Refactoring Fundamentals: Code Smells - Obfuscators

Steve Smith

Ardalis.com

Twitter: @ardalis

In This Course

- ~~What is Refactoring?~~
- ~~Why do it?~~
- ~~What's the process?~~
- ~~What are some tools that can assist with it?~~
- ~~What is a *Code Smell*?~~
- What are some examples of Code Smells?
- What are some common refactorings?
- How does one apply them correctly?

Organizing Code Smells

- **Taxonomy proposed by Mäntylä, M. V. and Lassenius, C.**
  - http://www.soberit.hut.fi/~mmantyla/ESE_2006.pdf
- **Organization of Code Smells into 5 Groups**
  - ~~The Bloaters~~
  - The Object-Orientation Abusers
  - The Change Preventers
  - The Dispensables
  - The Couplers
- **I've added three more:**
  - The Obfuscators
  - Environment Smells
  - Test Smells

Code Smells: The Obfuscators

- Coding constructs or techniques that obfuscate the intent of the code
- Impede clear communication

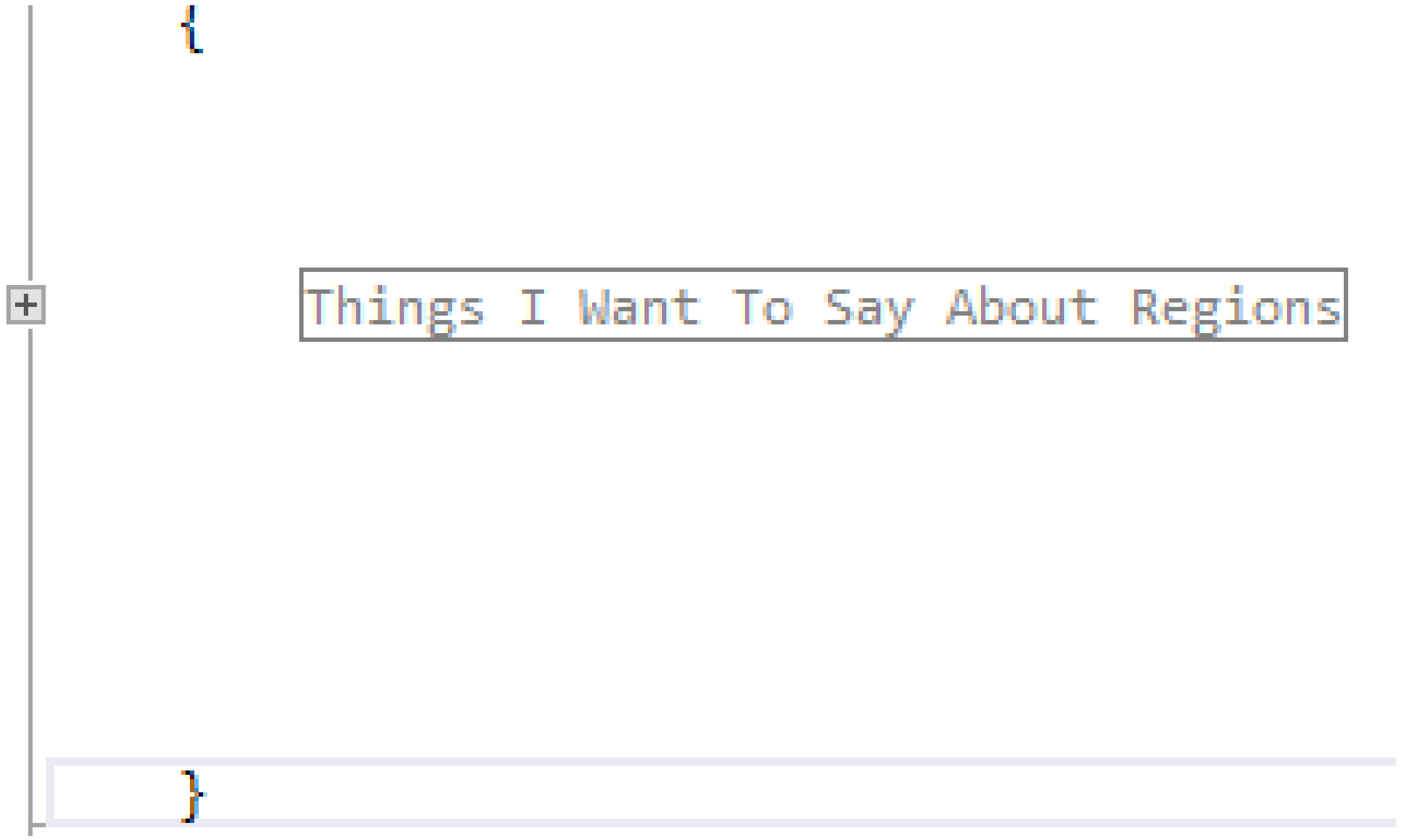
The party of the first part
hereinafter known as Jack ...
and ...

The party of the second part
hereinafter known as Jill ...

Ascended or caused to be
ascended an elevation of
undetermined height and
degree of slope, hereinafter
referred to as "hill".

http://wordstodeeds.com/2013/01/28/monday-smiles-jack-and-jill/

Bloaters and Obfuscators: Regions



```
{
        Things I Want To Say About Regions
}
```

Bloaters and Obfuscators: Regions

- **C# regions tend to bloat code, while trying to hide bloat**
  - Regions are rugs under which to sweep smelly code.

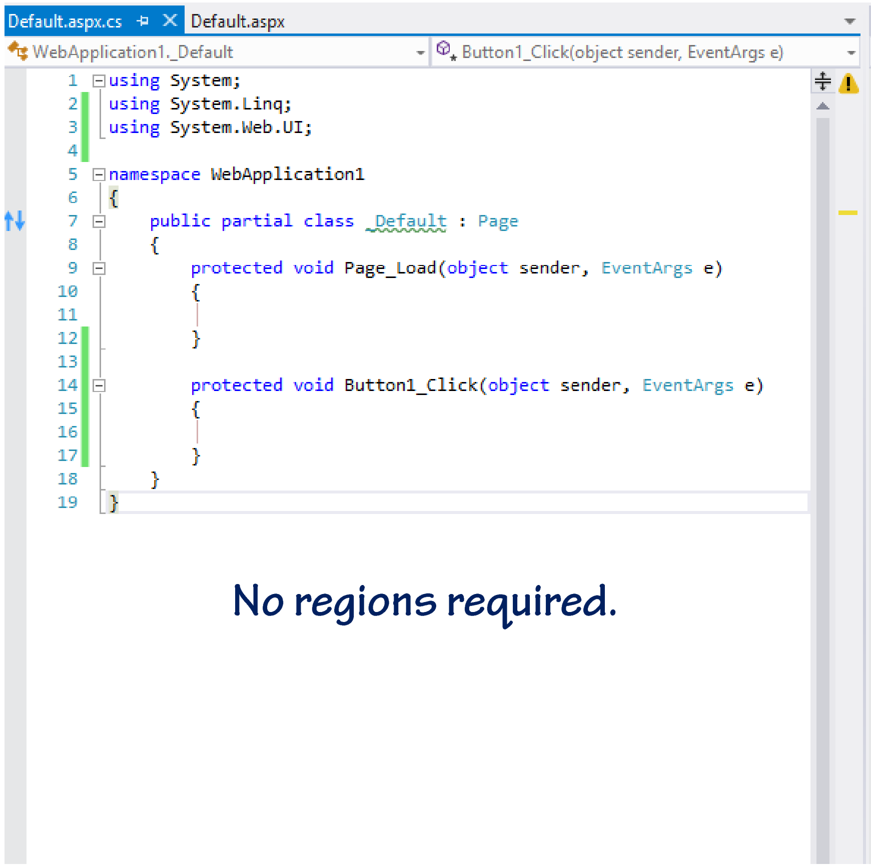
- **Regions do two things:**
  - Provide a single line comment
  - Wrap some code and allow it to be hidden on demand

