**@Countinous Education - ARISP  
Programando com C# .NET**

Sumário

[**1.** **NET Framework** 4](#_Toc22282403)

[1.1. O que é .NET FRAMEWORK? 4](#_Toc22282404)

[1.2. E que é .NET CORE? 4](#_Toc22282405)

[1.3. Serviços e tecnologias 4](#_Toc22282406)

[1.4. Linguagens Homologadas - CLS 5](#_Toc22282407)

[1.5. O que podemos fazer com o .net? 5](#_Toc22282408)

[1.6. A Base do .NET Framework 6](#_Toc22282409)

[1.7. Como funciona a Estrutura do .NET Framework 6](#_Toc22282410)

[1.8. Principais Bibliotecas do Framework Class Library – FCL 7](#_Toc22282411)

[1.9. Just-in-time compiler - Jit 7](#_Toc22282412)

[1.10. Garbage colector 7](#_Toc22282413)

[1.11. Intermediate language - IL 7](#_Toc22282414)

[1.12. Debugging 8](#_Toc22282415)

[1.13. Arquivos Configuration 8](#_Toc22282416)

[1.13.1. Web.config 8](#_Toc22282417)

[1.13.2. IConfiguration 9](#_Toc22282418)

[**2.** **Linguagem C#** 9](#_Toc22282419)

[2.1. Benefícios da linguagem C# 9](#_Toc22282420)

[2.2. C# - Uma das linguagens mais usadas atualmente 10](#_Toc22282421)

[**3.** **Estrutura da Linguagem C#** 10](#_Toc22282422)

[3.1. Meu primeiro App “Hello World” 10](#_Toc22282423)

[3.2. Variáveis, tipos primitivos, recebendo valor e conversão de tipos 11](#_Toc22282424)

[3.3. Operadores 12](#_Toc22282425)

[3.3.1. Exercício - 01 13](#_Toc22282426)

[3.4. Estruturas Condicionais/ Controle 13](#_Toc22282427)

[3.4.1. If/Else 13](#_Toc22282428)

[3.4.2. Switch 13](#_Toc22282429)

[3.4.3. Exercício - 02 14](#_Toc22282430)

[3.5. Arrays e Lists 14](#_Toc22282431)

[3.5.1. Array 14](#_Toc22282432)

[3.5.2. List<T> 14](#_Toc22282433)

[3.6. Estruturas de Repetição (loop, laço) 14](#_Toc22282434)

[3.6.1. For 14](#_Toc22282435)

[3.6.2. Foreach 14](#_Toc22282436)

[3.6.3. Do – While 15](#_Toc22282437)

[3.6.4. Exercício – 03 15](#_Toc22282438)

[3.7. Classes e Objetos 16](#_Toc22282439)

[3.7.1. Instanciando uma classe - objeto 17](#_Toc22282440)

[3.7.2. Modificadores de acesso 18](#_Toc22282441)

[3.7.3. Exercício - 04 18](#_Toc22282442)

[3.7.4. Classes Abstratas 19](#_Toc22282443)

[3.7.5. Classes e membros estáticos 20](#_Toc22282444)

[3.7.6. Interfaces 20](#_Toc22282445)

[3.7.7. Exercício - 05 20](#_Toc22282446)

[3.8. Exceptions 21](#_Toc22282447)

[3.9. Enumeradores 21](#_Toc22282448)

[3.9.1. Exercício – 06 22](#_Toc22282449)

[3.10. Boas práticas no desenvolvimento 22](#_Toc22282450)

[3.11. Capitalization Styles 22](#_Toc22282451)

[3.11.1. Dicas 22](#_Toc22282452)

[3.11.2. Clean Code 22](#_Toc22282453)

[3.12. ADO.Net 23](#_Toc22282454)

[3.13. Linq to SQL 23](#_Toc22282455)

[3.14. EntityFramework 23](#_Toc22282456)

[3.15. Web Forms 23](#_Toc22282457)

[3.16. MVC 23](#_Toc22282458)

# **NET Framework**

## O que é .NET FRAMEWORK?

* O .NET Framework é um ambiente de execução gerenciado para “Windows” oferecem uma variedade de serviços aos aplicativos em execução. Ele consiste em dois componentes principais: o CLR (Common Language Runtime), o mecanismo de execução que manipula aplicativos em execução, e a biblioteca de classes .NET Framework, que oferece uma biblioteca de códigos testados e reutilizáveis que os desenvolvedores podem chamar de seus próprios aplicativos.
* Surgiu em: **13 de fevereiro de 2002**
* Versão atual: **.NET Framework 4.8**

## E que é .NET CORE?

* O .NET Core é uma plataforma de desenvolvimento de código aberto de uso geral mantida pela Microsoft e pela comunidade .NET no GitHub. É uma plataforma cruzada (compatível com Windows, macOS e Linux) que pode ser usada no desenvolvimento de dispositivos, na nuvem e em aplicativos de IoT (Internet Of Things).
* Versão atual: **.NET Core 3.0**

## Serviços e tecnologias



## Linguagens Homologadas - CLS

* *Common Language Specification - CLS é responsável pela padronização das linguagens para seu perfeito funcionamento com a Máquina Virtual Net Framework- CLR.*

*Linguagens homologadas pela Microsoft:*

**Visual Basic (VB),**

**C# (Vs. 7.0),**

**F#,**

**C++/CLI**

Outras linguagens suportadas (tool for Visual Studio):

