Sumário

Apresentação

O propósito deste documento é padronizar o desenvolvimento de todos os projetos da SGI no que se refere à sua estrutura, código fonte e controles do ambiente de desenvolvimento. Esta padronização tem como finalidade manter a clareza e a manutenibilidade da aplicação, permitindo compreensão homogênea do código fonte de todos os projetos.

Os padrões adotados neste documento seguem as diretrizes da linguagem de programação C# da Microsoft, adotada pela SGI como linguagem de programação oficial para o desenvolvimento de novos projetos.

Estilos de Nome

A nomenclatura proposta segue duas convenções bem conhecidas:

- ***P**ascal**C**asing*: Esta convenção torna maiúscula a primeira letra de cada palavra que compõe um nome. Exemplo: **N**ome**P**essoa, **G**erenciador**D**e**U**suarios;

- ***c**amel**C**asing*: Esta convenção torna maiúscula a primeira letra de cada palavra, exceto a da primeira. Exemplo: **n**ome**P**essoa**F**isica, **e**mail**U**suario.

Idioma

De maneira geral, todos os nomes devem ser escritos em **português**.

Pelo fato da API .NET ser definida no idioma inglês, exceções a esta regra ocorrerão nos casos em que houver a necessidade de referenciar e/ou estender recursos da API.

Organização dos Arquivos

Namespaces e Layout de Diretórios

Um *layout* de diretório consiste numa estrutura de pastas no sistema de arquivos. Em se tratando de espaços de nomes, sua estrutura lógica deverá coincidir com a estrutura física. Por exemplo:

Namespaces	Layout de Diretórios
`Cadastro.Usuario`	.../Cadastro/Usuario
`Cadastro.Colaborador`	.../Cadastro/Colaborador

Todo *namespace* está prefixado com o nome do modulo o qual pertence, seguido da sua hierarquia interna especifica. No exemplo, “Cadastro” é o módulo, enquanto que “Usuario” e “Colaborador” correspondem à hierarquia interna específica de cada *namespace*.

Arquivos Fontes C#

- Por padrão, cada classe/interface deverá ser mantida em um arquivo separado cujo nome deverá coincidir com o nome da classe/interface. Veja o exemplo abaixo:
  - Nome do arquivo da classe: *UsuarioDAO.cs.*
  - Nome da classe: *UsuarioDAO.*

- Há a possibilidade de manter mais de uma classe/interface em um mesmo arquivo quando:
  - Houver a implementação de *inner classes* (classes internas);
  - Classes/Interfaces que possuírem vínculos consideravelmente dependentes entre si.

- Para arquivos *.aspx do projeto web, que são compostos pela página (*.aspx) e os arquivos de códigos (*.cs e *.designer.cs) o padrão ficará da seguinte forma:

- Nome dos arquivos aspx, cs e designer.cs :
  - <Substantivo> + <Assunto> + .extensao.
  - Exemplos:
    - CadastroUsuario.aspx
    - CadastroUsuario.cs
    - CadastroUsuario.designer.cs

Código Fonte

Nas seções seguintes são abordadas as padronizações das principais estruturas da linguagem C# e algumas convenções de estilo.

Cabeçalhos

Um cabeçalho deve existir em cada arquivo de código fonte para identificar e explicar o seu real propósito. O modelo de cabeçalho é tal como segue:

```
/************************************************************
 * Nome do Sistema.
 * **********************************************************
 * Este arquivo...
 * SGI - All rights reserved ©.
 *
 * Criado por:
 * Data da Criação:
 * Modificado por:
 * Data da modificação:
 * Observação:
 * **********************************************************
 */
```

Comentários

Há vários tipos de comentários que normalmente ocorrem dentro de trechos de código fonte. Utilizaremos os que seguem:

Comentário de linha

- Usar para explicar uma linha de código **sempre** quando esta não for suficientemente óbvia. Exemplo:

```
int id = 10;    //Numero de identificação do usuário
```

Comentário de bloco

- Usar para explicar aspectos do programa que requerem mais de uma linha.

```
/*
Este é um comentário
de bloco que faz uso
de mais de uma linha.
*/
```

Summary.