- **Why is this useful?**
  - The intent of the code is unclear (a smell)
  - The code is too long to quickly understand at a glance (a smell)
  - Both of the above
  - Author prefers an “outline” view of classes (personal preference)

Bloaters and Obfuscators: Regions

```
#region Web Form Designer generated code
    override protected void OnInit(EventArgs e)
    {
        //
        // CODEGEN: This call is required by the ASP.NET Web Form Designer.
        //
        InitializeComponent();
        base.OnInit(e);
    }

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {

    }
#endregion
```

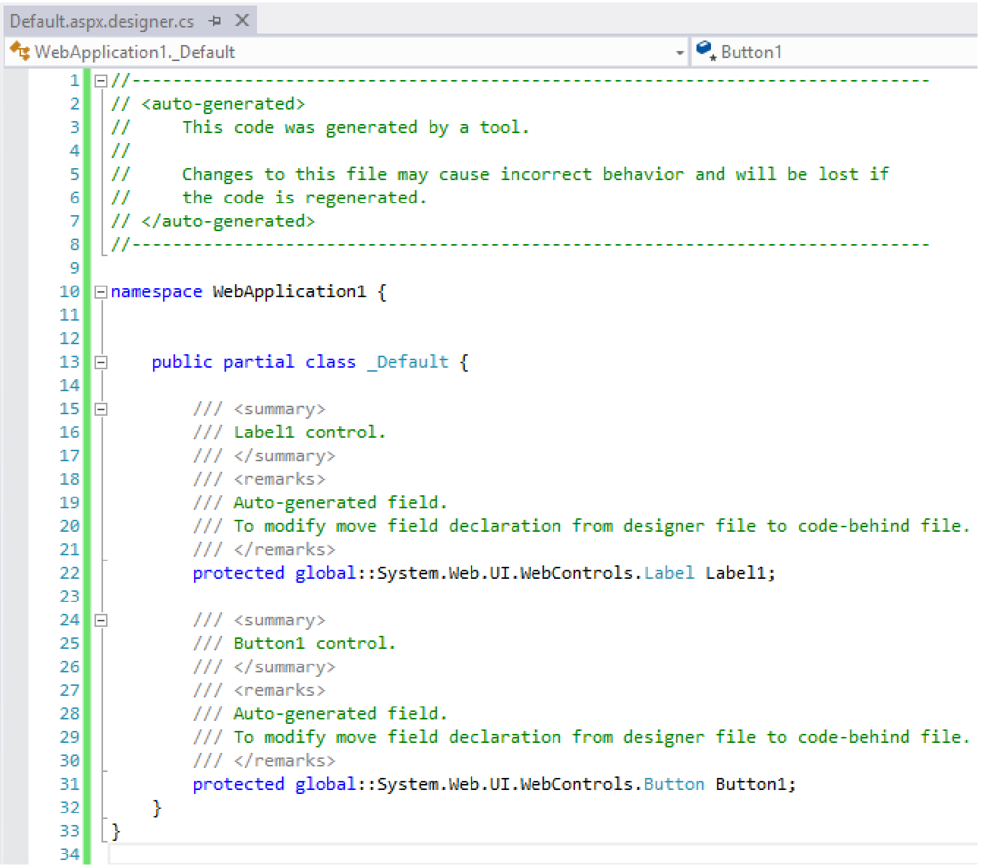
Bloaters and Obfuscators: Regions



Default.aspx.cs | Default.aspx

WebApplication1._Default | Button1_Click(object sender, EventArgs e)

```
using System;
using System.Linq;
using System.Web.UI;

namespace WebApplication1
{
    public partial class _Default : Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void Button1_Click(object sender, EventArgs e)
        {

        }
    }
}
```



