**Desenvolvimento de Software – Aula 1**

**Professor: Fabiano Gonçalves**

ABES – Pesquisa sobre o mercado mundial de Software

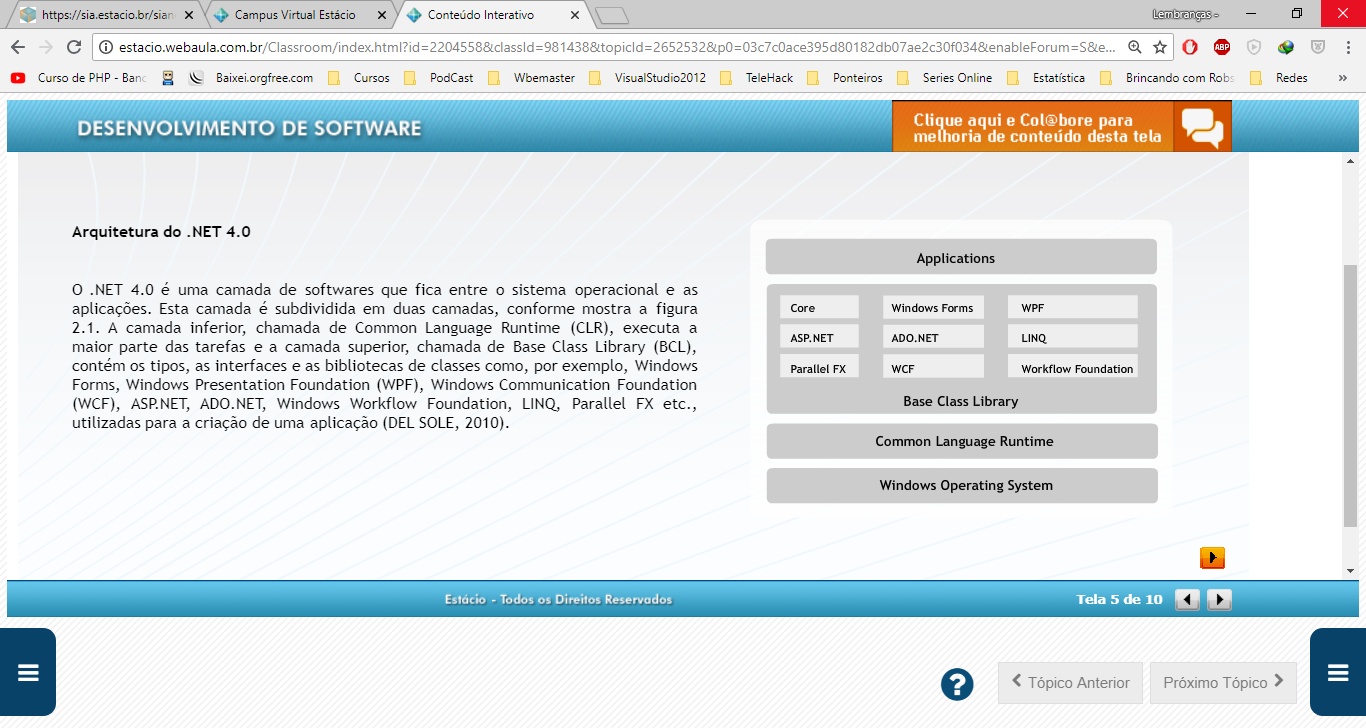
1. **Plataforma .NET Framework 4.0**

É uma **plataforma** que *interage* *com diversos* *conjuntos de linguagem de programação*, *auxilia na produção de aplicações*, *implantação* e *execução de aplicações* em gerenciamento de ambiente como o MSDN-2010.

O .NET é reconhecido como **tecnologia** por *possuir diversas bibliotecas*. Entre as diversas, podemos encontrar a **vbc.exe**, o *compilador do Visual Basic* (VB).

1. **Arquitetura do .NET 4.0**

O .NET é uma *camada de software* que *encontra-se* *entre a camada de software e o aplicativo*. Essa camada subdivide-se me duas camadas:



Conforme vemos na imagem, a **camada Inferior**, a denominamos de **COMMON LANGUAGE RUNTIME (CLR)**, responsável por *carregar a maior parte das tarefas*.

A **camada superior**, a chamamos de **BASE CLASS LIBRARY (BCL)**, onde encontramos os *tipos, interfaces e as bibliotecas de classes*. Alguns dos componentes que podemos encontrar são: *Windows Form, Windows Presentation Fundation (WPF), Windows Communication Fundation (WCF), ASP.NET, ADO.NET, Windows Workflow Fundation (WWF), LINQ, Parallel FX*, entre outros que utilizamos para criação de aplicação.

Com o .NET podemos encontrar uma grande vantagem na produtividades de aplicação, uma vez que ele nos oferece uma grande quantidade de classes necessárias para o desenvolvimento de software, sem haver a necessidade de o desenvolvedor ter que cria-las.

O .NET 4.0 é denominado como ***standalone***, pois contém *todos os CLR e BCL*, dispensando assim a necessidade de versões anteriores instaladas.

O que difere o .NET 4.0 das versões anteriores, que eram dependentes uma das outras por não conter todos os componentes. Por exemplo, o *.NET 3.5 que continha o LINQ*, necessitava do *.NET 3.0 que continha o WPF* e também o *.NET 2.0 por conter o Windows Form*. Essas versões, também eram conhecidas como versões incrementáveis.

Nesta ferramenta várias linguagens compartilham o mesmo ambiente de desenvolvimento integrado (IDE). Este ambiente possui um analisador de código que pode oferecer, enquanto o código é digitado, sugestões ao programador, como por exemplo tipos de dados e membros de classes e estruturas, esse analisador é chamado de **IntelliSense**.

1. **Componentes do .NET**



* 1. **Common Languagem Runtime**

**CLR** – Common Language Runtime, em todas as linguagens .NET contém uma camada comum. Permite que a aplicação rode em um sistema gerenciável em que a aplicação se comunica com o .NET. O CLR é responsável pelo controle de execução da aplicação, gerenciamento de memória, gerenciamento de recursos, e acesso aos recursos do sistema e pelos serviços de segurança. Qualquer código escrito em uma linguagem .NET para um ambiente gerenciado, é denominado de **MANAGED CODE**, ou seja, código gerenciado.

* 1. **.NET Assemblies**

Um pacote é gerado, quando os compiladores das linguagens .NET compila os códigos das aplicações. Esse pacote é denominado **.NET ASSEMBLIES (montagem)**. É um arquivo que armazena os códigos em **Microsoft Intermediate Language** (**MSIL**) e **Metadados** (**metadata**).

**MSIL** – É uma linguagem orientada a objetos de alto nível (o mais próximo da linguagem humana), com conjuntos de instruções que independem da CPU.

Também é chamada de Linguagem comum, levando em consideração os códigos implementados em diferentes linguagens .NET que quando compilados, *são denominados de MSIL*.

**Metadados** – esses, contém os tipos, funções, procedimentos e assinaturas, ou seja, contém informações sobre o código implementado.

* 1. **Base Class Library**

**BCL ­**– É onde podemos encontrar diversos tipos, *interfaces* e *classes* que utilizamos na aplicação. A **BCL** Também é *composta* de *diversos Assemblies*. Entre eles, podemos encontrar um dos principais **MsCorlib.dll** (*Microsoft Core Library*). Alguns são para tecnologia específica como *System.ServiceModel.dll* que integra a **BCL** ao **Windows Comunication Foundation** (**WCF**).

**Namespaces** – São os pacotes ou Interfaces que agrupam componentes como interfaces e classes relacionadas a uma tecnologia como por exemplo, o namespace:

- System.Windows.Forms para Windows Forms  
- System. Web para aplicações Web

* 1. **Execução do código**

Ao chamar um .NET Assembly, o código será traduzido em **MSIL** *do .NET Assembly*, por um compilador **just-in-time** (**JIT**), para um código compatível e nativo ao sistema operacional onde o .NET Assembly está instalado. Chamamos de **jitted code** - o código executável nativo.

Podemos também usar uma outra opção denominada **prejitting**, que é *pré-compilar o código MSIL* com a ferramenta **ngen.exe**. *Todos os* *assemblies do BCL são pré-compilados* (**prejitted**).

1. **Instalação do .NET**

Acesse a página web [*http://msdn.microsoft.com/en-us/library/5a4x27ek.aspx*](http://msdn.microsoft.com/en-us/library/5a4x27ek.aspx) para conhecer os requisitos de sistema para instalação do .NET 4.0 e faça o download do programa de instalação via web (Web installer) dotNetFx40\_Full\_setup.exe, execute o programa e siga as instruções para instalação.

O .NET Framework é instalado em uma pasta de caminho %windir%\Microsoft.NET\Framework\4.0 ou similar, dependendo do ambiente.

1. **Linguagens de programação para .NET Framework 4.0**

A Microsoft oferece um conjunto de linguagens de programação para o .NET 4.0. Podemos desenvolver as aplicações .NET em linguagem *Visual Basic 2010 (VB 10), Visual C# 4.0, Visual F# 2010 e Visual C++ 2010*. Estas linguagens permitem desenvolver aplicações fazendo uso das ferramentas visuais disponíveis na interface gráfica do usuário (graphical user interface - GUI) do Visual Studio 2010. Podem ser utilizadas as linguagens dinâmicas como IronRuby IronPython, exceto a linguagem J#, pois não faz parte da família de linguagens para .NET 4.0.

1. **Visual Studio**

É uma IDE (integrated development environment), que possui recursos aprimorados que auxiliam no desenvolvimento da aplicação, oferecendo praticidade e rapidez no desenvolvimento, desde o projeto à implementação.

A MS (Microsoft), oferece diversas edições do VS 2010 com recursos diversificados e distintos, diversos preços e para que o desenvolvedor tenha flexibilidade para escolher a edição que melhor servir.

**VS 2010 Professional:** *O Visual Studio 2010 Professional é indicado para profissionais que executam tarefas básicas de desenvolvimento. Ele simplifica a criação, depuração e implantação de aplicações em várias plataformas que incluem Windows, Web SharePoint e Azure.*

**VS 2010 Premium:** *O Visual Studio 2010 Premium é indicado para profissionais e equipes que necessitem de teste automatizado da interface para usuário, de identificação do impacto de testes ao mudar os códigos, de geração de dados para testes e de simplificação no desenvolvimento de bancos de dados.*

**VS 2010 Test Professional:** *O Visual Studio Test Professional 2010 é um conjunto de ferramentas especializado para equipes de controle de qualidade, que simplifica o planejamento de testes e a execução de testes manuais. Funciona em conjunto com as outras edições do Visual Studio para desenvolvedores, permitindo a colaboração efetiva entre os desenvolvedores e os testadores durante todo o ciclo de desenvolvimento da aplicação.*

**VS 2010 Ultimate:** *O Visual Studio 2010 Ultimate é indicado para profissionais e equipes que necessitem de um conjunto completo de ferramentas de gerenciamento de ciclo de vida da aplicação (Application Life Management - ALM). Permite a elaboração de diagramas em UML e o gerenciamento de projetos com metodologias iterativas e ágeis.*

1. Conclusão

Nesta aula, você:

* Conheceu o mercado de software;
* Aprendeu as características do .NET Framework 4.0;
* Conheceu as linguagens de programação para .NET Framework 4.0;
* Aprendeu a baixar e instalar o .NET Framework 4.0;
* Conheceu as edições do Visual Studio 2010;
* Aprendeu a baixar e instalar o Visual Basic 2010 Express.

1. Bibliografia e pesquisa

Para saber mais sobre os tópicos estudados nesta aula, pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto. Se ainda tiver alguma dúvida, fale com seu professor online utilizando os recursos disponíveis no ambiente de aprendizagem.

