Nesne Tabanlı Programlama

Giriş

- Nesne sınıfları (Object Classes) veri ve metodları içerir
- Nesneler diğer nesnelerden bilgi gizleyebilirler
- Methodlar : programlama üniteleridir
- User-defined type: programcı tarafından yazılan sınıf
- Sınıflar
  - Bilgi üyeleri - Data members (member variable veya instance variables)
  - Metodlar - Methods that manipulate the data members

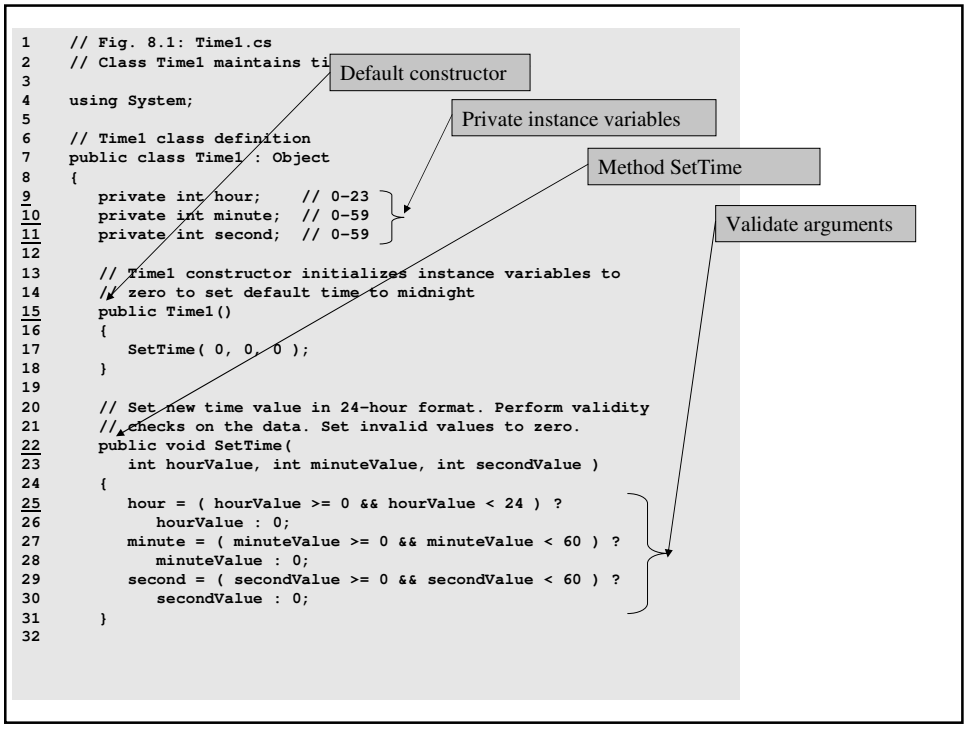
sahiptirler

Time isimli bir sınıf oluşturma

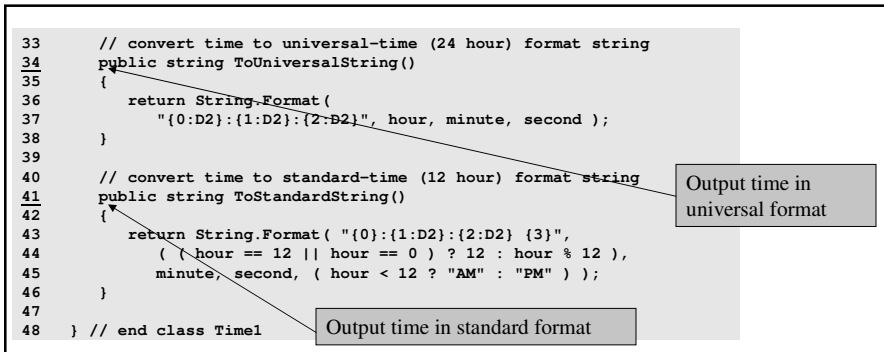
- Açılan sol parantez ({) ve kapatılan sağ parantez (}) sınıfın gövde kısmını belirler
- Değişkenler tanımlamaları *instance variables* olarak isimlendirilir
- *Member Access Modifiers (Erişim Değiştiricileri)*
  - *public* : bu sınıfa ait örnek değişken nerne olursa olsun heryerden erişime açık tanımlama yapar
  - *private :* sadece bu sınıf içinden erişime açık olarak tanımlama yapar

