SESI SENAI
Disciplina que gera conhecimento

Programação de Aplicativos

Framework .NET – C#

Instrutor Bruno Manzoli

SESI SENAI
Disciplina que gera conhecimento

Material Didático

- C# - Como Programar (A Bíblia)
- Linguagem C#.Net – Baseada em pequenos Projetos
- K19 Treinamentos – C# e Orientação a Objetos

Referências Online:

https://sites.google.com/site/profbrunomanzoli

ou

profbrunomanzoli@gmail.com

Metodologia e Avaliação

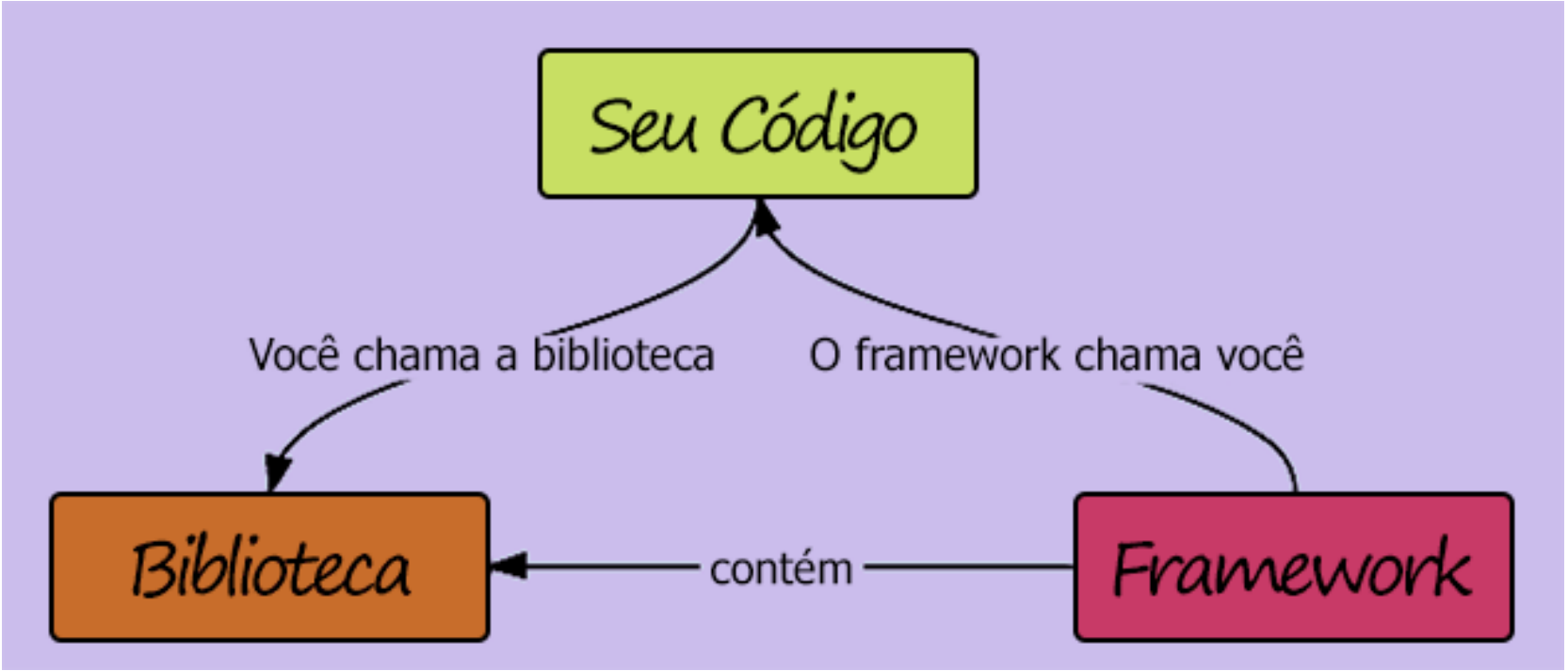
- Abordagem do conteúdo:
  - ✓Aulas expositivas;
  - ✓Seminários;
  - ✓Videoaulas;
  - ✓Gameficação;
  - ✓Exercícios práticos e teóricos.

- Sistema avaliativo:
  - ✓Duas provas práticas
  - ✓Uma prova teórica
  - ✓Um trabalho individual
  - ✓Um trabalho em grupo

O ambiente de desenvolvimento

- Definições:
  - API: "Application Programming Interface" que significa em tradução para o português "Interface para Programação de Aplicações";
  - Biblioteca: Conjunto de códigos organizados para a solução de problemas específicos;
  - Framework: Conjunto de bibliotecas para conseguir executar uma operação maior.

O ambiente de desenvolvimento



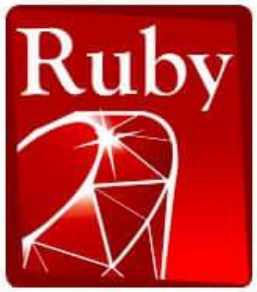
A Plataforma .NET

- Permitir que uma aplicação .NET seja executada em diferentes versões do Sistema Operacional Window.
- IDE: Integrated Development Environment (Ambiente de Desenvolvimento Integrado).

O que é um Programa?