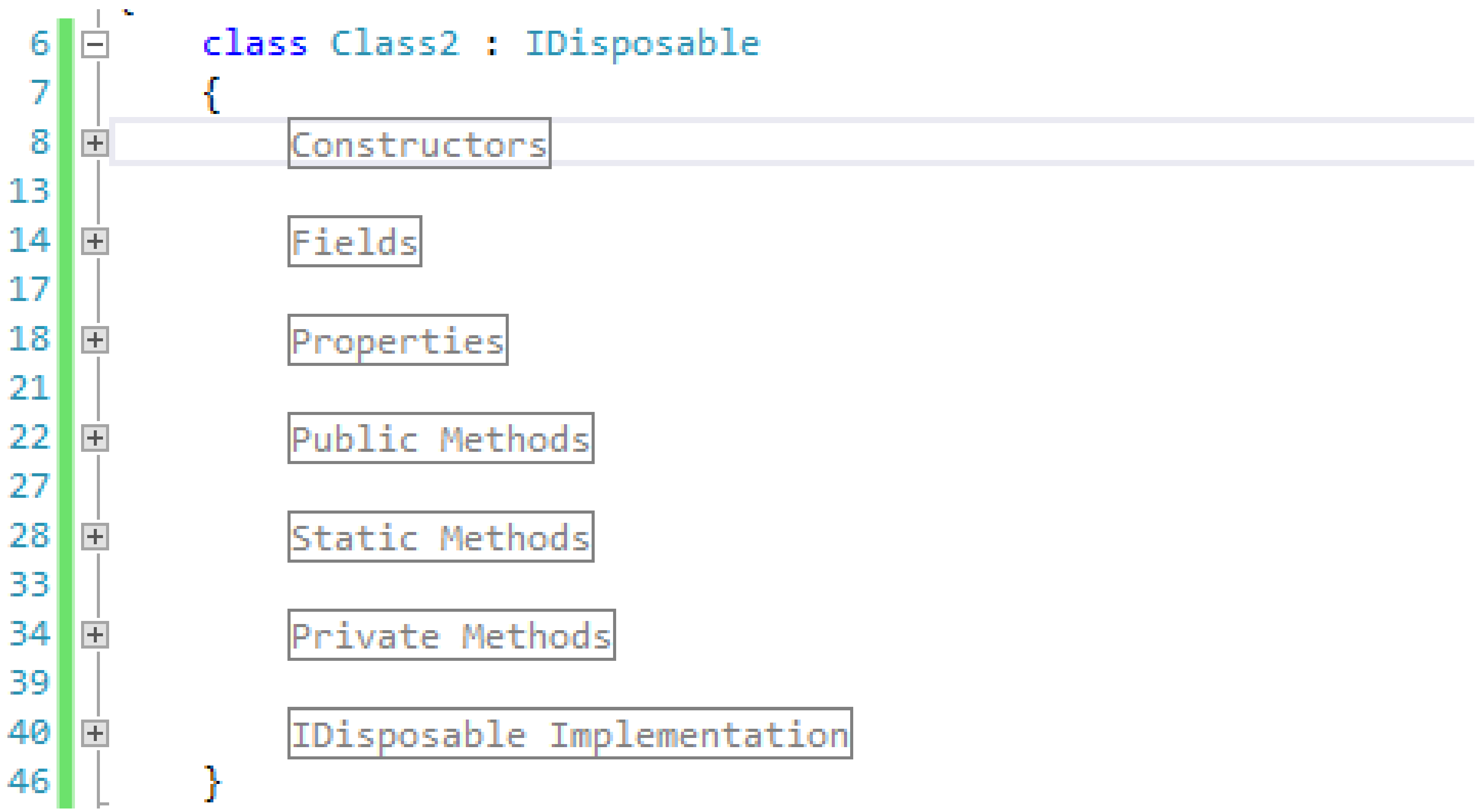
Default.aspx.designer.cs

WebApplication1._Default | Button1

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace WebApplication1 {


    public partial class _Default {

        /// <summary>
        /// Label1 control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer file to code-behind file.
        /// </remarks>
        protected global::System.Web.UI.WebControls.Label Label1;

        /// <summary>
        /// Button1 control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer file to code-behind file.
        /// </remarks>
        protected global::System.Web.UI.WebControls.Button Button1;
    }
}
```

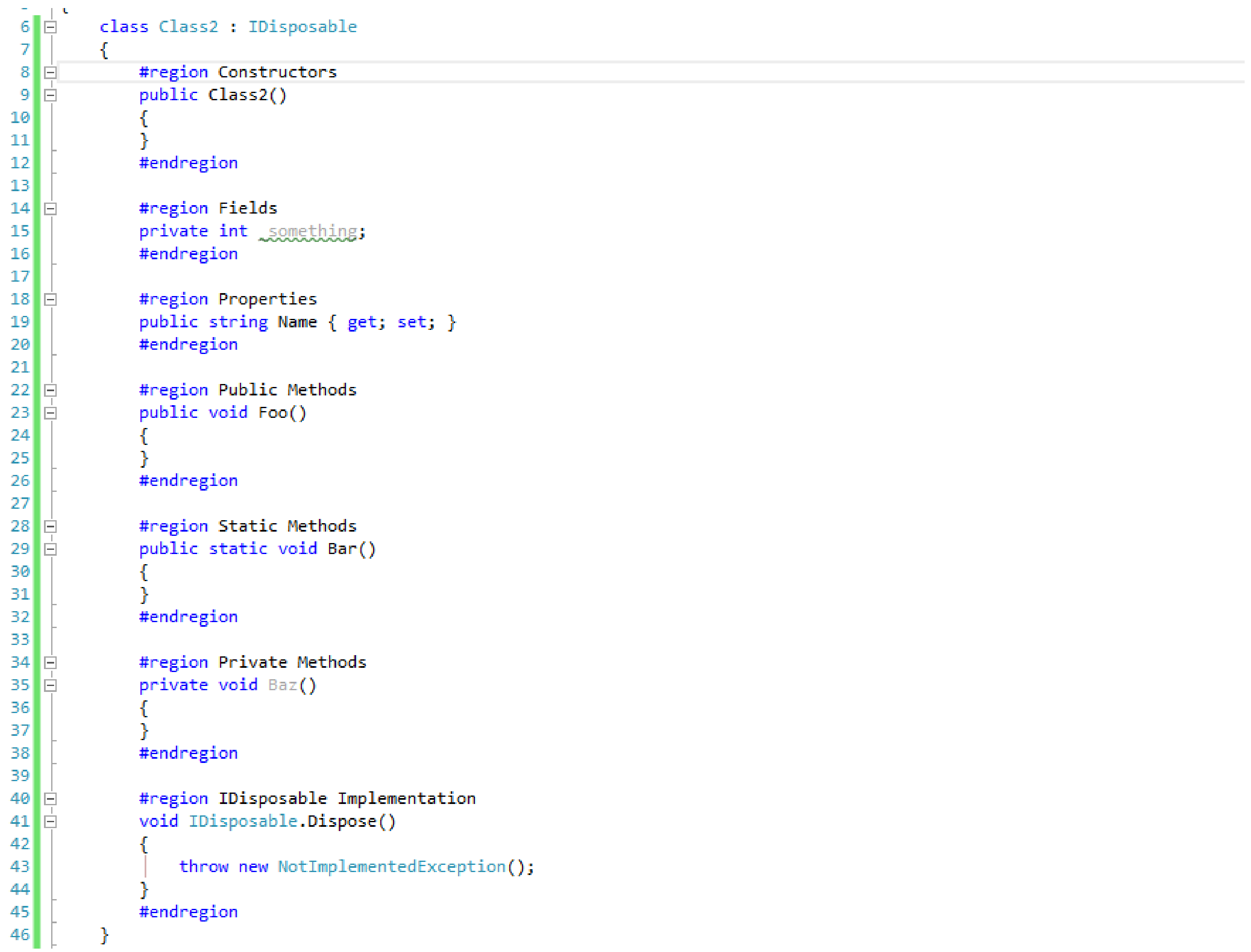
No regions required.

Bloaters and Obfuscators : Regions



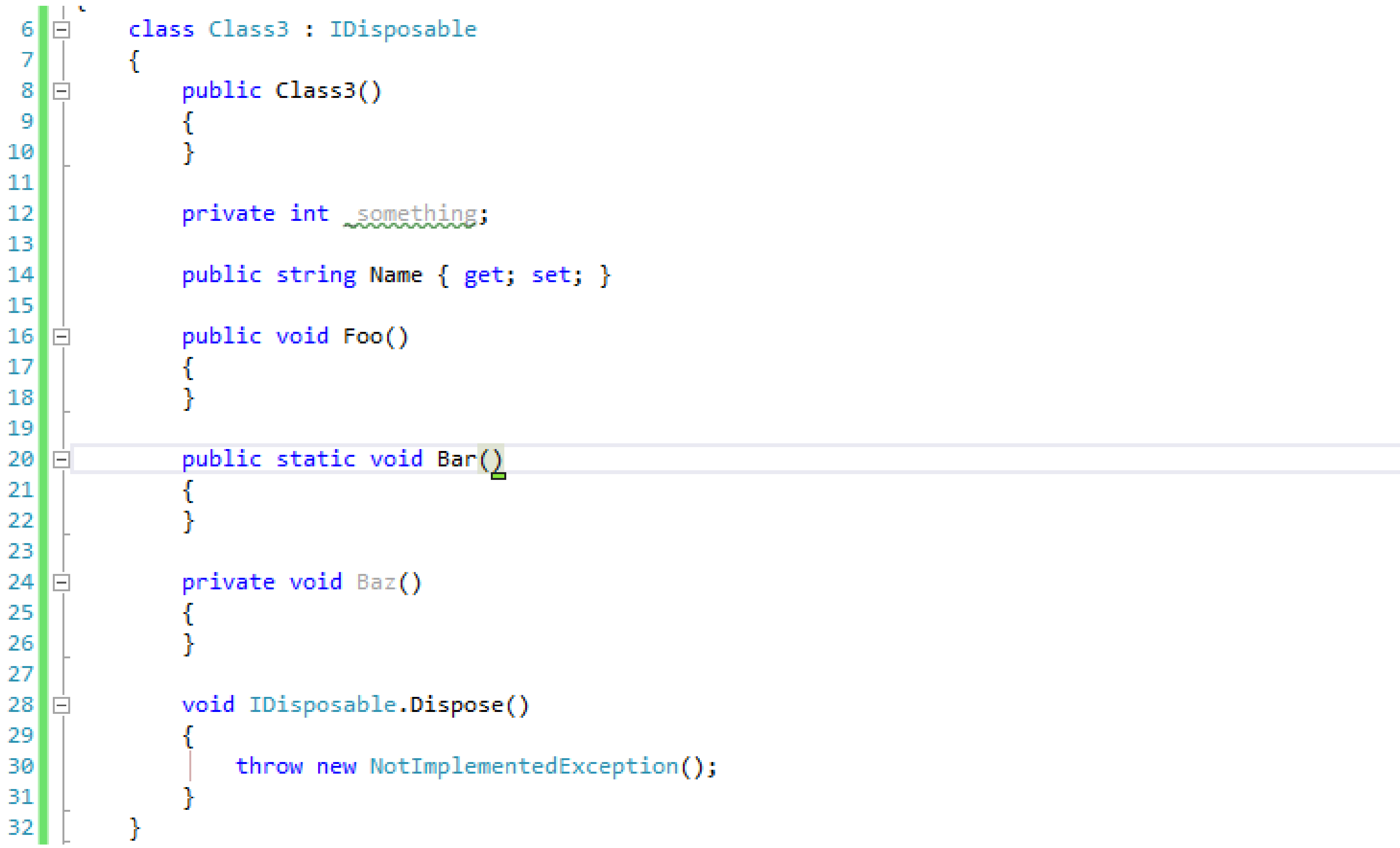
```
class Class2 : IDisposable
{
    Constructors