Time isimli bir sınıf oluşturma

- Erişim metodları: okuma veya ekrana yazma
- Constructor (Sınıf yapılandırıcısı)
  - Başlatma işlemini yapar
  - Parametre alabilirler
  - Değer dömdürmezler
  - Bir sınıfta birden fazla constructor olabilir (overloaded constructors)
- Bir sınıfı örneklemek (instantiate) için **new** operatörü kullanılır
- Bir projeye yeni bir sınıf eklemek için **Project < Add Class** kullanılır




```
1    // Fig. 8.1: Time1.cs
2    // Class Time1 maintains ti
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
```




```
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
```

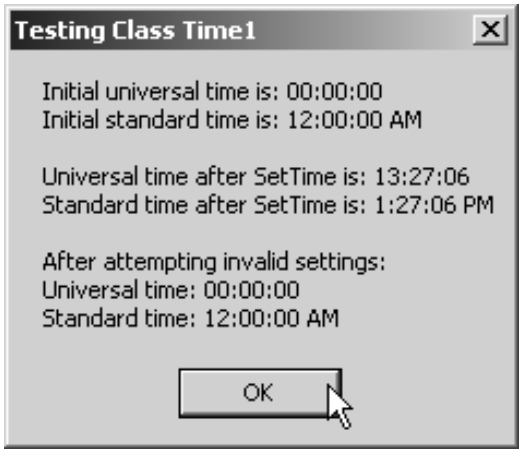
```
1    // Fig. 8.2: TimeTest1.cs
2    // Demonstrating class Time1.
3
4    using System;
5    using System.Windows.Forms;
6
7    // TimeTest1 uses creates and uses a Time1 object
8    class TimeTest1
9    {
10      // main entry point for application
11      static void Main( string[] args )
12      {
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

```
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
```



Class Scope

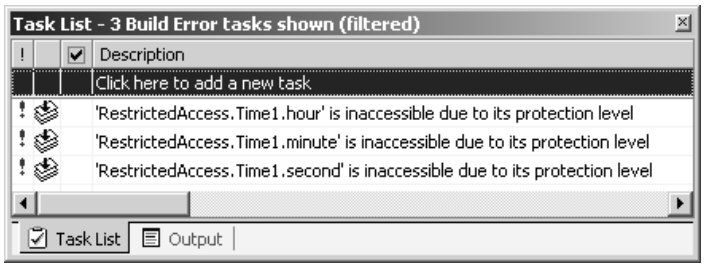
- Bir sınıf kendi içerisindeki tüm üyelere metodları ile adını kullanarak erişebilir
- Sınıf dışında, üyeler doğrudan adıyla referans edilemezler (çağrılamazlar), public üyeler nokta ile birlikte çağrılabilirler (*referenceName.memberName*)

Üyelere Erişim Denetimi (Controlling Access to Members)

- Public metodlar sınıfın sunduğu işlemler için kullanılır
- Metodlar sadece bir işi gerçekleştirmelidir
  - Eğer bir metod başka bir işleme ihtiyaç duyarsa yardımcı metod kullanılmalıdır
  - Sınıfı kullananlar sadece public metoda erişmelidir ve yardımcı metod **private** tanımlanmalıdır
- Properties
  - **get** accessor : bir bilgiyi okuma işlemini yapar
  - **set** access : bir bilgiyi değiştirme işlemini yapar

