



## UNIDADE II

---

### Paradigmas de Linguagens

Prof. Me. Luiz Forçan

# Paradigma de Programação Estruturada

- Um paradigma de programação é uma metodologia que oferece a visão que o programador possui sobre a estrutura e a execução do programa.
- Por exemplo, na programação orientada a objetos, os programadores podem abstrair um programa como uma coleção de objetos que interagem entre si, enquanto na programação estruturada os programadores abstraem o programa como uma sequência de funções executadas de modo empilhado.
- Da mesma forma que os engenheiros de *software* propõem metodologias diferentes para o desenvolvimento de sistemas; diferentes linguagens possuem diferentes paradigmas de programação.
  - Os paradigmas são definidos, muitas vezes, pelo que permitem ou não permitem que uma linguagem realize.
  - Por exemplo, a linguagem Pascal é uma linguagem estruturada e, por esse motivo, não permite a criação de objetos, característica exclusiva de linguagens que seguem o paradigma orientado a objetos.

# Paradigma de Programação Estruturada

Como era a programação de computadores antes da programação estruturada?

- Antes da programação estruturada, a programação era feita com o uso de desvios incondicionais no fluxo de execução das instruções;
- Dependendo do tamanho do programa, essa abordagem podia tornar muito difícil a manutenção do programa para a correção de erros ou para a simples evolução do programa para atender aos requisitos do usuário;
- A programação linear era a técnica usada, nessa época, e fazia uso de desvios incondicionais para construir a lógica do programa;
  - A instrução GOTO era usada para implementar tais desvios incondicionais e o seu uso foi descontinuado com o aparecimento da programação estruturada;
  - Diversas discussões podem ser encontradas na literatura a respeito de saltos incondicionais;
  - Grande parte das opiniões relatam que os saltos não são considerados uma boa prática de programação.

# Exemplo de utilização do comando GOTO em VB

```
1. Sub ExemploUsoComandoGoTo()  
2. Dim numeroEntrada As Integer = 1  
3. Dim retornaTexto As String  
4. ' Avalia o número informado na variável  
   numeroEntrada e se este número for 1 segue  
   pelo fluxo 1, senão pelo fluxo 2  
5. If numeroEntrada = 1 Then GoTo Fluxo1  
   Else GoTo Fluxo2  
6. Fluxo1:  
7. retornaTexto = "O número informado é  
   igual a 1"  
8. GoTo SegueFluxoPrincipal  
9. Fluxo2:  
10. retornaTexto = "O número informado é  
   diferente de 1"  
11. GoTo SegueFluxoPrincipal  
12. SegueFluxoPrincipal:  
13. ' Imprime o valor da variável  
   retornaTexto na janela de Depuração.  
14. Debug.WriteLine(retornaTexto)  
15. End Sub
```

# Paradigma de Programação Estruturada

- Antes de decidir qual linguagem ou paradigma usar para resolver um problema, é necessário ter o conhecimento da forma como irá resolver o problema, ou seja, criar o algoritmo, que é independente de linguagem de programação.
- O algoritmo possui a lógica de tudo o que o programa deverá fazer.
- A partir daí, a linguagem poderá ser escolhida de acordo com a afinidade ou o gosto do programador.
  - O paradigma estruturado foi um dos principais modelos de programação desenvolvidos e, até os dias atuais, é muito utilizado para a construção de programas de computadores.

# Paradigma de Programação Estruturada

Programação estruturada é uma forma de programação de computadores que preconiza que todos os programas possíveis podem ser reduzidos a apenas três estruturas:

- Sequência (estrutura de controle Sequencial);
  - Decisão (estrutura de controle condicional);
  - Iteração ou repetição (estrutura de controle repetitiva).
- 
- O fluxo de execução do programa é evidente; não são admitidos saltos condicionais (comando GOTO).

# Estrutura sequencial

- Os comandos são escritos na sequência e executados.

São as operações básicas que um computador pode realizar, como, por exemplo:

- Operações de atribuição. Exemplo: `int x = 1;`

`variavelNumerica = 7;`

`variavelTexto = "Luiz Roberto";`

`variavelFaturamentoMedio = variavelFaturamentoTotal / qdeProdutosVendidos;`

operações aritméticas. Exemplo: `int y = 2 + 2;`

- Operações de entrada e saída de informações ou dados.  
Exemplo de trecho de código:

```
int main (void )
```

```
{
```

```
    int numero;
```

```
    printf ("Digite um número de 1 a 3: ");
```

```
    scanf("%d", &numero);
```