Realize as seguintes atividades para saber mais:

Conheça o portal do centro do desenvolvedor para .NET, .NET Framework Developer Center, disponível em <http://msdn.microsoft.com/pt-br/netframework/default.aspx>

Leia sobre a visão geral conceitual do .NET Framework disponível em <http://msdn.microsoft.com/pt-br/library/zw4w595w.aspx>

Leia sobre os estudos de casos do .NET Framework disponível em <http://www.microsoft.com/net/CaseStudies/>

Leia sobre a visão geral do CLR disponível em <http://msdn.microsoft.com/library/ddk909ch.aspx>

Leia sobre a visão geral da BCL disponível em <http://msdn.microsoft.com/library/hfa3fa08.aspx>

Leia sobre o comparativo das edições do Visual Studio 2010 disponível em <http://www.microsoft.com/visualstudio/pt-br/products>

Leia sobre o histórico da linguagem VB disponível em <http://visualbasic.freetutes.com/?gclid=COGuurSx9KICFYMe7godMD-uhQ> e<http://www.johnsmiley.com/visualbasic/vbhistory.htm>

**Desenvolvimento de Software – Aula 2**

* 1. **Visual Basic 2010**
  2. **Palavra-chave**

são palavras reservadas e não devem ser usadas como nomes de identificadores de variáveis, constantes, classes, procedimentos, módulos, sub-rotinas, funções, objetos e etc.

Isso também serve para mantermos a legibilidade do código.

* 1. Comentários

Tem como objetivo informar outros programadores sobre o código. Não são interpretados pelo compilador, apenas para leitura humana.

Prefixamos os comentários no VB com apóstrofes ( ‘ ). O texto fica marcado de verde.

Exemplo:

*‘Isso é um comentário e não um código’*

* 1. **Variável**

Como em outras linguagens, nada mais é que um espaço alocado na memória para armazenamento de valores. Toda variável tem um nome (identificador), um tamanho no espaço que irá alocar na memória e um tipo.

* 1. **Constante**

Tem o mesmo conceito de variável, porém após ser atribuído um valor a ela, esse valor não poderá mais ser alterado.

* 1. **Tipos**

Serve para definir o tipo de valores que uma variável ou Constante irá receber, o espaço a ser alocado deste tipo e a faixa de valores. Assim, fica bem definido o tipo de dados ou a característica dessa variável ou constante.

As linguagens de programação .NET são baseadas no *Common Type System* (**Sistema de Tipo Comum**) do .NET Framework.

* 1. **Common Type System**: fornece tipos unificados para todas as linguagens .NET.

Balena afirma que, os milhares de tipos suportados pelo .NET Framwork podem *ser agrupados em duas amplas categorias*:

**Tipos de valores (value types)   
Tipos de referências (reference types)**

Os tipos de valores definem a variável ou constante. Esses valores são armazenados em um local da memória denominado **Stack** (pilha).

* 1. **Tipos de Valores:** VB são: **booleanos**, **numéricos**, **Char**, **Date**, **Structure** e **Enum**.
  2. **Tipos de Referência**: definem *variáveis* ou *constantes*, *conhecidas como ponteiros*, onde são armazenados os endereços iniciais aos valores (apontam para os valores). Conclui-se que os ponteiros são variáveis que armazenam endereço de memória.

Os ponteiros são armazenados na **Stack**, enquanto os valores referenciados pelo ponteiro são armazenados em uma área chamada **Heap** **gerenciado**. *Esta área é gerenciada pelo CLR*, que determina *quando deve realizar o processo conhecido como coleta de lixo que libera as posições de memórias ocupadas por valores que deixam de ser referenciados pelos ponteiros após a destruição destes*.

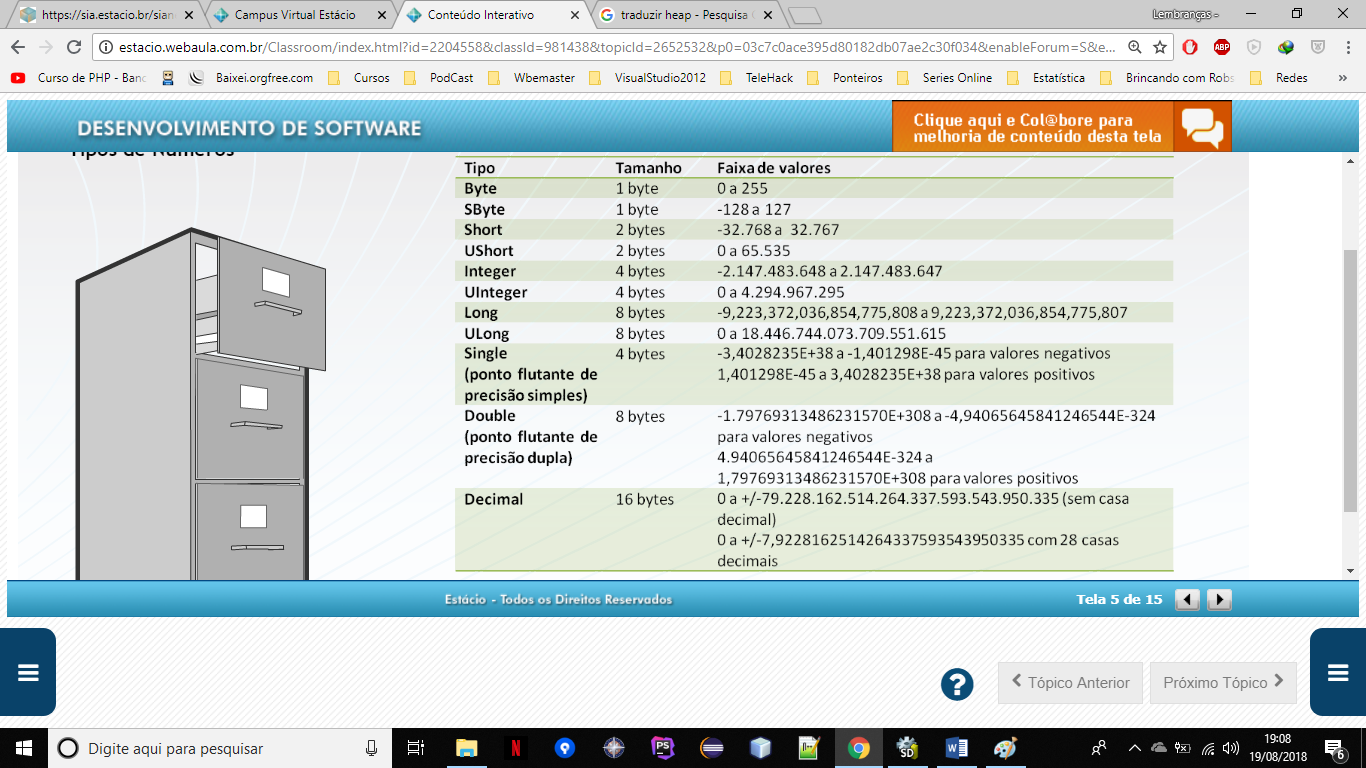
Os **tipos de referência** da **linguagem VB 2010** são: **String**, **array** e **classes**.

1. **Tipo Booleano:**

|  |  |  |
| --- | --- | --- |
| **Tipo** | **Tamanho** | **Faixa de Valores** |
| Boolean | Depende da Plataforma | True ou False |

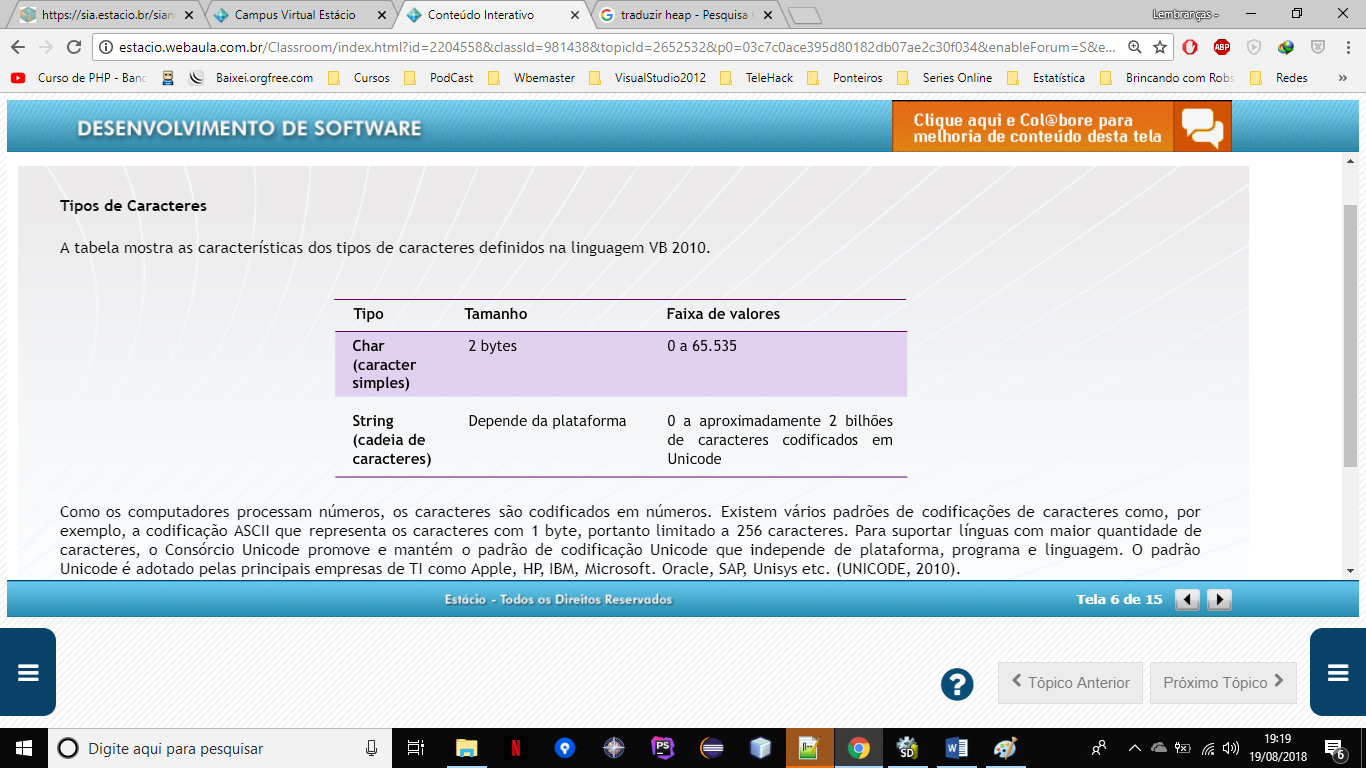
Defina a uma variável que o valore que ela irá receber possa ser True ou False, que representam dois estados lógicos possíveis: Verdadeiro ou Falso, Ligado ou Desliga, Aberto ou Fechado etc. Ao declarar esse tipo a uma variável, o valor padrão, que ela passa a receber é FALSE.

