" یا ارتباط اشیاء متفاوت که که تنها از طریق رابط با یکدیگر ارتباط دارند استفاده می شوند.

10.9 مطالعه موردی: ایجاد و استفاده رابط

- رابط با کلمه کلیدی **interface** تعریف می شود
- از نشانه گذاری ارث بری برای تایین اینکه یک کلاس یک رابط را پیاده سازی می کند استفاده می شود. (*ClassName* : *InterfaceName*)
- یک کلاس ممکن است بیش از یک رابط را پیاده سازی کند (کاما لیست رابط ها را از همدیگر جدا می کند)
- کلاسهایی که یک رابط را پیاده سازی می کنند، باید هر تابع یا خصوصیت تعریف شده در رابط را نیز پیاده سازی کنند.
- به عنوان مثال: رابط **IAge** ی که اطلاعاتی درباره سن اشیا برگشت می دهد.
  - می تواند بوسیله کلاسهای people، cars، trees استفاده شود (همه دارای سن می باشند)


```
1    // Fig. 10.15: IAge.cs
2    // Interface IAge declares property for setting and getting age.
3
4    public interface IAge
5    {
6       int Age { get; }
7       string Name { get; }
8    }
```

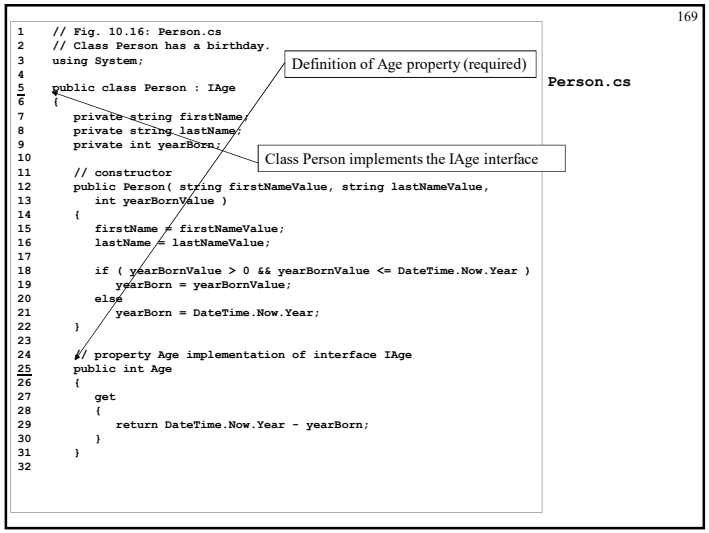
IAge.cs

Definition of interface IAge

Classes implementing this interface will have to define read-only properties Age and Name

169

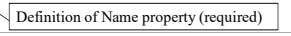
Person.cs




```
1    // Fig. 10.16: Person.cs
2    // Class Person has a birthday.
3    using System;
4
5    public class Person : IAge
6    {
7       private string firstName;
8       private string lastName;
9       private int yearBorn;
10
11      // constructor
12      public Person( string firstNameValue, string lastNameValue,
13         int yearBornValue )
14      {
15         firstName = firstNameValue;
16         lastName = lastNameValue;
17
18         if ( yearBornValue > 0 && yearBornValue <= DateTime.Now.Year )
19            yearBorn = yearBornValue;
20         else
21            yearBorn = DateTime.Now.Year;
22      }
23
24      // property Age implementation of interface IAge
25      public int Age
26      {
27         get
28         {
29            return DateTime.Now.Year - yearBorn;
30         }
31      }
32
```