- Deverá ser utilizado obrigatoriamente em métodos da arquitetura.
- Opcionalmente poderá ser utilizado em métodos criados pelos desenvolvedores, utilizando o bom senso. Em geral, todos os métodos das classes da Aplicação e da DAO deverão possuir summary para facilitar o entendimento do código fonte e para geração da documentação do projeto.

```
/// <summary>
/// Método para a criação do registro de feriado
/// </summary>
/// <param name="feriado">Objeto a ser criado no banco de dados</param>
public void Inserir(Feriado feriado)
{
    //...
}
```

Declarações

Nas seções seguintes são abordadas as regras de declaração seguindo as convenções de nomes e a semântica de cada tipo: classes e seus membros, interfaces, enumerações e exceções.

Classes

- Convenção de nome:
  - ***P**ascal**C**asing*.
- Semântica:
  - São substantivos ou frase-substantivo;
  - Se uma instância da classe representar uma única ocorrência individual segundo alguma regra de negócio, então deixar o nome da classe no **singular**. Não faz sentido uma classe se chamar Usuarios, se uma instância desta classe representar um único indivíduo. Entretanto, faz sentido deixar uma classe no plural se uma instância desta classe for utilizada para representar **uma coleção de outros objetos**. Exemplo:

```
//Instância de um único objeto
Usuario usuario = new Usuario();

//Instancia de uma coleção de objetos
List<Usuario> usuarios = new List<Usuario>();
```

Atributos

- Convenção de nome:
  - Atributos públicos seguem o estilo ***P**ascal**C**asing*, protegidos e privados seguem o estilo **c**amel**C**asing.
  - Atributos estático e constante de classe são tabulados e terão todas as letras maiúsculas. Exemplo:

```
public class Matematica
{
   public const PI    = 3.1415;
   public const DELTA = 0.01;
   public string Nome = string.Empty;
   private string nome = "Renato Tozzi";
}
```

- Semântica:
  - Substantivo ou frase-substantivo.

Propriedades

- Convenção de nome:
  - Propriedades públicas seguem o estilo **P**ascal**C**asing, protegidas e privadas seguem o estilo **c**amel**C**asing.
  - Uma propriedade pode ser definida em modo de leitura, escrita, o ou leitura e escrita. A definição do modo da propriedade é feito através das palavras reservadas **get** e **set**. Por esta razão, não justifica a presença do prefixo **Set** ou **Get** em métodos de classes. Exemplo:

Não usar	Usar

`class Pessoa`
`{`
`  public string GetNome()`
`  {`
`  return this.nome;`
`  }`
`}`
	`class Pessoa`
	`{`
	`  public string Nome`
	`  {`
	`  get {return this.nome;}`
	`  }`
	`}`

Métodos

- Convenção de nome:
  - ***P**ascal**C**asing*
- Semântica: verbos.
- O nome da classe já deve representar o assunto do negócio, então não faz sentido repetir o assunto na assinatura do método. Exemplo:

Não usar	Usar

`class UsuarioDAO`
`{`
`  public bool InserirUsuario(Usuario usuario)`
`  {`
`  //...;`
`  }`
`}`
	`class UsuarioDAO`
	`{`
	`  public bool Inserir(Usuario usuario)`
	`  {`
	`  //...;`
	`  }`
	`}`

  - Aplicar **sobrecarga de métodos** e não "inventar" nomes para os métodos. Exemplo:

Não usar	Usar

`class UsuarioDAO`
`{`
`  public Usuario ObterPorID(int id)`
`  {`
`  //...;`
`  }`
 	
`  public Usuario ObterPorNome(string nome)`
`  {`
`  //...;`
`  }`
`}`
	`class UsuarioDAO`
	`{`
	`  public Usuario Obter(int id)`
	`  {`
	`  //...;`
	`  }`
	
	`  public Usuario Obter(string nome)`
	`  {`
	`  //...;`
	`  }`
	`}`

Variáveis Locais

- Convenção de nome:
  - ***c**amel**C**asing*.
- Regras:
  - Declarar as variáveis em português e com um nome intuitivo. Exemplo:

Não usar	Usar

`string user = "Renato";`
`int x = 1;;`
`Usuario user = new Usuario()`
	`string usuario = "Renato";`
	`int idUsuario = 1;`
	`Usuario usuario = new Usuario()`

Interfaces

- Convenção de nome: ***P**ascal**C**asing* prefixado com a letra ***I***.
- Semântica:
  - Deverão ser de caráter qualificativo (adjetivo), um substantivo ou frase-substantivo que descreva um comportamento comum. Exemplo: **I**Comparavel, **I**Enumeravel, **I**Conectavel, **I**Componente.

