Ambientes de Desenvolvimento Avançados

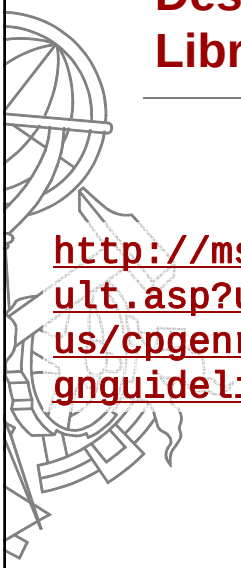
http://www.dei.isep.ipp.pt/~jtavares/ADAV/ADAV.htm

Aula Teórico-Prática

Engenharia Informática

2006/2007

José António Tavares
jrt@isep.ipp.pt



Design Guidelines for Class Library Developers

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp

Design Guidelines for Class Library Developers

Capitalization Styles

Use the following conventions for capitalizing identifiers.

Pascal case

The first letter in the identifier and the first letter of each subsequent concatenated word are capitalized. You can use Pascal case for identifiers of three or more characters. For example:

```
BackColor
```

Camel case

The first letter of an identifier is lowercase and the first letter of each subsequent concatenated word is capitalized. For example:

```
backColor
```


Design Guidelines for Class Library Developers

Capitalization Styles

Uppercase

All letters in the identifier are capitalized. Use this convention only for identifiers that consist of two or fewer letters. For example:

```
System.IO

System.Web.UI
```

You might also have to capitalize identifiers to maintain compatibility with existing, unmanaged symbol schemes, where all uppercase characters are often used for enumerations and constant values. In general, these symbols should not be visible outside of the assembly that uses them.

Design Guidelines for Class Library Developers

Identifier	Case	Example
Class	Pascal	**AppDomain**
Enum type	Pascal	**ErrorLevel**
Enum values	Pascal	**FatalError**
Event	Pascal	**ValueChange**
Exception class	Pascal	**WebException** **Note** Always ends with the suffix **Exception**.
Read-only Static field	Pascal	**RedValue**
Interface	Pascal	**IDisposable** **Note** Always begins with the prefix `I`.
Method	Pascal	**ToString**
Namespace	Pascal	**System.Drawing**
Parameter	Camel	**typeName**
Property	Pascal	**BackColor**
Protected instance field	Camel	**redValue** **Note** Rarely used. A property is preferable to using a protected instance field.
Public instance field	Pascal	**RedValue** **Note** Rarely used. A property is preferable to using a public instance field.

Design Guidelines for Class Library Developers

Case Sensitivity

To avoid confusion and guarantee cross-language interoperation, follow these rules regarding the use of case sensitivity:

- Do not use names that require case sensitivity. Components must be fully usable from both case-sensitive and case-insensitive languages. Case-insensitive languages cannot distinguish between two names within the same context that differ only by case. Therefore, you must avoid this situation in the components or classes that you create.
- Do not create two namespaces with names that differ only by case. For example, a case insensitive language cannot distinguish between the following two namespace declarations.

```
namespace ee.cummings;

namespace Ee.Cummings;
```

Design Guidelines for Class Library Developers

Case Sensitivity

Do not create a function with parameter names that differ only by case. The following example is incorrect.

```
void MyFunction(string a, string A)
```

Do not create a namespace with type names that differ only by case. In the following example, `Point p` and `POINT p` are inappropriate type names because they differ only by case.

```
System.Windows.Forms.Point p

System.Windows.Forms.POINT p
```


Design Guidelines for Class Library Developers

Case Sensitivity

Do not create a type with property names that differ only by case.

In the following example, `int Color` and `int COLOR` are inappropriate property names because they differ only by case.

```
int Color {get, set}

int COLOR {get, set}
```

Design Guidelines for Class Library Developers

Case Sensitivity

Do not create a type with method names that differ only by case.

In the following example, `calculate` and `Calculate` are inappropriate method names because they differ only by case.

```
void calculate()

void Calculate()
```


Design Guidelines for Class Library Developers

Abbreviations

To avoid confusion and guarantee cross-language interoperation, follow these rules regarding the use of abbreviations:

- Do not use abbreviations or contractions as parts of identifier names. For example, use `GetWindow` instead of `GetWin`.
- Do not use acronyms that are not generally accepted in the computing field.
- Where appropriate, use well-known acronyms to replace lengthy phrase names. For example, use `UI` for `User Interface` and `OLAP` for `On-line Analytical Processing`.