1. **Tipos de Números:**



* 1. **Tipo de números:** inteiros permite *declarar variáveis* ou *constantes* que *ocupam um determinado espaço de memória e armazenam números inteiros de uma determinada faixa de valores*. O programador deve escolher o tipo que ocupe a menor quantidade de memória em bytes e seja capaz de armazenar a faixa de valores possíveis de um determinado dado a ser armazenado em uma variável ou constante. Esta abordagem permite otimizar o programa, tendo em vista que as variáveis ocupam somente os espaços de memória necessários e suficientes e consumem menos tempo e recursos, como pilha e registradores, para processamento. Por exemplo, vamos supor a necessidade de se declarar uma variável para armazenar a quantidade de filhos de uma pessoa. Neste caso, basta declarar uma variável do tipo Byte que ocupa apenas um byte e permite armazenar valores de 0 a 255. Uma variável do tipo Byte seria suficiente para armazenar a idade de uma pessoa em anos, a quantidade de dias em um mês, a quantidade de meses em um ano etc.
  2. **Tipo Single:** define números de *ponto flutuante com precisão até a sexta* casa decimal.
  3. **Double:** define números de ponto flutuante com o dobro de dígitos significativos em *relação ao tipo Single ou com precisão até a décima casa decimal*, que evita os erros de arredondamento que podem acontecer com números do tipo Single. Desta forma, o **tipo Double** é *indicado para números utilizados em aplicações científicas, financeiras ou de engenharia*.
  4. **Decimal:** define números inteiros de 12 bytes ou números com 0 a 28 casas decimais.
  5. **Default:** das variáveis de tipos de números é zero.
  6. **Tipos Caracteres**

A tabela mostra as características dos tipos de caracteres definidos na linguagem VB 2010:



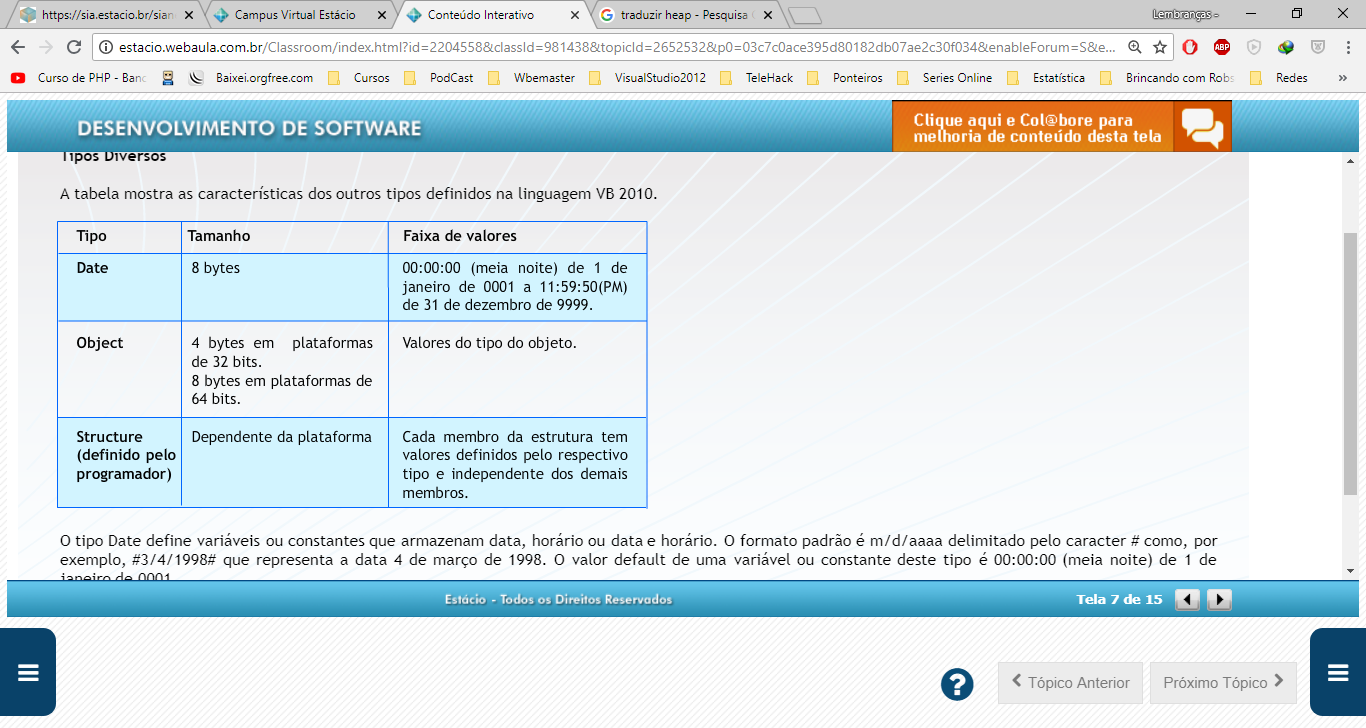
Como os computadores processam números, os caracteres são codificados em números. Existem vários padrões de codificações de caracteres como, por exemplo, a **codificação ASCII** que representa os caracteres com 1 byte, portanto limitado a 256 caracteres. Para suportar línguas com maior quantidade de caracteres, o **Consórcio Unicode** *promove e mantém o padrão de codificação Unicode que independe de plataforma*, *programa e linguagem*. O **padrão Unicode** é adotado pelas principais empresas de TI como Apple, HP, IBM, Microsoft. Oracle, SAP, Unisys etc. (UNICODE, 2010).

**Char:** define variáveis ou constantes que armazenam o código de 2 bytes de um caracter na *codificação Unicode*. *O valor default de uma variável do tipo Char é zero*.

**String:** define um ponteiro para uma cadeia de caracteres codificados em Unicode. A cadeia de caracteres pode ter até aproximadamente 231 caracteres. O valor default do ponteiro deste tipo é **Nothing**, isto é, sem endereço.

* 1. **Tipos diversos**

A tabela mostra as características dos outros tipos definidos na linguagem VB 2010:



**Date:** define variáveis ou constantes que armazenam data, horário ou data e horário. O formato padrão é m/d/aaaa delimitado pelo caracter # como, por exemplo, #3/4/1998# que representa a data 4 de março de 1998. *O valor default de uma variável ou constante deste tipo é 00:00:00 (meia noite) de 1 de janeiro de 0001*.

**Object:** *define um ponteiro que armazena o endereço de referência a qualquer valor ou objeto*. O *valor default do ponteiro deste tipo é Nothing*, isto é, *sem endereço*.

**Structure:** permite que o programador defina estruturas com membros de diferentes tipos.

1. **Declaração de Variável**

Para declararmos uma variável, usamos a palavra reservada **Dim.**

Sintaxe:

**Dim** nome **As** **Tipo**

Identificadores:

1. Não devem iniciar com (“\_”) – conhecido com *undescore*.
2. Não devem conter espaços
3. Não devem conter Símbolos
4. É usual usar acentos nos identificadores

Exemplos corretos:

**Dim** Idade **As Byte**  
**Dim** Endereco **As String**

Podemos declarar duas ou mais variáveis na mesma sentença:

**Dim** Dia, Mês **As Byte**

Podemos usar na mesma sentença variáveis e tipos diferentes:

**Dim** Numero **As UShort**, Complemento **As String**

1. **Atribuir Valor**
   1. A Atribuição de Valor é feita através de um sinal de (“=”)

*Nome = valor*

**Exemplos**:

Atribuindo Valor:  
**Idade = 5**

Atribuindo literal:  
**Endereco = “Av. Darcy Vargas”**

1. **Declaração de Variável e atribuição de Valor**

Podemos atribuir o valor na mesma sentença da declaração da variável:

Dim Nome As String = valor

Exemplo:

*Dim Dependentes As Byte = 1*

1. **Declaração de Constantes e Atribuição de valor**

A declaração de uma constante é feita através da palavra-chave reservada **Const** e a atribuição de um valor é feita com **sinal de igual**, conforme a seguinte sintaxe:

**Const** Nome **As Tipo =** valor

Obs.: A linguagem VB 2010 tem constantes predefinidas que podem ser conhecidas acessando:

<http://msdn.microsoft.com/en-us/library/dy7yth1w.aspx>

Uma constante deve ter o seu valor atribuído na sentença de declaração.

O exemplo a seguir declara uma constante de nome Versão do tipo Decimal com valor igual a 1.0 e outra constante de nome PI do tipo Double com valor igual a 3.1415.

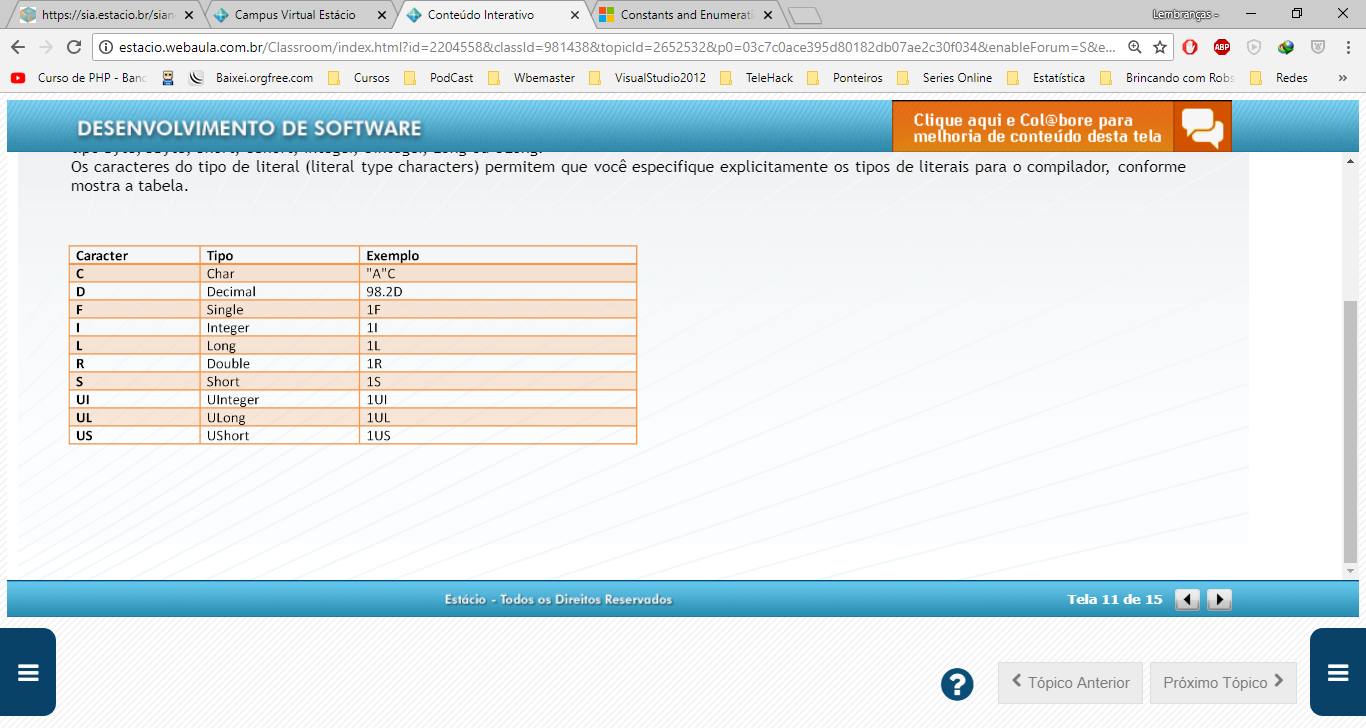
Exemplos:

