

Programação de Aplicativos

Framework .NET – C#

Instrutor Bruno Manzoli

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.

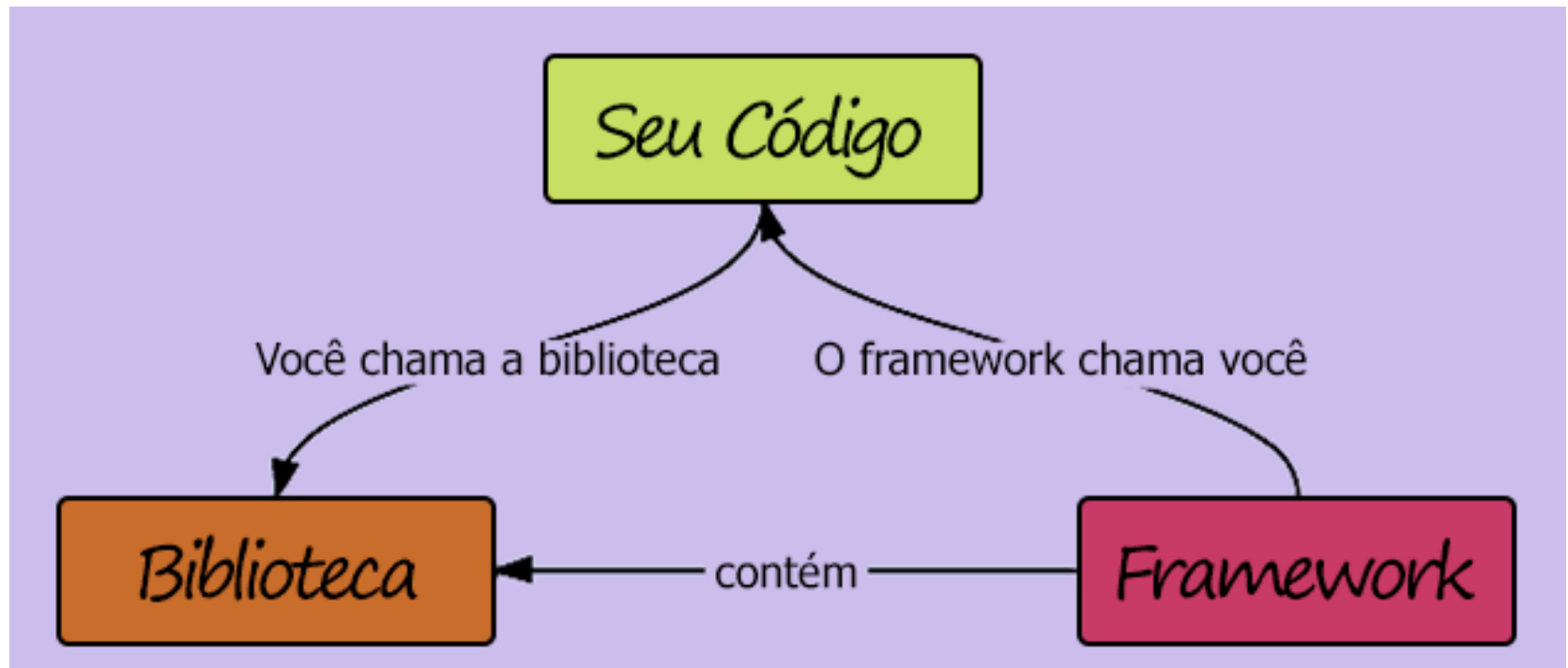
- Sistema avaliativo:
 - ✓ Duas provas práticas
 - ✓ Uma prova teórica
 - ✓ Um trabalho individual
 - ✓ Um trabalho em grupo



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).