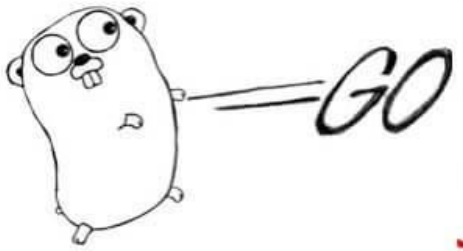
- Sequência de instruções
- Linguagem de máquina
- Linguagem de programação
- Compilador

Para onde ir? Por onde começar?



Objective-C

python



JavaScript

THE C PROGRAMMING LANGUAGE

Exemplo de um Programa C#

```
1  class OlaMundo
2  {
3    static void Main()
4    {
5      System.Console.WriteLine("Olá Mundo");
6    }
7  }
```

- Os códigos são colocados em arquivos com extensão .cs
- Obrigação de um método principal “Main”
- Compilar o código fonte para gerar um .exe

Método Main – Ponto de Entrada

- Primeiro método a ser chamado quando o programa é executado
- Precisa ser ***static*** e seu tipo de retorno ***void***

```
class Principal
{
    static void Main(string[] args)
    {
        System.Console.WriteLine("Meu primeiro programa em C#!");
    }
}
```

Método Main – Ponto de Entrada

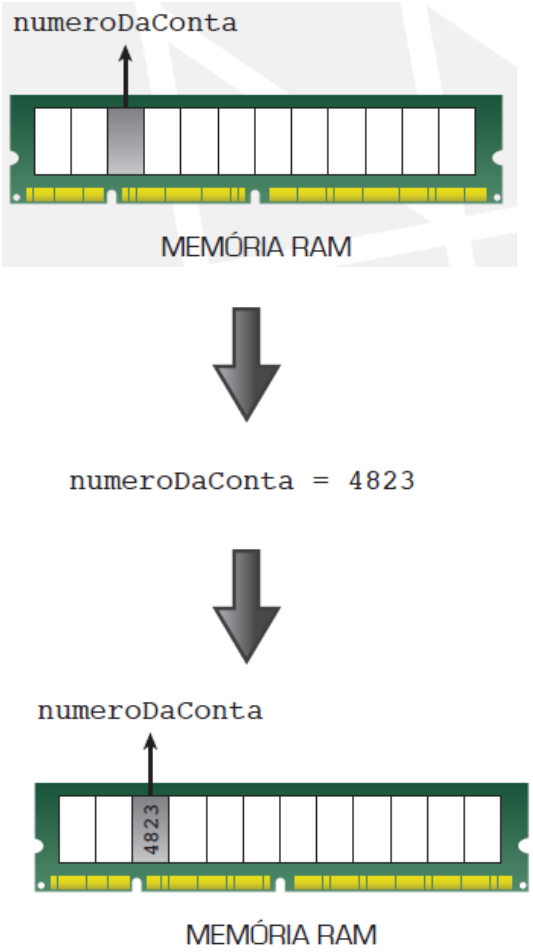
- Trabalhando com namespace:

```
using System;

namespace Aula1
{
    class Principal
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Meu primeiro programa em C#!");
        }
    }
}
```

Variáveis

- Basicamente, o que um programa faz é manipular dados. Em geral, esses dados são armazenados em **variáveis localizadas na memória RAM do computador. Uma variável pode guardar dados de** vários tipos: números, textos, booleanos (verdadeiro ou falso), referências de objetos. Além disso, toda variável possui um nome que é utilizado quando a informação dentro da váriavel precisa ser manipulada pelo programa.

Variáveis - Exemplo



Variáveis - Declaração

- As variáveis devem ser declaradas para que possam ser usadas
- Isso constitui um nome, que deve ser único e um tipo de valor

```
//Uma variável do tipo int chamada numeroDaConta
int numeroDaConta;

//Uma variável do tipo double chamada precoDoProduto
double precoDoProduto;
```

Variáveis - Característica da Linguagem de Programação

- Estaticamente Tipada: Exige que os tipos das variáveis sejam definidas antes da compilação
- Fortemente Tipada: Exige que os valores atribuídos a uma variável sejam compatíveis com o tipo da variável

Variáveis

- Inicialização:

```
int numero = 10;
double preco = 137.3;

System.Console.WriteLine(numero);
System.Console.WriteLine(preco);
```

- Erro de compilação:

```
int numero = 10;
int numero = 20;
```

Variáveis - Tipos Primitivos

Tipo	*Descrição*	*Tamanho*
sbyte	Valor inteiro entre -128 e 127 (inclusivo)	1 byte
byte	Valor inteiro entre 0 e 255 (inclusivo)	1 byte
short	Valor inteiro entre -32.768 e 32.767 (inclusivo)	2 bytes
ushort	Valor inteiro entre 0 e 65.535 (inclusivo)	2 bytes
int	Valor inteiro entre -2.147.483.648 e 2.147.483.647 (inclusivo)	4 bytes
uint	Valor inteiro entre 0 e 4.294.967.295 (inclusivo)	4 bytes

Variáveis - Tipos Primitivos

Tipo	*Descrição*	*Tamanho*
long	Valor inteiro entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807 (inclusivo)	8 bytes
ulong	Valor inteiro entre 0 e 18.446.744.073.709.551.615 (inclusivo)	8 bytes
float	Valor com ponto flutuante entre $1,40129846432481707x10^{-45}$ e $3,40282346638528860x10^{38}$ (positivo ou negativo) – Precisão de 7 dígitos nas casas decimais. Atribuição direta com uso da letra "f".	4 bytes