```
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



Sınıf Nesnelerini Başlatma: Constructors

- Sınıfların örnekleri constructor'lar ile başlatılır
- Constructor'lar bir nesnenin örnek değişkenlerini başlatırlar
- Overloaded constructor'lar bir nesnenin farklı şekillerde başlatılması için kullanılır
- Constructor içerisinde tüm değişkenler başlatılmasa bile tüm değişkenlere bir başlangıç değeri atanır
  - Primitive numeric tipindeki değişkenler 0 yapılır
  - Boolean tipindeki değişkenler false yapılır
  - Reference tipleri ise null yapılır
- Eğer bir sınıfın constructor metodu yoksa, bir default constructor sağlanır
  - Kodu olmaz ve parametre almaz

```
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

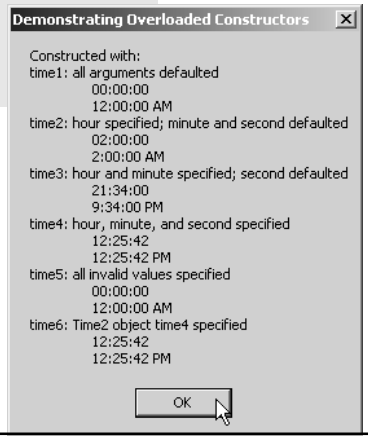
Constructor which takes the hour and minute as input

```
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
```

```
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

```
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
```



Özellikler (Properties)

- Public özellikler kullanıcıya aşağıdaki işlemleri sağlar:
  - Get: private datayı elde etmeyi sağlar
  - Set: private data'ya değer atamayı sağlar
- Get accessor
  - Data'nın formatını kontrol eder veya değiştirir
- Set accessor
  - Yeni değerin uygunluğunu denetler

```
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
```

```
34      // Time3 constructor: hour, minute and se
35      public Time3( int hour, int minute, int s
36      {
37         SetTime( hour, minute, second );
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
```

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

Property Hour

```
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
```

Property Second

Property Minute

```
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
```

```
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
```

```
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
```

```
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
```

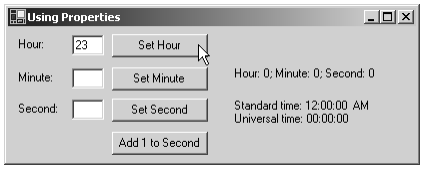
Set Minute property of Time3 object

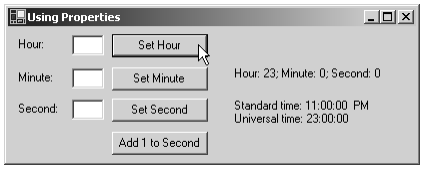
Set Second property of Time3 object

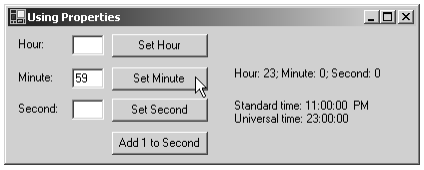
Set Hour property of Time3 object

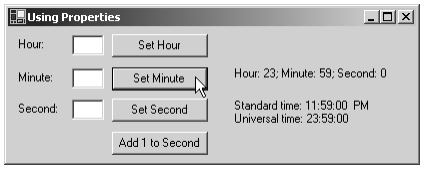
Add 1 second to Time3 object

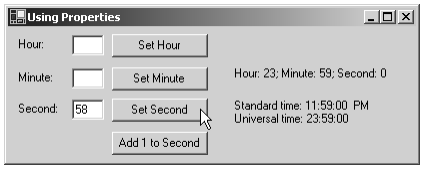
```
98          if ( time.Second == 0 )
99          {
100            time.Minute = ( time.Minute + 1 ) % 60;
101
102            if ( time.Minute == 0 )
103               time.Hour = ( time.Hour + 1 ) % 24;
104         }
105
106         UpdateDisplay();
107      }
108
109   } // end class TimeTest3
```