Const Versao As Decimal = 1.0  
Const PI As Double = 3.1415

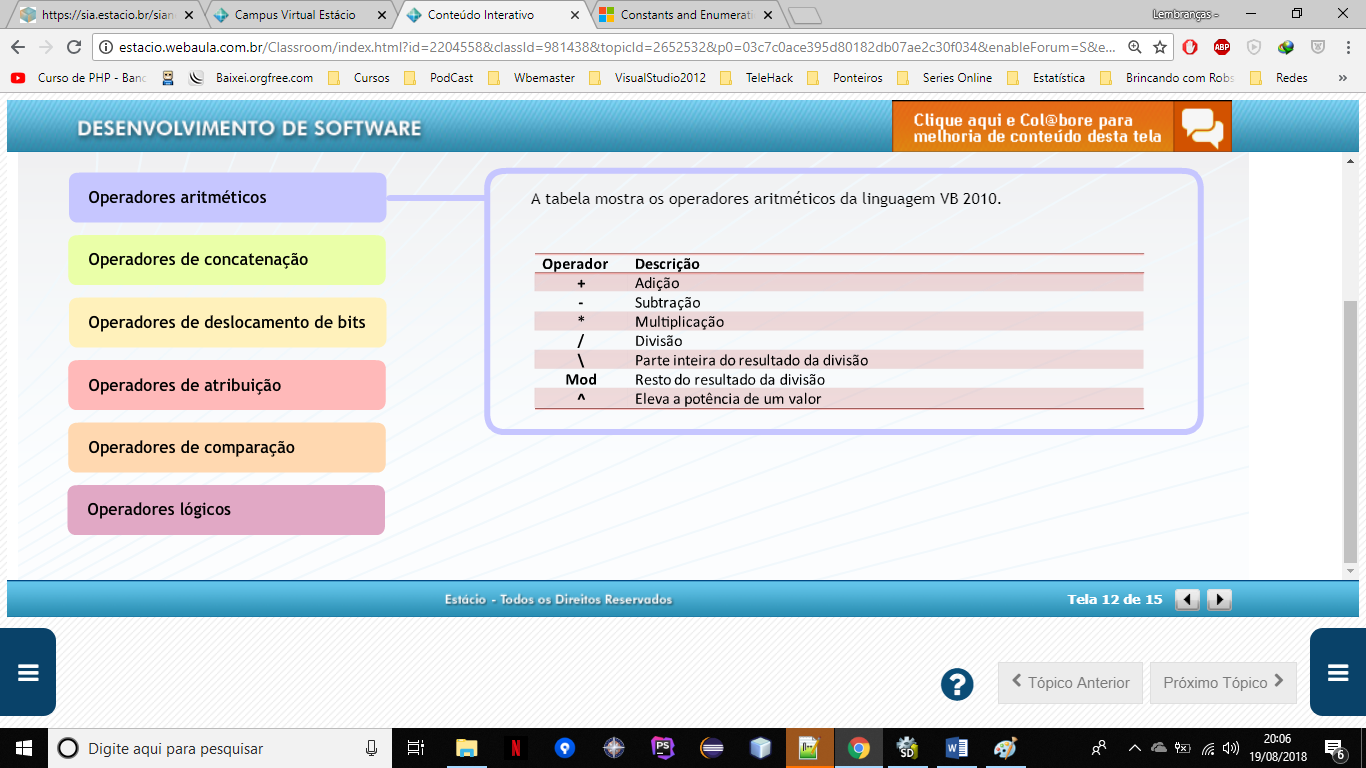
1. **Caracteres do tipo Literal**

Alguns valores literais podem ser compatíveis com diferentes tipos como, por exemplo, "**A**" pode ser do tipo **Char ou String** e o valor literal **1** pode ser do tipo *Byte, SByte, Short, UShort, Integer, UInteger, Long ou ULong*.

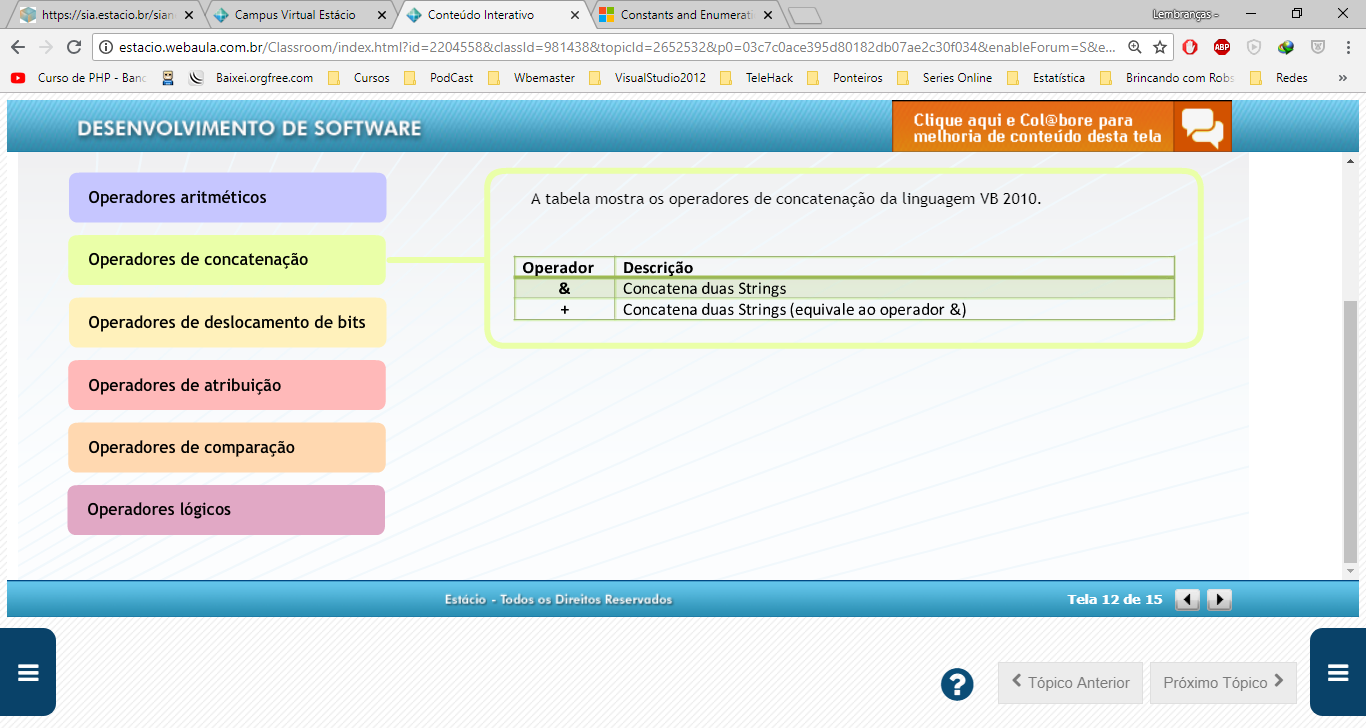
Os caracteres do tipo de literal (**literal type characters**) permitem que você especifique explicitamente os tipos de literais para o compilador, conforme mostra a tabela.



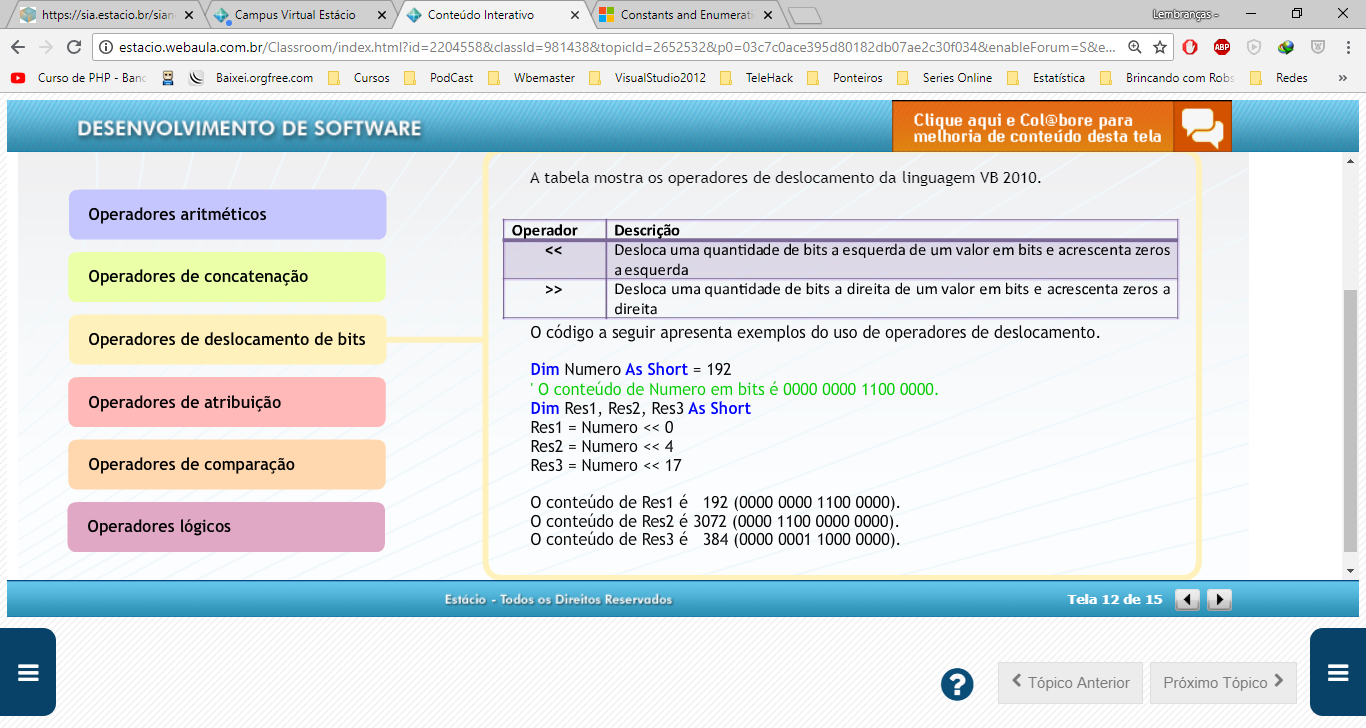
1. **Operadores Matemáticos**:



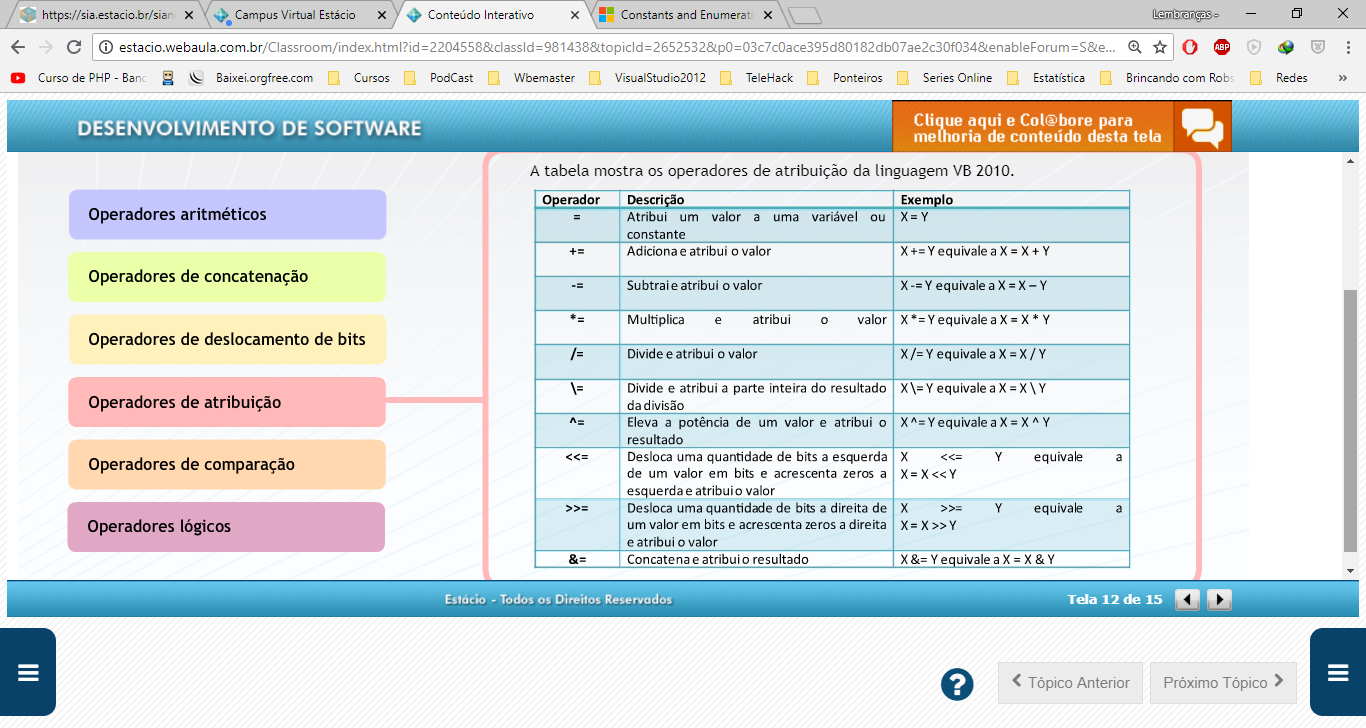
1. **Operadores de Concatenação:**



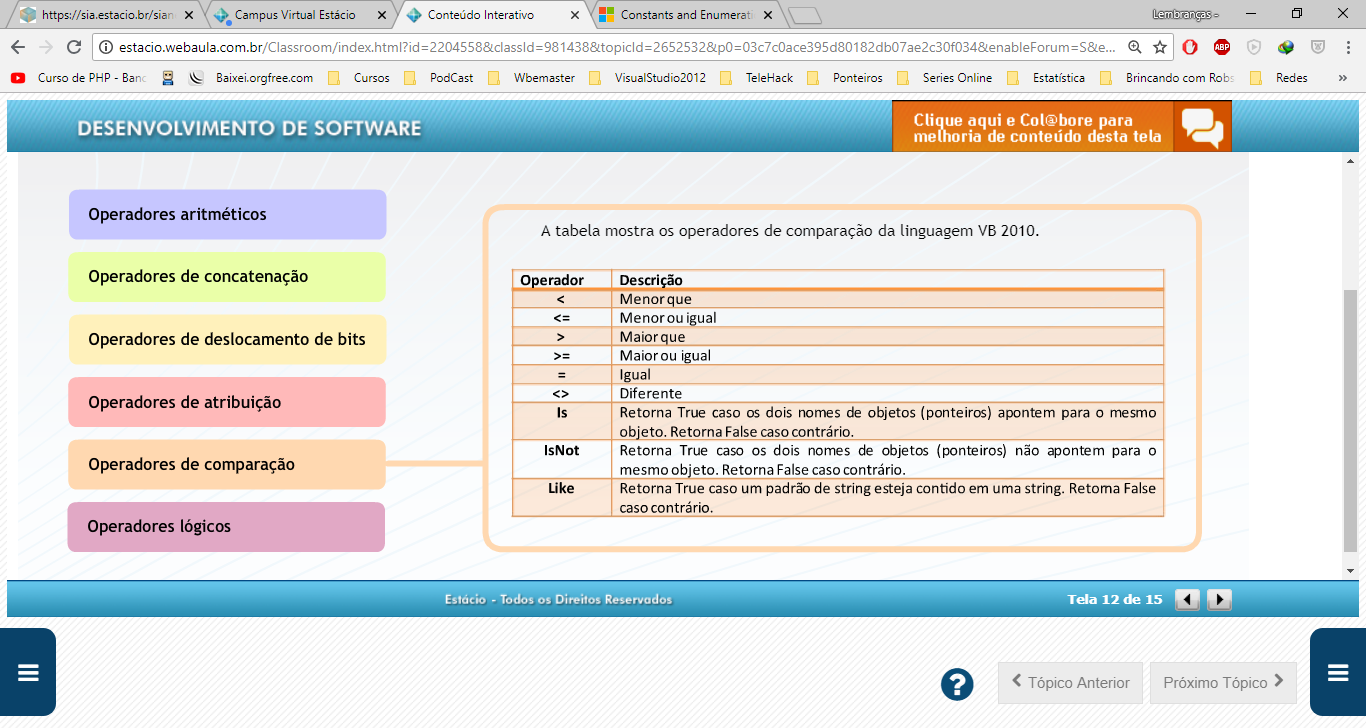
1. **Operadores de Deslocamento de Bits**



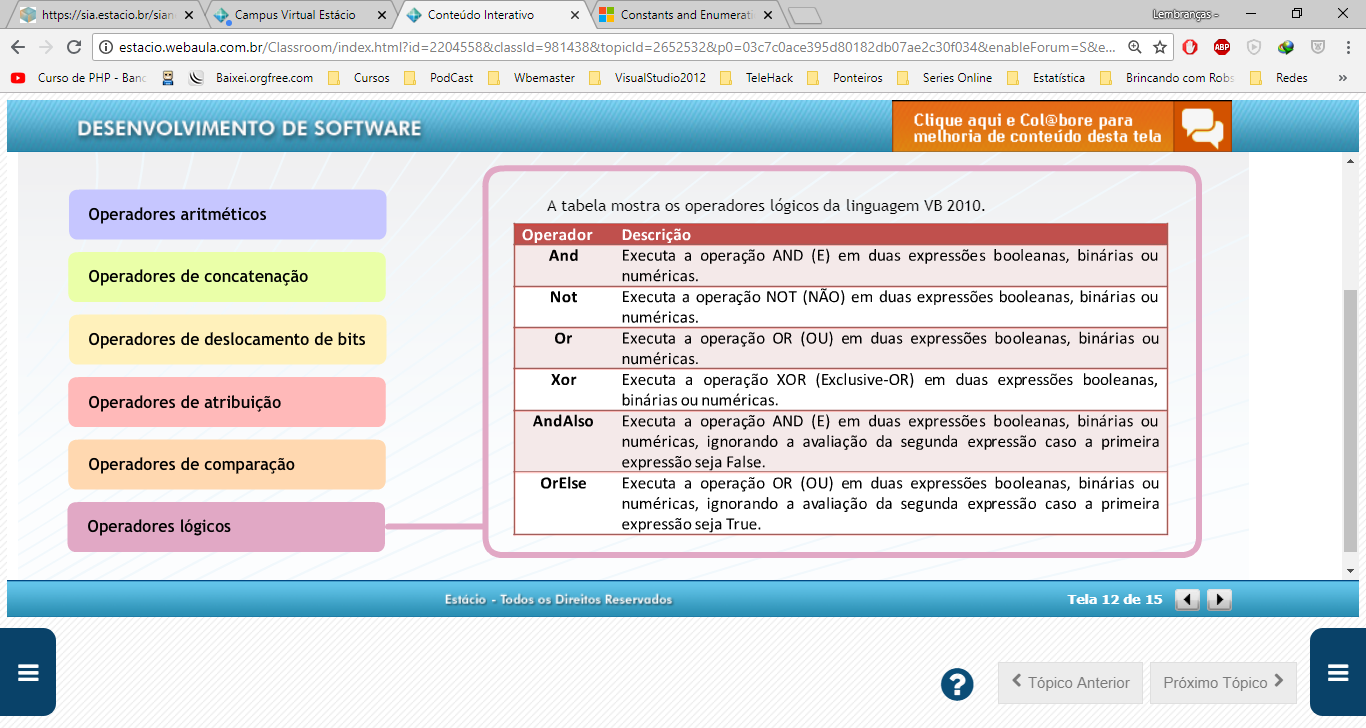
1. **Operadores de Atribuição**



1. **Operadores de comparação**



1. **Operadores Lógicos**



1. **Referências**

BALENA, Francesco. Programando com Microsoft Visual Basic 2005: A linguagem, ed. Bookman, 2006.  
DEL SOLE, Alessandro. Visual Basic 2010 Unleashed. ed. Pearson, 2010.   
  
MSDN. Keywords (Visual Basic). Disponível em:   
http://msdn.microsoft.com/en-us/library/dd409611.aspx > Acesso em: 17 de julho de 2010.

FREETUTES. The Tutorial Website. Disponível em:   
http://visualbasic.freetutes.com/?gclid=COGuurSx9KICFYMe7godMD-uhQ> Acesso em: 17 de julho de 2010.

KUNK, Joe. What's New in Visual Basic 2010. Visual Studio Magazine: Expert Solutions for .NET Development. Disponível em:   
http://visualstudiomagazine.com/articles/2010/06/01/whats-new-in-visual-basic-2010.aspx> Acesso em: 17 de julho de 2010.

SMILEY, John. Interested in learning more about Visual Basic? Disponível em:   
http://www.johnsmiley.com/visualbasic/vbhistory.htm> Acesso em: 17 de julho de 2010.

UNICODE. Unicode Consortium. Disponível em:   
http://www.unicode.org/standard/translations/portuguese.html > Acesso em: 17 de julho de 2010.

VISUAL STUDIO. Microsoft Visual Studio. Disponível em:   
http://msdn.microsoft.com/en-us/vstudio/default.aspx> Acesso em: 17 de julho de 2010.

Para saber mais sobre os tópicos estudados nesta aula, pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto. Se ainda tiver alguma dúvida, fale com seu professor online utilizando os recursos disponíveis no ambiente de aprendizagem.

Realize as seguintes atividades para saber mais.

* Assista ao vídeo de introdução à linguagem Visual Basic disponível em <http://msdn.microsoft.com/pt-br/beginner/bb308774.aspx>
* Leia sobre o histórico da linguagem VB disponível em <http://www.johnsmiley.com/visualbasic/vbhistory.htm>
* Conheça o portal da referência da linguagem VB 2010 disponível em <http://msdn.microsoft.com/en-us/library/sh9ywfdk.aspx>
* Conheça as palavras-chave da linguagem VB 2010 disponível em <http://msdn.microsoft.com/en-us/library/dd409611.aspx>
* Leia sobre os tipos da linguagem VB 2010 disponível em <http://msdn.microsoft.com/en-us/library/47zceaw7.aspx>
* Conheça as constantes predefinidas da linguagem VB 2010 disponível em <http://msdn.microsoft.com/en-us/library/dy7yth1w.aspx>
* Leia sobre os operadores da linguagem VB 2010 disponível em <http://msdn.microsoft.com/en-us/library/f15ea9k8.aspx>

**Desenvolvimento de Software – Aula 3**

1. **Módulo**

***Módulo*** é *um container onde são definidos declarações de variáveis, constantes e métodos*.

Os ***módulos*** são delimitados pelas palavras chaves e um por um nome que o identifica:

***Module Nome***

***End Module***

Ao criarmos um **projeto** no VB10 Express, um *módulo padrão* é criado automaticamente junto com uma **sub-rotina** chamada ***Main(),*** que é o método principal do programa.

Exemplo:

**Module** *Programa*

**Sub** *Main*()

**End** *Sub*

**End** *Module*

1. **Módulos Predefinidos**

A linguagem de programação VB 2010 possui diversos ***Módulos Prefefinidos*** com a finalidade de serem usadas em nossos códigos. Cada uma possui diversos membros podendo ser utilizados.