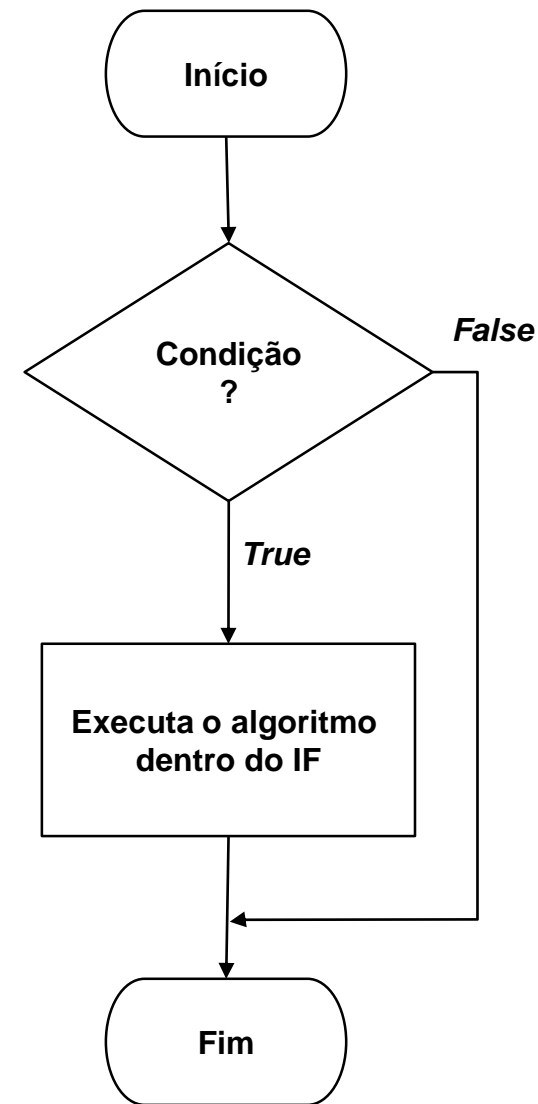
```
    ...
```

# Estrutura condicional

É utilizada para a tomada de decisão ao longo do fluxo de execução do programa, conforme o exemplo em C#:

```
int mediaAluno;  
if (mediaAluno >= 7);  
    Console.WriteLine("Aluno aprovado com média maior ou igual 7")  
else;  
    Console.WriteLine("Aluno reprovado na disciplina").
```

Todas as LPs dão suporte à estrutura condicional, por meio da instrução que é chamada de IF, que se utiliza de uma expressão booleana para decidir se determinado trecho de código deverá ou não ser executado, conforme a figura ao lado:





# Estrutura de repetição

- As estruturas de repetição, também conhecidas como estruturas iterativas ou de recursão, permitem aos desenvolvedores construir os ciclos ou os laços de repetição no fluxo de controle do programa, permitindo que um bloco de instruções ou uma instrução seja executada a zero, uma ou mais vezes, mediante à satisfação de alguma condição lógica.

É empregada em trechos do código que necessitam de execuções constantes, conforme o exemplo em C# na figura ao lado:

```
while (mediaAluno >=7)
    Console.WriteLine("Média do
aluno maior ou igual a 7")
}
```



# Paradigma estruturado – Procedimentos e funções

- Este paradigma possui uma estrutura organizada em módulos ou subprogramas, que são denominados de procedimentos ou funções.