    Fields

    Properties

    Public Methods

    Static Methods

    Private Methods

    IDisposable Implementation
}
```

Bloaters and Obfuscators : Regions



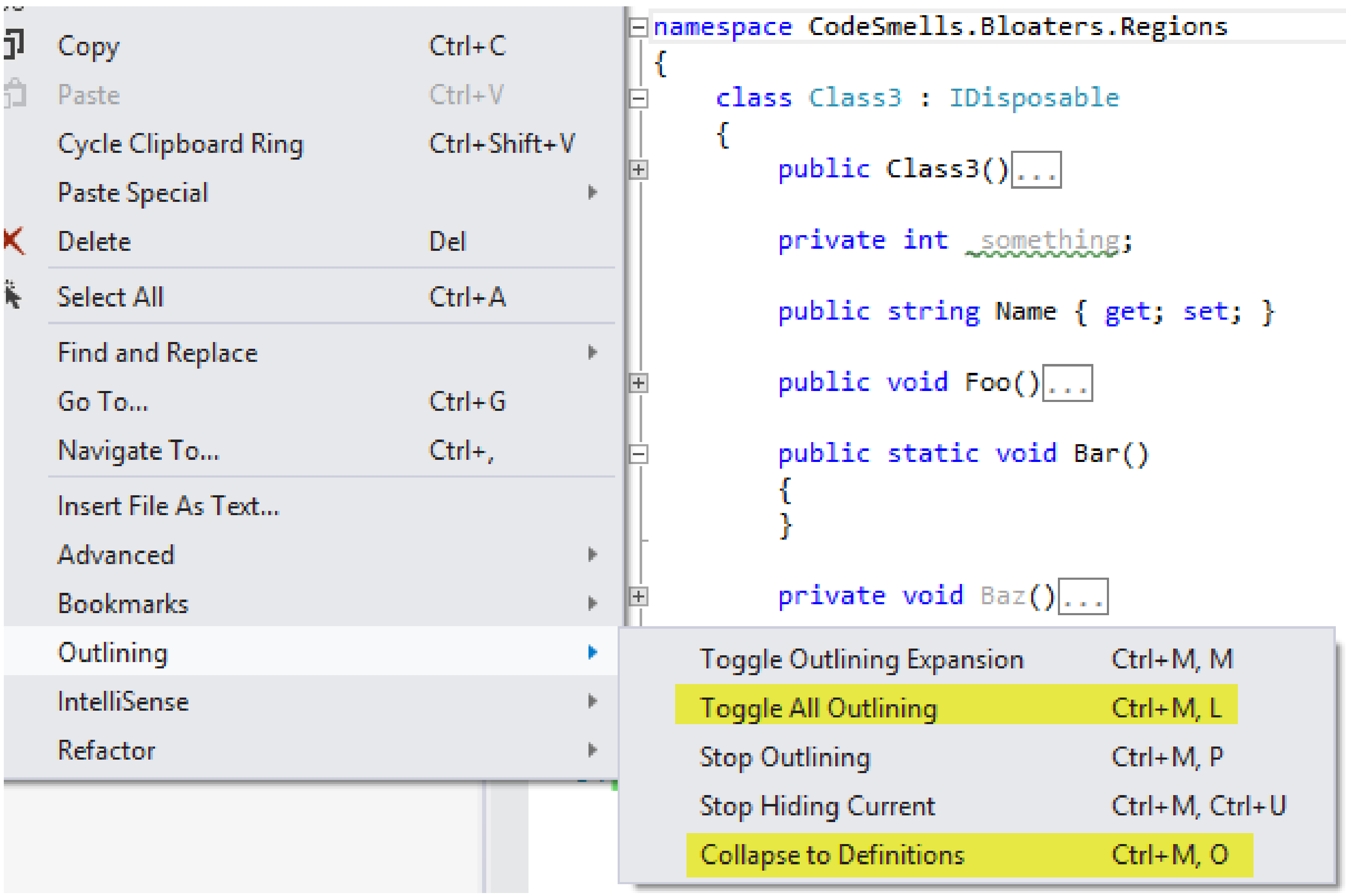
```
class Class2 : IDisposable
{
    #region Constructors
    public Class2()
    {
    }
    #endregion

    #region Fields
    private int _something;
    #endregion

    #region Properties
    public string Name { get; set; }
    #endregion

    #region Public Methods
    public void Foo()
    {
    }
    #endregion

    #region Static Methods
    public static void Bar()
    {
    }
    #endregion

    #region Private Methods
    private void Baz()
    {
    }
    #endregion

    #region IDisposable Implementation
    void IDisposable.Dispose()
    {
        throw new NotImplementedException();
    }
    #endregion
}
```

Bloaters and Obfuscators : Regions