Vejamos alguns desses módulos predefinidos:

* + **Constantes**: Contém diversos **membros** *constantes*
  + **ControlChars**: Contém **membros** *constantes de controle de caracteres para impressão e apresentação de textos*.
  + **Convertion**: Contém **membros** que *convertem números decimais em outras bases*, números para strings, strings para números e um tipo de dado para outro.
  + **DateAndTime**: Contém **membros** que *permitem obter a data e a hora corrente*, realizar cálculos com datas e horários e atribuir a data ou o horário.
  + **ErrObject**: Contém **membros** para *obter informações sobre erros gerados em tempo de execução*, para *gerar erro* ou para *inicializar (limpar) as propriedades do* ***object Err***, que representa o erro. (obtém erros)
  + **FileSystem**: Contém ***métodos*** para *realizar operações sobre arquivos, diretórios ou pastas*.
  + **Financial**: Contém ***métodos*** para *realizar cálculos financeiros*.
  + **Globals**: Contém **membros** para *obter informações sobre a versão do interpretador de script*.
  + **Information**: Contém ***métodos*** que retornam o *resultado dos testes de verificação de tipo*, ou *retornam as informações sobre erro em tempo de execução*, *o nome do tipo*, o *valor da cor em RGB*, etc. (mostra erros)
  + **Interection**: Contém ***métodos*** usados para *interagir com objetos*, *aplicações e sistema*.
  + **Strings**: Contém ***métodos*** para *tratamento de manipulação de strings* como, por exemplo, para *formatar, buscar um caracter ou sub-string na string, obter o código ASC do caracter, obter comprimento da string* etc.
  + **VBMath**: Contém ***métodos*** para realizar *operações matemáticas*.

1. **Métodos**
   1. ***Método*** ou ***Procedimento***.
   2. Também é um bloco identificado por um nome.
   3. Chamamos pelo nome do método para que o código do seu bloco seja executado.
   4. Um Módulo pode ter um ou mais métodos.

Vantagens do uso dos métodos:

* 1. Eliminação de repetição em um mesmo bloco de código, bastando chamar o método pelo seu nome para que o código do seu bloco seja executado quantas vezes o chamarmos. – Evitando Redundâncias.
  2. Podemos utilizar os métodos em diferentes aplicações. Para que o reaproveitamento dos Métodos é preciso que o código desse bloco, realize finalidades específicas. Como por exemplo, calcular a idade de uma pessoa.
  3. Possibilita a modularização, tornando-o legível e menos suscetível a erros de programação.
  4. Mais de um programador pode desenvolver em conjunto, ou seja, enquanto um desenvolve o método específico, o outro desenvolve outras partes da programação que chama esse método. Para isso é preciso que a assinatura do método seja especificada e informada à equipe, para evitar erros de assinatura na chamada do método.

**Obs.: UM *MÉTODO* PODE SER UMA *SUB-ROTINA* OU UMA *FUNÇÃO***

4. **Sub-rotina**

* Uma sub-rotina **é um método** que executa um bloco de código, ***mas não retorna um valor***.
* As sub-rotinas são delimitadas pelas palavras-chaves **Sub** e **End Sub**
* Uma sub-rotina **tem um nome** para que a ***identifica para que seja chamada***.

Sintaxe:

Sub Nome(ByVal NomeParametro1 As Tipo..., ByVal NomeParametro2 As Tipo)

End Sub

Exemplo1:

*Module Module1  
 Sub main()  
 Console.WriteLine(“Olá, “ + “Fernando“ + “, conheça a TI Verde.”)  
 Console.WriteLine(“Olá, ”+”Ana”+”, conheça a TI Verde.”)  
 Console.WriteLine(“Olá, ”+”Cláudia”+”, conheça a TI Verde.“)  
 ‘Aguarde a digitação de uma tecla antes de terminar  
 Console.ReadKey()  
 End Sub  
End Module*

No exemplo 2, veremos como um método (sub-rotina) permite realizar a mesma tarefa anterior de forma coesa.

*Module Module1*

*Sub ExibirMensagem(ByVal Nome As String)  
 Console.WriteLine(“Olá “+Nome+”, conheça a TI Verde”)  
End Sub*

*Sub Main()   
 ExibirMensagem(“Fernando”)  
 ExibirMensagem(“Cláudia”)  
 ExibirMensagem(“Ana”)*

*‘Aguarde a digitação de uma tecla antes de terminar  
 Console.ReadKey()  
End Sub*

*End Module*

Obs.: Usamos a palavra-chave **ByVal** para definir parâmetros em um método.

Podemos ver que nesse último exemplo, não encontramos redundância de código comum para acréscimo do prefixo ou do sufixo. E se quisermos mudar o prefixo “Olá” para “OI”, basta mudar apenas no método.

5. **Função**

* Uma função é um método que executa um bloco de código mas que retorna um valor.
* As funções são delimitadas pelas palavras-chave **Function e End Function**
* Uma função, também tem um nome que a identifica

Sintaxe:

Function Nome(ByVal NomeParametro1 As Tipo... ByVal NomeParametroN As Tipo)  
 ‘código da função  
 Return expressão  
End Function

Exemplo 1:

Vamos definir uma função chamada **AreaQuadrado** que recebe como ***Argumento*** o *comprimento* e **retorna** ***o valor*** da *área do quadrado*. A sub-rotina Main() chama a ***função AreaQuadrado*** passando o **valor 3.0** para o parâmetro definido no método, e depois novamente com um novo **valor 4.5**.

Module Module1  
 Function AreaQuadrado(ByVal Comprimento As Double)  
 ‘calcula a área do quadrado  
 Return Comprimento \* Comprimento  
 End Function

Sub Main()  
 Console.WriteLine(AreaQuadrado(3.0))  
 Console.WriteLine(AreaQuadradi(4.5))  
 Console.ReadKey()  
 End Sub  
End Module

6. **Encadeamento de Métodos**

Um método pode chamar outro método, permitindo que métodos mais especializados sejam criados para aumentar a coesão. Denominamos isso como: Encadeamento de Métodos.

Exemplo:

Vamos criar um método chamado de VolumeCubo (função) que recebe como argumento o comprimento, chama a função AreaQuadrado para obter a área e retorna o valor do volume do cubo. A Sub-Rotina Main(), chama a função VolumeCubo passando o valor 3.0.

**Module** Module1  
 **Function** AreaQuadrado(ByVal Comprimento As Double)   
 **Retorn** Comprimento \* Comprimento  
 **End Function**  
   
 **Function** VolumeCubo(ByVal Comprimento As Double)  
 **Return** AreaQuadrado(Comprimento) \* Comprimento  
 **End Function  
  
 Sub Main()**  
 Console.WriteLine(VolumeCubo(3.0))  
 Console.ReadKey() **End Sub**  
**End Module**

7. **Escopo**

É o que define acesso a uma variável ou método.

* Escopo de uma variável:

Geralmente, definimos uma variável com a palavra chave DIM, e dessa forma, a variável só poderá ser acessada dentro desse método onde ela foi criada.

Para que possamos acessar uma variável ou método fora do escopo, ou seja compartilhar com outros métodos, declaramos as variáveis com as palavras-chave **Private** ou **Public**, fora do método.

As variáveis **Public** são *acessadas por todos os métodos da aplicação.*

As variáveis **Private** são *acessadas apenas pelos métodos situados dentro do mesmo* ***módulo*** *ou da mesma* ***classe***onde ***foram declaradas****.*

Exemplo:

Definiremos uma variável **Private** chamada **Numero** do **tipo Double**, **a** **sub-rotina Main()** ***atribui o valor 1 a esta variavel***, depois ela é ***acessada pela* sub-rotina Duplicar()** para **duplicar o valor dela**, depois ela é ***acessada pela pela sub-rotina* Triplicar()** para **triplicar o valor dela** e no final ela é ***acessada pela Main() para exibir o valor dela***.

***Module*** *Module1* ***Private*** *Numero* ***As*** *Double* ***Sub*** *Duplicar()  
 Numero = Numero \* 2* ***End Sub******Sub*** *Triplicar()  
 Numer = Numero \* 3* ***End Sub******Sub Main()*** *Numero = 1.0  
 Dublicar()  
 Triplicar()* ***Console.WriteLine****(Numero)* ***Console.ReadKey()*** ***End Sub******End Module***

8. **Funções Internas**

A linguagem VB2010 tem várias funções internas predefinidas que podemos usar em nossas aplicações.

Classificação das Funções Internas:

* + Funções de Conversões de Tipo:
  + Funções de Manipulação de String
  + Funções Matemáticas

Para saber mais sobre os tópicos estudados nesta aula, pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto. Se ainda tiver alguma dúvida, fale com seu professor online utilizando os recursos disponíveis no ambiente de aprendizagem.

Realize as seguintes atividades para saber mais.

Conheça os módulos predefinidos da linguagem VB 2010 disponível em   
<http://msdn.microsoft.com/pt-br/library/y76404kz.aspx> e clique no nome de cada módulo para saber mais detalhes de cada um deles.

Conheça as funções internas da linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/32s6akha.aspx>.

Estude os exemplos das funções de conversão de tipo da linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/s2dy91zy.aspx>.

Estude o exemplo da função CType de conversão de tipo da linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/4x2877xb.aspx>.

Estude os exemplos das funções de manipulação de string da linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/dd789093.aspx>.

Estude os exemplos das funções matemáticas da linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/thc0a116.aspx>.

**Desenvolvimento de Software – Aula 4**

1. Programação Estruturada

A linguagem VB possui estrutura de controle de fluxo que nos permite desenvolver aplicações estruturadas, ou seja, programas que fazer o uso do GoTo, facilitando a manutenção e leitura do código.

1. Estrutura de Controle de Fluxo

O VB 2010 suporta as seguintes estruturas de fluxo:

* Estrutura de controle de decisão simples
* Estrutura de controle de decisão estendida
* Estrutura de controle de decisão aninhada
* Estrutura de controle de decisão em sequência na mesma linha
* Estrutura de controle de múltiplas decisões (escolha)
* Estrutura de controle de repetição controlada por **contador** (**para**)
* Estrutura de controle de repetição **While** (enquanto)
* Estrutura de controle de repetição **Do** (faça)

1. Estrutura de Controle de Decisão Simples

Estrutura: (se – então - fim\_se)

Executa um bloco de código baseado em uma condição, condição lógica ou expressão booleana.

**Sintaxe**:

***If****(condição)* ***Then*** *Bloco de código****End If***

O bloco de código só será executado se a condição for verdadeira (true).

**Exemplo**:

O exemplo a seguir, exibe a pergunta "É sustentável (s/n)? ", obtém a entrada de dados que é atribuída na variável Sustentável. Se o valor da variável Sustentável for igual a "s", exibe "Parabéns!".

***Module*** *Module1* ***Sub******Main****()* ***Dim*** *Sustentavel* ***As******String******Console.Write****(“É sustentável s/n?”)  
 Sustentavel =* ***Console.ReadLine()******If****(Sustentável == “S”)* ***Then******Console.WriteLine****(“Parabéns!“)* ***End If******Console.ReadKey()******End Sub******End Module***

2. Estrutura de Controle de Decisão Estendida

Sintaxe:

***If*** *(condição)* ***Then*** *Bloco de código 1****Else*** *Bloco de código 2****End If***

O bloco de código 1 será executada se a condição 1 for verdadeira (true) e o segundo bloco só será executado se a condição 1 for falsa (false).

Exemplo

O exemplo a seguir, exibe "Digite a sua média final: ", obtém a  entrada de dados que é atribuída na variável MediaFinal. Se o valor da variável MediaFinal for maior ou  igual a 6.0, exibe "Aprovado", senão exibe "Reprovado".