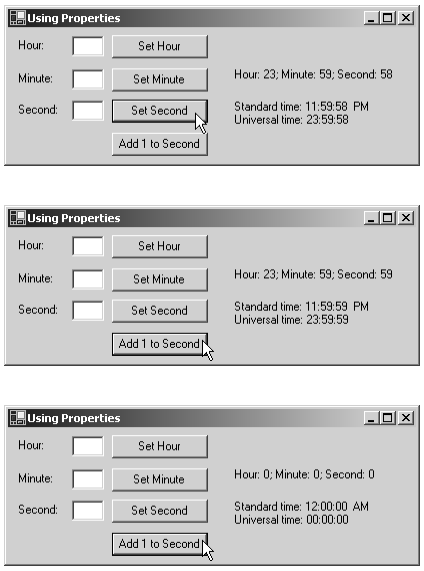












Composition: Diğer Sınıfları Instance Değişkenlerle Çağırma

- Software Reuse – Varolan bir nesnenin referans ile kullanımı kolay ve hızlıdır
- User-defined tipler instance değişkenleri olarak kullanılır

```
1    // Fig. 8.8: Date.cs
2    // Date class definition e
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
```

Constructor that receives the month, day and year arguments. Arguments are validated; if they are not valid, the corresponding member is set to a default value

```
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
```

Validate that the given month can have a given day number

```
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
```

Two Date objects are members of the Employee class

Constructor that initializes the employee's name, birth date and hire date

```
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
```

```
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
```

this keyword'ün kullanımı

- Her nesne kendisini **this** keyword ile gösterebilir
- Genellikle metodların değişkenleriyle instance değişkenlerini ayırmak için kullanılır

```
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
```

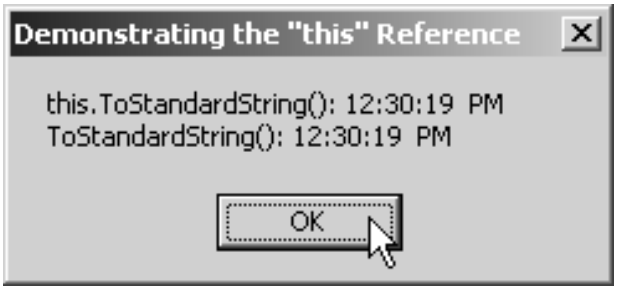
The this reference is used to set the class member variables to the constructor arguments

The this reference is used to refer to an instance method

```
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

```
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   }
```



Garbage Collection

- **new** operatörü hafızaya yerleştirir
- Nasneler daha fazla kullanılmayacaksa, CLR (Common Language Runtime) garbage collection çalışır
- Garbage collection hafıza yönetimine yardımcı olur (kullanılmayan hafızayı yeniden kullanılır duruma getirir)
- Diğer kaynakların (database connections, file access, etc.) hafızaya alınması ve hafızadan atılması programcı tarafından yapılmalıdır

Garbage Collection

- *Finalizer*'lar garbage collector ile birlikte kaynakların hafızadan atılması için kullanılır
- Garbage collector bir nesneye ayrılmış hafızayı yeniden kullanılır yapmak için, nesnenin finalizer metodunu çağırır
- Her sınıf sadece bir tane finalizer'a sahiptir (destructor)
- Destructor metodu ~ karakteri ile başlar ve sınıfın adıyla devam eder
- Destructor'lar parametre almazlar

static Sınıf Üyeleri

- Her sınıf nesneleri kendi örnek değişkenlerine sahiptir
- Bazen bir sınıfın tüm örneklerinin aynı değere sahip olması daha uygun olabilir
- Bu tür değişkenler **static** keyword ile tanımlanır ve sadece bir değere sahiptirler (bu türdeki tüm nesneler tarafından paylaşılırlar)
- Scope tanımlaması **static** değişkenler içinde tanımlanabilir (**public**, **private**, etc.)