Variáveis - Tipos Primitivos

Tipo	Descrição	Tamanho
double	Valor com ponto flutuante entre 4,94065645841246544x 10^{-324} e 1,79769313486231570x10^{308} (positivo ou negativo) – Precisão de 15 a 16 dígitos nas casas decimais. Atribuição direta com uso da letra “d”.	8 bytes
decimal	Valor com ponto flutuante entre 1,0x10^{-28} e 7,9x10^{28} (positivo ou negativo) – Precisão de 28 a 29 dígitos nas casas decimais. Atribuição direta com uso da letra “m”.	16 byte
bool	true ou false	1 bit
char	Um único caractere Unicode de 16 bits. Valor inteiro e positivo entre 0 e 65.535	2 bytes
String	Guardar qualquer valor delimitado por aspas duplas. Não é um tipo primitivo.	

Operadores

- Aritmético (+, -, *, /, %)
- Atribuição (=, +=, -=, *=, /=, %=)
- Relacional (==, !=, <, <=, >, >=)
- Lógico (&&, ||)

Comandos de Entrada e Saída

- Segue abaixo a sintaxe dos comando utilizados para se fazer interação entre usuário e máquina:

```
//COMANDO DE SAÍDA
System.Console.Write("Texto a ser exibido na tela!");

//COMANDO DE ENTRADA
System.Console.Read();
```

Exercícios

1. Faça um programa para ler o salário mensal atual de um funcionário e o percentual de reajuste. Calcular e escrever o valor do novo salário.

Estruturas para tomada de decisão (IF e ELSE)

- O comportamento de uma aplicação pode ser influenciado por valores definidos pelos usuários. A partir de então, deve ser implementado qual decisão tomar.

```
if (preco < 0)
{
  System.Console.WriteLine("O preço do produto não pode ser negativo");
}
else
{
  System.Console.WriteLine("Produto cadastrado com sucesso");
}
```

Exercícios

2. As maçãs custam R$ 1,30 cada se forem compradas menos de uma dúzia, e R$ 1,00 se forem compradas pelo menos 12. Escreva um programa que leia o número de maçãs compradas, calcule e escreva o custo total da compra.

3. Ler as notas da 1ª. e 2ª. avaliações de um aluno. Calcular a média aritmética simples e escrever uma mensagem que diga se o aluno foi ou não aprovado (considerar que nota igual ou maior que 6 o aluno é aprovado). Escrever também a média calculada.

Estruturas de repetição

- **FOR**: Através desse comando**, é possível definir quantas vezes um determinado trecho de código** deve ser executado pelo computador.

```
for(int contador = 0; contador < 100; contador++) {
  System.Console.WriteLine("Bom Dia");
}
```

Estruturas de repetição

- **WHILE**: O comando **WHILE é análogo ao FOR. A diferença entre esses dois comandos é que o WHILE não recebe** três argumentos.

```
int contador = 0;

while(contador < 100)
{
  System.Console.WriteLine("Bom Dia");
  contador++;
}
```

- Derivação do WHILE é o DO WHILE.

Exercícios

4. Ler um valor N e imprimir todos os valores inteiros entre 1 (inclusive) e N (inclusive). Considere que o N será sempre *maior que ZERO.*

Exercícios

5. Faça um algoritmo para calcular n! (fatorial de n), sendo que o valor inteiro de n é fornecido pelo usuário.

   Sabendo que:

   n! = 1 * 2 * 3 * ... * (n – 1) * n;

   0! = 1, por definição;

   Negativo! = Não existe

Exercícios

6. Escreva um programa que calcule a média dos números digitados pelo usuário se eles forem pares. Termine a leitura se o usuário digitar 0.

Vetores e Matrizes

- Conjunto de dados de mesmo tipo.
  - Exemplo 1:

```
int[] valores = new int[5];

for (int i = 0; i < 5; i++)
{
    Console.Write("Entre com o " + (i+1) + "º valor: ");
    valores[i] = int.Parse(Console.ReadLine());
}

for (int j = 0; j < 5; j++)
{
    Console.WriteLine(valores[j]);
}
```

Vetores e Matrizes

- Conjunto de dados de mesmo tipo.
  - Exemplo 2:

```
int[,] matriz = new int[2, 4];

string[] nomes = new string[2];
nomes[0] = "Bruno";
nomes[1] = "Manzoli";

for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < 4; j++)
    {
        Console.Write("Entre com o valor da " + (i + 1) + "ª linha da " + (j+1) + "ª coluna: ");
        matriz[i, j] = int.Parse(Console.ReadLine());
    }
}

for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < 4; j++)
    {
        Console.Write(matriz[i,j]);
    }
}
```

Vetores e Matrizes

- Uso do foreach:
  - Para acessar todos os elementos de um array, é possível aplicar o comando foreach.

```
void ImprimeArray(int[] numeros)
{
   foreach (int numero in numeros)
   {
      System.Console.WriteLine(numero);
   }
}
```

- Método Array.Sort:
  - Classificar valores em um vetor.
    - Array.Sort(nome_do_vetor);

Valores aleatórios

- Método que retorna um número inteiro de forma aleatória.
  - Ex.:

```
Random rdm = new Random();

int sorteio = rdm.Next(6);
```

Exercícios

7. Faça um programa que imprima a média de n números (n é um valor lido do teclado) excluindo o menor e o maior deles. Seu programa deve tratar casos em que n < 3 exibindo uma mensagem de erro.