***Module*** *Module 1* ***Sub Main()******Dim*** *MediaFinal* ***As******Single******Console.Write****(“Digite a sua média final: “)  
 MediaFinal =* ***Console.ReadLine()******if****(MediaFinal >= 6.0)* ***Then******Console.ReadLine****(“Aprovado”)* ***Else******Console.ReadLine****(“Reprovado”)* ***End If******Console.ReadKey()******End Sub******End Module***

1. Estrutura de Controle de Decisão Aninhada

Sintaxe:

***If*** *(condição 1)* ***Then*** *Bloco de Código 1****Else******If****(condição 2)* ***Then*** *Bloco de Código 2* ***End If******End If***

No entanto, podemos condensar essa estrutura, para uma melhor legibilidade de código:

Vejamos:

***If****(condição 1)* ***Then*** *Bloco de código 1****ElseIf*** *Bloco de código 2****End If***

O VB 2010 permite diversas estruturas aninhadas com a sintaxe Condensada, que equivale a **Estrutura de Múltiplas Decisões** (Escolha)

Vejamos:

***If****(condição 1)* ***Then*** *Bloco de código 1****ElseIf****(condição 2)* ***Then*** *Bloco de código 2****ElseIf****(condição 3)* ***Then*** *Bloco de código 3  
...****End If***

O exemplo a seguir, exibe "Digite um número de 0 a 99: ", obtém a entrada de dados que é atribuída na variável Numero. Se o valor da variável Numero estiver entre 0 a 9, exibe " O número está no intervalo de 0 a 9 inclusive.", senão testa se o valor da variável Numero está entre 10 e 99 e, se estiver, exibe " O número está no intervalo de 10 a 99 inclusive.", senão exibe "O número não está no intervalo de 0 a 99 inclusive.".

***Module*** *Module1* ***Sub Main()******Dim*** *Numero* ***As*** *Byte* ***Console.Write****(“Digite um número de 0 à 99: “)  
 Numero =* ***Console.ReadLine()******if****(Numero >= 0* ***And*** *Numero <= 9)* ***Then******Console.Write****(“O número está no intervalo entre 0 a 9”)* ***ElseIf****(Numero >= 10* ***And*** *Numero <= 99)* ***Console.Write****(“O número está no intervalo entre 10 e 99, inclusive”)* ***Else******Console.Write****(“O número não está no intervalo de 0 a 99”)* ***End If******Console.ReadKey()******End Sub******End Module***

1. Controle de Decisão em uma única linha

Podemos expressar uma expressão de controle de decisão em uma única linha no VB 2010

Sintaxe:

**If**(condição 1) **Then** instrução 1 **Else** instrução 2

O ramo verdadeiro (Then) ou Falsol (Else) pode ser composto por mais de uma instrução, separadas por “ : ” – dois pontos.

Sintaxe:

**If**(condição 1) **Then** instrução 1 **:** Instrução 2 **Else** Instrução 3 **:** Instrução 4

1. Estrutura de Controle de Múltiplas Decisões

No VB podemos usar uma estrutura denominada **Select Case**, que permite definir uma estrutura de múltiplas decisões (escolha), mais otimizado que o uso de estrutura de decisão aninhadas.

Sintaxe:

***Select Case****(condição)* ***Case*** *valor ou intervalo de valores ou lista de valores 1  
 Bloco de código 1* ***Case*** *valor ou intervalo de valores ou lista de valores 2  
 Bloco de código 2  
 ...* ***Case Else*** *Bloco de código N****End Select***

Onde o intervalo de valores é definido usando-se a palavra **To** na forma valor inicial **To** valor final e a lista de valores é definida com os valores separados por vírgula na **forma valor1, valor2,.., valor n**.

O exemplo apresentado a seguir, que usa estrutura de decisão aninhadas, modificado para fazer uso da instrução **Select Case**.

***Module*** *Module1* ***Sub Main()******Dim*** *Numero* ***As Byte******Console.Write****(“Digite um numero de 0 à 99: “)  
 Numero =* ***Console.WriteLine******Select Case*** *(Numero)* ***Case*** *0* ***To*** *9* ***Console.Write****(“O número está no intervalo de 0 a 9”)* ***Case*** *10* ***To*** *99* ***Console.Write****(“O número está no intervalo de 10 a 99”)* ***Case Else******Console.Write****(“O número não está no intervalo de 0 a 100”)* ***End Select******Console.ReadKey()******End Sub******End Module***

1. Estrutura de Controle de Repetição Controlada por Contador (para)

A estrutura de controle de repetição ***While (enquanto)*** ***executa um bloco de código enquanto a condição for verdadeira (True)***. Para essa estrutura, será preciso definir uma instrução antes para uma variável e uma instrução de alteração dentro do laço, para que o loop (laço) não entre em uma repetição perpétua.

Sintaxe:

*Instrução de inicialização da variável da condição****While****(condição)  
 Bloco de código   
 Com instrução de alteração da variável da condição****End While***

O exemplo a seguir, é o exemplo ConsoleApplication4-6.sln modificado para fazer uso da estrutura de controle de repetição **While** ao invés do For.

***Module*** *Module1* ***Sub Main()******Dim*** *Contador* ***As******Byte*** *= 1 ‘instrução de inicialização* ***While****(Contador* ***<=*** *10)* ***Console.WriteLine****(Contador)  
 ‘Instrução de alteração da variável da condição  
 Contador = Contador* ***+ 1******End While******Console.ReadKey()******End Sub******End Module***

1. Estrutura de Repetição Do (Faça)

Na linguagem VB podemos encontrar duas sintaxes para definir essa estrutura. As duas sintaxes do mesmo tratamento de inicialização e alteração da variável, assim como na sintaxe do While.

Sintaxe 1:

*Instrução de inicialização da variável da condição****Do While*** *ou* ***Until****(condição)  
 Bloco de código com instrução de alteração da variável da condição****Loop***

Sintaxe 2:

*Instrução de inicialização da variável da condição****Do****(condição)  
 Bloco de código com instrução de alteração da variável da condição****Loop While ou Until (****condição****)***

Quando o **Loop** ou **Until** ***While é utilizado***, o laço será ***repetido enquanto a condição for verdadeira*** ***(True)***. Quando o ***Until*** é utilizado, o ***laço será repetido até que a condição seja falsa*** (False). A sintaxe da opção 1 define um laço (loop) que poderá não ser executado ou repetido várias vezes (0 a n vezes). Enquanto a sintaxe da opção 2 define um laço (loop) que será repetido uma (pelo menos) ou várias vezes (1 a n vezes).

O exemplo a seguir, é o exemplo ConsoleApplication4-6.sln modificado para fazer uso da estrutura de controle de repetição Do While ao invés do For, e equivalente ao exemplo anterior.

***Module*** *Module1* ***Sub Main()******Dim*** *Contador* ***As Byte*** *= 1 ' instrução de incialização* ***Do While*** *(Contador* ***<=*** *10)* ***Console.WriteLine****(Contador) ' instrução de alteração da variável da condição   
 Contador = Contador* ***+ 1******Loop******Console.ReadKey()******End Sub******End Module***

O exemplo a seguir, modificado para fazer uso da estrutura de controle de repetição **Do Until** ao invés do **For**, e equivalente ao exemplo anterior.

***Module*** *Module1* ***Sub Main()******Dim*** *Contador* ***As Byte*** *= 1 ' instrução de incialização* ***Do Until Not*** *(Contador* ***<=*** *10)* ***Console.WriteLine****(Contador)   
 ' instrução de alteração da variável da condição   
 Contador = Contador* ***+ 1******Loop******Console.ReadKey()******End Sub   
End Module***

**Observação**

Como vimos no controle de seleção Do While, o loop funciona até que o Teste Lógico seja Falso. Por exemplo:

**N = 5  
Do While(N<10)**

Os comandos aqui irão ser executados até que N seja Igual ou Maior que 10 (quando o teste lógico for Falso).

Para forçar o teste lógico a mudar o seu valor de True Pra False, basta acrescentar a notação de negação no teste lógico:  
**N= 5  
Do While Not(N<10)** = Nesse caso, o retorno dessa estrutura seria False. O Código já pararia como está.

O mesmo pode ser aplicado no uso do **Do Until** (quando a expressão deve ser VERDADEIRA), então a forçamos a ser **FALSE** com o uso do **NOT**

Para saber mais sobre os tópicos estudados nesta aula, pesquise na internet sites, vídeos e artigos relacionados ao conteúdo visto. Se ainda tiver alguma dúvida, fale com seu professor online utilizando os recursos disponíveis no ambiente de aprendizagem.

Realize as seguintes atividades para saber mais.

Estude a estrutura de controle de decisão simples, a estrutura de controle de decisão estendida, a estrutura de controle de decisão aninhada, a estrutura de controle de decisão em seqüência na mesma linha da linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/752y8abs.aspx>

Estude a estrutura de controle de múltiplas decisões (escolha) da linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/cy37t14y.aspx>

Estude a estrutura de controle de repetição controlada por contador (para) da linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/5z06z1kb.aspx>

Estude a estrutura de controle de repetição While (enquanto) da linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/zh1f56zs.aspx>

Estude a estrutura de controle de repetição Do (faça) da linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/eked04a7.aspx>

**Desenvolvimento de Software – Aula 5**

1. **Tratamento de Exceções**

Exceção de Erros, é uma estrutura de fluxo de código em uma aplicação. É comparada a um If e Else (diga-se de passagem). O .NET Framework, possui uma estrutura única que identifica dois tipos de Exceções:  
**Hardware**: por exemplo, é quando uma aplicação possui um erro de violação de acesso a memória.

**Software**: quando uma aplicação possui um erro que se refere a software, por exemplo, quando tentamos atribuir um valor incompatível em uma variável.

1. **O Tratamento de Exceções Estruturadas (Structured Exception Handling - SEH)**

É uma forma para tratar tanto as exceções de hardware quanto de software. Dessa forma o código se torna mais unificado para tratar de ambos os erros, tratando ambas as exceções, sessões de códigos, acesso a banco de dados, operações com arquivos e etc.

1. **Tipos de Exceções**

Os tipos de Exceções, derivam da classe **System.Exception** que é definida no **.NET Framework**.

Os principais tipos de Exceções são:

* + ArgumentException
  + ArgumentNullException
  + IndexOutOfRangeException
  + NotImplementedException
  + OutOfMemoryException
  + OverflowException
  + FileNotFoundException

1. **Try** (tente) ... **End Try**:

A estrutura delimita-se o bloco de texto protegido para tratar as exceções ocorridas, onde o início e o fim do bloco, são definimos pela palavra-chave **Try ... End Try**.

A sintaxe:

***Try  
 ...  
Catch*** *ex* ***As Exception  
 ...  
End Try***

1. A Sessão **CATCH (pegar)**

Essa palavra-chave é sessão do bloco que captura e trata as exceções, que podem ocorrer. Normalmente, usa-se essa seção, para capturar cada tipo de exceção e uma seção Catch, denominada genérica, por fim, captura todos os demais tipos de exceções que não foram capturadas pelas seções Catch anteriores.

