Model Pattern Library

Donald Belcham
www.igloocoder.com
@dbelcham

INotifyPropertyChanged

- Largely boilerplate code
- Required in a lot of places
- Easy to mess up
- Clutters the codebase
- Is an infrastructure concern

```
public class Person : INotifyPropertyChanged
{
    private string firstName;
    private string lastName;

    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged(string propertyName)
    {
        if ( this.PropertyChanged != null )
        {
            this.PropertyChanged( this,
            new PropertyChangedEventArgs(propertyName) );
        }
    }

    public string FirstName
    {
        get { return this.firstName; }
        set
        {
            if ( this.firstName != value )
            {
                this.firstName = value;
                this.OnPropertyChanged("FirstName");
                this.OnPropertyChanged("FullName");
            }
        }
    }

    public string LastName
    {
        get { return this.lastName; }
        set
        {
            if ( this.lastName != value )
            {
                this.lastName = value;
                this.OnPropertyChanged("LastName");
                this.OnPropertyChanged("FullName");
            }
        }
    }

    public string FullName
    {
        get { return this.FirstName + " " + this.LastName; }
    }
}
```

NotifyPropertyChanged Aspect

- Implements all logic required
- Class level aspect
- Can be multicast

```
[NotifyPropertyChanged]
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public string FullName
    {
        get { return this.FirstName + " " + this.LastName; }
    }
}
```

INotifyPropertyChanged Approaches

Without AOP

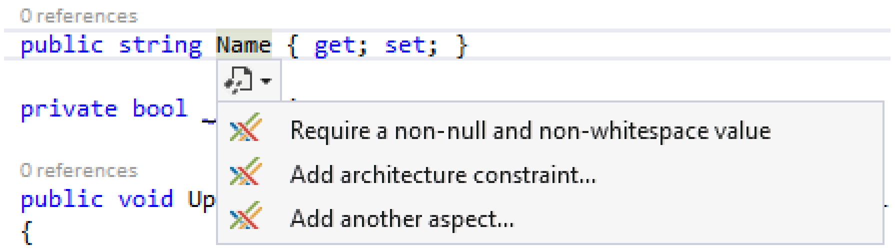
```
public class Person : INotifyPropertyChanged
{
    private string firstName;
    private string lastName;

    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged(string propertyName)
    {
        if ( this.PropertyChanged != null )
        {
            this.PropertyChanged( this,
            new PropertyChangedEventArgs(propertyName) );
        }
    }

    public string FirstName
    {
        get { return this.firstName; }
        set
        {
            if ( this.firstName != value )
            {
                this.firstName = value;
                this.OnPropertyChanged("FirstName");
                this.OnPropertyChanged("FullName");
            }
        }
    }

    public string LastName
    {
        get { return this.lastName; }
        set
        {
            if ( this.lastName != value )
            {
                this.lastName = value;
                this.OnPropertyChanged("LastName");
                this.OnPropertyChanged("FullName");
            }
        }
    }

    public string FullName
    {
        get { return this.FirstName + " " + this.LastName; }
    }
}
```

With PostSharp

```
[NotifyPropertyChanged]
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public string FullName
    {
        get { return this.FirstName + " " + this.LastName; }
    }
}
```

Contracts

- Operate like .NET code contracts
- Validating parameters, fields and properties

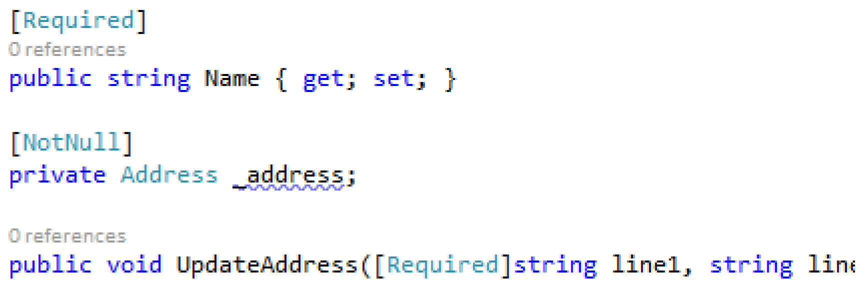


```
0 references
public string Name { get; set; }

private bool _

0 references
public void Up
{
```

Require a non-null and non-whitespace value
Add architecture constraint...
Add another aspect...

Contract Behavior

- **Throws an exception**
  - ArgumentException
  - ArgumentOutOfRangeException
  - ArgumentNullException
- **Can be applied to**
  - Properties
  - Fields
  - Parameters



```
[Required]
0 references
public string Name { get; set; }

[NotNull]
private Address _address;

0 references
public void UpdateAddress([Required]string line1, string line
```

- **Much like OnEntry of OnMethodBoundaryAspect, but faster**

Ready-Made Contracts

- GreaterThan
- LessThan
- NotEmpty
- NotNull
- Positive
- Range
- RegularExpression
- Required
- StrictlyGreaterThan
- StrictlyPositive
- StringLength
- CreditCard
- EmailAddress
- EnumDataType
- Phone
- Url

Custom Contracts

- Implement ILocationValidationAspect<T>
- As many types of T as necessary
- No type conversion of the value
- Can require a lot of implementations of ILocationValidationAspect<T>
  - int, int?, long, long?, decimal, decimal?, etc…

Contract Inheritance

- Inherited if applied on interfaces, abstract or virtual methods
- Only applicable on method parameters

```
interface IFoo
{
   void Bar( [Required] string fooBar );
}

class Foo : IFoo
{
   public void Bar( string fooBar )
   {
      // PostSharp will inject the [Required] contract at the top of this method body.
   }
}
```

Contract Limitations

- Do not support type conversions
- Cannot validate return values or output arguments
- Are an opt-in feature

Summary

- **NotifyPropertyChangedAspect**
  - Less code
  - Faster to implement
- **Contracts**
  - Out of the box
  - Extensible