170

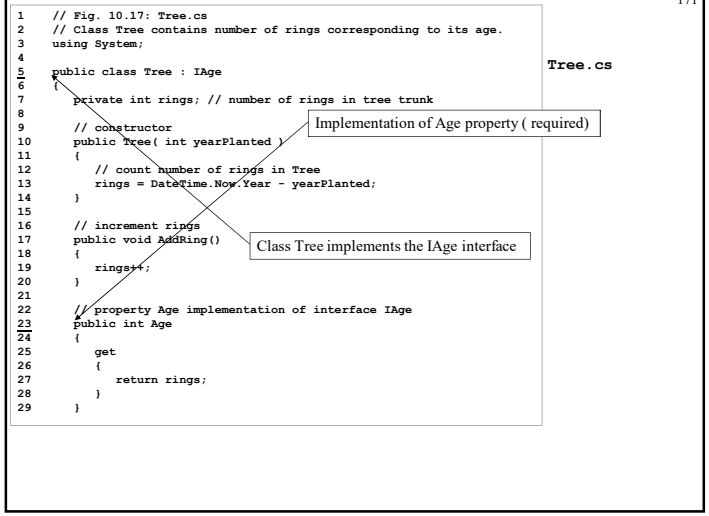
Person.cs




```
33      // property Name implementation of interface IAge
34      public string Name
35      {
36         get
37         {
38            return firstName + " " + lastName;
39         }
40      }
41
42   } // end class Person
```

171

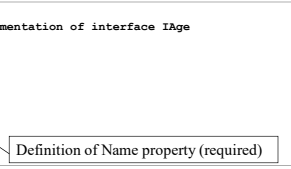
Tree.cs




```
1    // Fig. 10.17: Tree.cs
2    // Class Tree contains number of rings corresponding to its age.
3    using System;
4
5    public class Tree : IAge
6    {
7       private int rings; // number of rings in tree trunk
8
9       // constructor
10      public Tree( int yearPlanted )
11      {
12         // count number of rings in Tree
13         rings = DateTime.Now.Year - yearPlanted;
14      }
15
16      // increment rings
17      public void AddRing()
18      {
19         rings++;
20      }
21
22      // property Age implementation of interface IAge
23      public int Age
24      {
25         get
26         {
27            return rings;
28         }
29      }
```

172

Tree.cs




```
30
31      // property Name implementation of interface IAge
32      public string Name
33      {
34         get
35         {
36            return "Tree";
37         }
38      }
39
40   } // end class Tree
```

173

InterfacesTest.cs

```
1   // Fig. 10.18: InterfacesTest.cs
2   // Demonstrating polymorphism with interfaces.
3   using System.Windows.Forms;
4
5   public class InterfacesTest
6   {
7      public static void Main( string[] args )
8      {
9         Tree tree = new Tree( 1978 );
10        Person person = new Person( "Bob", "Jones", 1971 );
11
12        // create array of IAge references
13        IAge[] iAgeArray = new IAge[ 2 ];
14
15        // iAgeArray[ 0 ] refers to Tree object polymorphically
16        iAgeArray[ 0 ] = tree;
17
18        // iAgeArray[ 1 ] refers to Person object polymorphically
19        iAgeArray[ 1 ] = person;
20
21        // display tree information
22        string output = tree + ": " + tree.Name + "\nAge is " +
23           tree.Age + "\n\n";
24
25        // display person information
26        output += person + ": " + person.Name + "\nAge is: "
27           + person.Age + "\n\n";
28
```

Create array of IAge references

Assign an IAge reference to reference a Person object

reference a Tree object

174

InterfacesTest.cs

```
29        // display name and age for each IAge object in iAgeArray
30        foreach ( IAge ageReference in iAgeArray )
31        {
32           output += ageReference.Name + ": Age is " +
33              ageReference.Age + "\n";
34        }
35
36        MessageBox.Show( output, "Demonstrating Polymorphism" );
37
38     } // end method Main
39
40  } // end class InterfacesTest
```

Use foreach loop t… each element of the array

Use polymorphism to call the property of the appropriate class

Program Output

175

IShape.cs

```
1   // Fig. 10.19: IShape.cs
2   // Interface IShape for Point, Circle, Cylinder Hierarchy.
3
4   public interface IShape
5   {
6      // classes that implement IShape must implement these methods
7      // and this property
8      double Area();
9      double Volume();
10     string Name { get; }
11  }
```

Definition of IShape interface

Classes implementing the interface must define methods Area and Volume (each of which take no arguments and return a double) and a read-only string property Name