```
class Class3 : IDisposable
{
    public Class3()
    {
    }

    private int _something;

    public string Name { get; set; }

    public void Foo()
    {
    }

    public static void Bar()
    {
    }

    private void Baz()
    {
    }

    void IDisposable.Dispose()
    {
        throw new NotImplementedException();
    }
}
```

Bloaters and Obfuscators : Regions



Copy	Ctrl+C
Paste	Ctrl+V
Cycle Clipboard Ring	Ctrl+Shift+V
Paste Special	
Delete	Del
Select All	Ctrl+A
Find and Replace	
Go To...	Ctrl+G
Navigate To...	Ctrl+,
Insert File As Text...	
Advanced	
Bookmarks	
Outlining	
IntelliSense	
Refactor	

Toggle Outlining Expansion	Ctrl+M, M
Toggle All Outlining	Ctrl+M, L
Stop Outlining	Ctrl+M, P
Stop Hiding Current	Ctrl+M, Ctrl+U
Collapse to Definitions	Ctrl+M, O

```
namespace CodeSmells.Bloaters.Regions
{
    class Class3 : IDisposable
    {
        public Class3()...

        private int _something;

        public string Name { get; set; }

        public void Foo()...

        public static void Bar()
        {
        }

        private void Baz()...
```

Bloaters and Obfuscators: Regions

Refactor away from Regions by

- **Deleting them**

- **Clean up smelly code**

- **Regions tend to hide other code smells, especially other bloaters:**
  - Long method
  - Long class
  - Long loop
  - Function does more than one thing

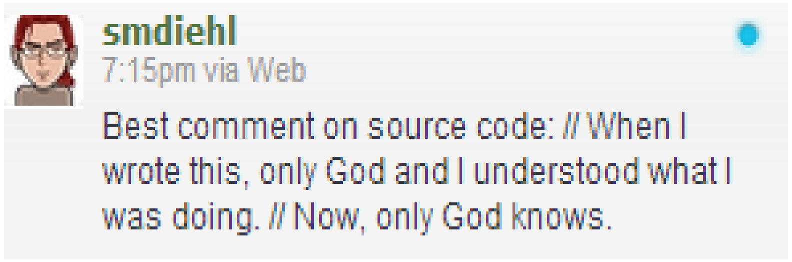
http://ardalis.com/regional-differences

Bloaters and Obfuscators: Comments

- Untrustworthy
- Should only be used to tell *why*, not *what* or *how*

Common Commenting Smells:

- Used to explain difficult code
- Used to hold inappropriate information
- May be obsolete (or simply wrong)
- Redundant
- May be poorly written
- Commented out code

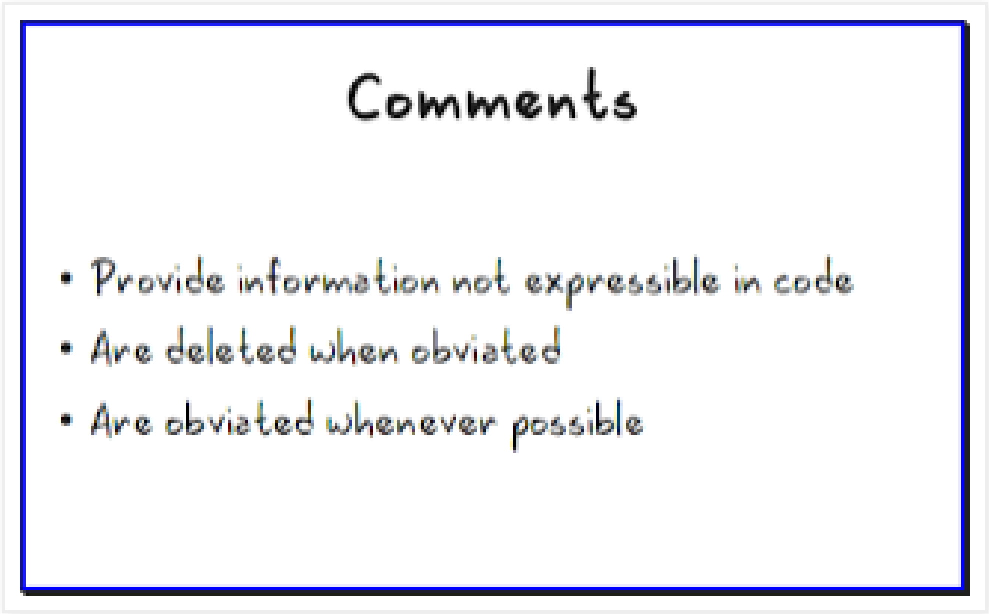

smdiehl
7:15pm via Web
Best comment on source code: // When I wrote this, only God and I understood what I was doing. // Now, only God knows.

http://ardalis.com/when-to-comment-your-code

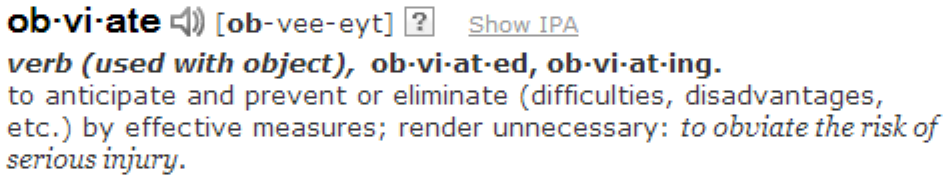
Rules for Commenting

Tim Ottinger and Jeff Langr suggest these rules for comments:

Rules for Commenting


Comments

- Provide information not expressible in code
- Are deleted when obviated
- Are obviated whenever possible


ob·vi·ate [ob-vee-eyt] ? Show IPA
verb (used with object), **ob·vi·at·ed, ob·vi·at·ing.**
to anticipate and prevent or eliminate (difficulties, disadvantages, etc.) by effective measures; render unnecessary: *to obviate the risk of serious injury.*

http://agileinaflash.blogspot.com/2009/04/rules-for-commenting.html

Examples of Explaining Difficult Code

```
// check if user has permission to edit article
if(user.IsInRole(“admin”) ||
  article.Author.Id == user.Id ||
  (user.IsInRole(“reviewer”) &&
   article.Reviewer.Id == user.Id))
{
}