```
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
```

Employee destructor

Update number of Employees

Decrease static member count, to signify that there is one less employee

```
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
```

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

Create 2 Employee objects

Set Employee objects to null

Force garbage collection

```
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
```

const ve readonly Üyeler

- Sabit değere sabit üyeler (değeri asla değişmez) **const** keyword ile tanımlanır
- **const** üyeler **static** özellik taşır
- **const** üyeler tanımlandığı değerlerini almak zorundadır
- **readonly** keyword ise başlangıç değerini constructor içerisinde alan ve daha sonra değişmeyen tanımlamalar için kullanılır

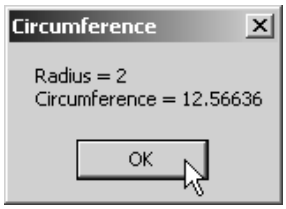
```
1    // Fig. 8.15: UsingConstAndReadOnly.cs
2    // Demonstrating constant values wit
3
4    using System;
5    using System.Windows.Forms;
6
7    // Constants class definition
8    public class Constants
9    {
10      // PI is constant variable
11      public const double PI = 3.14159;
12
13      // 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
```

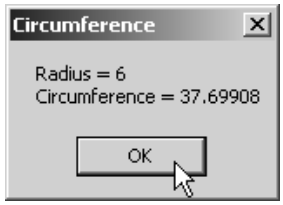
Readonly variable radius; must be initialized in constructor

Constant variable PI

Initialize readonly member radius

```
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
```





Namespaces

- Yazılım parçaları sürekli kullanılabilir olmalıdır
- Namespace sınıfların mantıksal bir grubunu sunar
- Aynı namespace içindeki iki ayrı sınıf aynı adı kullanamaz
- Ayrı namespace içindeki sınıflar aynı ada sahip olabilir
- Dinamik Bağlantı Kütüphaneleri (Dynamic Link Libraries -.dll files) sınıfların oluşturduğu paketleri yeniden kullanım için sunarlar

Namespaces

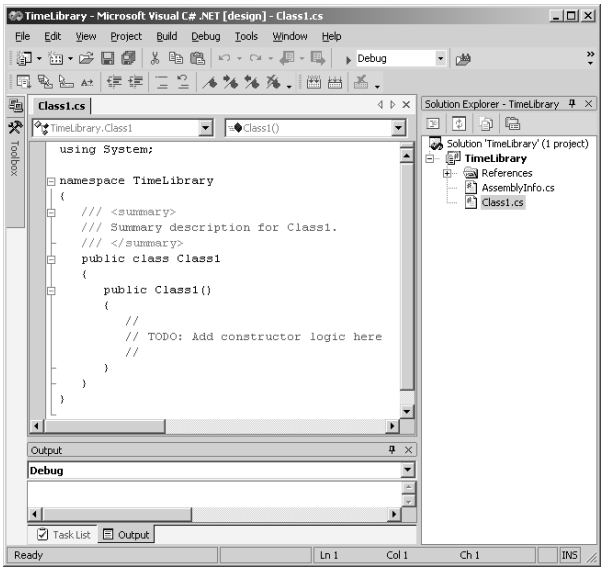


Fig. 8.18 Simple Class Library.

```
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

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

Class View ve Object Browser

- **Class View** ve **Object Browser** object-oriented uygulamaları tasarlamak için Visual Studio tarafından sunulan yardımcı araçlardır
- **Class View**
  - Bir projedeki tüm sınıflar için değişkenleri, metodları gösterir
  - Ağaç görünümde hiyerarşik yapıyı gösterir
  - + bir nodun alt parçalarının olduğunu gösterir
  - - bir nodun alt parçalarının görüntülenmiş olduğu durumu gösterir
  - **View < Class View** menüsünden seçilerek kullanılabilir

Class View ve Object Browser

- **Object Browser**
  - Bir kütüphanedeki tüm sınıfları listeler
  - **Object Browser**'ı görüntülemek için herhangi bir yerden kısayol menüsüyle **Go To Definition** seçilir