176

Point3.cs

```
1   // Fig. 10.20: Point3.cs
2   // Point3 implements interface IShape and represents
3   // an x-y coordinate pair.
4   using System;
5
6   // Point3 implements IShape
7   public class Point3 : IShape
8   {
9      private int x, y; // Point3 coordinates
10
11     // default constructor
12     public Point3()
13     {
14        // implicit call to Object constructor occurs here
15     }
16
17     // constructor
18     public Point3( int xValue, int yValue )
19     {
20        X = xValue;
21        Y = yValue;
22     }
23
24     // property X
25     public int X
26     {
27        get
28        {
29           return x;
30        }
31
```

Class Point3 implements the IShape interface

177

```
32          set
33          {
34             x = value;
35          }
36       }
37
38       // property Y
39       public int Y
40       {
41          get
42          {
43             return y;
44          }
45
46          set
47          {
48             y = value;
49          }
50       }
51
52       // return string representation of Point3 object
53       public override string ToString()
54       {
55          return "[" + X + ", " + Y + "]";
56       }
57
58       // implement interface IShape method Area
59       public virtual double Area()
60       {
61          return 0;
62       }
63
```

Point3.cs

Implementation of IShape method Area (required), declared virtual to allow deriving classes to override

178

```
64       // implement interface IShape method Volume
65       public virtual double Volume()
66       {
67          return 0;
68       }
69
70       // implement property Name of IShape
71       public virtual string Name
72       {
73          get
74          {
75             return "Point3";
76          }
77       }
78
79    } // end class Point3
```

Point3.cs

Implementation of IShape method Volume (required), declared virtual to allow deriving classes to override

Implementation of IShape property Name (required), declared virtual to allow deriving classes to override

179

```
1     // Fig. 10.21: Circle3.cs
2     // Circle3 inherits from class Point3 and overrides key members.
3     using System;
4
5     // Circle3 inherits from class Point3
6     public class Circle3 : Point3
7     {
8        private double radius; // Circle3 radius
9
10       // default constructor
11       public Circle3()
12       {
13          // implicit call to Point3 constructor occurs here
14       }
15
16       // constructor
17       public Circle3( int xValue, int yValue, double radiusValue )
18          : base( xValue, yValue )
19       {
20          Radius = radiusValue;
21       }
22
23       // property Radius
24       public double Radius
25       {
26          get
27          {
28             return radius;
29          }
30
```

Circle3.cs

Definition of class Circle3 which inherits from Point3

180

```
31          set
32          {
33             // ensure non-negative Radius value
34             if ( value >= 0 )
35                radius = value;
36          }
37       }
38
39       // calculate Circle3 diameter
40       public double Diameter()
41       {
42          return Radius * 2;
43       }
44
45       // calculate Circle3 circumference
46       public double Circumference()
47       {
48          return Math.PI * Diameter();
49       }
50
51       // calculate Circle3 area
52       public override double Area()
53       {
54          return Math.PI * Math.Pow( Radius, 2 );
55       }
56
57       // return string representation of Circle3 object
58       public override string ToString()
59       {
60          return "Center = " + base.ToString() +
61             "; Radius = " + Radius;
62       }
63
```

Circle3.cs

Override the Point3 implementation of Area

181

```
64      // override property Name from class Point3
65      public override string Name
66      {
67         get
68         {
69            return "Circle3";
70         }
71      }
72
73   } // end class Circle3
```

Circle3.cs

Override the Point3 implementation of Name

182

```
1    // Fig. 10.22: Cylinder3.cs
2    // Cylinder3 inherits from class Circle2 and overrides key members.
3    using System;
4
5    // Cylinder3 inherits from class Circle3
6    public class Cylinder3 : Circle3
7    {
8       private double height; // Cylinder3 height
9
10      // default constructor
11      public Cylinder3()
12      {
13         // implicit call to Circle3 constructor occurs here
14      }
15
16      // constructor
17      public Cylinder3( int xValue, int yValue, double radiusValue,
18         double heightValue ) : base( xValue, yValue, radiusValue )
19      {
20         Height = heightValue;
21      }
22
23      // property Height
24      public double Height
25      {
26         get
27         {
28            return height;
29         }
30
```