if(user.CanEdit(article))
{
}
```

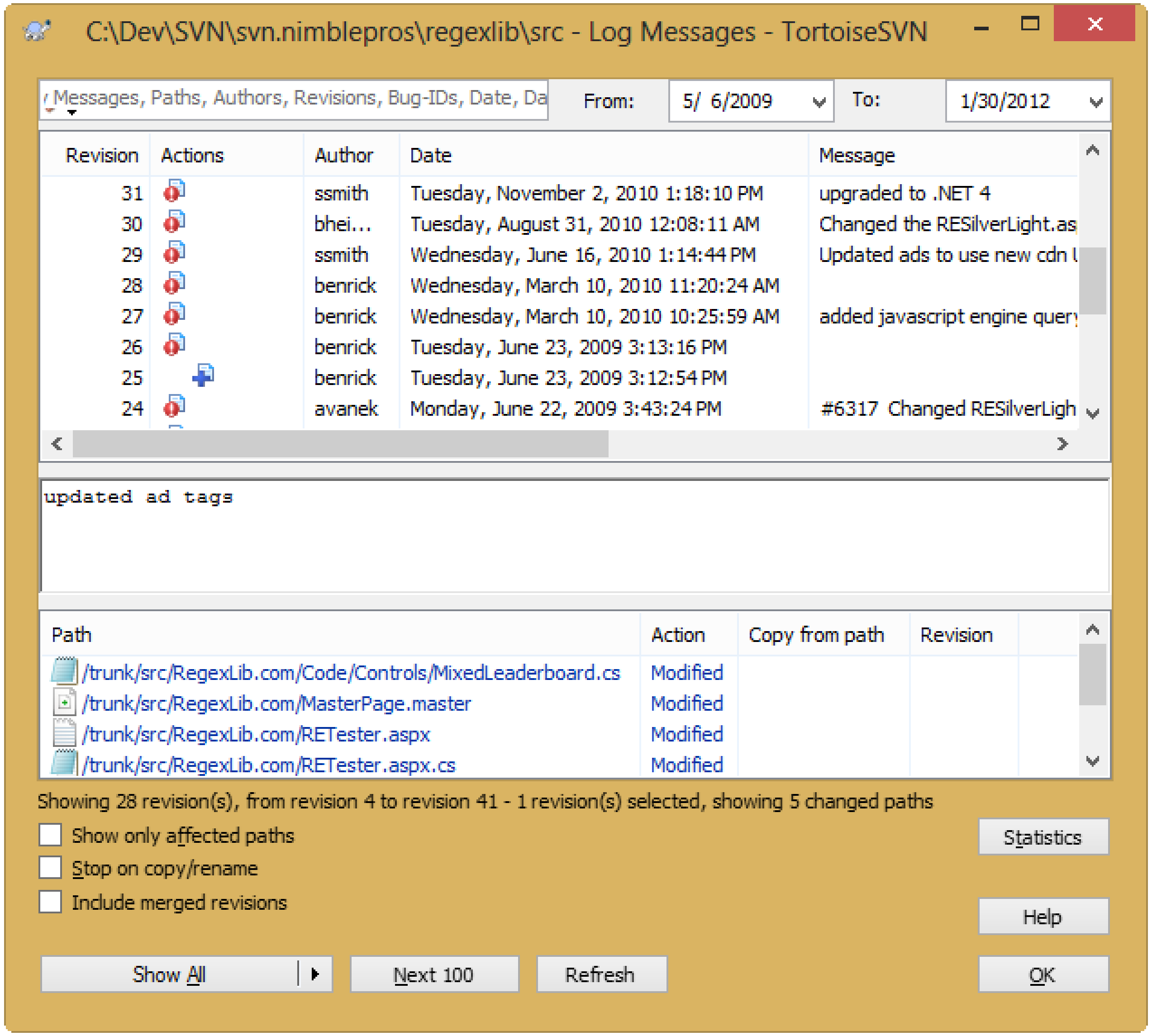
Examples of Inappropriate Information

```
#region Copyright Notice
// Copyright (c) 2014, My Company. All rights reserved.
#endregion Copyright Notice
```

Examples of Inappropriate Information

```
//////////////////////////////////////////////
// 2/2/12  : Added DateCreated property (SS)
// 1/12/12 : Added DateModified property (SS)
// 12/7/11 : Fixed bug in Foo() method (AG)
// 12/5/11 : Fixed bug in Bar() method (AG)
// 11/2/11 : Created class
//////////////////////////////////////////////
```

Examples of Inappropriate Information



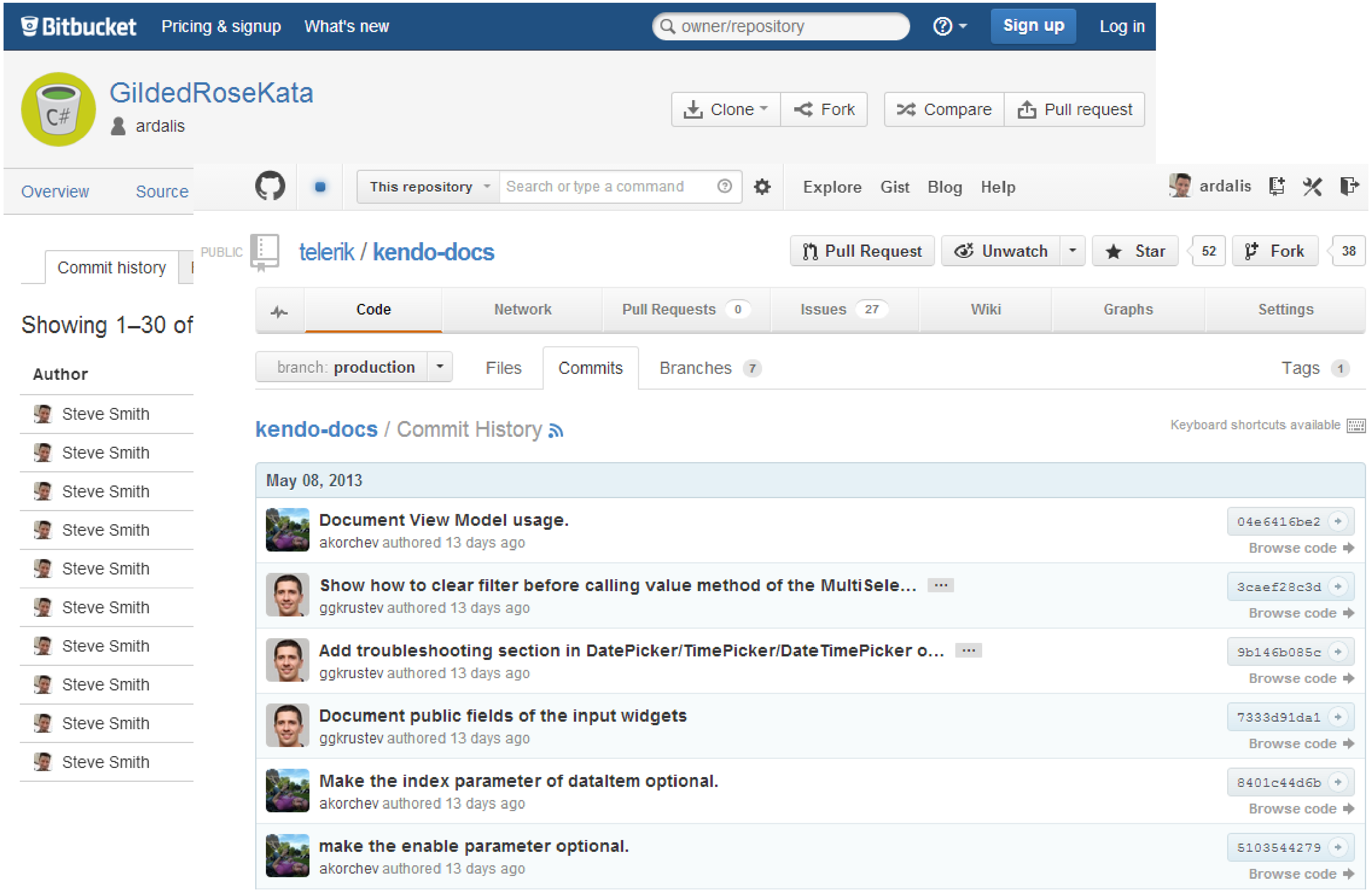
Revision	Actions	Author	Date	Message
31		ssmith	Tuesday, November 2, 2010 1:18:10 PM	upgraded to .NET 4
30		bhei...	Tuesday, August 31, 2010 12:08:11 AM	Changed the RESilverLight.as
29		ssmith	Wednesday, June 16, 2010 1:14:44 PM	Updated ads to use new cdn l
28		benrick	Wednesday, March 10, 2010 11:20:24 AM	
27		benrick	Wednesday, March 10, 2010 10:25:59 AM	added javascript engine query
26		benrick	Tuesday, June 23, 2009 3:13:16 PM	
25		benrick	Tuesday, June 23, 2009 3:12:54 PM	
24		avanek	Monday, June 22, 2009 3:43:24 PM	#6317 Changed RESilverLigh