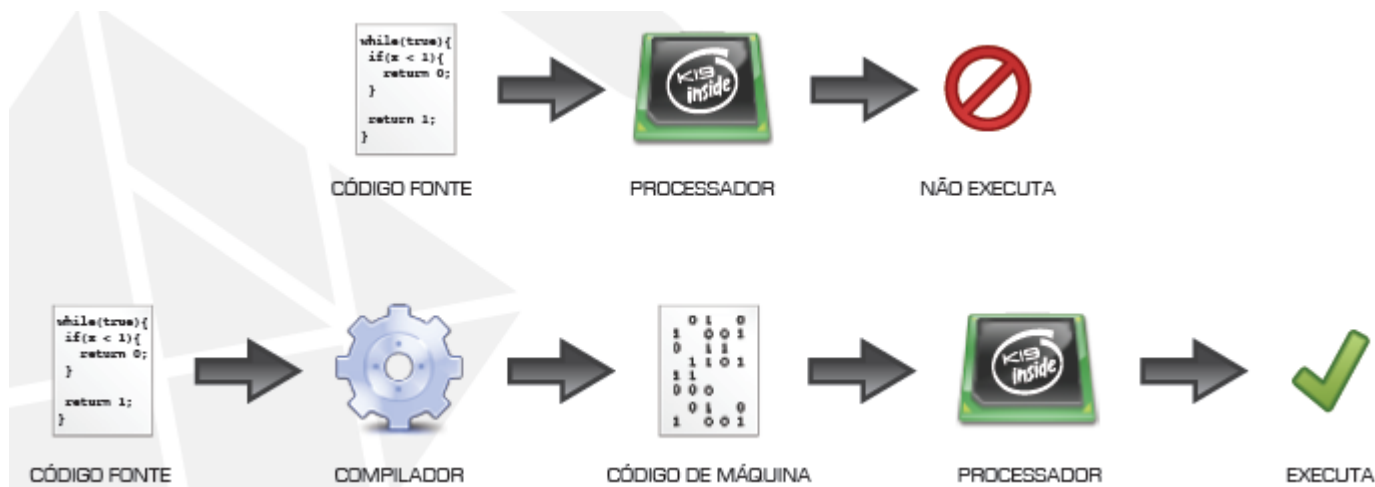
PLATAFORMA .NET

LINGUAGEM DE PROGRAMAÇÃO
ORIENTADA A OBJETOS

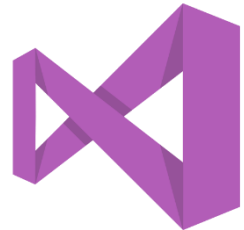
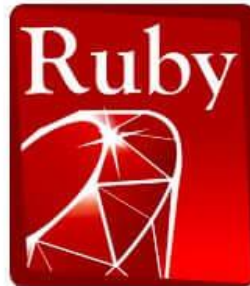
AMBIENTE DE EXECUÇÃO

O que é um Programa?

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



Para onde ir? Por onde começar?

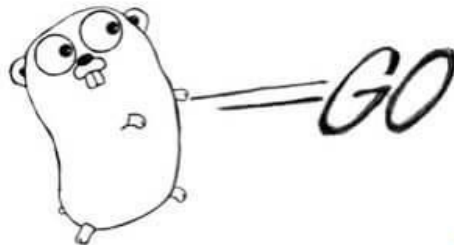


Objective-C



Perl

C++



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:

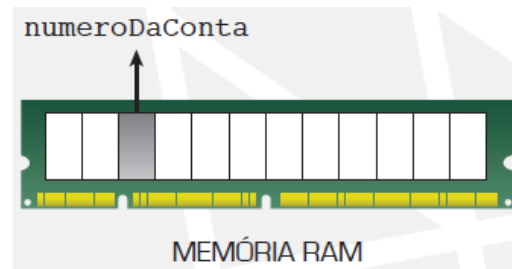
```
using System;

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

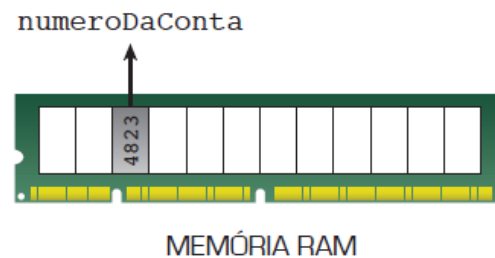
Variáveis

- 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 variável precisa ser manipulada pelo programa.**

Variáveis - Exemplo



`numeroDaConta = 4823`



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

<i>Tipo</i>	<i>Descrição</i>	<i>Tamanho</i>
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

<i>Tipo</i>	<i>Descrição</i>	<i>Tamanho</i>
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 $1,40129846432481707 \times 10^{-45}$ e $3,40282346638528860 \times 10^{38}$ (positivo ou negativo) – Precisão de 7 dígitos nas casas decimais. Atribuição direta com uso da letra “f”.	4 bytes

Variáveis - Tipos Primitivos

<i>Tipo</i>	<i>Descrição</i>	<i>Tamanho</i>
double	Valor com ponto flutuante entre $4,94065645841246544 \times 10^{-324}$ e $1,79769313486231570 \times 10^{308}$ (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 $1,0 \times 10^{-28}$ e $7,9 \times 10^{28}$ (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:

$$n! = 1 * 2 * 3 * \dots * (n - 1) * n;$$

$0! = 1$, 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.WriteLine("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.WriteLine(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.

DESAFIO 01

VALIDADOR DE E-MAIL



DESAFIO 02

SORTEIO PARA CONSÓRCIO



DESAFIO 03

QUADRADO MÁGICO



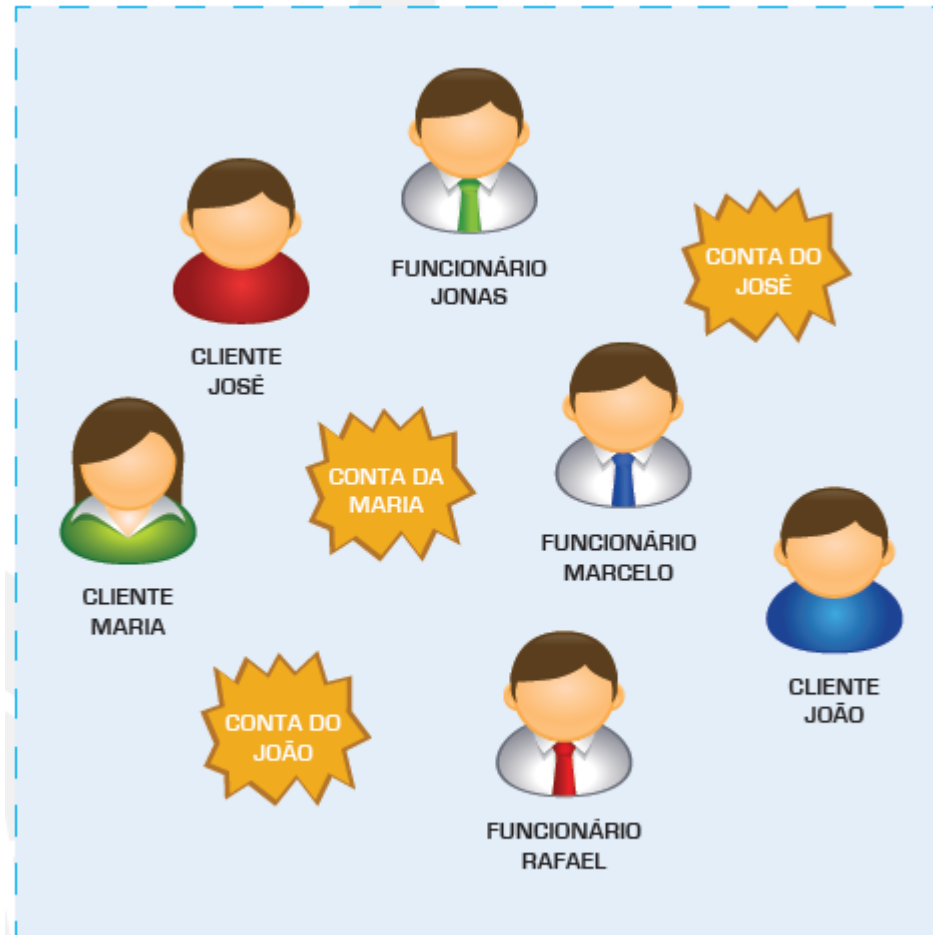
Orientação a Objetos

- **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



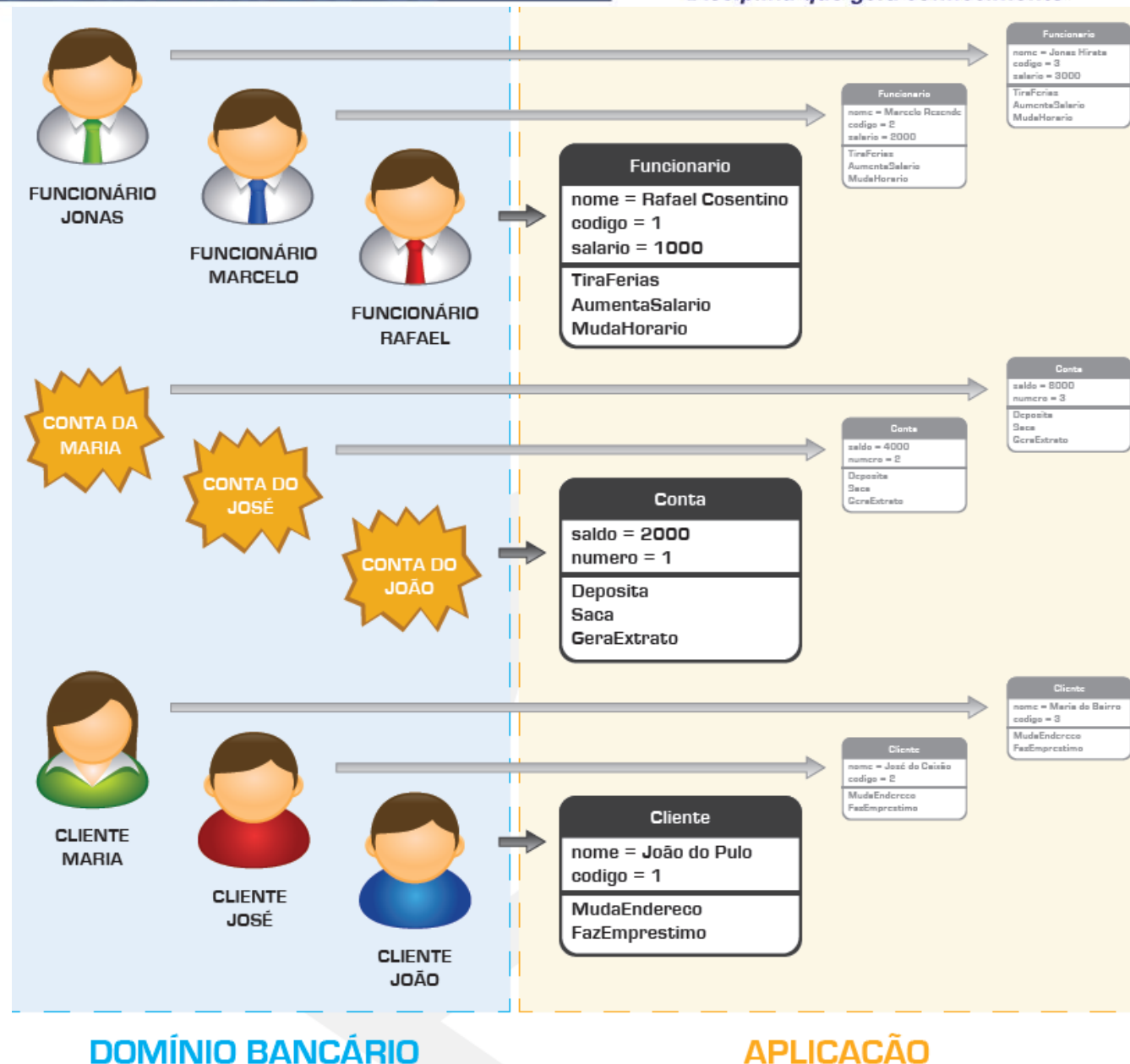
DOMÍNIO BANCÁRIO

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

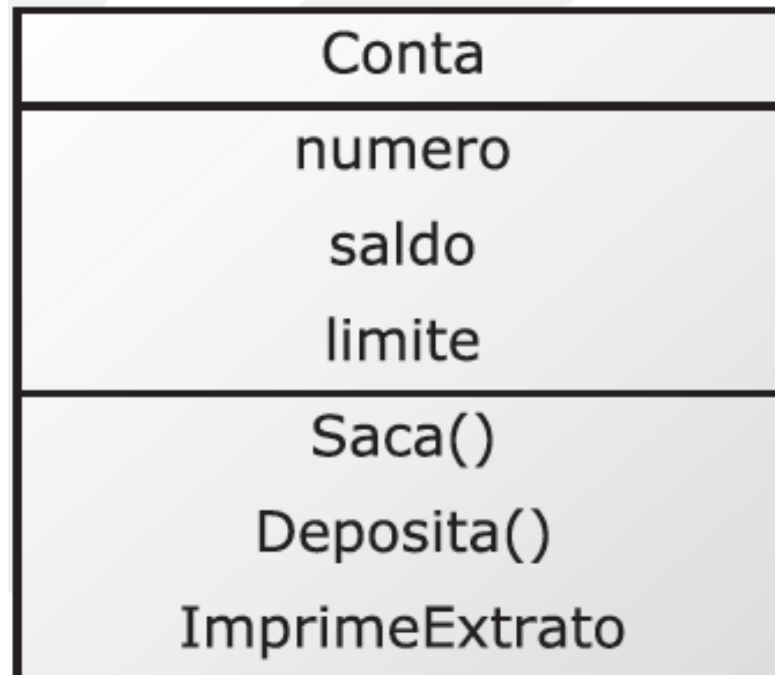


Orientação a Objetos

- **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 referencia = new Conta();
```

Orientação a Objetos

- **Manipulando Atributos:**

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

```
referencia.saldo = 1000.0;
```

```
referencia.limite = 500.0;
```

```
referencia.numero = 1;
```

```
System.Console.WriteLine(referencia.saldo);
```

```
System.Console.WriteLine(referencia.limite);
```

```
Svstem.Console.WriteLine(referencia.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.