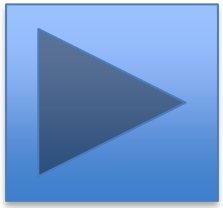
Exercícios

8. A prefeitura de uma cidade fez uma pesquisa com 5 famílias , coletando dados sobre o salário e número de filhos. A prefeitura deseja saber:
   a. Média do salário da população.
   b. Média do número de filhos.
   c. Maior salário.
   d. Percentual de famílias com salário até R$2.000,00.

Obs.: Faça uso de vetores para armazenamento de dados.

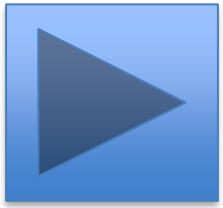
SESI SENAI
Disciplina que gera conhecimento

DESAFIO 01
VALIDADOR DE E-MAIL



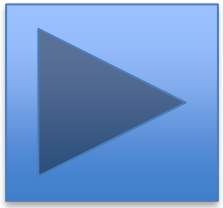
SESI SENAI
Disciplina que gera conhecimento

DESAFIO 02
SORTEIO PARA CONSÓRCIO



SESI SENAI
Disciplina que gera conhecimento

DESAFIO 03
QUADRADO MÁGICO



Orientação a Objetos

- **Domínio e Aplicação:**
  - Um **domínio é composto pelas entidades, informações e processos relacionados a um determinado** contexto. Uma **aplicação pode ser desenvolvida para automatizar ou tornar factível as tarefas** de um domínio. Portanto, uma aplicação é basicamente o “reflexo” de um domínio.

Orientação a Objetos

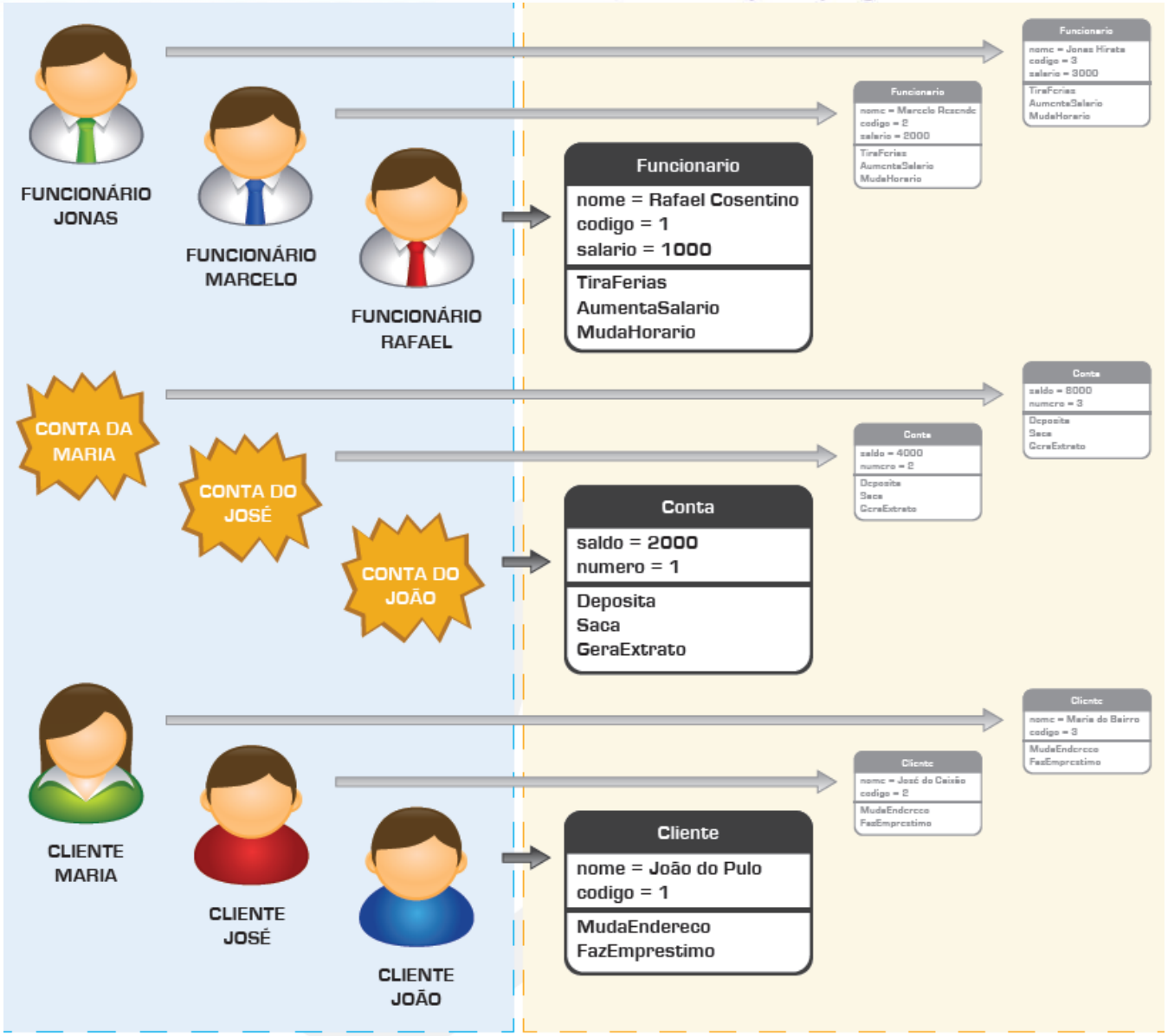
- **Definição:**
  - Programação orientada a objetos é um **paradigma** de programação baseado no conceito de "**objetos**", que podem conter dados na forma de campos, também conhecidos como **atributos**, e códigos, na forma de procedimentos, também conhecidos como **métodos**.