```
updated ad tags
```

Path	Action	Copy from path	Revision
/trunk/src/RegexLib.com/Code/Controls/MixedLeaderboard.cs	Modified		
/trunk/src/RegexLib.com/MasterPage.master	Modified		
/trunk/src/RegexLib.com/RETester.aspx	Modified		
/trunk/src/RegexLib.com/RETester.aspx.cs	Modified		

Showing 28 revision(s), from revision 4 to revision 41 - 1 revision(s) selected, showing 5 changed paths

Examples of Inappropriate Information



Showing 1–30 of

Author
Steve Smith
Steve Smith
Steve Smith
Steve Smith
Steve Smith
Steve Smith
Steve Smith
Steve Smith
Steve Smith
Steve Smith

kendo-docs / Commit History

May 08, 2013

Document View Model usage.	akorchev authored 13 days ago	04e6416be2
Show how to clear filter before calling value method of the MultiSele…	ggkrustev authored 13 days ago	3caef28c3d
Add troubleshooting section in DatePicker/TimePicker/DateTimePicker o…	ggkrustev authored 13 days ago	9b146b085c
Document public fields of the input widgets	ggkrustev authored 13 days ago	7333d91da1
Make the index parameter of dataItem optional.	akorchev authored 13 days ago	8401c44d6b
make the enable parameter optional.	akorchev authored 13 days ago	5103544279

Examples of Obsolete Comments

```
// only managers can view reports
if(!user.IsInRole(“managers”) &&
   !user.IsInRole(“admins”))
{
  throw new NotAuthorizedException();
}


if(!user.CanViewReport(report))
{
  throw new NotAuthorizedException();
}
```

Examples of Redundant Comments

```
/// <summary>
/// Gets the last error message.
/// </summary>
public string LastError
{
    get { return _lastError; }
}

/// <summary>
/// Gets the last exception.
/// </summary>
public Exception LastException
{
    get { return _lastException; }
}
```

Examples of Redundant Comments

```
public string LastError { get { return _lastError; } }
public Exception LastException { get { return _lastException; } }
```

Examples of Redundant Comments

```
public string LastError { get; private set;}
public Exception LastException { get; private set;}
```

Commented Out Code

- **When you see commented out code, *delete it***
- **There, isn't that better?**
- **Commented code rots quickly as the program changes around it**
- **Don't worry, the code is still in your version control system**
  - You are using source control, right?

The Obfuscators: Poor Names

SIMPLY EXPLAINED
geek & poke
LET ME TAKE A WILD GUESS: int doSomething() DOES SOMETHING
WHAT ELSE?
SELF DOCUMENTING CODE

http://bit.ly/115kWII

The Obfuscators: Poor Names

Uncle Bob's Naming Recommendations (from *Clean Code*)

- Choose Descriptive Names
- Choose Names at the Appropriate Level of Abstraction
- Use Standard Nomenclature Where Possible
- Choose Unambiguous Names
- Use Long Names for Long Scopes
- Avoid Encodings
- Names Should Describe Side Effects

http://deviq.com/naming-things

Choose Descriptive Names

```
public static List<int> Generate(int n)
{
    var x = new List<int>();
    for (int i = 2; n > 1; i++)
        for (; n % i == 0; n /= i)
            x.Add(i);
    return x;
}
```

Choose Descriptive Names

```
public static List<int> GeneratePrimeFactorsOf(int input)
{
    var primeFactors = new List<int>();
    for (int candidateFactor = 2; input > 1; candidateFactor++)
        while (input % candidateFactor == 0)
        {
            primeFactors.Add(candidateFactor);
            input /= candidateFactor;
        }
    return primeFactors;
}
```

Usage:

```
var factors = GeneratePrimeFactorsOf(input);
```

Choose Names at Appropriate Abstraction Level

```
public class User
{
    public string UserName { get; set; }

    public static int GetTotalUserCountInDatabaseTable()
    {
        throw new NotImplementedException();
    }

    public static SqlDataReader GetDataReaderWithRoles(string userName)
    {
        throw new NotImplementedException();
    }
}
```

Choose Names at Appropriate Abstraction Level

```
public class User
{
    public string UserName { get; set; }

    public IEnumerable<Role> IsInRoles()
    {
        throw new NotImplementedException();
    }
}

public class SqlUserRepository
{
    public int TotalUserCount()
    {
        throw new NotImplementedException();
    }

    public IEnumerable<Role> UserIsInRoles(string userName)
    {
        throw new NotImplementedException();
    }
}
```

Use Standard Nomenclature Where Possible

```
var customer = customerFactory.Create(123);
var order = orderBuilder.Make(234);
var orderItem = orderItemMaker.NewItem();

order.AddRow(orderItem);
customer.Append(order);
```

Use Standard Nomenclature Where Possible

```
var customer = customerFactory.Create(123);
var order = orderFactory.Create(234);
var orderItem = orderItemFactory.Create();

order.Add(orderItem);
customer.Add(order);
```

Choose Unambiguous Names

```
public string Format(string input)
{
    int n;
    if(int.TryParse(input, out n))
    {
        if (n == 0) return "Not Started";
        if (n == 100) return "Complete";
        return n.ToString() + '%';
    }
    return input.Trim().ToUpper();
}
```

Choose Unambiguous Names

```
public string FormatProgressForDisplay(string
   input)
{
    int n;
    if(int.TryParse(input, out n))
    {
        if (n == 0) return "Not Started";
        if (n == 100) return "Complete";
        return n.ToString() + '%';
    }
    return input.Trim().ToUpper();
}
```

Choose Long Names For Long Scopes

```
public string ListUsers()
{
    var sb = new StringBuilder();
    for (int i = 0; i < Application.CurrentUserCount; i++)
    {
        sb.Append("User " + i + Environment.NewLine);
    }
    return sb.ToString();
}
```

Choose Long Names For Long Scopes

```
public string ListUsers()
{
    var sb = new StringBuilder();
    for (int i = 0; i < A.UC; i++)
    {
        sb.Append("User " + i + E.NL);
    }
    return sb.ToString();
}
```

Avoid Encodings

```
string strName;
int iCount;
DateTime dtStart;
DateTime dtEnd;
User usrOne;
User usrTwo;
SqlUserRepository surDataAccess;
List<User> lstUsers;
```

Avoid Encodings

```
string name;
int count;
DateTime StartDate;
DateTime EndDate;
User user1;
User user2;
SqlUserRepository userRepository;
List<User> users;

string userName = UserNameTextBox.Text;
UserNameLabel.Text = userName;
```

Avoid Encodings

```
string name;
int count;
DateTime StartDate;
DateTime EndDate;
User user1;
User user2;
SqlUserRepository userRepository;
List<User> users;

string userName = UserNameTextBox.Text;
UserNameLabel.Text = userName;
```

Names Should Describe Side Effects