Class View ve Object Browser

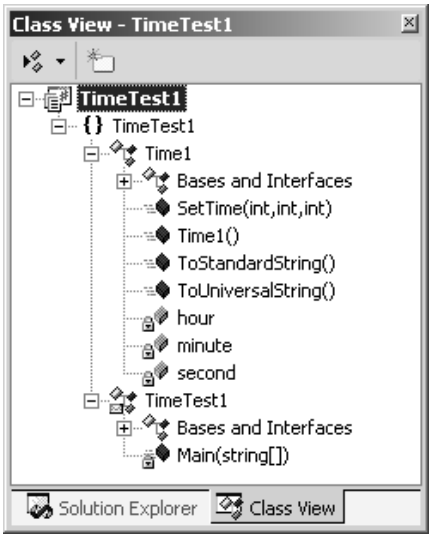


Fig. 8.20 **Class View** of class **`Time1`** (Fig. 8.1) and class **`TimeTest`** (Fig. 8.2).

Class View ve Object Browser

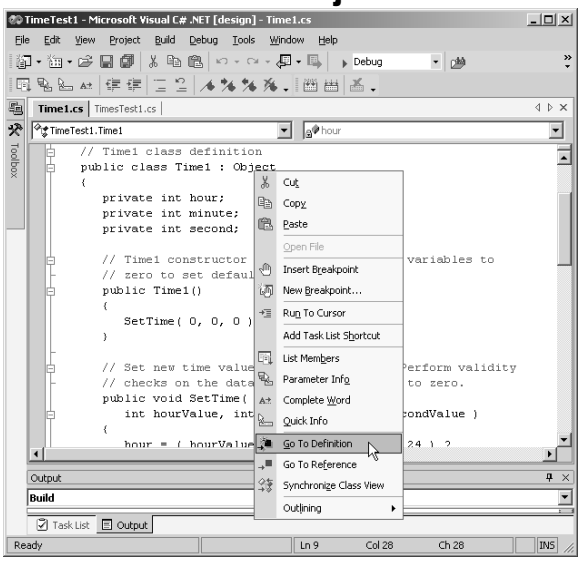


Fig. 8.21 **Object Browser** when user selects **`Object`** from **`Time1.cs`**. (part 1)

Class View ve Object Browser

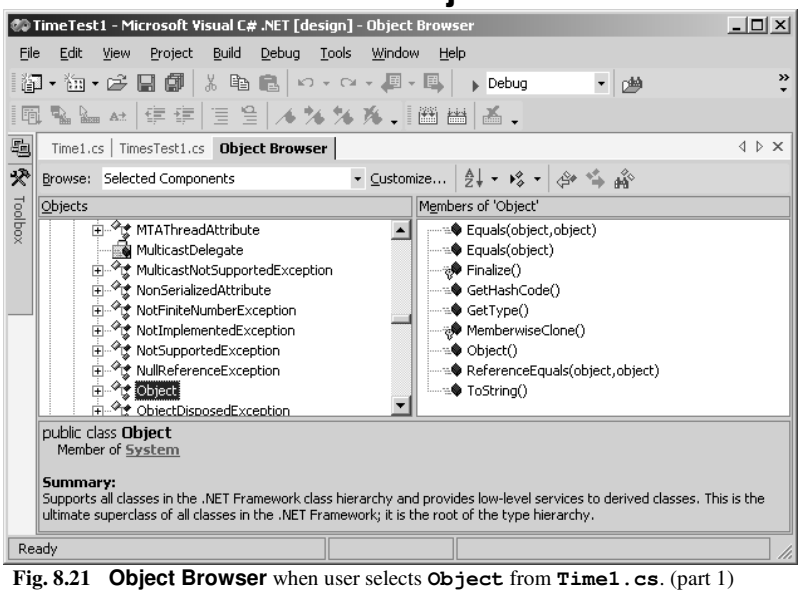


Fig. 8.21 **Object Browser** when user selects **`Object`** from **`Time1.cs`**. (part 1)