Orientação a Objetos

Orientação a Objetos

- **Objetos, Atributos e Métodos:**
  - As entidades identificadas no domínio devem ser representadas de alguma forma dentro da aplicação correspondente. Nas aplicações orientadas a objetos, as entidades são representadas por **objetos.**
  - Um **atributo** é uma variável que pertence a um objeto. Os dados de um objeto são armazenados nos seus atributos.
  - O próprio objeto deve realizar operações de consulta ou alteração dos valores de seus atributos. Essas operações são definidas nos **métodos** do objeto.

Orientação a Objetos

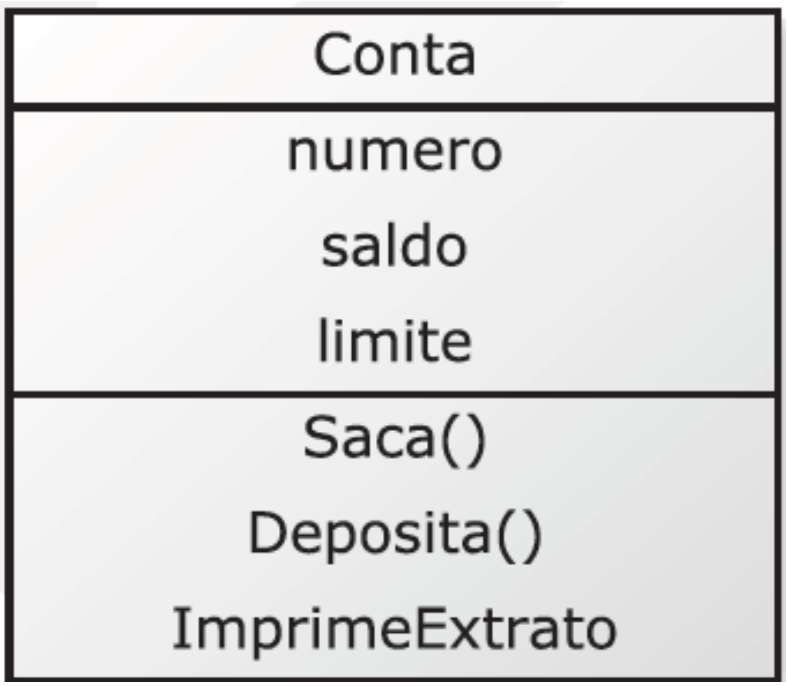


Orientação a Objetos

- **Classes:**
  - Antes de um objeto ser criado, devemos definir quais serão os seus atributos e métodos. Essa definição é realizada através de uma **classe elaborada por um programador. A partir de uma classe,** podemos construir objetos na memória do computador que serão executados por nossa aplicação.

Orientação a Objetos

- **Representação de uma classe através de um diagrama UML:**



Orientação a Objetos

- **Uma Analogia (Classe e Objeto)**

Orientação a Objetos

- **Instanciando um objeto (Referências em C#):**
  - Ao utilizar o comando new, um objeto é alocado em algum lugar da memória. Para que possamos acessar esse objeto, precisamos de sua referência. O comando new devolve a referência do objeto que foi criado.

```
Conta referecia = new Conta();
```

Orientação a Objetos

- **Manipulando Atributos:**

```
Conta referecia = new Conta();

referecia.saldo = 1000.0;
referecia.limite = 500.0;
referecia.numero = 1;

System.Console.WriteLine(referecia.saldo);
System.Console.WriteLine(referecia.limite);
System.Console.WriteLine(referecia.numero);
```

Instanciando uma Classe com um atributo de valor padrão

```
class Conta
{
    public double limite = 500;
}
```

```
class TestaConta
{
  static void Main()
  {
    Conta conta = new Conta();

    // imprime 500
    System.Console.WriteLine(conta.limite);
  }
}
```

Orientação a Objetos

- **Estrutura de uma classe:**
  - Modificadores (public, private ou protected)
    - Encapsulamento
      - Construtores e Destrutores
      - Propriedades
      - Métodos

Orientação a Objetos

- **Valores Padrão:**
  - Os atributos de tipos numéricos são inicializados com 0, os atributos do tipo boolean são inicializados com false e os demais atributos com null (referência vazia).

Exercício de Aprendizagem

- Implemente uma classe para definir os objetos que representarão os clientes de um banco. Essa classe deve declarar dois atributos: um para os nomes e outro para os códigos dos clientes. Faça o cadastro de três clientes e em seguida a impressão dos mesmos na tela.
- Os bancos oferecem aos clientes a possibilidade de obter um cartão de crédito que pode ser utilizados para fazer compras. Um cartão de crédito possui um número e uma data de validade. Crie uma classe para modelar os objetos que representarão os cartões de crédito.