```
public User GetUser(string userName)
{
    var user = GetUserFromDatabase(userName);

    return user ?? new User();
}
```

Names Should Describe Side Effects

```
public User GetOrCreateUser(string userName)
{
    var user = GetUserFromDatabase(userName);

    return user ?? new User();
}
```

The Obfuscators: Vertical Separation

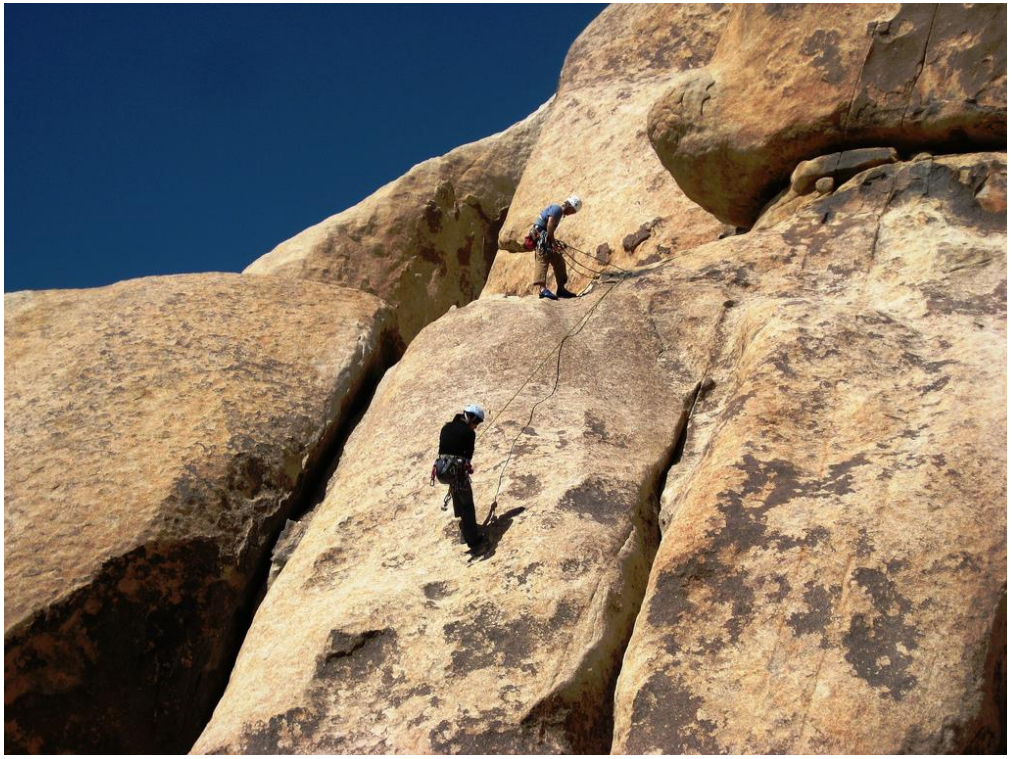
Define variables and functions near where they are used

Define local variables just before first use;

Define private functions just below their first usage

Avoid forcing reader to scroll

Exacerbated by Long Class



http://www.flickr.com/photos/brewbooks/6183737447/sizes/l/in/photostream/

The Obfuscators: Inconsistency

Follow the Principle of Least Surprise

If Something is Usually Called X, Always Call it X

Simple consistency, applied regularly, results in code that's much easier to read and modify

Arbitrary inconsistency is *confusing* <u>and</u> **distracting**

The Obfuscators: Obscured Intent

- Small and dense are not ends in and of themselves

- Take the time to make your code intention revealing, not obscuring

Intention Obscuring

```
public int m_otCalc()
{
    return iThsWkd * iThsRte +
        (int)Math.Round(0.5 * iThsRte *
        Math.Max(0, iThsWkd - 400));
}
```

Intention ~~Obscuring~~ Revealing

```
public int CalculateStraightPay()
{
    return tenthsWorked * tenthsRate;
}

public int CalculateOverTimePay()
{
    int overTimeTenths = Math.Max(0, tenthsWorked - 400);
    int overTimePay = CalculateOverTimeBonus(overTimeTenths);
    return CalculateStraightPay() + overTimePay;
}

private int CalculateOverTimeBonus(int overTimeTenths)
{
    double bonus = 0.5 * tenthsRate * overTimeTenths;
    return (int)Math.Round(bonus);
}
```

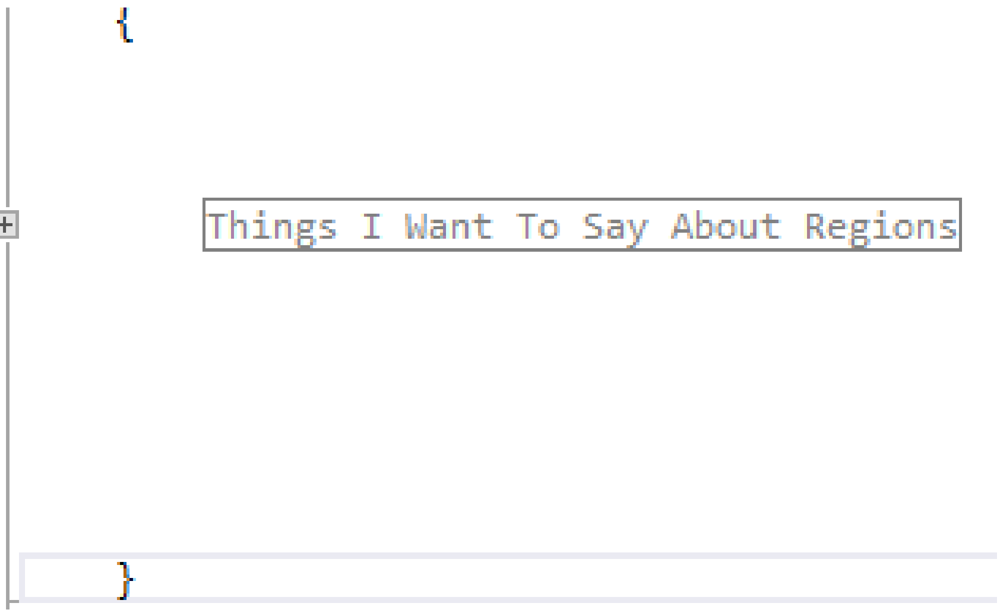
Summary

Organization of Code Smells into 5 Groups

- The Bloaters
- The Object-Orientation Abusers
- The Change Preventers
- The Dispensables
- The Couplers

I've added three more:

- The Obfuscators
- Environment Smells
- Test Smells



```
{

        Things I Want To Say About Regions

}
```

References

Related Pluralsight Courses

SOLID Principles of Object Oriented Design http://bit.ly/rKbR9a

Design Patterns Library http://bit.ly/SJmAX1

Books

Code Complete http://amzn.to/Vq5YLv

Clean Code http://amzn.to/YjUDI0

Web

When to Comment Your Code http://ardalis.com/when-to-comment-your-code

On Regions http://ardalis.com/regional-differences

Naming Things http://deviq.com/naming-things

Thanks!

Steve Smith

Ardalis.com

Twitter: @ardalis

pluralsight
hardcore developer training

Guest Opinion: Scott Hanselman