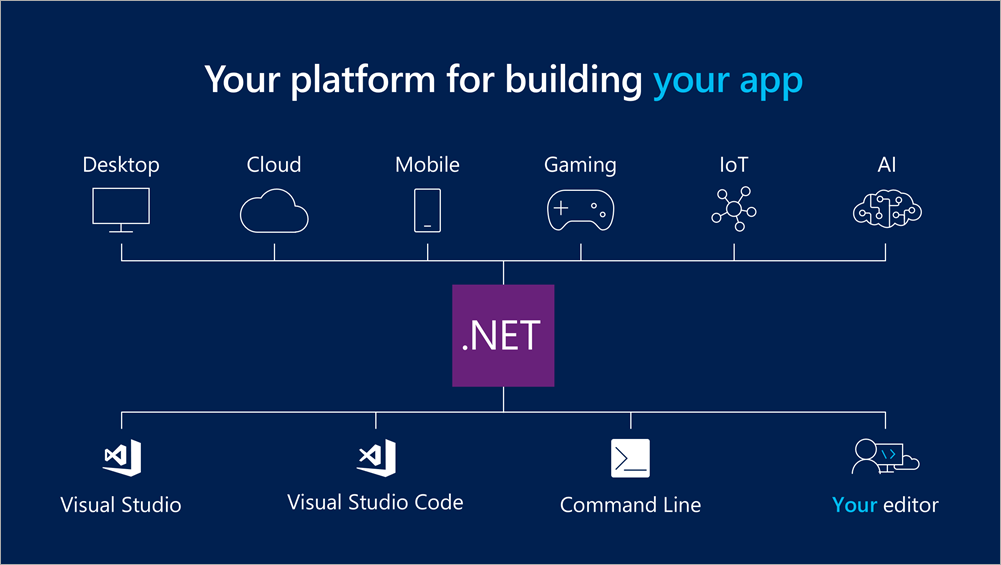
**Phyton,**

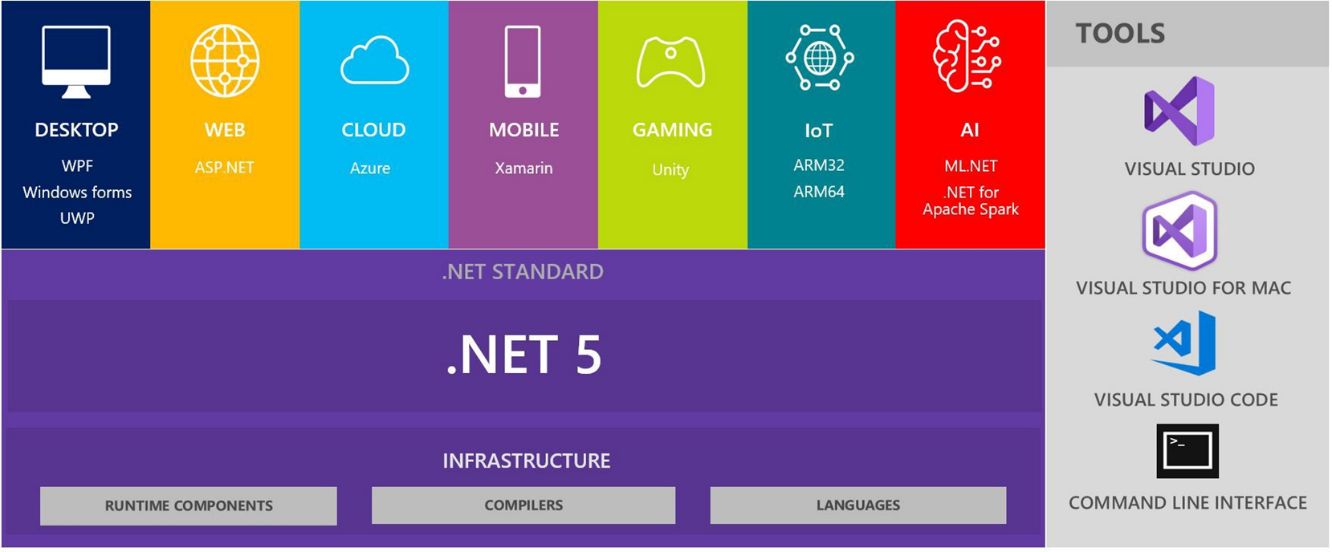
**PHP,**

**Delphi .Net,**

**[others]**

## O que podemos fazer com o .net?





## A Base do .NET Framework

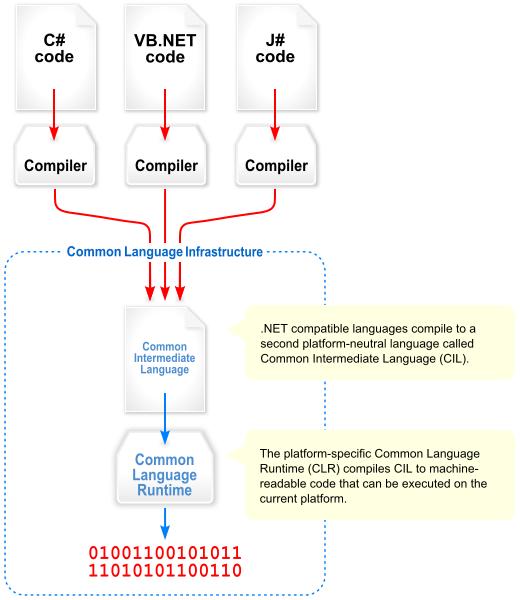
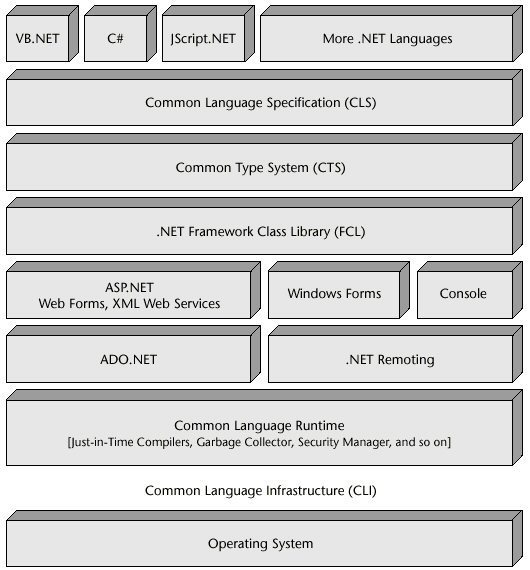
Common Language Runtime

(Maquina virtual)

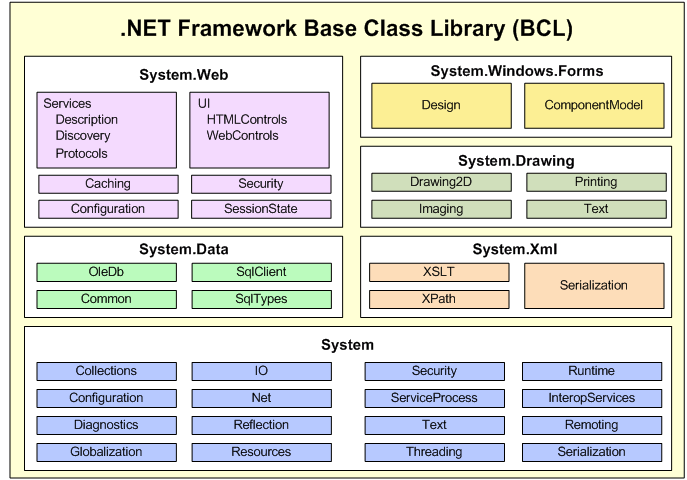
Framework Class Library

(Conjunto de Bibliotecas Unificadas)

## Como funciona a Estrutura do .NET Framework



## Principais Bibliotecas do Framework Class Library – FCL



## Just-in-time compiler - Jit

JiT é o compilador da CLR do .NET Framework com execução em desempenho máximo compilando o código escrito em linguagem de máquina com a vantagem de executar no ambiente integrado do .NET.