Exercício de Aprendizagem

- As agências do banco possuem número. Crie uma classe para definir os objetos que representarão as agências.
- Faça um teste criando dois objetos da classe Agencia.
- As contas do banco possuem número, saldo e limite. Crie uma classe para definir os objetos que representarão as contas.
- Faça um teste criando dois objetos da classe Conta.
- Altere a classe Conta para que todos os objetos criados a partir dessa classe possuam R$ 100 de limite inicial.

Exercícios Fixação

- Implemente uma classe chamada Aluno **para definir os objetos** que representarão os alunos de uma escola. Essa classe deve declarar três atributos: o primeiro para o nome, o segundo para o RG e o terceiro para a data de nascimento dos alunos.
- Crie dois objetos da classe Aluno atribuindo valores a eles. O sistema deve mostrar na tela as informações desses objetos.

Exercícios Fixação

- Em uma escola, além dos alunos temos os funcionários, que também precisam ser representados em nossa aplicação. Então implemente outra classe chamada **Funcionario** que contenha três atributos: o primeiro para o nome, o segundo para o cargo e o terceiro para o CPF.
- Crie dois objetos da classe Funcionario atribuindo valores a eles. Mostre na tela as informações desses objetos.

Exercícios Fixação

- Em uma escola, os alunos precisam ser divididos por turmas, que devem ser representadas dentro da aplicação. Implemente uma classe turma com quatro atributos: o primeiro para o turno, o segundo para definir a série, o terceiro para sigla e o quarto para o tipo de ensino.
- Criar dois objetos da classe Turma. Adicione informações a eles e depois mostre essas informações na tela.

Relacionamentos

Relacionamentos

```
class Cliente
{
   public string nome;
}
```

```
class CartaoDeCredito
{
    public int numero;
    public string dataDeValidade;
    public Cliente cliente;
}
```

Instanciando os objetos

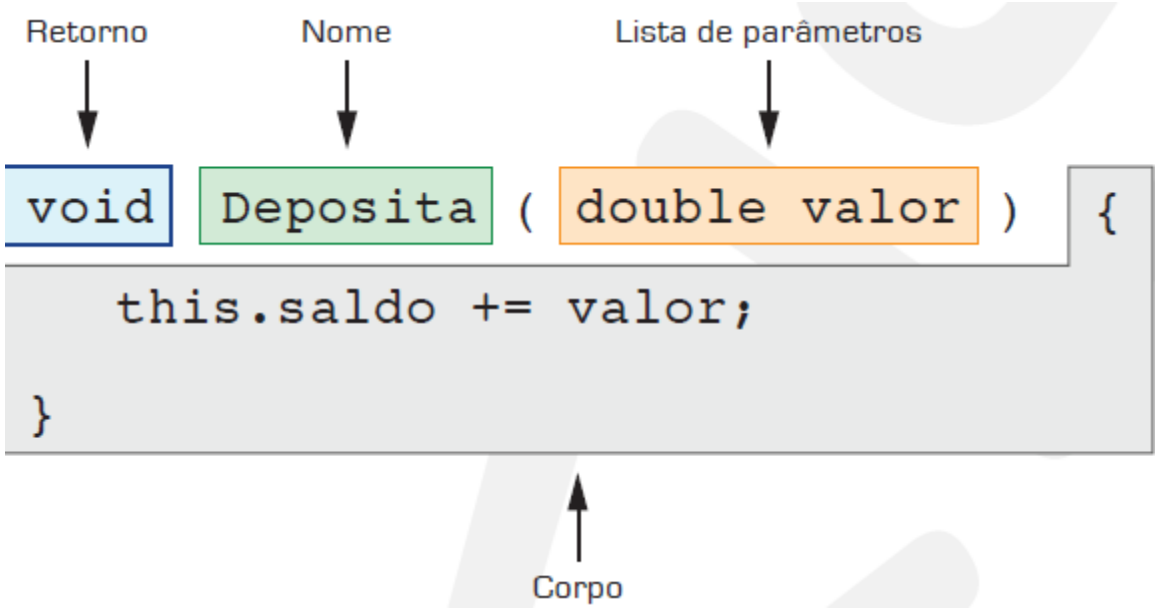
```
// Criando um objeto de cada classe
CartaoDeCredito cdc = new CartaoDeCredito();
Cliente c = new Cliente();

// Ligando os objetos
cdc.cliente = c;

// Acessando o nome do cliente
cdc.cliente.nome = "Rafael Cosentino";
```

Exercícios Fixação

- Defina um vínculo entre os alunos e as turmas, criando na classe Aluno um atributo do tipo Turma.
- Teste o relacionamento entre os alunos e as turmas, criando um objeto de cada classe e atribuindo valores a eles. Exiba na tela os valores que estão nos atributos da turma através do objeto da classe Aluno.

Métodos

- No banco, é possível realizar diversas operações em uma conta: depósito, saque, transferência, consultas e etc. Essas operações podem modificar ou apenas acessar os valores dos atributos dos objetos que representam as contas.



Métodos

- Implementando um método para depósito.

```
void Deposita(double valor)
{
  this.saldo += valor;
}
```

- Realizando um depósito.

```
// Referência de um objeto
Conta c = new Conta();

// Chamando o método Deposita()
c.Deposita(1000);
```

Exercício de Aprendizagem

- Agora, acrescente os demais métodos na classe Conta para realizar as operações de saque, impressão de extrato e consulta do saldo disponível.
- Teste os métodos da classe Conta.