Sintaxe:

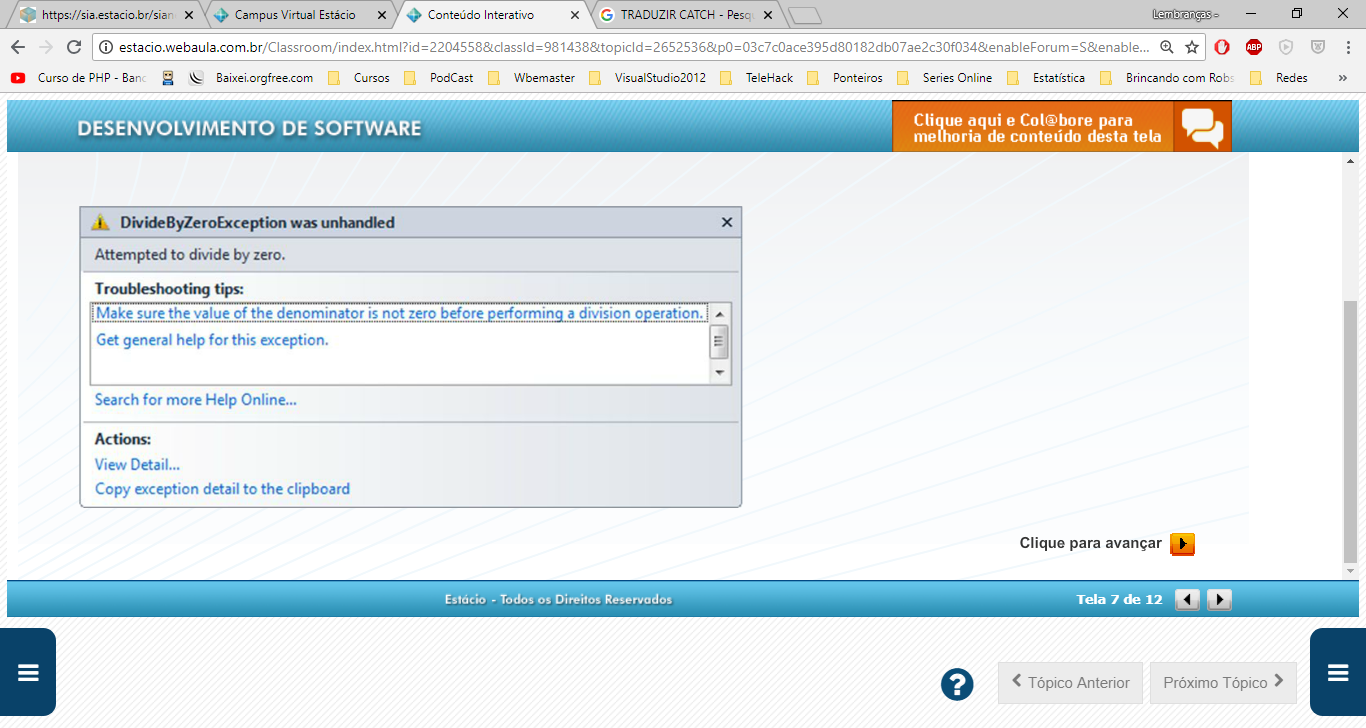
Try   
 ...  
Catch ex1 As Execption1  
 ‘Código para tratamento da Exception1  
 ...  
Catch ex2 As Exception2  
 ‘Código para tratamento da Exception2  
 ...  
Catch exN As ExceptionN  
 ‘Código para tratamento da Exception3  
 ...  
Catch ex As Exception ‘captura genérica   
 ‘Código para tratamento das demais exceções  
 ...  
End Try

**Exemplo 1:**

O exemplo a seguir, declara a **variável i do tipo Decimal** e **atribui o valor 10**, declara a **variável j** do **Decimal** e **atribui o valor zero**, define **a divisão de i por j** sem captura e tratamento de exceção, que ao ser executado é abortado por gerar uma exceção de divisão por zero, Pois não é possível dividir tal valor por 0.

Module Module1  
 Sub Main()  
 Dim i As Decimal = 10  
 Dim j As Decimal = 0  
 Dim Resultado As Decimal  
   
 Resultado = i/j  
 Console.ReadKey()  
 End Sub  
End Module

O seguinte erro será exibido:



**Exemplo 2:**

O exemplo a seguir, é o exemplo anterior modificado para realizar a captura e o tratamento da exceção genérica, ou seja, de qualquer tipo de exceção.

Module Module 1  
 Sub Main()  
 Dim i As Decimal = 10  
 Dim j As Decimal = 0  
 Dim Resultado As Decimal  
   
 Try   
 Resultado = i/j  
 Catch ex As Exception   
 ‘captura genérica  
 Console.Writeline(“Houve um erro de exceção de divisão por zero”)  
 End Try  
 Console.ReadKey()  
 End Sub  
End Module

1. Os blocos Try ... End Try Aninhados

O exemplo a seguir, define um bloco **Try.**..**End Try** externo que contém um outro bloco **Try** ... **End Try** interno. ***Se ocorrer uma exceção na abertura do arquivo***, o programa exibe a mensagem "Exceção na abertura do arquivo." e ***se após a abertura do arquivo sem exceção, ocorrer uma exceção na leitura do arquivo***, o programa exibe a mensagem "Exceção na leitura do arquivo."

Imports System.IO  
Module Module1  
 Sub Main()  
 Dim FileName As String = “TestFile.data”  
 ‘Bloco externo  
 Try   
 Dim fs As New FileStream(FileName, FileMode.Open, FileAccess.Read)  
 ‘Bloco interno  
 Try   
 Dim reader As New BinaryReader(fs)  
 reader.ReadInt32()  
 Catch ex As Exception   
 Console.WriteLine(“Exceção na leitura do arquivo.”)  
 End Try   
   
 Catch ex As Exception  
 Console.WriteLine(“Exception na abertura do arquivo.”)  
 End Try  
 End Sub  
End Module

1. **A seção Finally**

A seção **Finally** define *o código que deverá ser sempre executado*, *sem ou com a ocorrência de exceção*, antes de terminar o bloco ***Try ... End Try***. Normalmente, contém o código para liberar recursos alocados no bloco Try ... End Try ou o código comum a ser executado antes de terminar o bloco Try ... End Try.

O exemplo a seguir, é o exemplo anterior modificado para conter a seção **Finally** que sempre emite um sinal sonoro, **Beep()**, antes de terminar o bloco **Try...End Try**.

Module Program  
    Sub Main()  
        Dim i As Decimal = 10   
        Dim j As Decimal = 0  
        Dim Resultado As Decimal  
          
        Try   
            Resultado = i/j  
        Catch exDivideByZero As DivideByZeroException  
            'Captura uma exceção específica de divisão por 0 -- procurar API para estudar  
            Console.WriteLine("Houve um erro devido a uma divisão por 0")              
        Finally   
            Beep()  
            Console.WriteLine("O programa finalizou")  
        End try  
        Console.ReadKey()  
    End Sub  
End Module

1. Lançamentos de Exceções

Em algumas situações, desejamos lançar uma exceção para que seja tratada por uma rotina de tratamento de exceção ou para gerar uma exceção.

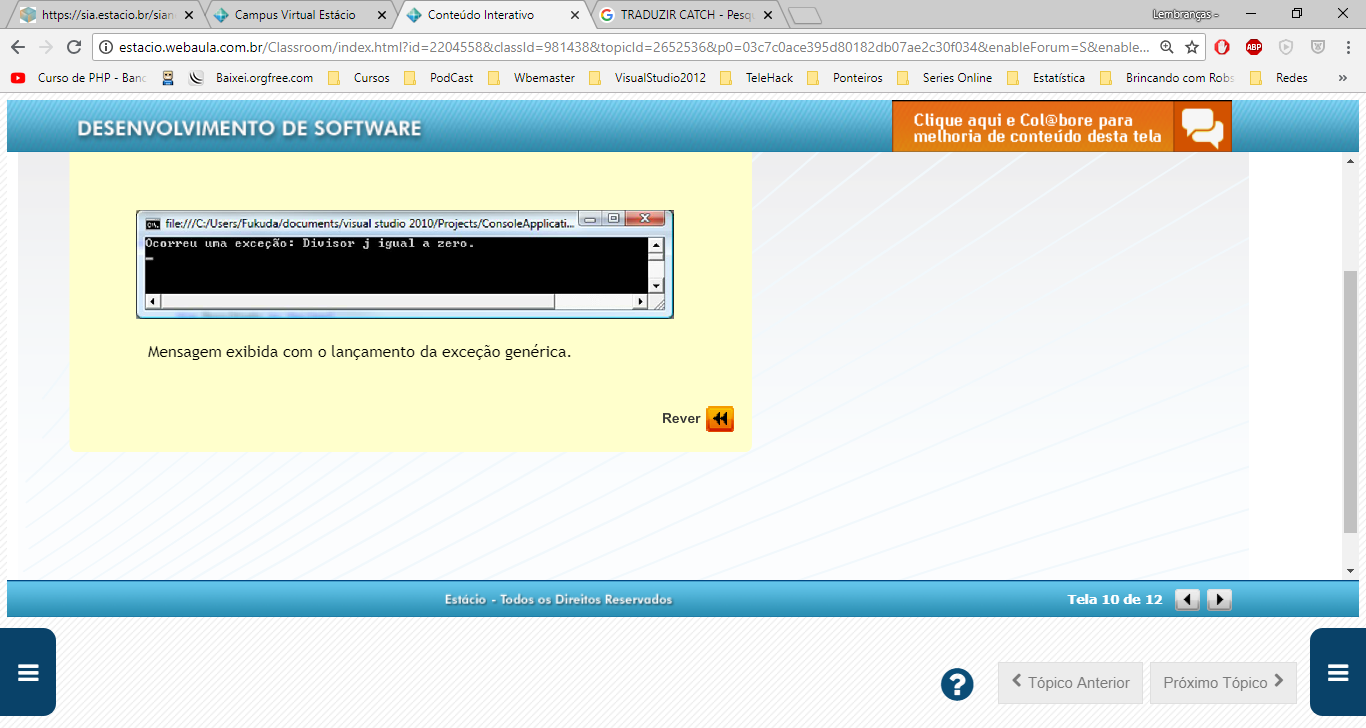
Sintaxe:

**Throw** *[ expressão ]*

**Exemplo:**

O exemplo a seguir, é o exemplo anterior modificado para lançar uma exceção genérica com a mensagem "Divisor j igual a zero" *se o valor da variável j for igual a zero*. Como a variável **j é declarada e inicializada com valor igual a zero**, o programa lança a exceção, que é capturada pela seção Catch, a qual exibe a mensagem "Ocorreu uma exceção: Divisor j igual a zero.", conforme mostra a imagem.

***Module Module1  
 Sub Main()  
 Dim*** *i* ***As Decimal = 10  
 Dim*** *j* ***As Decima = 0  
 Dim*** *Resultado* ***As Decimal   
 Try*** *i = i+10* ***if(****j = 0****) Then  
 Throw New System.Exception(****“Divisor j igual a zero.”****)  
 End if*** *Resultado = i/j* ***Catch*** *ex* ***As Exception*** *‘captura genérica* ***Console.WriteLine(“****Ocorreu uma exceção: {0}.****”,ex.Message)  
 Finally  
 Beep()  
 End Try  
 Console.ReadKey()  
 End Sub  
End Module***



Realize as seguintes atividades para saber mais.

Estude a Introdução ao Tratamento de Exceção da linguagem VB 2010 disponível em   
<http://msdn.microsoft.com/pt-br/library/c68xfk56.aspx>.

Estude a Visão Geral sobre a Manipulação Estruturada de Exceções para o Visual Basic da linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/8a9f2ew0.aspx>

Estude a Classe Exception da linguagem VB 2010 disponível em   
<http://msdn.microsoft.com/pt-br/library/system.exception.aspx>

Estude Como Capturar uma exceção em linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/ys1b32h3.aspx>

Estude a Declaração Try... Catch... Finally... End Try da linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/fk6t46tz.aspx>

Estude a Visão Geral do Tratamento de Exceção Não Estruturado da linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/sf1hwa21.aspx>

Estude a Declaração On Error da linguagem VB 2010 disponível em   
<http://msdn.microsoft.com/pt-br/library/5hsw66as.aspx>

Aprenda Quando Usar Manipulação de Exceção Estruturada ou Não Estruturada em linguagem VB 2010 disponível em <http://msdn.microsoft.com/pt-br/library/s74yzzb5.aspx>

Estude Como Lançar uma exceção em linguagem VB 2010 disponível em  
<http://msdn.microsoft.com/pt-br/library/z7w9ff54.aspx>

Estude a Declaração Throw em linguagem VB 2010 disponível em  
<http://msdn.microsoft.com/pt-br/library/ty79csek.aspx>