## Garbage colector

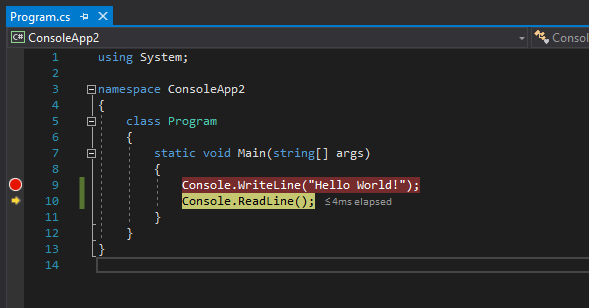
* O coletor de lixo (Garbage Colector) atua como um gerenciador automático de memória. Ele oferece os seguintes benefícios:
* Permite que você desenvolva seu aplicativo sem precisar liberar a memória manualmente para objetos criados.
* Aloca objetos no heap gerenciado com eficiência.
* Recupera os objetos que não estão sendo usados, limpa a memória e mantém a memória disponível para alocações futuras. Os objetos gerenciados obtêm automaticamente conteúdo limpo com o qual começar, portanto, seus construtores não precisam inicializar cada campo de dados.
* Fornece segurança de memória, assegurando que um objeto não possa usar o conteúdo de outro objeto.

## Intermediate language - IL

É a linguagem desenvolvida pela Microsoft, gerada no processo de compilação dos programas desenvolvidos na plataforma .NET  
As instruções IL são armazenadas em um container Assembly com um números binários conhecidos como OpCode.



## Debugging



* Insert breakpoint (F9)
* Step Over (F10)
* Step In (F11)
* Step Out (SHIFT + F11)
* Call back – Pilha de Chamadas (Ctrl+ALT+C)
* Add Watch
* Quick Watch

## Arquivos Configuration

### Web.config

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<system.web>

<connectionStrings>

<add name=“myConnectionString” connectionString=“server=localhost;

database=myDb; uid=myUser; password=myPass;” />

</connectionStrings>

</system.web>

</configuration>

using System.configuration;

string SqlConnection = ConfigurationManager.ConnectionStrings["myConnectionString"].ConnectionString;

### IConfiguration

AppSetthings.Json

{

"ConnectionStrings": {

"DefaultConnection": "Data Source=SCHEMA;Persist Security Info=True;User ID=USUARIO;Password=SENHA"

}

}

# **Linguagem C#**

* Um novo projeto de linguagem de programação foi iniciado, o projeto COOL (C-like Object Oriented Language). Anders Hejlsberg foi escolhido como engenheiro chefe desse novo projeto. COOL teve seu design baseado em diversas outras linguagens do mercado como Java, C, C++, Smalltalk, Delphi e VB. A ideia era estudar os problemas existentes e incorporar soluções.
* Em 2002, o projeto COOL foi lançado como linguagem C# 1.0, junto com o ambiente .Net 1.0.
* Atualmente, a linguagem C# está em sua versão 8.0, e o .Net na versão 4.7, tendo evoluído com expressiva velocidade, adotando novidades na sua sintaxe que a diferenciaram bastante do Java e outras concorrentes.

## Benefícios da linguagem C#

* Fast Speed
* Simple
* Object-Oriented
* Modern Programming Language
* Type-Safe
* Interoperability
* Scalable and Updateable
* Structured Programming Language
* Rich Library
* Component Oriented

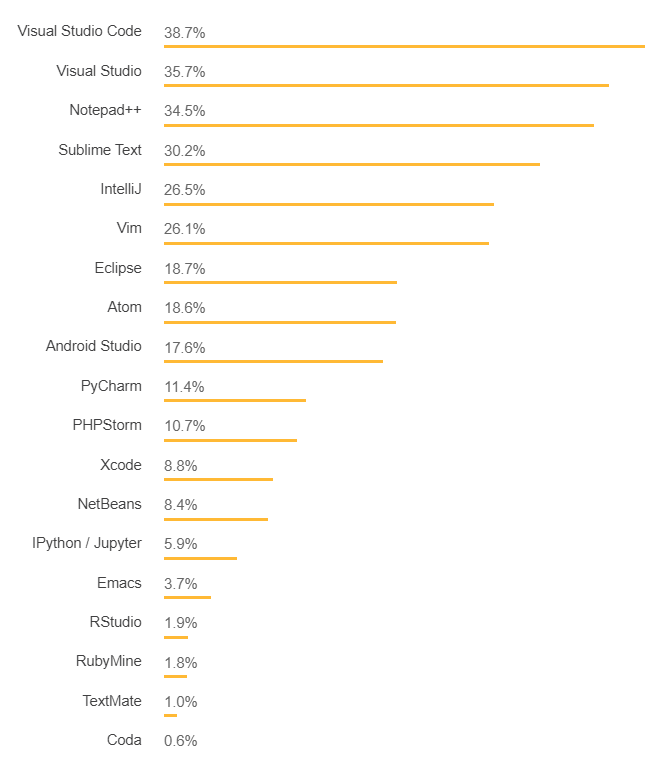
## C# - Uma das linguagens mais usadas atualmente

Linguagens mais utilizadas no mundo e as IDE’s favoritas para o desenvolvimento

1ª

2ª

3ª



1ª

2ª

# **Estrutura da Linguagem C#**

## Meu primeiro App “Hello World”

* Abrindo o Visual Studio;
* Conhecendo o ambiente;
* Criando um novo Projeto;
* Comentando Código: // or /\* \*/
* Sumary: “///”
* Region: “#region #endregion”

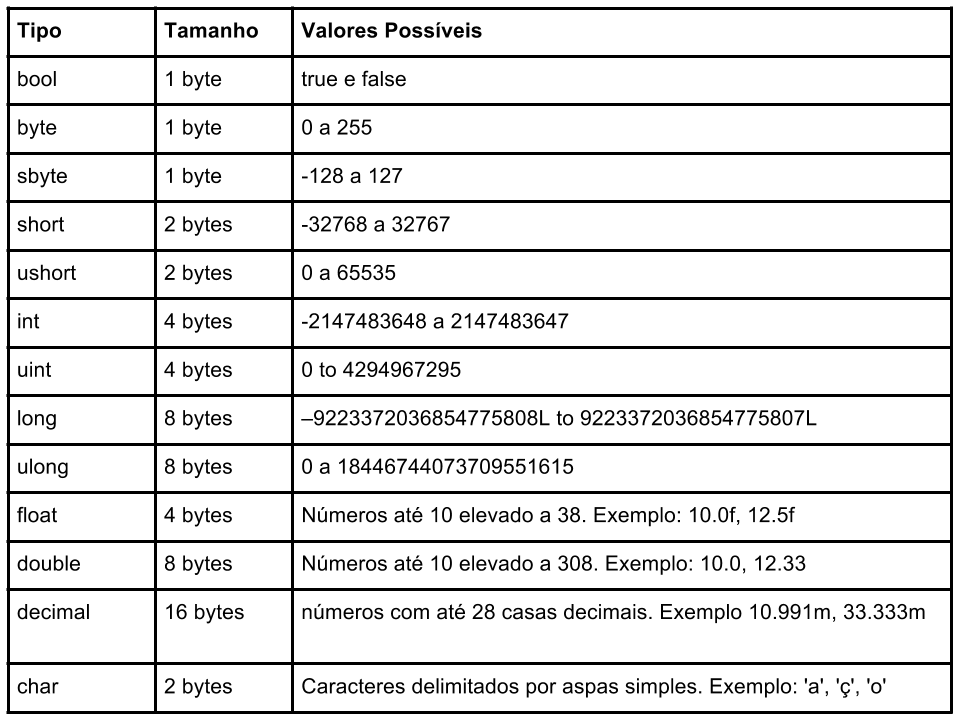
## Variáveis, tipos primitivos, recebendo valor e conversão de tipos