Design Guidelines for Class Library Developers

Abbreviations

Other rules regarding the use of abbreviations:

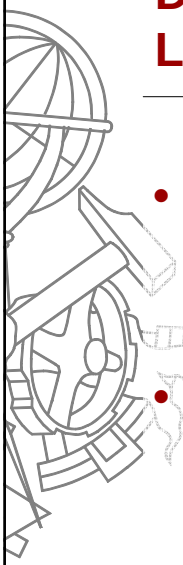
- When using acronyms, use Pascal case or camel case for acronyms more than two characters long. For example, use `HtmlButton` or `htmlButton`. However, you should capitalize acronyms that consist of only two characters, such as `System.IO` instead of `System.Io`.
- Do not use abbreviations in identifiers or parameter names. If you must use abbreviations, use camel case for abbreviations that consist of more than two characters, even if this contradicts the standard abbreviation of the word.

Design Guidelines for Class Library Developers

Word Choice

- Avoid using class names that duplicate commonly used .NET Framework namespaces. For example, do not use any of the following names as a class name: `System`, `Collections`, `Forms`, or `UI`. See the Class Library for a list of .NET Framework namespaces.
- In addition, avoid using identifiers that conflict with keywords

Design Guidelines for Class Library Developers



Avoiding Type Name Confusion

- Different programming languages use different terms to identify the fundamental managed types. Class library designers must avoid using language-specific terminology. Follow the rules described in this section to avoid type name confusion.
- Use names that describe a type's meaning rather than names that describe the type. In the rare case that a parameter has no semantic meaning beyond its type, use a generic name.

Design Guidelines for Class Library Developers

For example, a class that supports writing a variety of data types into a stream might have the following methods.

```
[Visual Basic]
Sub Write(value As Double);
Sub Write(value As Single);
Sub Write(value As Long);
Sub Write(value As Integer);
Sub Write(value As Short);

[C#]
void Write(double value);
void Write(float value);
void Write(long value);
void Write(int value);
void Write(short value);
```

Design Guidelines for Class Library Developers

Do not create language-specific method names, as in the following example.

```
[Visual Basic]
Sub Write(doubleValue As Double);
Sub Write(singleValue As Single);
Sub Write(longValue As Long);
Sub Write(integerValue As Integer);
Sub Write(shortValue As Short);

[C#]
void Write(double doubleValue);
void Write(float floatValue);
void Write(long longValue);
void Write(int intValue);
void Write(short shortValue);
```


Design Guidelines for Class Library Developers

In the extremely rare case that it is necessary to create a uniquely named method for each fundamental data type, use a universal type name. For example, a class that supports reading a variety of data types from a stream might have the following methods.

```
[Visual Basic]
ReadDouble()As Double
ReadSingle()As Single
ReadInt64()As Long
ReadInt32()As Integer
ReadInt16()As Short

[C#]
double ReadDouble();
float ReadSingle();
long ReadInt64();
int ReadInt32();
short ReadInt16();
```

Design Guidelines for Class Library Developers

The preceding example is preferable to the following language-specific alternative.

```
[Visual Basic]
ReadDouble()As Double
ReadSingle()As Single
ReadLong()As Long
ReadInteger()As Integer
ReadShort()As Short

[C#]
double ReadDouble();
float ReadFloat();
long ReadLong();
int ReadInt();
short ReadShort();
```


Design Guidelines for Class Library Developers

Namespace Naming Guidelines

The general rule for naming namespaces is to use the company name followed by the technology name and optionally the feature and design as follows.

```
CompanyName.TechnologyName[.Feature][.Design]
```

For example:

```
Microsoft.Media
Microsoft.Media.Design
```

Prefixing namespace names with a company name or other well-established brand avoids the possibility of two published namespaces having the same name.

For example, `Microsoft.Office` is an appropriate prefix for the Office Automation Classes provided by Microsoft.

Design Guidelines for Class Library Developers

Namespace Naming Guidelines

- Use a stable, recognized technology name at the second level of a hierarchical name. Use organizational hierarchies as the basis for namespace hierarchies. Name a namespace that contains types that provide design-time functionality for a base namespace with the .Design suffix. For example, the System.Windows.Forms.Design Namespace contains designers and related classes used to design System.Windows.Forms based applications.
- A nested namespace should have a dependency on types in the containing namespace. For example, the classes in the System.Web.UI.Design depend on the classes in System.Web.UI. However, the classes in **System.Web.UI** do not depend on the classes in **System.Web.UI.Design**.

Design Guidelines for Class Library Developers

Namespace Naming Guidelines

- You should use Pascal case for namespaces, and separate logical components with periods, as in Microsoft.Office.PowerPoint. If your brand employs nontraditional casing, follow the casing defined by your brand, even if it deviates from the prescribed Pascal case. For example, the namespaces NeXT.WebObjects and ee.cummings illustrate appropriate deviations from the Pascal case rule.
- Use plural namespace names if it is semantically appropriate. For example, use System.Collections rather than System.Collection. Exceptions to this rule are brand names and abbreviations. For example, use System.IO rather than System.IOs.