Exercícios de Fixação

- Sabendo que qualquer empresa possui funcionários, crie uma classe chamada **Funcionário** para representá-los. Acrescente os atributos nome e salário a essa classe. Além disso, você deve criar dois métodos: um para aumentar o salário e outro para consultar os dados dos funcionários.
- Teste os métodos implementados na classe Funcionário.

Construtores e Destrutores

```
class CartaoDeCredito
{
    public int numero;

    public CartaoDeCredito(int numero)
    {
        this.numero = numero;
    }
}
```

Exercícios Complementares (lâmpada)

- Desenvolva uma abstração de uma lâmpada, a qual pode ser ligada e desligada. Também deve ser possível observar o estado da lâmpada (se desligada ou ligada).
- Desenvolva um novo tipo de abstração para a lâmpada de forma a incluir as características de potência e voltagem. Garanta que seja possível tanto ler quanto alterar os valores de potência e voltagem de uma lâmpada.

Exercícios Complementares (lâmpada)

- No método Main da classe principal teste as classes desenvolvidas nos exercícios 1 e 2. Crie uma nova lâmpada, apresente no console as informações de estado (se ligada ou desligada, potência e voltagem), ligue a lâmpada e apresente novamente as informações de estado.

Exercícios Complementares (lâmpada)

- Modifique a abstração da lâmpada criada anteriormente para incluir o caso de uma lâmpada queimar ao ser ligada. Sabe-se que existe uma chance de 15% da lâmpada queimar ao ser ligada. Dica: neste exercício é importante pesquisar na biblioteca de classes fornecida pela linguagem de programação uma classe que dê suporte à geração de números aleatórios. (melhorias: incluir o teste de lâmpada queimada dentro do método ligar)

Exercícios Complementares (ponto)

- Crie uma classe Ponto, com a capacidade de armazenar a localização de ponto no plano cartesiano, de modo que ela possua três construtores: um construtor sem parâmetros, que cria um ponto nas coordenadas (0,0), um construtor que recebe dois parâmetros de coordenadas X e Y, e um construtor que inicializa o ponto através das coordenadas de um outro ponto recebido como parâmetro.

Exercícios Complementares (ponto)

- Seja a classe Ponto, implementada, com a capacidade de armazenar a localização de um ponto no plano cartesiano. Adicionalmente, deseja-se que esta classe seja capaz de calcular a distância entre dois pontos. Para tal é desejado o seguinte comportamento:
  - calcular a distância entre dois objetos ponto passados como parâmetro;
  - a fórmula a ser utilizada é

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Exercícios Complementares (Academia)

- Implemente uma classe Pessoa que possua como atributos nome, idade, peso (em quilogramas) e altura (em metros). Faça com que os dados sejam inicializados através do construtor da classe. Adicione métodos para ler e alterar cada um dos atributos em separado.

Exercícios Complementares (Academia)

- Altere a classe Pessoa do exercício anterior de modo que ela seja capaz de calcular o Índice de Massa Corporal (IMC). O cálculo é feito através da fórmula IMC = peso/altura^2. Adicionalmente, implemente um método que informa a faixa de categoria do IMC que a pessoa se encontra, utilizando a seguinte tabela.

IMC	Categoria
< 18,5	Abaixo do peso
[18,5 ; 25[	Peso normal
[25 ; 30[	Sobrepeso
[30 ; 35[	Obesidade grau I
[35; 40[	Obesidade grau II
>= 40	Obesidade grau III

Exercícios Complementares (Elevador)

- Crie uma classe denominada Elevador para armazenar as informações de um elevador dentro de um prédio. A classe deve armazenar o andar atual através de um número inteiro (sendo o térreo igual a 0), total de andares no prédio, excluindo o térreo, capacidade do elevador e quantas pessoas estão presentes nele. A classe deve também disponibilizar os seguintes métodos:

Exercícios Complementares (Elevador)

- **Construtor**: Este método deve receber como parâmetros a capacidade do elevador e o total de andares no prédio (o elevador sempre começa no térreo e vazio);
- **Entrar**: Este método é utilizado para acrescentar uma quantidade de pessoas no elevador. O método também deve verificar se existe espaço disponível no elevador;
- **Sair**: Este método remove uma quantidade de pessoas do elevador. O método deve verificar se existe a quantidade de pessoas solicitada para sair;
- **Subir e Descer**: Estes métodos serão utilizados para o controle interno do deslocamento do elevador. Ambos devem receber a quantidade de andares que o elevador terá que subir ou descer.
- **Deslocar**: Este método será utilizado para escolher o andar de destino do elevador. Além disso, deve se preocupar com a ação que será realizada para chegar ao andar desejado. Os métodos Subir e Descer devem ser utilizados;

Exercícios Complementares (Calculadora)

- Crie uma classe chamada Calculadora, que possua métodos para realizar as 4 operações básicas com números decimais. Cada um dos quatro métodos da classe deve apenas retornar o valor da operação realizada, recebendo apenas 2 números como parâmetros.