Cylinder3.cs

Declaration of class Cylinder3 which inherits from class Circle3

183

```
31         set
32         {
33            // ensure non-negative Height
34            if ( value >= 0 )
35               height = value;
36         }
37      }
38
39      // calculate Cylinder3 area
40      public override double Area()
41      {
42         return 2 * base.Area() + base.Circumference() * Height;
43      }
44
45      // calculate Cylinder3 volume
46      public override double Volume()
47      {
48         return base.Area() * Height;
49      }
50
51      // return string representation of Cylinder3 object
52      public override string ToString()
53      {
54         return "Center = " + base.ToString() +
55            "; Height = " + Height;
56      }
57
```

Cylinder3.cs

Override the Point3 implementation of Volume

Override the Circle3 implementation of Area

184

```
58      // override property Name from class Circle3
59      public override string Name
60      {
61         get
62         {
63            return "Cylinder3";
64         }
65      }
66
67   } // end class Cylinder3
```

Cylinder3.cs

Override the Circle3 implementation of Name

185

```
1    // Fig. 10.23: Interfaces2Test.cs
2    // Demonstrating polymorphism with interfaces in
3    // Point-Circle-Cylinder hierarchy.
4
5    using System.Windows.Forms;
6
7    public class Interfaces2Test
8    {
9       public static void Main( string[] args )
10      {
11         // instantiate Point3, Circle3 and Cylin
12         Point3 point = new Point3( 7, 11 );
13         Circle3 circle = new Circle3( 22, 8, 3.5
14         Cylinder3 cylinder = new Cylinder3( 10, 1
15
16         // create array of IShape references
17         IShape[] arrayOfShapes = new IShape[ 3 ];
18
19         // arrayOfShapes[ 0 ] references Point3 object
20         arrayOfShapes[ 0 ] = point;
21
22         // arrayOfShapes[ 1 ] references Circle3 object
23         arrayOfShapes[ 1 ] = circle;
24
25         // arrayOfShapes[ 2 ] references Cylinder3 object
26         arrayOfShapes[ 2 ] = cylinder;
27
28         string output = point.Name + ": " + point + "\n" +
29            circle.Name + ": " + circle + "\n" +
30            cylinder.Name + ": " + cylinder;
31
```

Create an array of IShape references

Assign an IShape reference to reference a Point3 object

Assign an IShape reference to reference a Circle3 object

Assign an IShape reference to reference a Cylinder3 object

Interfaces2Test.cs

186

```
32         foreach ( IShape shape in arrayOfShapes )
33         {
34            output += "\n\n" + shape.Name + ":\nArea = " +
35               shape.Area() + "\nVolume = " + shape.Volume();
36         }
37
38         MessageBox.Show( output, "Demonstrating Polymorphism" );
39      }
40   }
```

Use polymorphism to call the appropriate class's method or property

Interfaces2Test.cs

Program Output

10.10 نماینده ها (Delegates)

- بعضی مواقع ارسال توابع به عنوان آرگومان به توابع دیگر مفید است.
- نماینده ها مجموعه ای از ارجاع ها به توابع می باشد.
- شی نماینده می تواند به تابع ارسال شود؛ توابع سپس می توانند شی نماینده تابع را برای ارجاع به کار ببرند.
- نماینده هایی که شامل یک تابع می باشند به عنوان *singlecast delegates* شناخته می شوند و از کلاس **Delegate** ایجاد یا مشتق می شوند.
- نماینده هایی که شامل چندین تابع می باشند به عنوان *multicast delegates* شناخته می شوند و از کلاس **MulticastDelegate** ایجاد و یا مشتق می شوند.

187

10.10 نماینده ها (Delegates)

- نماینده ها باید قبل از استفاده اعلان شوند.
- اعلان نماینده پارامترها و نوع برگشتی را تابعی را که به آن می تواند ارجاع کند را مشخص می کند.
- توابعی که می توان به آنها با استفاده از نماینده رجوع کرد باید امضایی همانند نماینده داشته باشد.
- نمونه های نماینده سپس می تواند برای رجوع به توابع ایجاد شود.
- یک نمونه نماینده ایجاد شده
- هنگامی که یک نمونه نماینده ایجاد می شود، تابعی که به آن اشاره می کند می توانید فراخوانی شود.