Enumerações

- Convenção de nome: ***P**ascal**C**asing.*
- Semântica:
  - Deverão ser substantivos ou frase-substantivo previxado com a letra ***E***. Os itens que o compõe devem ser declarados em letra maiúscula. Exemplo:

```
public enum ETipoPessoa
{
     FISICA = 1,
     JURIDICA
}
```

Exceções

- Convenção de nome:
  - ***P**ascal**C**asing* prefixado com a palavra *Excecao*. (naturalmente ela será derivada direta ou indiretamente da classe da API .NET “Exception”). Exemplo:

```
class ExcecaoImpossivelAutenticar: System.Exception
{
    private string nomeBanco;

    public ExcecaoImpossivelAutenticar(string msg, string nomeBanco): base(msg)
    {
         this.nomeBanco = nomeBanco;
    }

    public override string Message
    {
         get
         {
                return base.Message + “Bd: ” + this.nomeBanco;
         }
    }
}
```

Sentenças

Simples

Cada linha deve conter apenas uma sentença. Exemplo:

Não use	Use
`nome = “joão”; sobrenome = “silva”;`	`nome = “Renato”;` `sobrenome = “Tozzi”;`

Retorno

Uma sentença de retorno não deve usar parênteses mais externos desnecessários:

Não use	Use
`return (n * (n + 1) / 2);` `return (n*(n+1)/2);`	`return n * (n + 1) / 2;`

Condicionais

Sentenças **if**, **if-else** e **else-if** devem obedecer a seguinte estrutura:

```
if (condiçao)
{
    FacaAlgo();
}

if (condição)
{
    FacaAlgo();
}
else
{
    FacaAlgo();
}
if (condição)
{
    FacaAlgo();
}
else if (condição)
{
    FacaAlgo();
}
```

A sentença **switch** deve ser como segue:

```
switch (condição)
{
   case A:
   {
     ...
     break;
   }
   case B:
   {
     ...
     break;
   }
   case C:
   {
     ...
     break;
   }
   default:
   {
     ...
     break;
   }
}
```

Repetição

As sentenças **for** e **foreach** devem ser como segue (note o espaçamento entre as variáveis):

```
for (int i = 0; i < 5; ++i)
{
      ...
}
```

```
foreach (Usuario usuario in usuarios)
{
        ...
}
```

As sentenças **while** e **do**-**while** devem ser como segue:

```
while (condição)
{
        //...
}

do
{
        //...
} while (condição);
```

Tratamento de Erro

A sentenças **try-catch** deve ser como segue:

```
try
{
    //...
}
catch (Exception excecao)
{
    //...
}

//...

try
{
    //...
}
catch (Exception excecao)
{
    //...
}
finally
{
    //...
}
```

Estética

Linhas em branco

Uma linha (**apenas uma linha**) em branco sempre deve ser usada:

- Entre métodos e construtores;
- Entre a última definição de atributo e os membros seguintes;
- Entre variáveis locais dentro de um método e sua primeira sentença;
- Antes de comentários de uma linha ou um bloco.
- Seções do código com um propósito lógico em comum.

Identação

- Sempre use TAB ao invés de espaços em branco.
- Use quebras de linhas identadas quando os limites horizontais de impressão do código forem extrapolados. Exemplo:

```
Pessoa pessoaFisica = new PessoaFisica(
      txtNomePessoaFisica.Text,
      txtCpfPessoa.Text,
      Estado.ObterObjeto(ddlEstadoPessoaFisica.SelectedValue)
);
```

Espaçamento Entre Termos

- O espaçamento entre os argumentos de um método deve obedecer ao que segue:

Não use	Use

`Metodo(a,b,c);`
`Metodo( a, b, c );`
`Metodo( a , b , c );`
	`Metodo(a, b, c);`

- Use um único espaço em branco entre os operadores binários. Exemplo:

Não use	Use
`v = 2  +  3;  //excesso de espaço`	`v = 2 + 3;`
`v = 2+3;  //sem espaço`	
`logico = a&&b;  //sem espaço`	`logico = a && b;`
`v=2;  //sem espaço`	`v = 2;`