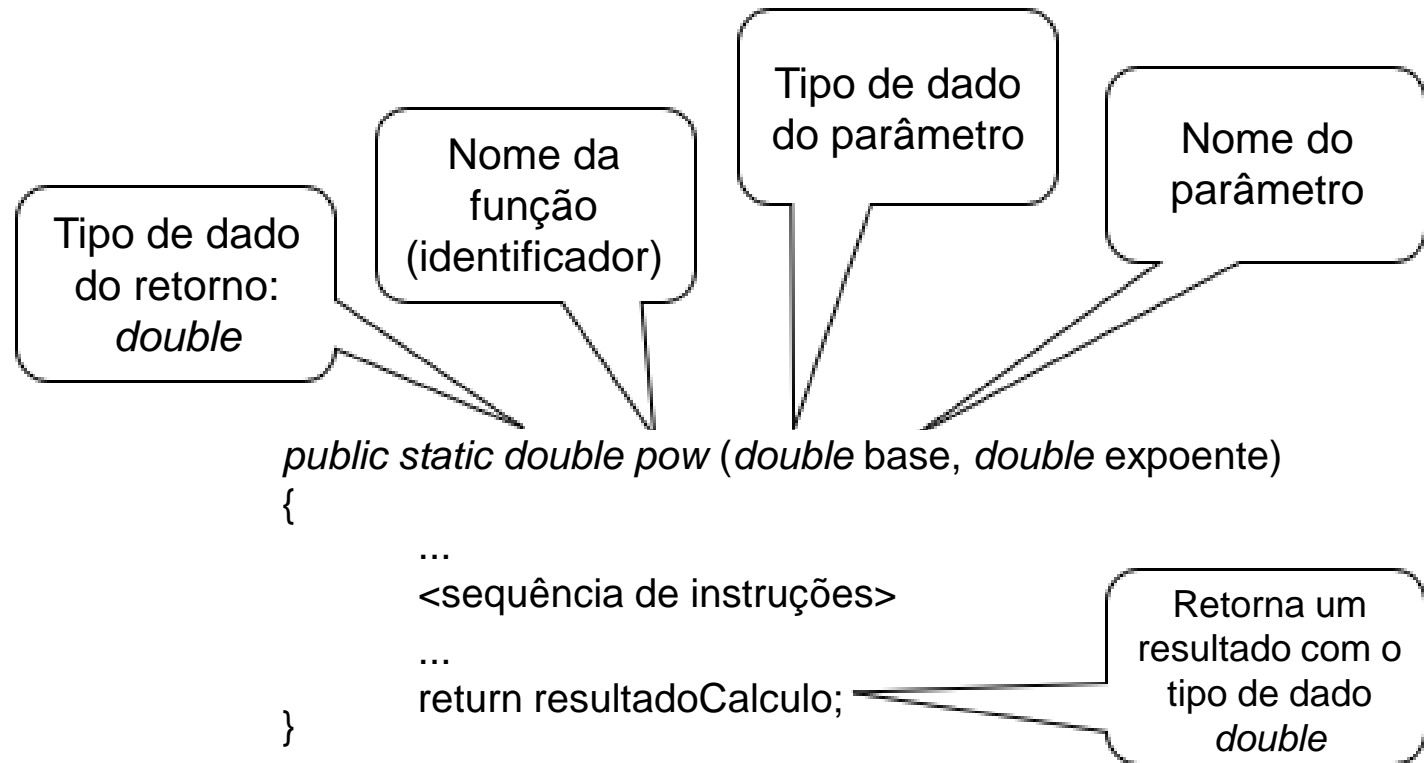
## Função:

- A diferença entre a função e o procedimento está no fato de que uma função sempre deve retornar um resultado para a unidade chamadora;
- As funções são constituídas por uma sequência de instruções para executar uma tarefa específica, porém, no final, retornam um valor ou resultado para o programa principal;
- Por exemplo, uma função pode fazer um cálculo e retornar o seu resultado.

# Procedimentos e funções

- A função *pow*, que calcula o valor de um número elevado a uma potência, recebe dois parâmetros e retorna um valor com o tipo de dado *double*.

```
public class ExemploRetornoFuncao {  
    System.out.println("Calcula a potência de um número: " + Math.pow(3, 2));  
}
```



Fonte: autoria própria.

# Procedimentos e funções

- Este paradigma possui uma estrutura organizada em módulos ou subprogramas, que são denominados de procedimentos ou funções.

## Procedimento:

- Um procedimento é um subprograma que executa determinada tarefa e não retorna nenhuma informação para a unidade chamadora.

Exemplo de procedimento:

```
public static void main(String args[]) {  
    System.out.println("Exemplo simples de procedimento");  
}
```

# Interatividade

Sobre o paradigma estruturado, considere as seguintes afirmações:

- I. Os *softwares*, neste paradigma, devem ser desenvolvidos utilizando apenas três estruturas essenciais: sequencial, condicional e repetição.
- II. Os códigos-fontes dos programas estruturados são organizados utilizando-se módulos ou subprogramas.
- III. O estado de um programa representa a situação das variáveis em determinado instante de tempo no fluxo de execução do programa.

Assinale a alternativa correta:

- a) Apenas a alternativa I está correta.
- b) Apenas as alternativas II e III estão corretas.
- c) Apenas as alternativas I e II estão corretas.
- d) Apenas as alternativas I e III estão corretas.
- e) Todas as alternativas estão corretas.

# Resposta

Sobre o paradigma estruturado, considere as seguintes afirmações:

- I. Os *softwares*, neste paradigma, devem ser desenvolvidos utilizando apenas três estruturas essenciais: sequencial, condicional e repetição.
- II. Os códigos-fontes dos programas estruturados são organizados utilizando-se módulos ou subprogramas.
- III. O estado de um programa representa a situação das variáveis em determinado instante de tempo no fluxo de execução do programa.