188

189

DelegateBubbleSort.cs

```
1    // Fig. 10.24: DelegateBubbleSort.cs
2    // Demonstrating delegates for sorting numbers.
3
4    public class DelegateBubbleSort
5    {
6       public delegate bool Comparator( int element1,
7          int element2 );
8
9       // sort array using Comparator delegate
10      public static void SortArray( int[] array,
11         Comparator Compare )
12      {
13         for ( int pass = 0; pass < array.Length; pass++ )
14
15            for ( int i = 0; i < array.Length
16
17               if ( Compare( array[ i ], array
18               Swap( ref array[ i ], ref array[ i + 1 ] );
19      }
20
21      // swap two elements
22      private static void Swap( ref int firstElement,
23         ref int secondElement )
24      {
25         int hold = firstElement;
26         firstElement = secondElement;
27         secondElement = hold;
28      }
29   }
```

Call delegate method to compare array elements

Method Swap, swaps the two arguments (passed by reference)

Delegate Comparator definition declaration; defines a delegate to a method that takes two integer parameters and returns a boolean

Method SortArray which takes an integer array and a Comparator delegate

190

BubbleSortForm.cs

```
1    // Fig. 10.25: BubbleSortForm.cs
2    // Demonstrates bubble sort using delegates to determine
3    // the sort order.
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9
10   public class BubbleSortForm : System.Windows.Forms.Form
11   {
12      private System.Windows.Forms.TextBox originalTextBox;
13      private System.Windows.Forms.TextBox sortedTextBox;
14      private System.Windows.Forms.Button createButton;
15      private System.Windows.Forms.Button ascendingButton;
16      private System.Windows.Forms.Button descendingButton;
17      private System.Windows.Forms.Label originalLabel;
18      private System.Windows.Forms.Label sortedLabel;
19
20      private int[] elementArray = new int[ 10 ];
21
22      // create randomly generated set of numbers to sort
23      private void createButton_Click( object sender,
24         System.EventArgs e )
25      {
26         // clear TextBoxes
27         originalTextBox.Clear();
28         sortedTextBox.Clear();
29
30         // create random-number generator
31         Random randomNumber = new Random();
32
```

191

BubbleSortForm.cs

```
33         // populate elementArray with random integers
34         for ( int i = 0; i < elementArray.Length; i++ )
35         {
36            elementArray[ i ] = randomNumber.Next( 100 );
37            originalTextBox.Text += elementArray[ i ] + "\r\n";
38         }
39      }
40
41      // delegate implementation for ascending sort
42      private bool SortAscending( int element1, int element2 )
43      {
44         return element1 > element2;
45      }
46
47      // sort randomly generated numbers in ascendi
48      private void ascendingButton_Click( object sender,
49         System.EventArgs e )
50      {
51         // sort array, passing delegate for SortAsc
52         DelegateBubbleSort.SortArray( elementArray,
53            new DelegateBubbleSort.Comparator(
54               SortAscending ) );
55
56         DisplayResults();
57      }
58
59      // delegate implementation for descending sort
60      private bool SortDescending( int element1, int element2 )
61      {
62         return element1 < element2;
63      }
64
```

To sort in ascending order, send a delegate for the SortAscending method to method SortArray

Method Sort… the first argument is smaller then the second; returns false otherwise

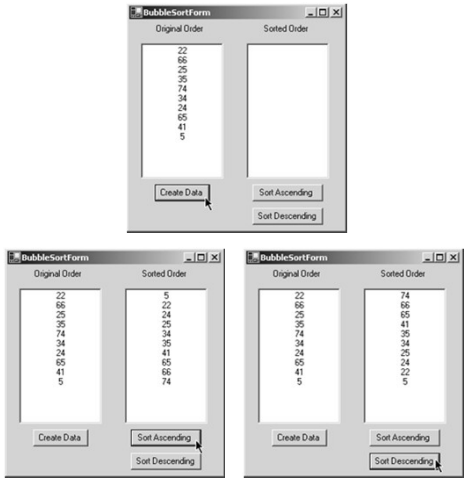
Method SortAscending returns true if the first argument is larger then the second; returns false otherwise

