

C# : Crafting Code

Make it readable



Overview

- Standards and best practices
- What is language oriented programming?
- Fluent Interfaces
- Declarative Programming
- Domain specific languages

Language Oriented Programming

Domain
Specific
Language



General
Purpose
Language



Language Oriented

- **Examples**

- SQL, regular expressions, CSS selectors

- **Benefits**

- Specialized language can increase productivity
- Narrows gap between you and customer
- Easy to separate concerns

- **Drawbacks**

- Can be expensive to create (and possibly learn)
- Difficult to implement in C#

Language Oriented C#

- **Extension methods**
 - For better APIs
- **Expression trees**
 - For static reflection
- **Funcs and Actions**
 - For functional, declarative programming

Example Scenario

- Scheduling tasks for periodic execution

```
public class ScheduledTask {  
    public ScheduledTask(ITask task,  
                        TimeSpan interval,  
                        TimeSpan expiration) {  
        Task = task;  
        Interval = interval;  
        Expiration = expiration;  
    }  
  
    public ITask Task { get; protected set; }  
    public TimeSpan Interval { get; protected set; }  
    public TimeSpan Expiration { get; protected set; }  
    ...  
}
```

```
var task = new ScheduledTask(  
    new AccountSynchronizationTask(),  
    new TimeSpan(0, 0, 2, 0),  
    new TimeSpan(2, 0, 0, 0));
```



Goals

- **Readability**
 - Easier to maintain
- **Essence over ceremony**
 - Remove language clutter

```
var task = new ScheduledTask(  
    new AccountSynchronizationTask(),  
    new TimeSpan(0, 0, 2, 0),  
    new TimeSpan(2, 0, 0, 0));
```

Named parameters

- Only a small step forward
 - Particularly useful when combined with optional parameters
 - Gives reader a clue when using constants

```
var task = new ScheduledTask(  
    Tasks.AccountSynchronization,  
    runEvery: new TimeSpan(0, 0, 2, 0),  
    expiresIn: new TimeSpan(2, 0, 0, 0));
```


Extension Methods

- Extend types!
 - Even sealed types, generic types, and interfaces

```
public static class StringExtensions
{
    public static int ToInt32(this string value)
    {
        return Int32.Parse(value);
    }
}
```

```
int value = "32".ToInt32();
```

Fluent APIs

- A readable API
 - Often uses method chaining

```
var then = 2.Minutes().Ago();
```

```
public static TimeSpan Minutes(this int value)
{
    return new TimeSpan(0, 0, value, 0, 0);
}

public static DateTime Ago(this TimeSpan value)
{
    return SystemTime.Now() - value;
}
```

Putting It Together

```
public static class Tasks
{
    public static ScheduledTask Schedule(ITask task,
                                        TimeSpan runEvery,
                                        TimeSpan runOnce,
                                        TimeSpan expireIn)
    {
        return new ScheduledTask(
            new AccountSynchronizationTask(),
            new TimeSpan(0, 0, 2, 0),
            new TimeSpan(2, 0, 0, 0));
    }

    public static AccountSynchronizationTask AccountSynchronization
    {
        get { return new AccountSynchronizationTask(); }
    }
}

var task = Tasks.Schedule(Tasks.AccountSynchronization,
                          runEvery: 2.Minutes(),
                          expireIn: 5.Days());
```