- Não use espaço em branco entre operadores unários. Exemplo:

Não use	Use
`++ valor;  //com espaço`	`++valor;`
`valor = - 7;  //com espaço`	`valor = -7;`
`valor --;  //com espaço`	`valor--;`

Uso de Chaves

A chave de abertura para classes, interfaces, construtores e métodos deve ocorrer na linha abaixo da declaração, sem deixar nenhuma linha em branco. A chave de fechamento deve ser na ultima linha do escopo (bloco) e deve ficar sozinha na linha, como segue o exemplo:

```
class Pessoa
{
      private int nome;

      public Pessoa(string nome)
      {
            this.nome = nome;
      }
```

```
        public string Nome
        {
            get { return this.nome;  }
        }
    }
```

No código acima, a estrutura get que defini o corpo da propriedade fugiu à regra, visto que a sentença return que a compôs (resumida em uma única linha) foi simples o suficiente para não atrapalhar a legibilidade do código.

Padronização de Controles

Todos os controles/componentes do Visual Studio 2010 são nomeados (atributo ID) segundo o estilo ***c**amel**C**ase*. O identificador do controle, entretanto, é prefixado com letras que sugerem o tipo do controle. Exemplos:

```
Menu                    ID = "mnuCabecalho"
TreeView                ID = "tvwMenu"
DropDownList            ID = "ddlDepartamento
GridView                ID = "gvNota"
Image                   ID = "imgSaida"
HyperLink               ID = "lnkkTitulo"
Label                   ID = "lblEmail"
TextBox                 ID = "txtCorpo"
Panel                   ID = "pnlEditor"
ObjectDataSource        ID = "odsComentario"
DetailsView             ID = "dvComenatrio"
DataList                ID = "dlstCategoria"
Button                  ID = "btnVote"
Localize                ID = "locTema"
CheckBox                ID = "chkFechamento"
CheckBoxList            ID = "chklCargos"
Calendar                ID = "calCalendario"
ListBox                 ID = "lstCursos"
RadioButton             ID = "rbtProva"
RadioButtonList         ID = "rbtlOpcoes"
Wizard                  ID = "wizEncomenda"
LinkButton              ID = "lnkbtnTitulo"
ImageButton             ID = "imgbtnCadastrarUsuario"
FileUpload              ID = "fupFotos"
MultiView               ID = "mvEncomenda"
FormView                ID = "fvCursos"
Repeater                ID = "repItens"
CalendarExtender        ID = "calData"
MaskedEditExtender      ID = "mkeValor"
TabPanel                ID = "tbpCliente"
PasswordStrength        ID = "pwsSenha"
TabContainer            ID = "tbcCliente"
AccordionPane           ID = "acpDetalhe"
TextBoxWatermarkExtend  ID = "txtwNome"
Accordion               ID = "accDetalhe"
FilteredTextBoxExtender ID = "ftbNumero"
CollapsiblePanelExtender ID = "cpexDetalhe"
ModalPopupExtender      ID = "mpexConsulta"
ContentPlaceHolder      ID = "cphConteudo"
HiddenField             ID = "hfID"
```

Formatação de tags HTML e ASP.Net

- Organize as tags HTML e ASP.Net sempre em TAB ao invés de espaços em branco.
- Ao iniciar uma tag mantenha a identação para fechá-la na mesma coluna.
- Não faça quebra de linha no meio de uma tag a não ser que exceda a largura de 80 dígitos.
- Tag HTML ou tags clientes deverão estar em letras minusculas, assim como seus atributos, seguindo o padrão W3C.
- Tag ASP.Net deverá estar em letra maiúscula, e seus atributos em letras minúsculas.
- Ao colar tags no editor do Visual Studio, fazer com que o ambiente formate automaticamente.

Histórico do Documento

- HISTÓRICO DO DOCUMENTO -			
Data	Versão	Autor	Descrição
17/08/2010	1.0	Renato Tozzi	Criação do documento.
19/08/2010	1.0	Renato Tozzi	Exemplificação do tópico “Métodos” e “Variáveis”, sugerido pelo Luiz Augusto.
20/08/2010	1.0	Renato Tozzi	Inserção do “Summary”, sugerido pelo Cláudio.