Properties and Indexers

Programiranje korisničkih interfejsa

Bojan Furlan

Enkapsulacija

```
class Racun
{
	private decimal stanjeDin; // polje

	decimal getStanje()
	{
		return stanjeDin;
	}
	…
}

//ili

class Racun
{
	private decimal stanjeEur; // polje
	decimal getStanje()
	{
		return stanjeEur*Kurs;
	}
	…
}
```

Why Use Properties?

- **Properties provide:**
  - A useful way to encapsulate information inside a class
  - Concise syntax
  - Flexibility

```
o.SetValue(o.GetValue( ) + 1);

ili

o.Value++;
```

Using Accessors

- **Properties provide field-like access**
  - Use **get** accessor statements to provide read access
  - Use **set** accessor statements to provide write access

```
class Button
{
    public string Text// Property
    {
        get { return text; }
        set { text = value; }
    }
    private string text; // Field
}
```

Upotreba

```
Button myButton;
...
string txt = myButton.Text; // "myButton.Text.get“
...
myButton.Text = "OK"; // "myButton.Text.set"
```

Comparing Properties to Fields

- **Properties are "logical fields"**
  - The **get** accessor can return a computed value
- **Similarities**
  - Syntax for creation and use is the same
- **Differences**
  - Properties are not values; they have no address
  - Properties cannot be used as **ref** or **out** parameters to methods

Comparing Properties to Methods

- **Similarities**
  - Both contain code to be executed
  - Both can be used to hide implementation details
  - Both can be virtual, abstract, or override
- **Differences**
  - Syntactic – properties do not use parentheses
  - Semantic – properties cannot be **void** or take arbitrary parameters

Property Types

- **Read/write properties**
  - Have both **get** and **set** accessors
- **Read-only properties**
  - Have **get** accessor only
  - Are not constants
- **Write-only properties – very limited use**
  - Have **set** accessor only
- **Static properties**
  - Apply to the class and can access only static data

Property Example

```
public class Console
{
    public static TextReader In
    {
        get {
            if (reader == null) {
                reader = new StreamReader(...);
            }
            return reader;
        }
    }
    ...
    private static TextReader reader = null;
}
```

Indexers

What Is an Indexer?

- **An indexer provides array-like access to an object**
  - Useful if a property can have multiple values
- **To define an indexer**
  - Create a property called *this*
  - Specify the index type
- **To use an indexer**
  - Use array notation to read or write the indexed property

Primer

```
class StringList
{
   private string[ ] list;
   public string this[int index]
   {
      get { return list[index]; }
      set { list[index] = value; }
   }
...
// Other code and constructors to initialize list
}

// upotreba
...
StringList myList = new StringList( );
...
myList[3] = "o"; // Indexer write
...
string myString = myList[3]; // Indexer read
...
```

Comparing Indexers to Arrays

- **Similarities**
  - Both use array notation
- **Differences**
  - Indexers can use non-integer subscripts
  - Indexers can be overloaded—you can define several indexers, each using a different index type
  - Indexers are not variables, so they do not denote storage locations—you cannot pass an indexer as a **ref** or an **out** parameter

Primer

```
class NickNames
{
   private Hashtable<string> names
   = new Hashtable<string>( );

   public string this[string realName]
   {
      get { return names[realName]; }
      set { names[realName] = value; }
   }
   public string this[int nameNumber]
   {
      get
      {
         string nameFound;
         // Code that iterates through the Hashtable
         // and populates nameFound
         return nameFound;
      }
   }
   ...
}
```

Comparing Indexers to Properties

- **Similarities**
  - Both use **get** and **set** accessors
  - Neither have an address
  - Neither can be void
- **Differences**
  - Indexers can be overloaded
  - Indexers cannot be static

Using Parameters to Define Indexers

- **When defining indexers**
  - Specify at least one indexer parameter
  - Specify a value for each parameter you specify

```
class MultipleParameters
{
    public string this[int one, int two]
    {
    get { ... }
    set { ... }
}
}
...
MultipleParameters mp = new MultipleParameters( );
string s = mp[2,3];
```

String Example

- **The String class**
  - Is an immutable class
  - Uses an indexer (**get** accessor but no **set** accessor)

```
class String
{
    public char this[int index]
    {
        get {
          if (index < 0 || index >= Length)
              throw new IndexOutOfRangeException( );
          ...
        }
    }
    ...
}
```

String Example

```
string s = "Sharp";
Console.WriteLine(s[0]); // Ok
s[0] = 'S'; // Compile-time error
s[4] = 'k'; // Compile-time error

String s = “ C ”+ ”Sharp”; //new object
```