Validation Example

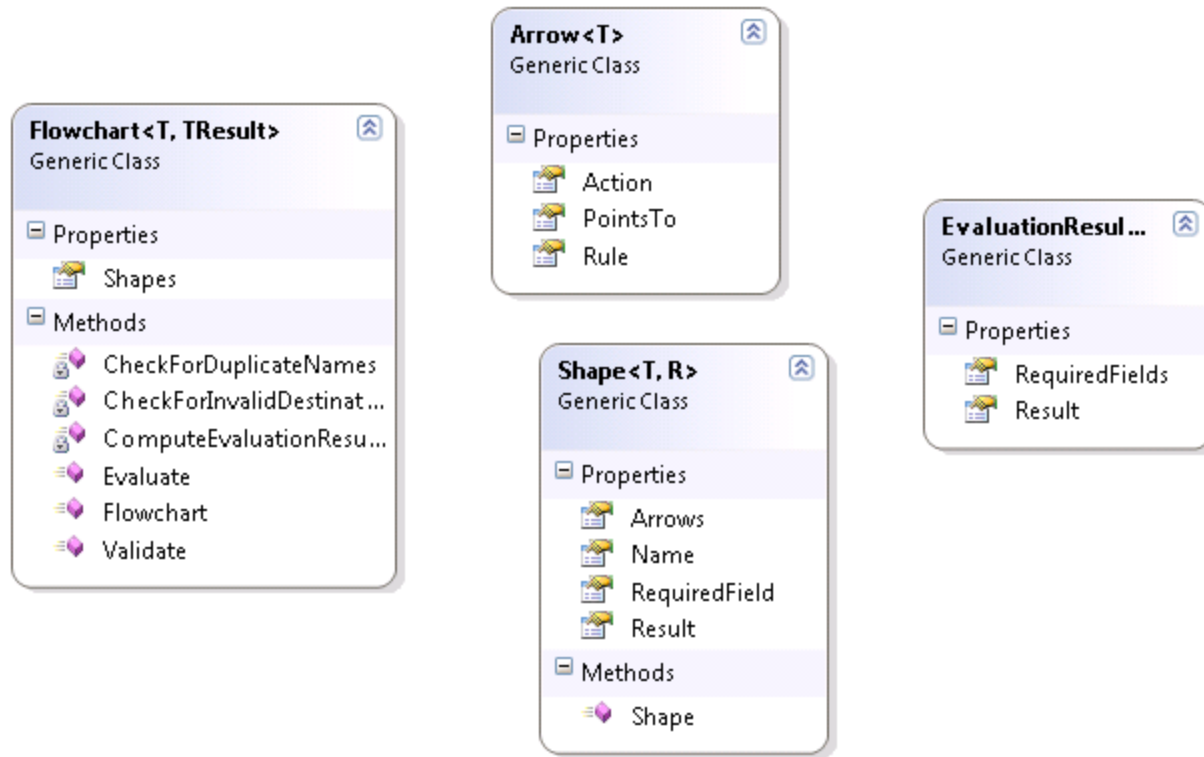
- Dealing with requirements in the form of complex flowcharts
 - Model them with procedural if/else code
 - Or ...

```
var chart = new MovieFlowchart();

chart.WithShape("CheckTitle")
    .RequiresField(m => m.Title)
    .WithArrowPointingTo("CheckLength")
        .AndTheRule(m => !String.IsNullOrEmpty(m.Title))
    .WithShape("CheckLength")
        .RequiresField(m => m.Length)
        .WithArrowPointingTo("BadMovie")
            .AndTheRule(m => m.Length > 120)
        .WithArrowPointingTo("GoodMovie")
            .AndTheRule(m => m.Length == 75)
        .WithArrowPointingTo("CheckReleaseDate")
            .AndTheRule(m => m.Length.HasValue)
        . . .
    .WithShape("BadMovie").YieldingResult(MovieResult.BadMovie)
    .WithShape("GoodMovie").YieldingResult(MovieResult.GoodMovie);
```

Semantic Model

- In-memory representation of what a DSL describes
 - A domain model built by the DSL



Building the Fluent API / DSL

- Heavy use of extension methods

```
public static Flowchart<T, R> WithShape<T, R>(
    this Flowchart<T, R> chart, string shapeName)
{
    var shape = new Shape<T, R> { Name = shapeName };
    chart.Shapes.Add(shape);
    return chart;
}

public static Flowchart<T, R> YieldingResult<T, R>(
    this Flowchart<T, R> chart, R result)
{
    chart.LastShape().Result = result;
    return chart;
}
```

Taking Advantage of Expression<T>

- Expression<T> can yield rich meta-data about a piece of code
 - “Static” reflection

```
public PropertySpecifier(Expression<Func<T, object>> expression)
{
    if(expression.Body is MemberExpression)
    {
        var me = expression.Body as MemberExpression;
        _propertyName = me.Member.Name;
    }
    else if(expression.Body is UnaryExpression)
    {
        var ue = expression.Body as UnaryExpression;
        var me = ue.Operand as MemberExpression;
        _propertyName = me.Member.Name;
    }
}
```

My Top 10 Rules



Rule #10: Avoid Regions

```
public class AccountManager
{
    #region UserManagement
    ...
#endregion

    #region RoleManagement
    ...
#endregion

    #region DataAccess
    ...
#endregion
}
```

Rule #9: Use Exceptions For Errors

- ... instead of status code or booleans
- ... but not for control flow

```
public bool Login(string userName, string password)
{
    if (String.IsNullOrEmpty(userName))
    {
        return false;
    }

    // ....

    return true;
}
```

Rule #8: Avoid Boolean Parameters

```
public void WriteFile(byte[] contents, bool flush)
{
    // ....
}
```

```
storage.WriteFile(data, false);
```

```
public void WriteFile(byte[] contents)
{
    WriteFile(contents, false);
}

public void WriteFileAndFlush(byte[] contents)
{
    WriteFile(contents, true);
}

private void WriteFile(byte[] contents, bool flush)
{
    // ...
}
```

Rule #7: Avoid Too Many Parameters

```
public void LoginUser(string username, string password,  
                    IPAddress ipAddress, bool persist)  
{  
    // ...  
}
```

```
public void LoginUser(UserLoginRequest request)  
{  
    // ...  
}
```

Rule #6: Warnings Are Errors

Application

Build*

Build Events

Debug

Resources

Services

Settings

Reference Paths

Signing

Security

Publish

Code Analysis

Configuration: Active (Debug) Platform: Active (x86)

General

Conditional compilation symbols:

☒ Define DEBUG constant

☒ Define TRACE constant

Platform target: x86

☐ Allow unsafe code

☐ Optimize code

Errors and warnings

Warning level: 4

Suppress warnings:

Treat warnings as errors

☐ None

☒ All

☐ Specific warnings:

Output

Output path: bin\Debug\ Browse...