192

BubbleSortForm.cs

```
65      // sort randomly generating numbers in descending order
66      private void descendingButton_Click( object sender,
67         System.EventArgs e )
68      {
69         // sort array, passing delegate for SortDescending
70         DelegateBubbleSort.SortArray( elementArray,
71            new DelegateBubbleSort.Comparator(
72               SortDescending ) );
73
74         DisplayResults();
75      }
76
77      // display the sorted array in sortedTextBox
78      private void DisplayResults()
79      {
80         sortedTextBox.Clear();
81
82         foreach ( int element in elementArray
83            sortedTextBox.Text += element + "\r\n";
84      }
85
86      // main entry point for application
87      public static void Main( string[] args )
88      {
89         Application.Run( new BubbleSortForm() );
90      }
91   }
```

To sort in descending order, send a delegate to the SortDescending method to method SortArray



BubbleSortForm.cs
Program Output

10.11 سربار گذاری عملگرها

- C# شامل عملگر های زیادی می باشد (مانند + و –) که برای بعضی انواع داده اولیه تعریف شده اند.
- برخی مواقع استفاده از عملگرها برای نوع های تعریف شده کاربر مفید می باشد (به عنوان مثال، کلاس اعداد مختلط)
- نماد اپراتور اغلب ممکن است بیشتر فرا خوانی متد قابل فهم باشد.
- C# به برنامه نویسان اجازه سربار گذاری عملگر ها را می دهد تا آنها را به زمینه ای که در آنها استفاده می شود حساس کنند.

10.11 سربار گذاری عملگرها

- متدها عملکردی که باید عملگرهای سربار گذاری شده اعمال شود را مشخص می کنند.
- آنها به فرم زیر تعریف می شوند
  - **public static** *ReturnType* **operator** *operator-to-be-overloaded(arguments)*
- این متد ها باید **public** و **static** اعلان شوند.
- *ReturnType*، نوع برگشتی است که به عنوان نتیجه ارزیابی عملیات می باشد.
- کلمه کلیدی operator در ادامه نوع برگشتی می آید تا مشخص کند که این متد یک سربار گذاری عملگر را تعریف می کند.
- آخرین بخش از اطلاعات عملگری می باشد که سربار گذاری خواهد شد (مانند +، -، * و غیره)
- اگر عملگر یکانی است، یک آرگومان باید مشخص شود، اگر عملگر دودویی است، دو آرگومان، و غیره

ComplexNumber.cs

```
1   // Fig. 10.26: ComplexNumber.cs
2   // Class that overloads operators for adding, subtracting
3   // and multiplying complex numbers.
4
5   public class ComplexNumber
6   {
7      private int real;
8      private int imaginary;
9
10     // default constructor
11     public ComplexNumber() {}
12
13     // constructor
14     public ComplexNumber( int a, int b )
15     {
16        Real = a;
17        Imaginary = b;
18     }
19
20     // return string representation of ComplexNumber
21     public override string ToString()
22     {
23        return "( " + real +
24           ( imaginary < 0 ? " - " + ( imaginary * -1 ) :
25           " + "  + imaginary ) + "i )";
26     }
27
28     // property Real
29     public int Real
30     {
31        get
32        {
33           return real;
34        }
35
```

Property Real provides access to the real part of the complex number

Class ComplexNumber definition

197

ComplexNumber.cs

```
36          set
37          {
38             real = value;
39          }
40
41       } // end property Real
42
43       // property Imaginary
44       public int Imaginary
45       {
46          get
47          {
48             return imaginary;
49          }
50
51          set
52          {
53             imaginary = value;
54          }
55
56       } // end property Imaginary
57
58       // overload the addition operator
59       public static ComplexNumber operator + (
60          ComplexNumber x, ComplexNumber y )
61       {
62          return new ComplexNumber(
63             x.Real + y.Real, x.Imaginary + y.Imaginary );
64       }
65
```

Overload the addition (+) operator for ComplexNumbers.