Declarando Variáveis: *string, char, int, short, long, bool, datetime, object, var, double, decimal, float.*

Variáveis Nulable (*?*)

Recebendo valor: “=“

Conversão: *Cast, Parse, Convert, Tryparse, ToString(), Tolower(), Toupper()*

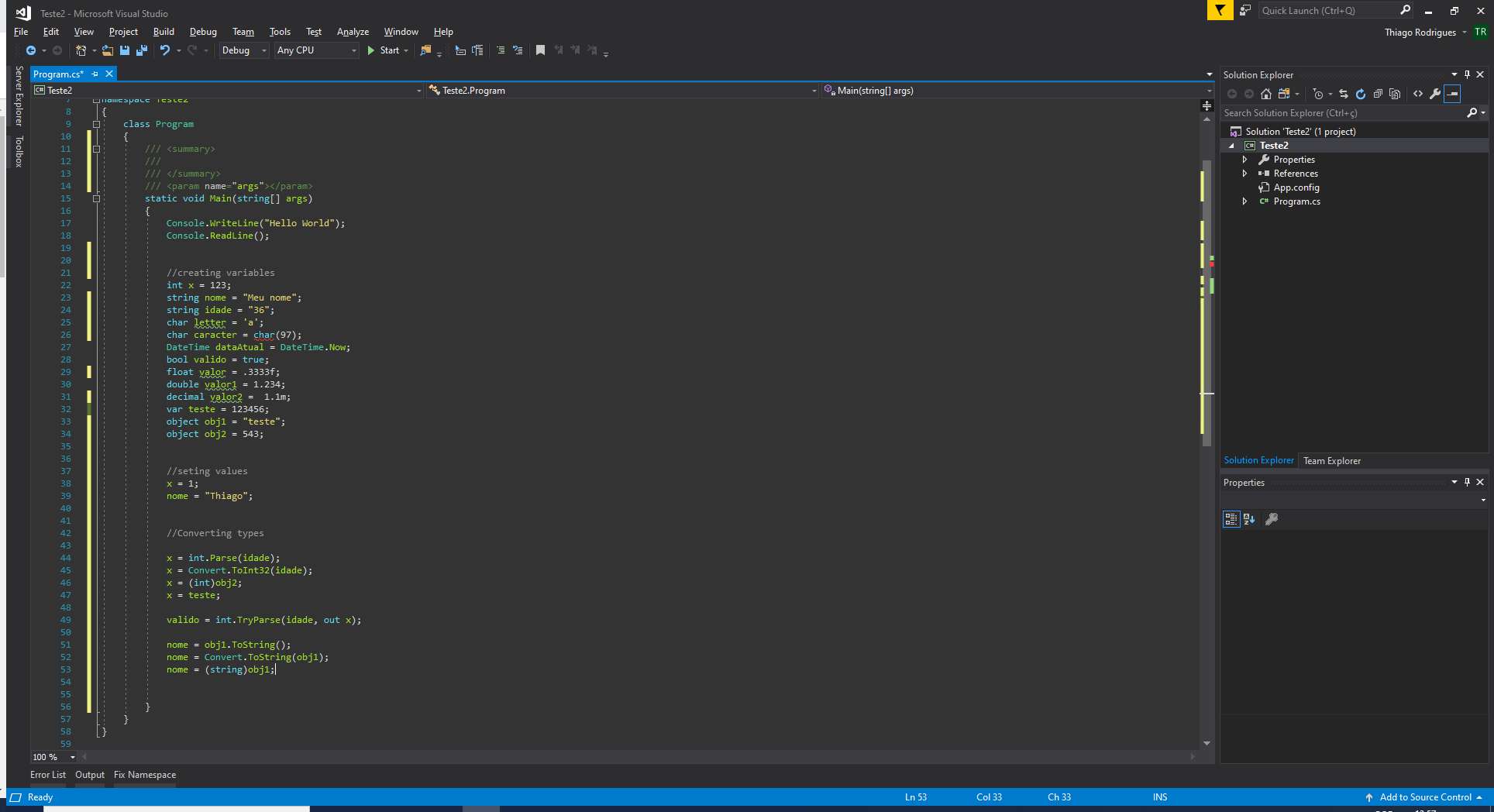


Alocação em memória:

Stack

Heap

Exemplo de variáveis:



## Operadores

* Aritiméticos: *+, -, \*, /, x++, x--, ++x, --x, %*
* Lógicos: *!, &, |, ^ , &&, ||, ..*
* Igualdade e comparação: *==, ===, <, >, <=, >=, !=, ??*
* Token / lambda: *=>*

## Exercício - 01

1. Crie 3 variáveis com as idades dos seus melhores amigos e/ou familiares. Algo como:

int idadeJoao = 10;

int idadeMaria = 25;

...

* Em seguida, pegue essas 3 idades e calcule a média delas. Exiba o resultado em um retorno no console.

1. No colegial, aprendemos a resolver equações de segundo grau usando a fórmula de Bhaskara. A fórmula é assim:

delta = b \* b - 4 \* a \* c;

a1 = (-b + raiz(delta)) / (2 \* a);

a2 = (-b - raiz(delta)) / (2 \* a);

* Crie um programa com três variáveis inteiras, a, b, c pergunte o valores ao usuário via console. Depois crie 3 variáveis double, delta, a1, a2, com a fórmula anterior e retorne o resultado no console.

Fórmula raiz quadrada: Math.Sqrt(variável)

## Estruturas Condicionais/ Controle

## If/Else

If(condição)

{

}

Else

{

},

Condição Ternária:

Var x = condição ? valor se verdadeiro : valor se falso;

## Switch

switch (variável)

{

case:

break;

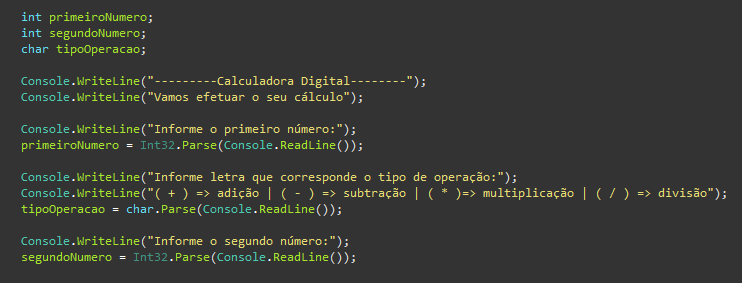
default:

break;

}

## Exercício - 02

Construindo uma calculadora:



Monte o programa acima, efetue o cálculo que o usuário escolheu e apresente o resultado no console.