☐ XML documentation file:

Rule #5: Encapsulate Complex Expressions

```
if (account.Balance > 0 && !account.IsVip && account.DueDate > CurrentDate)
{
    // send off a warning
}
```

```
if (account.IsPastDue)
{
    // ...
}
```

Rule #4: Try To Avoid Multiple Exits

```
if (account.Balance < 10000)
{
    return false;
}
else if (account.IsPastDue)
{
    return false;
}
else if (account.IsVip)
{
    return false;
}

return true;
```

```
var isValid = true;

if (account.Balance < 10000)
{
    isValid = false;
}
else if (account.IsPastDue)
{
    isValid = false;
}
else if (account.IsVip)
{
    isValid = false;
}

return isValid;
```

Rule #3: Try To Avoid Comments

```
// ensure we don't send a notification to
// important people
if (account.IsVip && account.IsPastDue)
{
    SendNotification = false;
}
```

```
CancelNotificationForVipAccounts();
```


Rule #2: Keep Methods Short

```
private string BuildLogParameters(HttpContext context, string SubModuleKey, string ownerType, string PageType, string Url)
{
    StringBuilder _parameters = new StringBuilder("");
    if (PageType == "Rapid") //should no longer have any overlap between indigo and rapid;
    {
        _parameters.Append("R,");
    }
    else if (PageType == "Indicator")
    {
        _parameters.Append("I,");
    }
    else
    {
        _parameters.Append("U,");//Unknown
    }
    _parameters.Append(SessionObject.UserId + ",");
    _parameters.Append(SessionObject.PatientDataAccess + ",");
    _parameters.Append(System.DateTime.Now);
    _parameters.Append(",");
    _parameters.Append(SubModuleKey + ",");
    _parameters.Append(ownerType + ",");
    if (context.Request.QueryString == null)
    {
        _parameters.Append("null");
    }
    else
    {
        _parameters.Append(context.Request.QueryString.ToString());
    }
    _parameters.Append(",");
    if (context.Request.QueryString["VisitIdentity"] != null)
    {
        _parameters.Append(context.Request.QueryString["VisitIdentity"].ToString());
    }
    else
    {
        _parameters.Append("null");
    }
    _parameters.Append(",");
    _parameters.Append(context.Request.RawUrl);
    _parameters.Append(",");
    _parameters.Append(PageType[0]);
    _parameters.Append(",");
    if (SubModuleKey == "0" || ownerType == "0")
        _parameters.Append(((Page)context.CurrentHandler).Title);
    else
        _parameters.Append("");
    _parameters.Append(",");
    _parameters.Append(Url);

    return (_parameters.ToString());
}
```

Rule #1: Keep Classes Small

```
private string BuildLogParameters(HttpContext context, string SubModuleKey, string ownerType, string PageType, string Url)
{
    StringBuilder _parameters = new StringBuilder("");
    if (PageType == "Rapid") //should no longer have any overlap between indigo and rapid;
    {
        _parameters.Append("R,");
    }
    else if (PageType == "Indicator")
    {
        _parameters.Append("I,");
    }
    else
    {
        _parameters.Append("U,");//Unknown
    }
    _parameters.Append(SessionObject.UserId + ",");
    _parameters.Append(SessionObject.PatientDataAccess + ",");
    _parameters.Append(System.DateTime.Now);
    _parameters.Append(",");
    _parameters.Append(SubModuleKey + ",");
    _parameters.Append(ownerType + ",");
    if (context.Request.QueryString == null)
    {
        _parameters.Append("null");
    }
    else
    {
        _parameters.Append(context.Request.QueryString.ToString());
    }
    _parameters.Append(",");
    if (context.Request.QueryString["VisitIdentity"] != null)
    {
        _parameters.Append(context.Request.QueryString["VisitIdentity"].ToString());
    }
    else
    {
        _parameters.Append("null");
    }
    _parameters.Append(",");
    _parameters.Append(context.Request.RawUrl);
    _parameters.Append(",");
    _parameters.Append(PageType[0]);
    _parameters.Append(",");
    if (SubModuleKey == "0" || ownerType == "0")
        _parameters.Append(((Page)context.CurrentHandler).Title);
    else
        _parameters.Append("");
    _parameters.Append(",");
    _parameters.Append(Url);

    return (_parameters.ToString());
}
```

Summary

- **Language Oriented Programming**
 - **Useful in complex domains**
 - **Increases productivity, maintainability, separation of concerns**
- **Trade Offs**
 - **Effort required to create DSL / Fluent API in C#**