Property Imaginary provides access to the imaginary part of a complex number

198

ComplexNumber.cs

```
66       // provide alternative to overloaded + operator
67       // for addition
68       public static ComplexNumber Add( ComplexNumber x,
69          ComplexNumber y )
70       {
71          return x + y;
72       }
73
74       // overload the subtraction operator
75       public static ComplexNumber operator - (
76          ComplexNumber x, ComplexNumber y )
77       {
78          return new ComplexNumber(
79             x.Real - y.Real, x.Imaginary - y.Imaginary );
80       }
81
82       // provide alternative to overloaded - operator
83       // for subtraction
84       public static ComplexNumber Subtract( ComplexNumber x,
85          ComplexNumber y )
86       {
87          return x - y;
88       }
89
90       // overload the multiplication operat
91       public static ComplexNumber operator
92          ComplexNumber x, ComplexNumber y )
93       {
94          return new ComplexNumber(
95             x.Real * y.Real - x.Imaginary * y.Imaginary,
96             x.Real * y.Imaginary + y.Real * x.Imaginary );
97       }
```

Method Subtract – provides an alternative to the subtraction operator

Overloads the multiplication (*) operator for ComplexNumbers

Method Add – provides an alternative to the addition operator

Overload the subtraction (-) operator for ComplexNumbers

199

ComplexNumber.cs

```
98
99       // provide alternative to overloaded * operator
100      // for multiplication
101      public static ComplexNumber Multiply( ComplexNumber x,
102         ComplexNumber y )
103      {
104         return x * y;
105      }
106
107   } // end class ComplexNumber
```

Method Multiply – provides an alternative to the multiplication operator

200

OperatorOverloading.cs

```
1    // Fig 10.27: OperatorOverloading.cs
2    // An example that uses operator overloading
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10
11   public class ComplexTest : System.Windows.Forms.Form
12   {
13      private System.Windows.Forms.Label realLabel;
14      private System.Windows.Forms.Label imaginaryLabel;
15      private System.Windows.Forms.Label statusLabel;
16
17      private System.Windows.Forms.TextBox realTextBox;
18      private System.Windows.Forms.TextBox imaginaryTextBox;
19
20      private System.Windows.Forms.Button firstButton;
21      private System.Windows.Forms.Button secondButton;
22      private System.Windows.Forms.Button addButton;
23      private System.Windows.Forms.Button subtractButton;
24      private System.Windows.Forms.Button multiplyButton;
25
26      private ComplexNumber x = new ComplexNumber();
27      private ComplexNumber y = new ComplexNumber();
28
29      [STAThread]
30      static void Main()
31      {
32         Application.Run( new ComplexTest() );
33      }
34
```

```
35    private void firstButton_Click(
36       object sender, System.EventArgs e )
37    {
38       x.Real = Int32.Parse( realTextBox.Text );
39       x.Imaginary = Int32.Parse( imaginaryTextBox.Text );
40       realTextBox.Clear();
41       imaginaryTextBox.Clear();
42       statusLabel.Text = "First Complex Number is: " + x;
43    }
44
45    private void secondButton_Click(
46       object sender, System.EventArgs e )
47    {
48       y.Real = Int32.Parse( realTextBox.Text );
49       y.Imaginary = Int32.Parse( imaginaryTextBox.Text );
50       realTextBox.Clear();
51       imaginaryTextBox.Clear();
52       statusLabel.Text = "Second Complex Number is: " + y;
53    }
54
55    // add complex numbers
56    private void addButton_Click( object sender, System.EventArgs e )
57    {
58       statusLabel.Text = x + " + " + y + " = " + ( x + y );
59    }
60
61    // subtract complex numbers
62    private void subtractButton_Click(
63       object sender, System.EventArgs e )
64    {
65       statusLabel.Text = x + " - " + y + " = " + ( x - y );
66    }
67
```

OperatorOverloading.cs

Use addition operator to add two ComplexNumbers

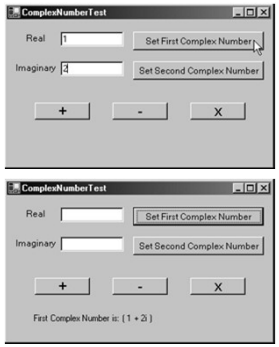
Use subtraction operator to subtract two ComplexNumbers


```
68    // multiply complex numbers
69    private void multiplyButton_Click(
70       object sender, System.EventArgs e )
71    {
72       statusLabel.Text = x + " * " + y + " = " + ( x * y );
73    }
74
75 } // end class ComplexTest
```

OperatorOverloading.cs

Use multiplication operator to multiply two ComplexNumbers

Program Output



OperatorOverloading.cs

Program Output