Assinale a alternativa correta:

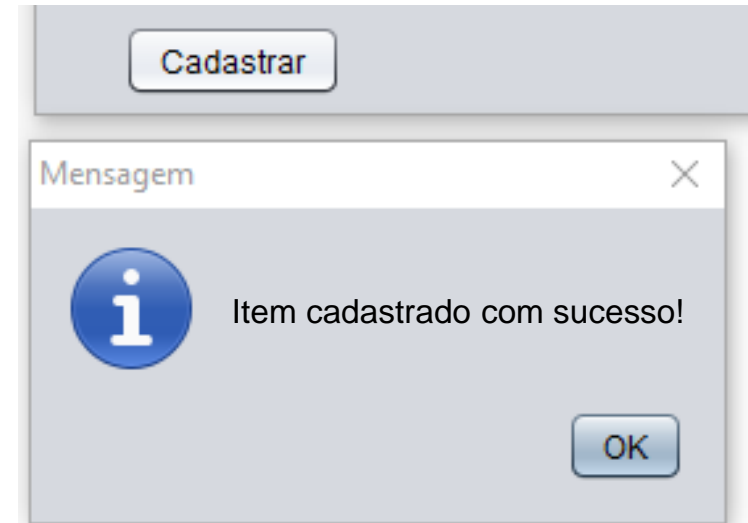
- a) Apenas a alternativa I está correta.
- b) Apenas as alternativas II e III estão corretas.
- c) Apenas as alternativas I e II estão corretas.
- d) Apenas as alternativas I e III estão corretas.
- e) Todas as alternativas estão corretas.

# Programação Orientada a Eventos

- O paradigma de Programação Orientada a Eventos tem adquirido grande importância devido ao crescimento exponencial da utilização de aplicações *mobile* e *web*, em que o usuário, para interagir com estas interfaces, faz uso de toques em botões, digita os textos e seleciona as diversas opções nos *menus* da aplicação.
- O Paradigma de Programação Orientado a Eventos é um modelo de desenvolvimento de *software* em que as partes do programa são executadas em momentos inesperados, em que não se controla a sequência na qual ocorrem os eventos de entrada de dados.
  - Normalmente, estes eventos de entrada são disparados por interações do usuário, por meio de componentes gráficos ou objetos, quando o *software* está em execução (SEBESTA, 2011).

# Programação Orientada a Eventos

- No paradigma imperativo tradicional, o desenvolvedor de *software* é quem determina a sequência de comandos que alteram o estado atual do sistema, desde a entrada dos dados, até atingir um estado final.
- No paradigma orientado a eventos, as ações do programa são estruturadas através de eventos, que podem ser programados pelo desenvolvedor para atender aos requisitos de *software*, criando, assim, os procedimentos específicos para o programa.
  - Para responder aos eventos dos objetos, o desenvolvedor precisa implementar os métodos manipuladores chamados de *handlers* de evento.
  - Estes métodos especiais são utilizados como tarefas para responder aos eventos detectados pelos *listeners*.



Fonte: autoria própria.



# Programação Orientada a Eventos – Exemplos

As aplicações desenvolvidas para a internet também têm controles acionados por eventos, como, por exemplo:

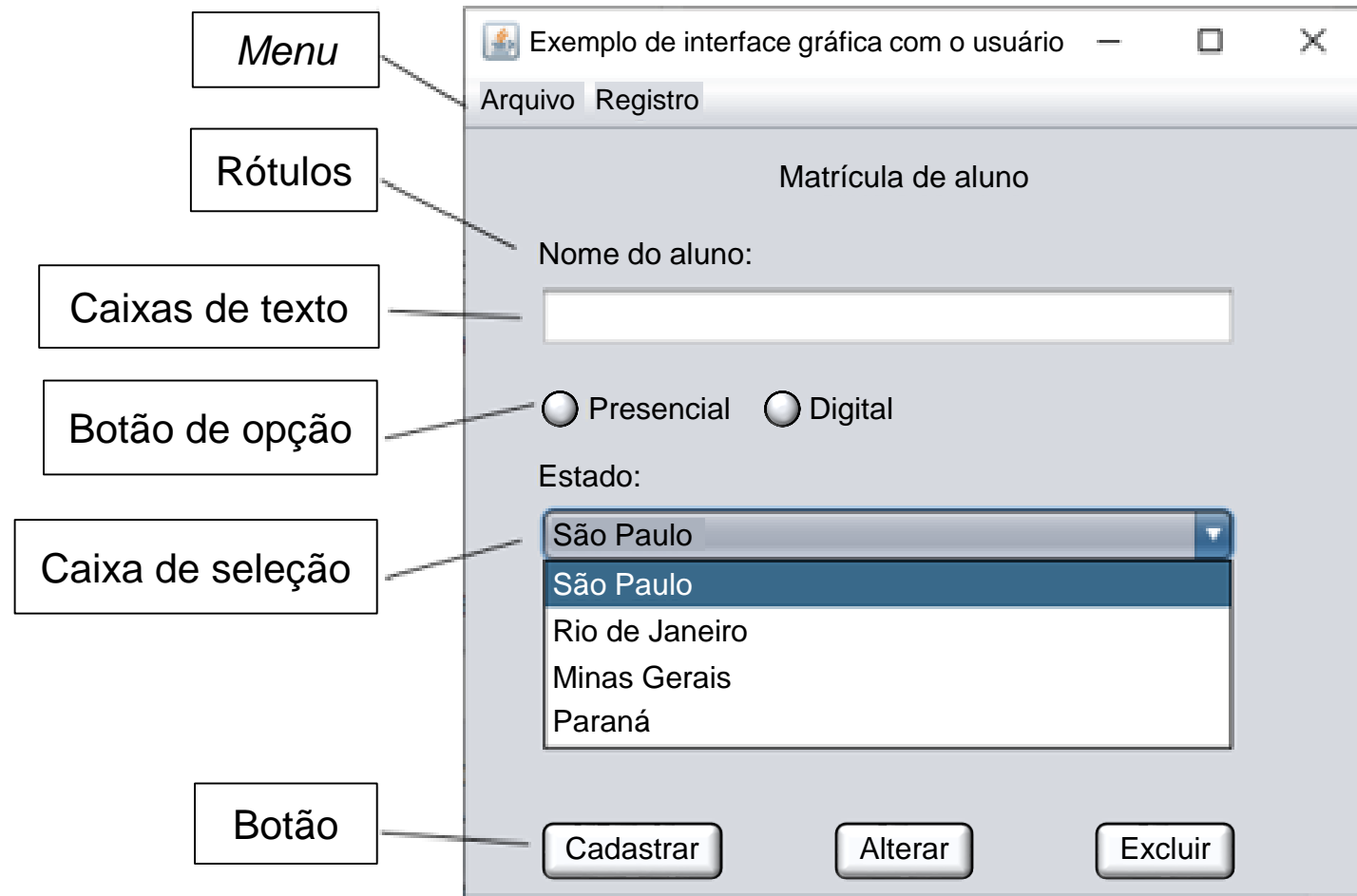
- Um sistema de registro de estudante *on-line* deve ser preparado para interagir com o usuário, independentemente de qual seja a sua próxima ação: cadastrar um curso, alterar os dados de um curso, escolher em qual classe vai assistir às suas aulas;
- Um outro exemplo seria um sistema de reservas *web* de uma companhia aérea que também deve estar preparado para responder a várias sequências de eventos de um usuário, como selecionar a data da viagem, o destino preferido ou o local da poltrona dentro do avião.

São alguns exemplos de eventos:

- Toda vez que o usuário movimenta o *mouse*;
- Aperta um botão do *mouse*;
- Aciona uma tecla no teclado do computador;
- Digita em uma caixa de texto;
- Seleciona um item no *menu*.

# Programação Orientada a Eventos – Interface gráfica com o usuário (GUI)

- Um evento é uma notificação de que alguma coisa aconteceu.
- Então o desenvolvedor de *software* pode, por exemplo, utilizar esses eventos para elaborar um programa cujo processamento ocorra a partir da sua execução.



Fonte: autoria própria.

# Programação Orientada a Eventos – Tipos de eventos disponíveis

Cada componente visual tem um conjunto de eventos disponíveis que pode ser utilizado pelo programador para estabelecer o fluxo do programa, de acordo com os seus objetivos de desenvolvimento. Especificamos, a seguir, alguns dos mais utilizados:

- **Eventos da Janela (WindowListener):** são disparados quando uma janela pode ser aberta (*Open*), fechada (*Close*), maximizada (*maximize*), minimizada (*minimize*) ou alterada de tamanho (*Resize*);
- **Eventos de Teclado (KeyListener):** são disparados quando uma tecla é pressionada ou é solta: (*Down*, *Press*, *Up*);
  - **Eventos do Mouse (MouseListener, MouseMotionListener):** são disparados por cliques do *mouse* ou por movimentos de *mouse*, como, por exemplo, o *mouse* entrando ou saindo de determinada área de um componente: Click, MouseHover, MouseLeave, MouseMove.

# Programação Orientada a Eventos – Tipos de eventos disponíveis

Alguns exemplos de eventos na linguagem Java:

- Por exemplo, a interface `KeyListener` que fica ouvindo se alguma tecla do teclado foi pressionada.