## Arrays e Lists

## Array

int[] numeros = new int[5];

numeros[0] = 1;

numeros[1] = 2;

numeros[2] = 3;

numeros[3] = 4;

numeros[4] = 5;

int[] numerosPares = { 2, 4, 6, 8 };

## List<T>

using System.Collections.Generic;

List<string> listaNomes = new List<string>();

listaNomes.Add("João");

listaNomes.Add("Maria");

## Estruturas de Repetição (loop, laço)

## For

For(inicialização; condição; incremento)

{

}

## Foreach

ForEach(variável x in lista)

{

}

## Do – While

while (condição)

{

}

Do

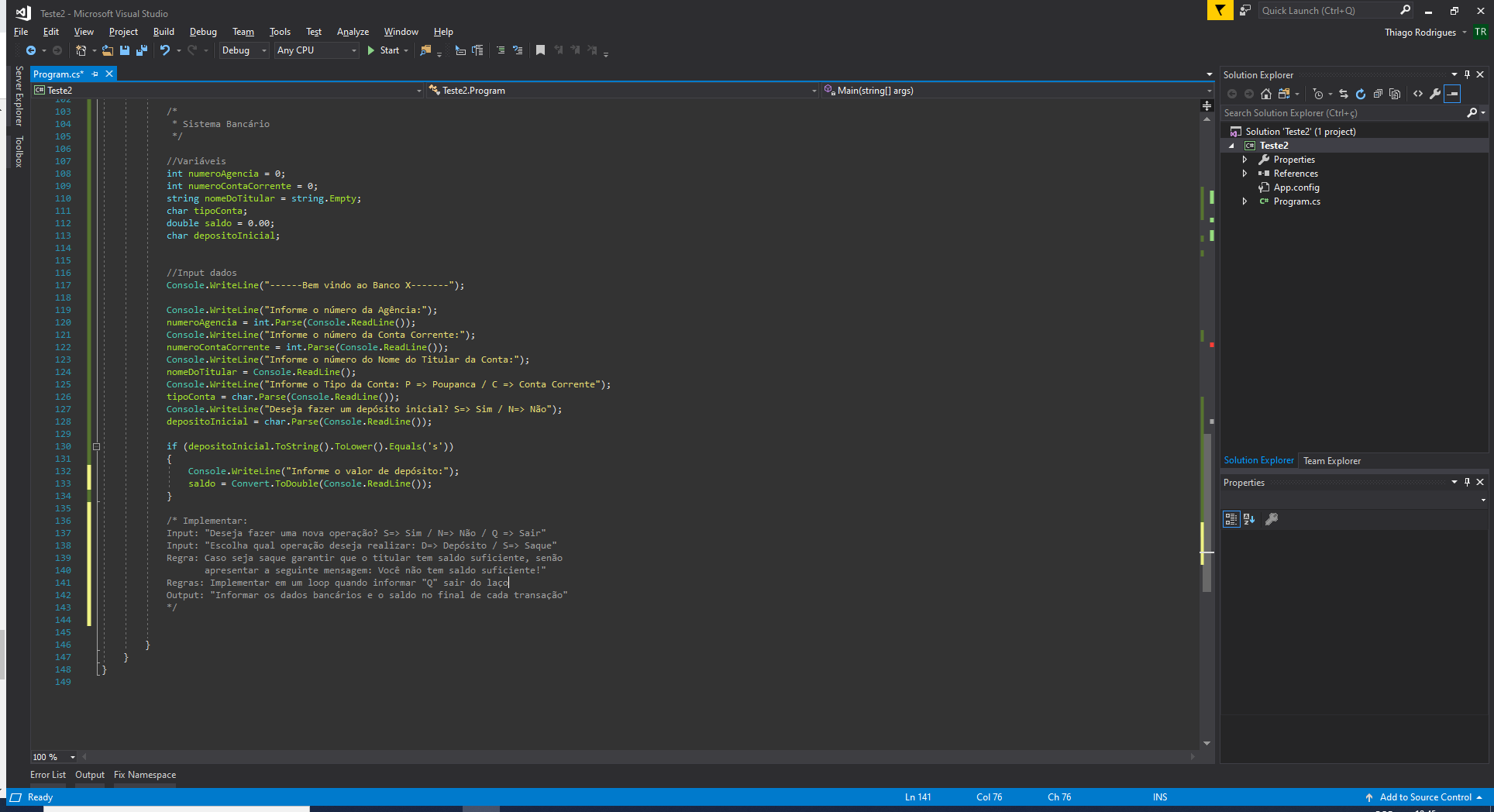
{

} while (condição)

## Exercício – 03

Montando um sistema bancário:  
implementar:

* Input: "Deseja fazer uma nova operação? S=> Sim / N=> Não / Q => Sair";
* Input: "Escolha qual operação deseja realizar: D=> Depósito / S=> Saque“;
* Input: "Informe o valor de depósito:" ou "Informe o valor de saque:“;
* Regra: Caso seja saque garantir que o titular tem saldo suficiente, senão apresentar a seguinte mensagem: Você não tem saldo suficiente!“;
* Regras: Implementar em um loop quando informar "Q" sair do laço;
* Output: "Informar os dados bancários e o saldo no final de cada transação“.



## Classes e Objetos

Em **orientação a objetos**, uma **classe** é uma descrição que abstrai um conjunto de **objetos** com características similares. Mais formalmente, é um conceito que encapsula abstrações de dados e procedimentos que descrevem o conteúdo e o comportamento de entidades do mundo real, representadas por **objetos**.

Exemplos de classe:

*Pessoa,*

*Animal,*

*Automóvel,*

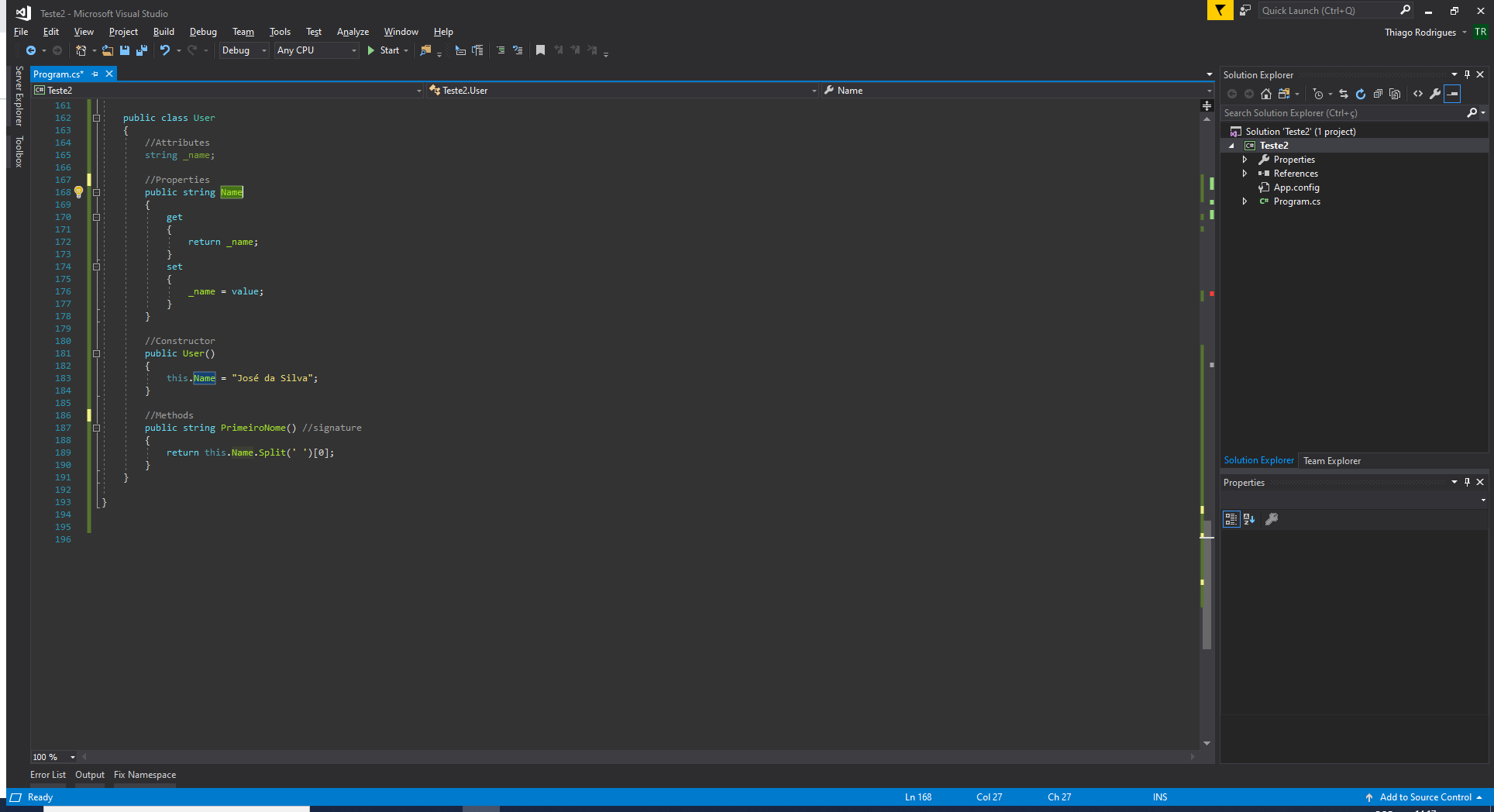
*Publicação ...*

Desta forma, a primeira coisa que se deve fazer em uma análise orientado a objetos e identificar as entidades que se pretende considerar no sistema e imediatamente transformá-las em classes.

Membros de uma classe:

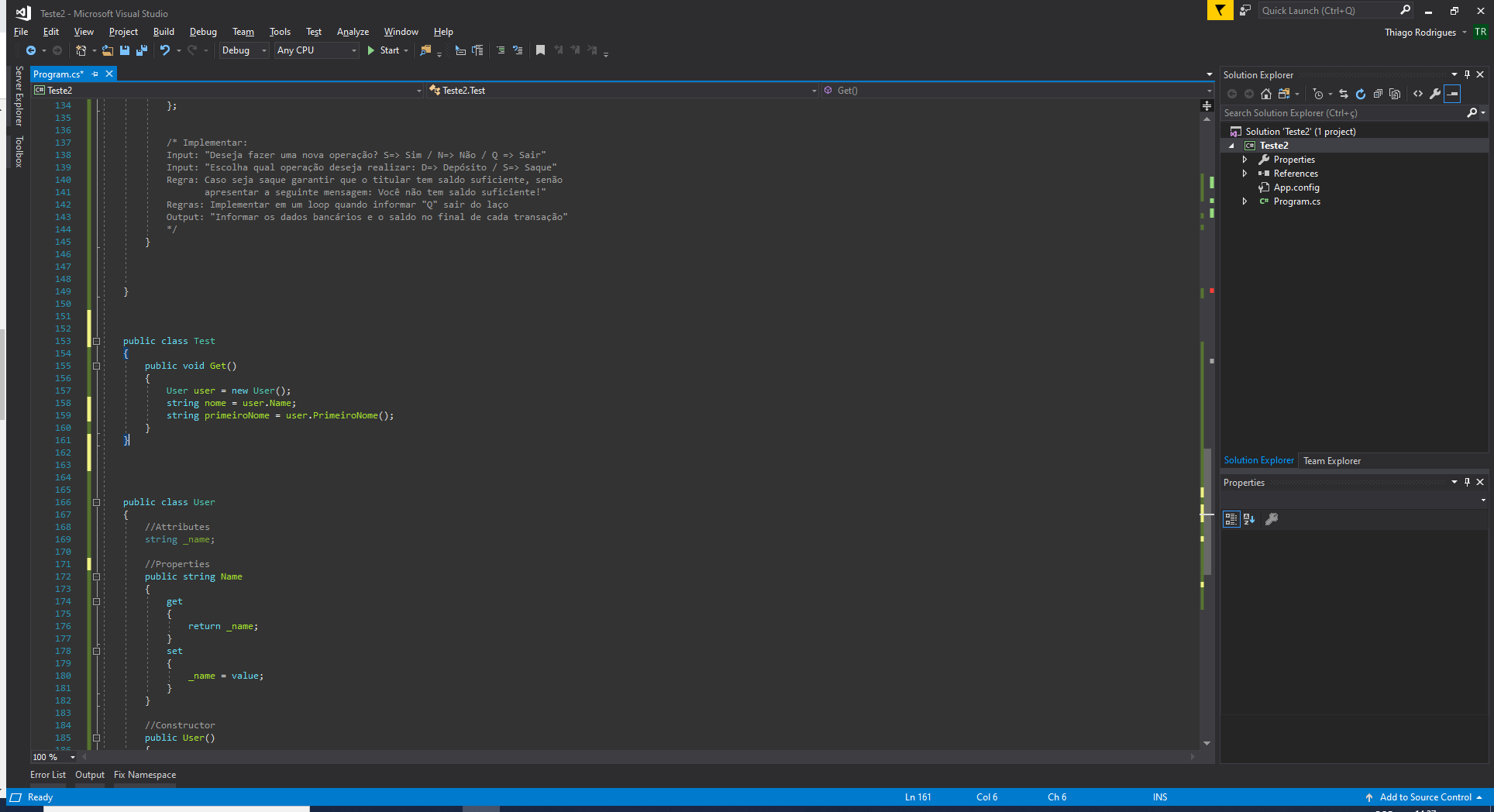
* 1. Classes
  2. Atributos
  3. Propriedades
  4. Métodos
  5. Construtores
  6. Finalizadores
  7. Tipos Aninhados
  8. Campos

Exemplo de uma Classe:



## Instanciando uma classe - objeto

**Instanciar uma classe** significa criar um objeto daquela **classe** ou seja, objeto é o conjunto de atributos e métodos da **classe.**



## Modificadores de acesso

Os modificadores de acesso são palavras-chave usadas para especificar a acessibilidade declarada de um membro ou de um tipo.