Design Guidelines for Class Library Developers

Namespace Naming Guidelines

- Do not use the same name for a namespace and a class. For example, do not provide both a Debug namespace and a Debug class.
- Finally, note that a namespace name does not have to parallel an assembly name. For example, if you name an assembly `MyCompany.MyTechnology.dll`, it does not have to contain a `MyCompany.MyTechnology` namespace.

Design Guidelines for Class Library Developers

Class Naming Guidelines

- The following rules outline the guidelines for naming classes:
  - Use a noun or noun phrase to name a class.
  - Use Pascal case.
  - Use abbreviations sparingly.
  - Do not use a type prefix, such as C for class, on a class name. For example, use the class name `FileStream` rather than `CFileStream`.
  - Do not use the underscore character (_).

Design Guidelines for Class Library Developers

Class Naming Guidelines

- Other rules for naming classes:
- Occasionally, it is necessary to provide a class name that begins with the letter I, even though the class is not an interface. This is appropriate as long as I is the first letter of an entire word that is a part of the class name. For example, the class name IdentityStore is appropriate.
- Where appropriate, use a compound word to name a derived class. The second part of the derived class's name should be the name of the base class. For example, `ApplicationException` is an appropriate name for a class derived from a class named `Exception`, because `ApplicationException` is a kind of `Exception`. Use reasonable judgment in applying this rule. For example, `Button` is an appropriate name for a class derived from `Control`. Although a button is a kind of control, making Control a part of the class name would lengthen the name unnecessarily.

Design Guidelines for Class Library Developers

Class Naming Guidelines

The following are examples of correctly named classes.

```
[Visual Basic]
Public Class FileStream
Public Class Button
Public Class String

[C#]
public class FileStream
public class Button
public class String
```

Design Guidelines for Class Library Developers

Interface Naming Guidelines

The following rules outline the naming guidelines for interfaces:

- Name interfaces with nouns or noun phrases, or adjectives that describe behavior. For example, the interface name **`IComponent`** uses a descriptive noun. The interface name **`ICustomAttributeProvider`** uses a noun phrase. The name **`IPersistable`** uses an adjective.
- Use Pascal case.
- Use abbreviations sparingly.
- Prefix interface names with the letter `I`, to indicate that the type is an interface.
- Use similar names when you define a class/interface pair where the class is a standard implementation of the interface. The names should differ only by the letter `I` prefix on the interface name.
- Do not use the underscore character (_).

Design Guidelines for Class Library Developers

Interface Naming Guidelines

The following are examples of correctly named interfaces.

[Visual Basic]
Public Interface IServiceProvider
Public Interface IFormatable

[C#]
public interface IServiceProvider
public interface IFormatable

Design Guidelines for Class Library Developers

Interface Naming Guidelines

The following code example illustrates how to define the interface **`IComponent`** and its standard implementation, the class **`Component`**.

```
[C#]
public interface IComponent
{
   // Implementation code goes here.
}
public class Component: IComponent
{
   // Implementation code goes here.
}
```


Design Guidelines for Class Library Developers

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp

- Relationship to the Common Type System and the Common Language Specification
  Describes the role of the common type system and the Common Language Specification in class library development.
- Naming Guidelines
  Describes the guidelines for naming types in class libraries.
- Class Member Usage Guidelines
  Describes the guidelines for using properties, events, methods, constructors, fields, and parameters in class libraries.

Design Guidelines for Class Library Developers

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp

- Type Usage Guidelines
  Describes the guidelines for using classes, value types, delegates, attributes, and nested types in class libraries.
- Guidelines for Exposing Functionality to COM
  Describes the guidelines for exposing class library types to COM.
- Error Raising and Handling Guidelines
  Describes the guidelines for raising and handling errors in class libraries.
- Array Usage Guidelines
  Describes the guidelines for using arrays in class libraries and how to decide whether to use an array vs. a collection.