<i>Event</i>	<i>Interface Listener</i>	<i>Método</i>
WindowEvent	WindowListener	<code>windowClosing(WindowEvent e)</code> <code>windowClosed(WindowEvent e)</code> <code>windowOpened(WindowEvent e)</code> <code>windowIconified(WindowEvent e)</code> <code>windowDeiconified(WindowEvent e)</code> <code>windowActivated(WindowEvent e)</code> <code>windowDeactivated(WindowEvent e)</code>
KeyEvent	KeyListener	<code>keyPressed(KeyEvent event)</code> <code>keyReleased(KeyEvent event)</code> <code>keyTyped(KeyEvent event)</code>
MouseEvent	MouseListener	<code>mouseClicked(MouseEvent event)</code> <code>mouseEntered(MouseEvent event)</code> <code>mouseExited(MouseEvent event)</code> <code>mousePressed(MouseEvent event)</code> <code>mouseReleased(MouseEvent event)</code>

# Vantagens do Paradigma Orientado a Eventos

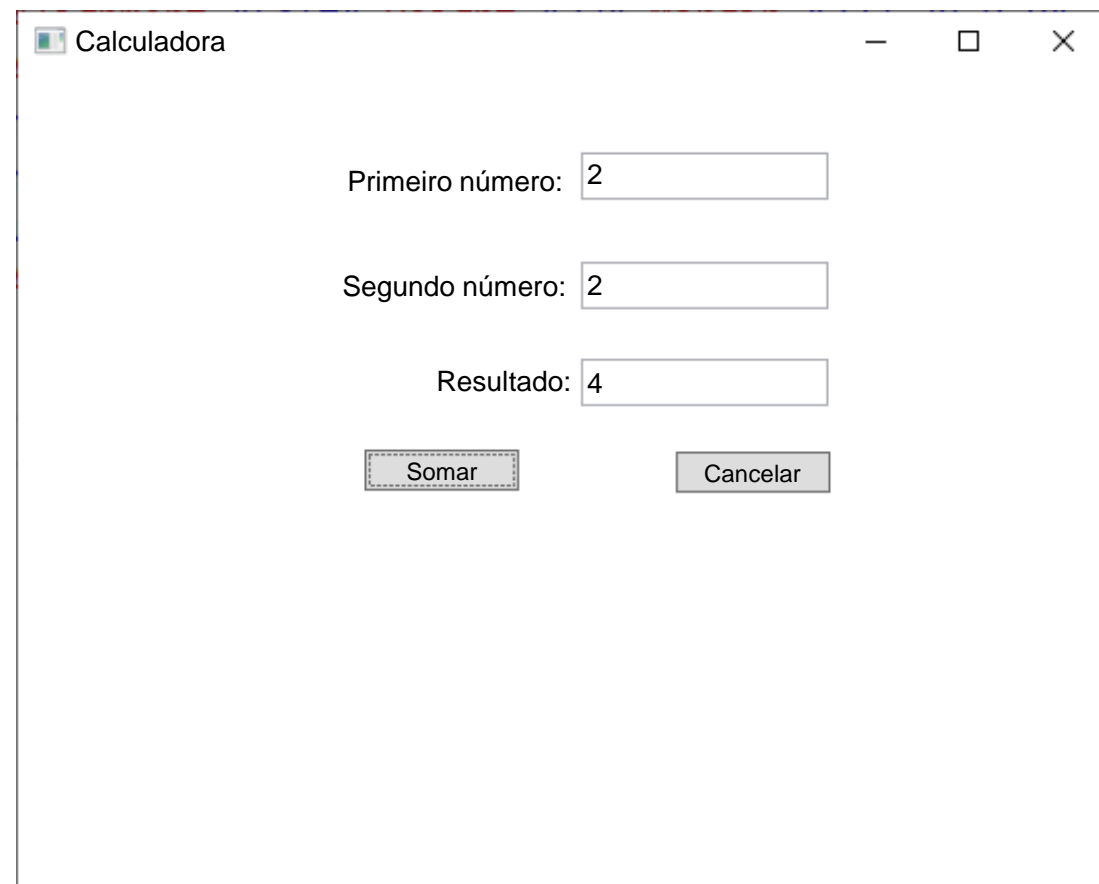
- Paradigma adequado para trabalhar com as interfaces gráficas, como *web* e *mobile*: permite que o usuário utilize a interface gráfica para disparar uma atividade sem que o sistema saiba, previamente, o momento da sua execução. Um exemplo é o envio de *e-mails* na interface do programa MS-Outlook.

 Enviar	Para	teste@teste.com.br
	Cc	
	Cco	
	Assunto	Teste

Fonte: autoria própria.

# Vantagens do Paradigma Orientado a Eventos

- O paradigma de programação orientado a eventos é amplamente utilizado em interfaces gráficas, programação *mobile* e jogos.
- O programa pode executar as ações utilizando pouco código-fonte, através dos eventos, como, por exemplo: em formulários, caixas de texto e botões.
- Também é possível utilizar diversos componentes e controles, inclusive de terceiros, e utilizá-los nos formulários.



# Vantagens do Paradigma Orientado a Eventos

- A programação visual exige menos esforço para o desenvolvedor, pois as aplicações são criadas a partir de janelas gráficas, com a utilização de ferramentas fornecidas pelas IDEs para o *design* de telas.