Exercícios Complementares (Calculadora)

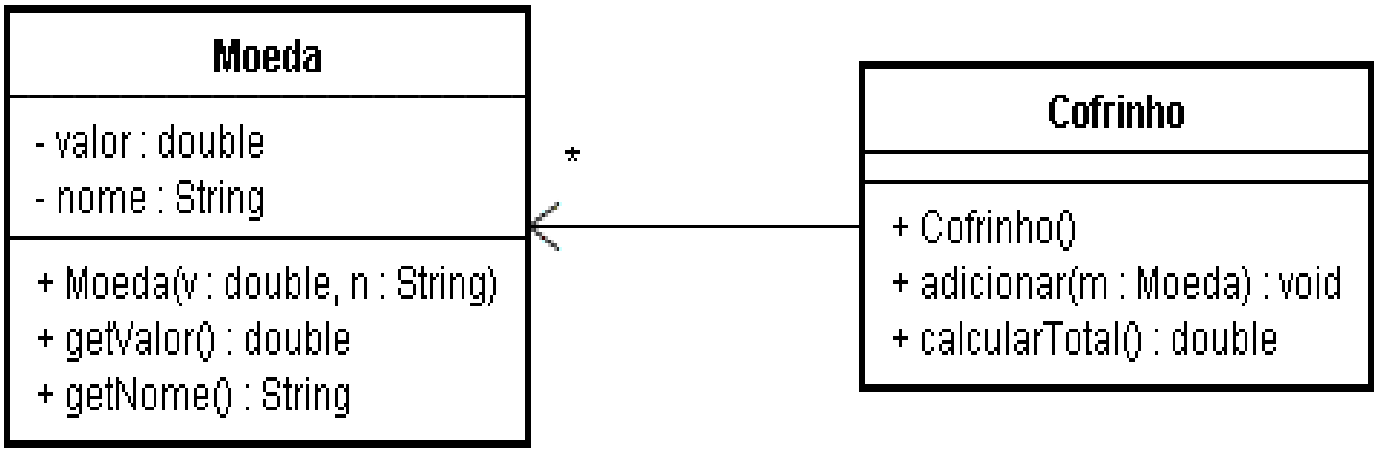
- Adicione um método na classe Calculadora, que seja capaz de realizar operações de potência entre 2 números inteiros. Considerando que o método tome como parâmetro X e Y , o resultado deverá ser igual a X elevado a Y . Por exemplo, para X = 2 e Y = 3, o resultado será 2 elevado a 3, que é o mesmo que 2 x 2 x 2. Para cálculo de potência utilize o método Math.Pow.

Exercícios Complementares (Calculadora)

- Adicione um método na classe Calculadora, para calcular a fórmula de Báskara. Esse método deve receber como parâmetro, 3 números decimais representando a, b e c na equação de segundo grau.

Exercícios Complementares (cofrinho)

- Deseja-se implementar um cofrinho de moedas com a capacidade de receber moedas e calcular o total depositado no cofrinho. O seguinte diagrama UML apresenta o projeto da solução desejada.



Exercícios Complementares (cofrinho)

- Altere a classe Cofrinho do exercício anterior de modo que ela implemente métodos para:
  - Contar o número de moedas armazenadas;
  - Contar o número de moedas de um determinado valor;
  - Informar qual a moeda de maior valor.

Exercícios Complementares (Matriz)

- Escreva uma classe chamada “MatrizDeInteiros” que tenha como atributo uma matriz de inteiros e um construtor que receba como parâmetro a ordem da matriz, a instancie e inicialize com zeros. Acrescente à classe os seguintes métodos:

a) Um método que receba como parâmetro três números inteiros indicando respectivamente linha, coluna e o valor que deve ser armazenado na linha e coluna indicada.

   Obs: Caso a linha ou a coluna passadas como parâmetro estejam fora da ordem da matriz indique com uma mensagem o erro

Exercícios Complementares (Matriz)

- Escreva uma classe chamada “MatrizDeInteiros” que tenha como atributo uma matriz de inteiros e um construtor que receba como parâmetro a ordem da matriz, a instancie e inicialize com zeros. Acrescente a classe os seguintes métodos:

b) Um método “eQuadrada”, que retorna true se a matriz for quadrada (isto é, tem o mesmo número de linhas e colunas).

c) Um método total que some todos os valores da matriz retornando o resultado.

d) Um método que receba como parâmetro um determinado valor e retorne a linha onde o elemento foi encontrado na matriz ou –1 caso contrário

Exercícios Complementares (sistema de informação)

- Seja a seguinte descrição de um sistema de informação: “Deseja-se criar um sistema de estoque de produtos que são vendidos em um supermercado. Cada produto possui uma descrição e um valor de venda. O sistema permite a emissão de relatórios dos produtos disponíveis em estoque. Também, é permitido ao gerente aplicar reajustes de preços sobre o produto que desejar.”

Exercícios Complementares (sistema de informação)

- Para a classe Produto, quais seriam seus métodos e atributos?
- Para a classe Estoque, quais seriam seus métodos e atributos?
- Pense em um pequeno sistema de informação. Descreva brevemente algumas funcionalidades dele. Entregue a descrição à outra pessoa e solicite que ela descreva um conjunto básico de classes (com seus métodos e atributos) que darão suporte à implementação orientada a objetos do sistema.

Tratamento de Exceções

- **Uso da estrutura Try - Catch**

```
static void Main(string[] args)
{
    double num1, num2, media;
    try
    {
        Console.Write("Entre com o primeiro valor: ");
        num1 = double.Parse(Console.ReadLine());
        Console.Write("Entre com o segundo valor: ");
        num2 = double.Parse(Console.ReadLine());
        media = (num1 + num2) / 2;
        Console.WriteLine("A média dos valore informados é: " + media);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
    Console.ReadKey();
}
```