**public:** Acesso ilimitado às classes.

**private:** Acesso limitado aos membros da classe.

**protected:** Acesso apenas aos membros e a tipos derivados da mesma.

**internal:** O acesso é limitado ao assembly.

**protected internal:** Acesso limitado aos membros, derivadas ou membros deste assembly.

**private protected:** Acesso é limitado aos membros e derivados no assembly atual.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Modificador** | **Classe e membros** | **Classe Derivada (herdada)** | **Classes no Mesmo pacote** | **Classes fora do pacote** | **Externo** |
| Public | **✓** | **✓** | **✓** | **✓** | **✓** |
| private | **✓** | **X** | **X** | **X** | **X** |
| protected | **✓** | **✓** | **X** | **X** | **✓** |
| Internal | **✓** | **✓** | **✓** | **✓** | **X** |

**Exemplos de modificadores para Classes:**

internal class User

{

string Nome;

}

class User

{

string Nome;

}

sealed class User

{

string Nome;

}

public class User

{

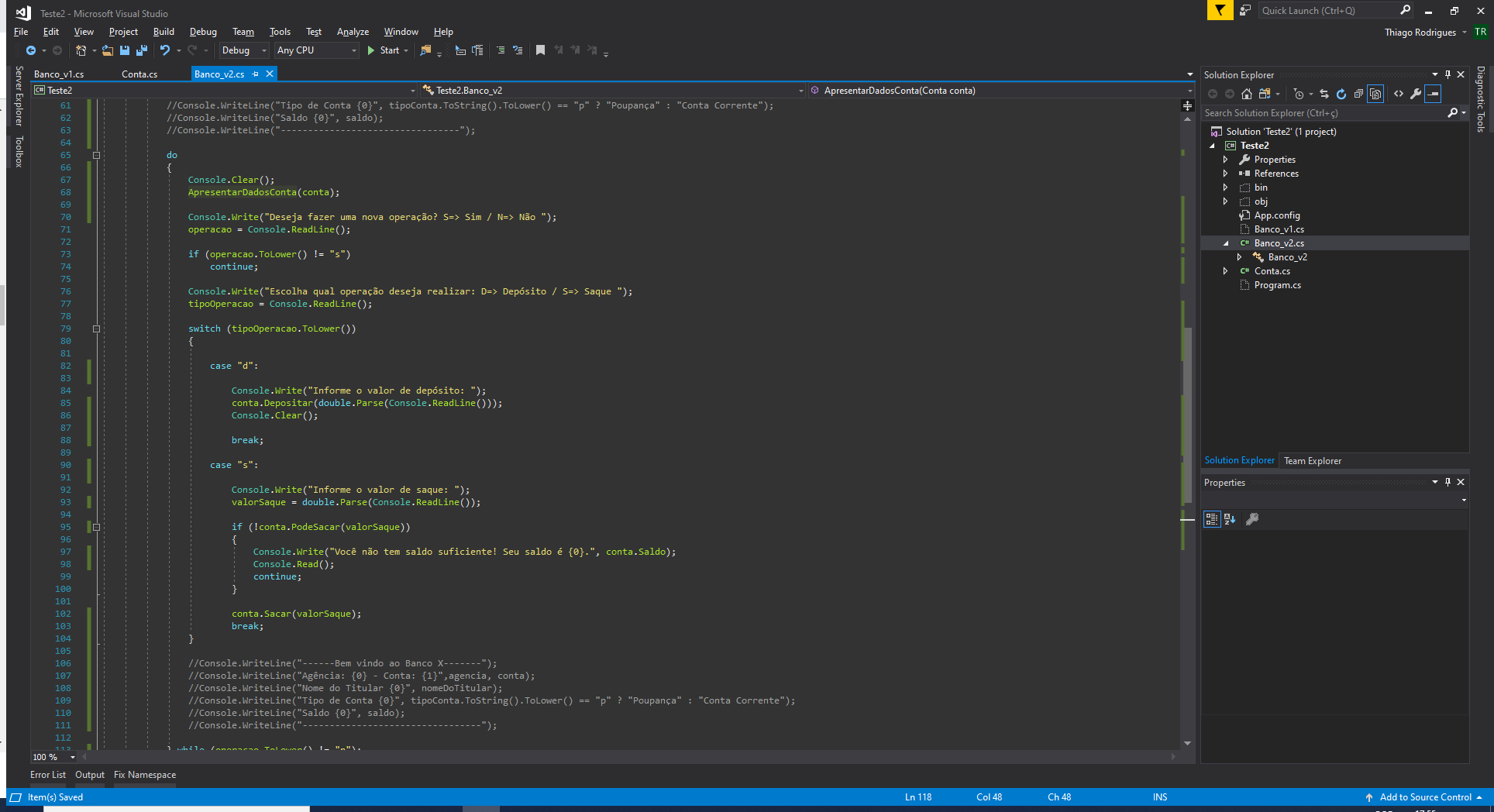
string Nome;

}

## Exercício - 04

Aprimorando o sistema bancário:

* Criar uma classe *Conta();*
* Criar as variáveis como propriedades da classe conta;
* Criar método *AbrirConta();*
* Criar métodos *Depositar()* e *Sacar();*
* Validar se o usuário tem saldo suficiente para sacar;
* Usar Refectoring no programa usando a classe Conta;
* Usar Refectoring criando um método no programa *ApresentarDadosConta();*



## Classes Abstratas

A classe abstrata é um tipo de classe que somente pode ser herdada e não instanciada, de certa forma pode se dizer que este tipo de classe é uma classe conceitual que pode definir funcionalidades para que as suas subclasses (classes que herdam desta classe) possam implementa-las de forma não obrigatória, ou seja ao se definir um conjunto de métodos na classe abstrata não é de total obrigatoriedade a implementação de todos os métodos em suas subclasses, em uma classe abstrata os métodos declarados podem ser abstratos ou não, e suas implementações devem ser obrigatórias na subclasse ou não, quando criamos um método abstrato em uma classe abstrata sua implementação é obrigatória, caso você não implemente o mesmo o compilador criará um erro em tempo de compilação.

public abstract class Conta

{

public abstract void Sacar(double valor);

public virtual void ResgatarValor()

{

}

}

Para criar uma classe abstrata devemos declarar na classe como sendo abstrata.  
Os métodos podem ser:

1. abstract: obriga as classes que e herdaram a implementarem o método utilizando “override”
2. virtual: É opcional para as classes derivadas herdarem os métodos ou não. Pode ser chamado o método da classe abstrata ou pode ser substituído por outro método utilizando o “override”

## Classes e membros estáticos

Os membros estáticos são campos, métodos, eventos ou propriedades que podem ser chamados em uma classe mesmo que não tenha sido criada uma instância desta classe.

Podemos usar membros estáticos para separar os dados e o comportamento, que independem de qualquer identidade do objeto, ou seja, os dados e funções nunca irão se modificar, independente do que acontecer com seu objeto.