Cadastro de vendedor

Nome	Peterson Foca
Endereço	Rua A, 45
E-mail	petfoca@email.com
Telefone	119707070

Vendedor cadastrado com sucesso

OK

Cadastrar Atualizar Excluir Pesquisar Sair

Fonte: autoria própria.

# Vantagens do Paradigma Orientado a Eventos

- O desenvolvimento orientado a eventos é realizado através dos componentes da tela que disponibilizam os eventos que fazem a interação com o usuário.
- Através dos tratadores de eventos o desenvolvedor orchestra a execução do programa.

```
Procedure Calculadora.SomarClick(Sender: TObject);
```

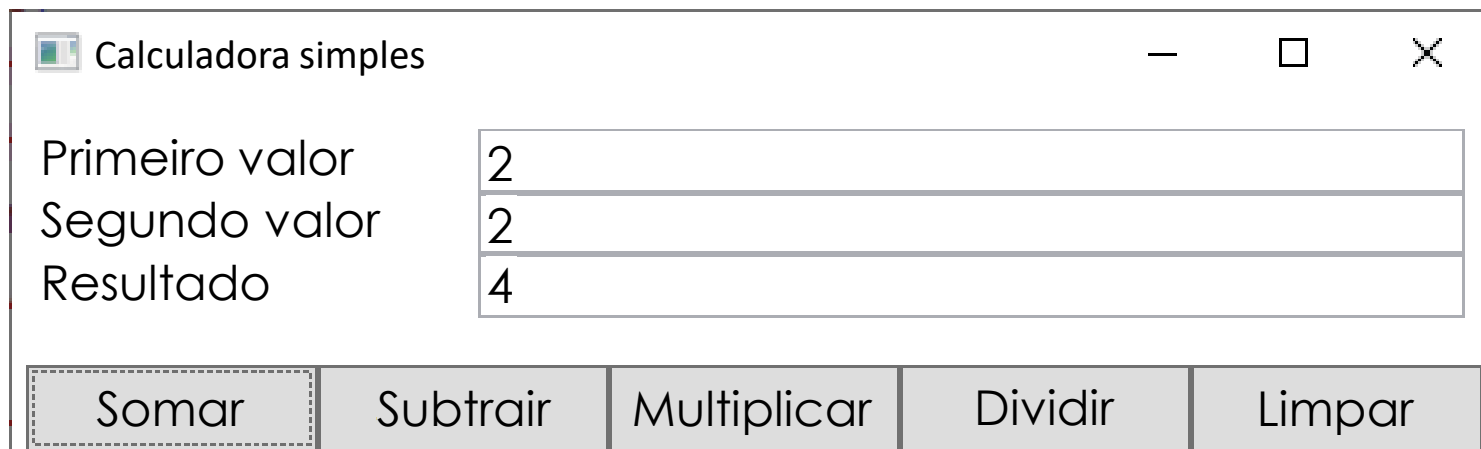
```
Begin
```

```
    if (primeiroNumero.Text <> "") and (segundoNumero.Text <> "") then
```

```
        resultado.Text := IntToStr(StrToInt(primeiroNumero.Text) +
```

```
        IntToStr(StrToInt(segundoNumero.Text);
```

```
end;
```



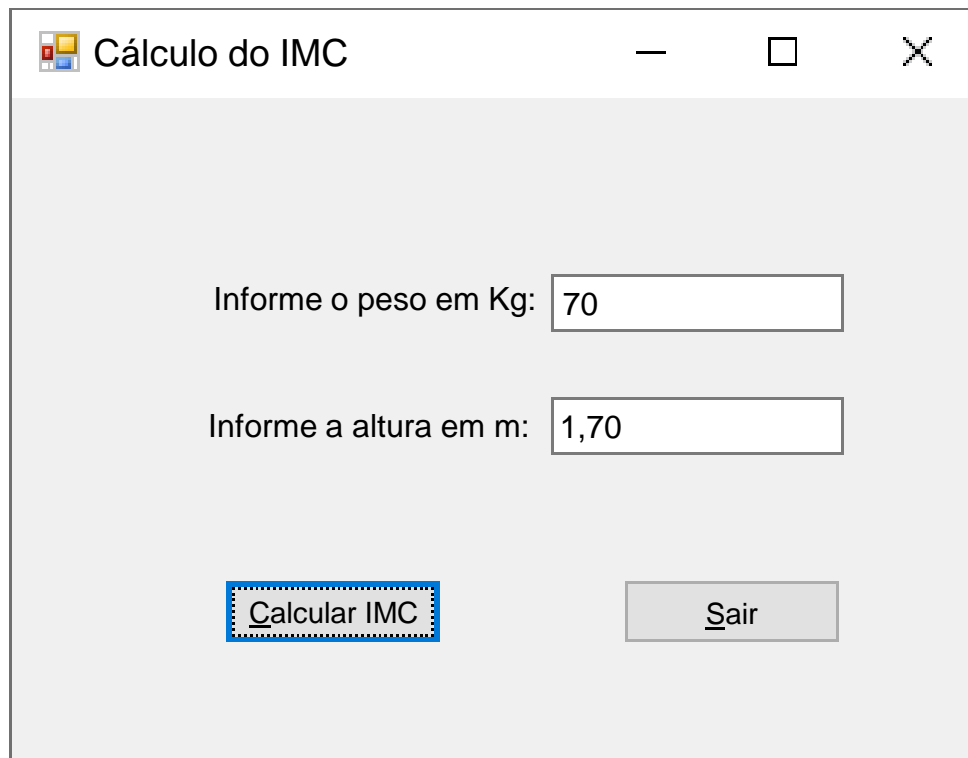
Primeiro valor	2
Segundo valor	2
Resultado	4