Design Guidelines for Class Library Developers

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp

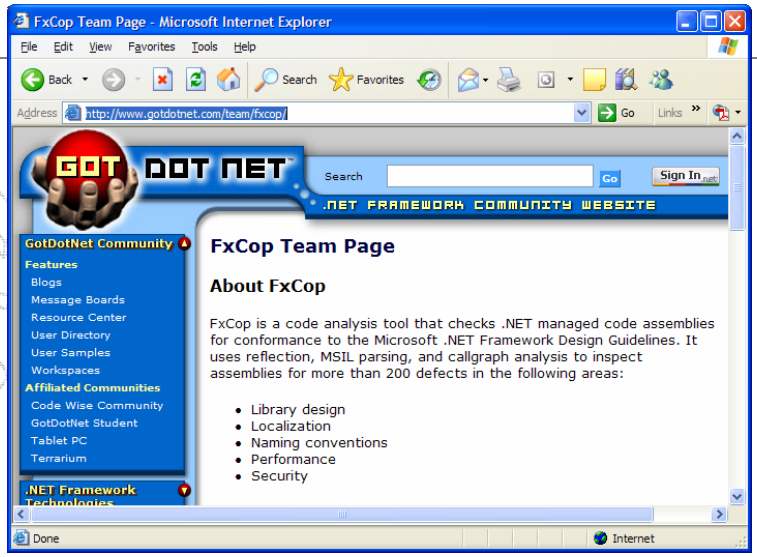
- Operator Overloading Usage Guidelines
  Describes the guidelines for implementing operator overloading in base class libraries.
- Guidelines for Implementing Equals and the Equality Operator (==)
  Describes the guidelines for implementing the **Equals** method and the equality operator (==) in class libraries.
- Guidelines for Casting Types
  Describes the guidelines for casting types in class libraries.
- Common Design Patterns
  Describes how to implement design patterns for **Finalize** and **Dispose** methods, the **Equals** method, callback functions, and time-outs.

Design Guidelines for Class Library Developers

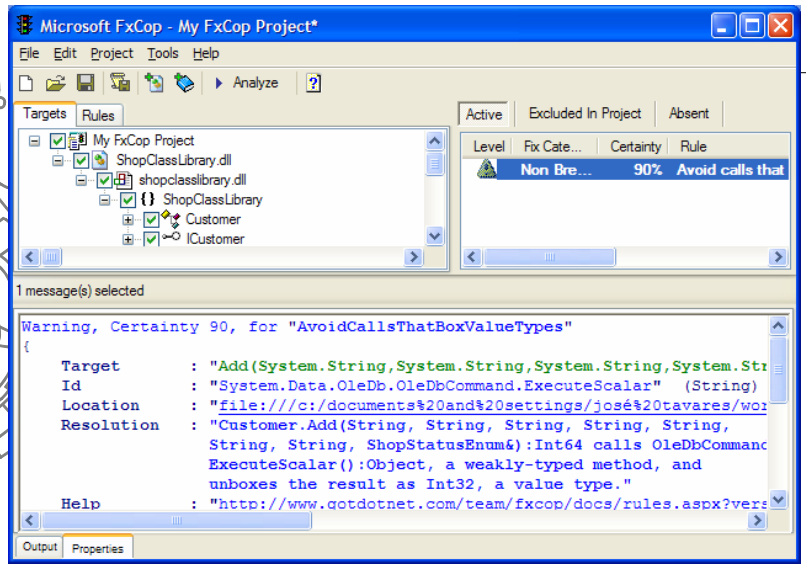
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp

- Security in Class Libraries
  Describes the precautions to take when writing highly trusted class library code, and how to help protect resources with permissions.
- Threading Design Guidelines
  Describes the guidelines for implementing threading in class libraries.
- Guidelines for Asynchronous Programming
  Describes the guidelines for implementing asynchronous programming in class libraries and provides an asynchronous design pattern.

FxCop (http://www.gotdotnet.com/team/fxcop/)



FxCop (http://www.gotdotnet.com/team/fxcop/)

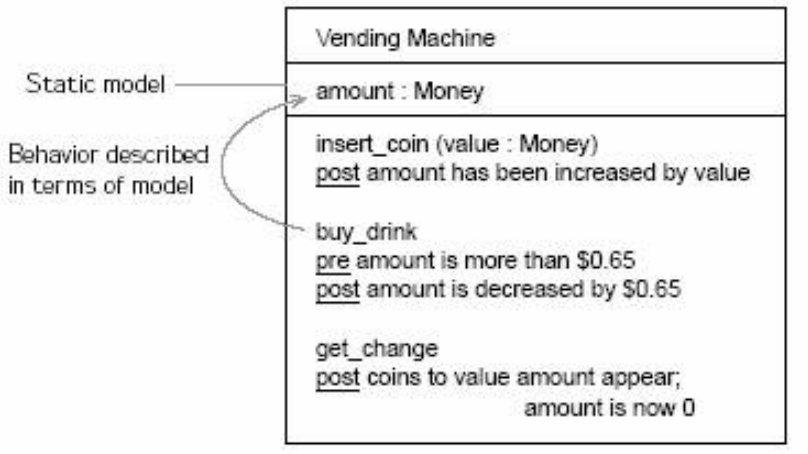


Exercício

```
public interface ISale
{
    /// <pre> ... </pre>
    /// <post> ... </post>
    ///
    DataSet CreateDetails(string user, string pass);

    /// <pre> ... </pre>
    /// <post> ... </post>
    ///
    ShopStatusEnum Add(string user, string pass,
                       long customerID, DateTime date,
                       DataSet dsDetails);
}
```

Pré & Post Conditions



Questões

?