A classe estática é alocada diretamente na memória.

public class Banco

{

private static string NomeBanco = "Banco .NET";

public static string ObterNomeBanco()

{

return NomeBanco;

}

}

## Interfaces

A Interface define um contrato que um objeto será obrigado a implementar. Para cada operação declarada por um objeto deve ser especificado o nome da operação, os objetos que esta operação aceita como parâmetro e o tipo de valor retornado pela operação; este conjunto de informações sobre uma determinada operação tem o nome de assinatura da operação, e um conjunto de assinaturas de operações dá-se o nome de interface.

É importante lembrar que uma interface nunca contém implementação, ou seja, numa interface não se pode definir campos, pois o mesmo é uma implementação de um atributo objeto, a interface também não permite construtores pois em um construtor temos as instruções usadas para inicializar campos.

interface IConta

{

void Sacar(double valor);

void Depositar(double valor);

}

## Exercício - 05

Implementando Abstract Class, Static e Interfaces:

* Crie um novo Project para Separar as classes de Dominio;
* Crie duas classes distintas **ContaCorrente** e **Poupança** e instancie a classe quando o usuário escolher o tipo de conta;
* Crie uma Interface **Conta** com os métodos e contratos e injete nas classes **Conta Corrente** e **Poupança;**
* Mude a classe **Banco** para **static** para guardar as informações gerais do banco;
* Crie um parâmetro **single** para gerar o número automático da conta a no ato criação de conta e altere para que no console mostre automaticamente;
* Permita na sua aplicação que o usuário crie várias contas correntes e poupança e persista em uma classe;
* Forneça uma opção para o usuário visualizar as contas criadas e poder criar mais se quiser;
* Forneça uma opção para o usuário fazer transferência em contas;
* Utilize Exceptions para tratar os erros e apresente para o Usuário.

## Exceptions

Para evitar problemas com erros e facilitar o tratamento de erros com regras de negócio, o C# nos permite tratar essas exceções à regra de uma maneira diferente: através de exceptions. Em vez de retornarmos um valor dizendo se uma operação foi bem sucedida, nós lançamos uma exceção à regra padrão, ao comportamento padrão, dizendo que algo de errado aconteceu.

try

{

}

catch (Exception)

{

throw;

}

finally

{

}

*Declarando Exceções:*

Throw new Exception();

Tipos de Exceções:

Exceptions*:* Exceção Genérica

ArgumentException*:* Argumento de um método Inválido

NullReferenceException*:* Tentativa de desreferenciar um objeto de referência nula;

OutOfMemoryException*:* Não há memória suficiente para continuar a execução de um programa;  
OverflowException*:* Operação aritmética, projeção ou de conversão em um contexto verificado resulta em um estouro;  
StackOverflowException: Pilha de execução estoura por conter chamadas de método aninhadas em excesso;

IndexOutOfRangeException*:* Tentativa de acessar um elemento de uma matriz ou coleção com um índice que está fora dos limites; *[...]*

**Criando uma Exceção Customizada:**

class CustomException : Exception

{

}

## Enumeradores

Enums (enumerations ou enumeradores) são constantes fortemente tipadas. Elas são estáticas e útil quando precisamos criar estruturas que serão pouco alteradas ao longo do desenvolvimento do projeto.

Exemplos:

public enum TipoConta

{

ContaCorrente = 'c',

Poupanca = 'p'

}

public enum Status

{

EmAnalise = 'E',

Ativo = 'a',

Desativado = 'd'

}

## Exercício – 06

* Implemente Exceptions para tratar os erros da sua aplicação bancária.

## Boas práticas no desenvolvimento

## Capitalization Styles

**Pascal Case**

Primeira letra do identificador e primeira letra de cada palavra concatenada em maiúsculo.

Exemplo: *BackColor*

**camel Case**

Primeira letra em minúsculo e cada palavra concatenada em maiúsculo.

Exemplo: *backColor*

**Uppercase**

Todas as letras iniciais são maiúsculas, quando separadas.

Exemplo: *System.IO, System.Web.UI*

**Regras Gerais:**

Não utilizar acentos.

Não utilizar espaços e underline *“\_”*.

Não utilizar *LETRASTODASMAIUSCULAS*

Não usar nomenclatura húngara: *strNome, intIdade, blnStatus*...

Abreviações são aceitáveis.

## Dicas

* Declaração de variáveis no início (camelCase);
* Declaração Propriedades e Métodos (UpperCase);
* Declaração de métodos “Ação/Infinitivo” Obter (Get), Validar (IsValid), Criar (Create) ...;
* Separar a classes por “#region” (Attributes, Properties, Methods e Constructors);
* Method “constructor” no final;
* Evitar duplicidades de código;
* Não usar muito IF e Else exageradamente, refatore sempre!;
* Objeto sempre válido (por Greg Young) “Nunca criar propriedades da classe inválidas”
* Código Clean Code;
* Usar refectorying (escrever um código de forma mais elegante, mais clara, mais performante sem alterar a funcionalidade do código.);
* Cada classe e seus membros devem representar um contexto único;
* Adotar a boa prática de comentar principais ações do código e usar sumary para os métodos;
* A clareza nos nomes dos métodos, variáveis e classes é essencial mesmo que o nome fique longo.

## Clean Code

**O que é Clean Code (código limpo)?**

Clean Code é uma filosofia de desenvolvimento cuja o principal objetivo é aplicar técnicas simples que visam facilitar a escrita e leitura de um código, tornando-o de fácil compreensão e revelando a sua real intenção.

**Porque devo usar Clean Code?**

“Alguma vez um código ruim já lhe atrasou consideravelmente? Se você for um programador, independentemente de sua experiência, então já se deparou várias vezes com esse obstáculo”

**Princípios:**

1. **Seja um verdadeiro autor do seu código**

*O código é uma história. Então, como um bom autor, devemos nos preocupar com a maneira de contar essa história.*

*“Elas precisam ser pequenas.”*

*Já a segunda regra das funções diz o seguinte:*

*“Elas têm de ser ainda menores.”*

1. **Comente. Mas só o necessário!**

*Mas não deixe de comentar, tenha um comentário para servir como documentação por mínimo que seja*

1. **Utilize D.R.Y**

*Don’t repeat yourself (Não repita a si mesmo)*

1. **Melhor prevenir do que remediar**

*Tente prever sempre antes o que pode acontecer*

*Uma dica excelente para não gerar erros em seu código é simplesmente não utilizar “null”*

1. **Regra de Escoteiro***“Deixe o código mais limpo do que estava antes de você mexer nele.”*
2. **Crie testes limpos**

*Testes limpos seguem as regras do acrônimo F.I.R.S.T (Fast, Indepedent, Repeatable, Self-validation, Timely)*

1. **Tenha orgulho do seu código**

*“Um código limpo sempre parece que foi escrito por alguém que se importava”*

*Se importar com um código é tomar cuidado na hora de:*

*Dar nomes*

*Criar funções pequenas e especificas*

*Não ser redundante*

*Estar sempre atento a erros*

*Não deixar de refatorar*

*Utilizar testes limpo para validações*

*No fim, a sensação de orgulho é recompensadora! O que compensa todo o trabalho duro durante o desenvolvimento!*

## ADO.Net

## Linq to SQL

## EntityFramework

## Web Forms

## MVC