Somar Subtrair Multiplicar Dividir Limpar



# Vantagens do Paradigma Orientado a Eventos

- A programação orientada a eventos permite que o programador projete, visualmente, a aplicação de acordo com os requisitos do produto de *software*, tornando a programação mais flexível.
- É possível programar os componentes por meio de um enorme leque de eventos que atendem aos diversos requisitos do que se quer que o programa faça e interagem com o usuário.

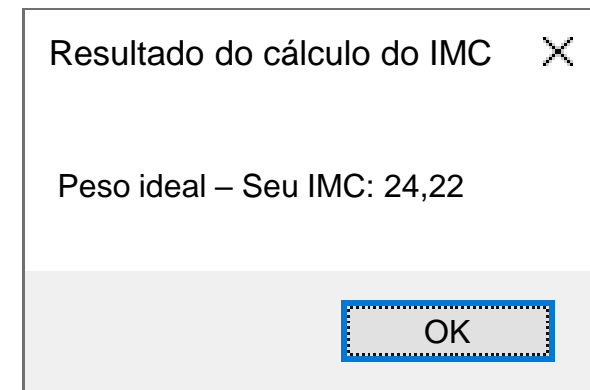


Cálculo do IMC

Informe o peso em Kg: 70

Informe a altura em m: 1,70

Calcular IMC Sair



Resultado do cálculo do IMC

Peso ideal – Seu IMC: 24,22

OK

Fonte: autoria própria.

# Interatividade

Com relação à programação orientada a eventos, é correto afirmar que:

- a) O fluxo do programa é determinado por uma sequência de comandos que alteram o estado atual do sistema, desde a entrada dos dados até atingir um estado final.
- b) O fluxo do programa é determinado por eventos normalmente disparados por interações do usuário.
- c) É uma tecnologia que, hoje, foi substituída pela programação funcional.
  - d) Concentra-se na modelagem de uma aplicação orientada por objetos do mundo real, restringindo-se o uso de uma interface gráfica.
  - e) Resolve os problemas computacionais utilizando-se de funções matemáticas, evitando-se os estados ou os dados mutáveis.

# Resposta

Com relação à programação orientada a eventos, é correto afirmar que:

- a) O fluxo do programa é determinado por uma sequência de comandos que alteram o estado atual do sistema, desde a entrada dos dados até atingir um estado final.
- b) O fluxo do programa é determinado por eventos normalmente disparados por interações do usuário.
- c) É uma tecnologia que, hoje, foi substituída pela programação funcional.
  - d) Concentra-se na modelagem de uma aplicação orientada por objetos do mundo real, restringindo-se o uso de uma interface gráfica.
  - e) Resolve os problemas computacionais utilizando-se de funções matemáticas, evitando-se os estados ou os dados mutáveis.

# Paradigma de Programação Orientada a Objetos

- O paradigma de programação orientado a objetos (POO) é um dos paradigmas mais requisitados pelas empresas de *software* na atualidade.
- Desde o final da década de 1980, a POO começou a ser considerada uma abordagem de desenvolvimento de *software* poderosa e prática.
- A POO, por meio de uma metodologia clara e objetiva, fornece todo o suporte para que o desenvolvedor possa entregar um programa com o custo, o prazo e a qualidade adequadas aos processos de desenvolvimento de *software*.

# Paradigma de Programação Orientada a Objetos

- Um modo importante de entender o funcionamento de um programa orientado a objetos é entender que ele é uma coleção de objetos que interagem entre si e se comunicam por meio da troca de mensagens.
- Cada um destes objetos pode ser visto como uma máquina separada que possui os dados e realiza as operações sobre estes dados.
- Uma das vantagens da POO é que vários objetos podem ser instâncias distintas da mesma classe, sendo que as operações são as mesmas, operando sobre os diferentes dados (TUCKER; NOONAN, 2009).

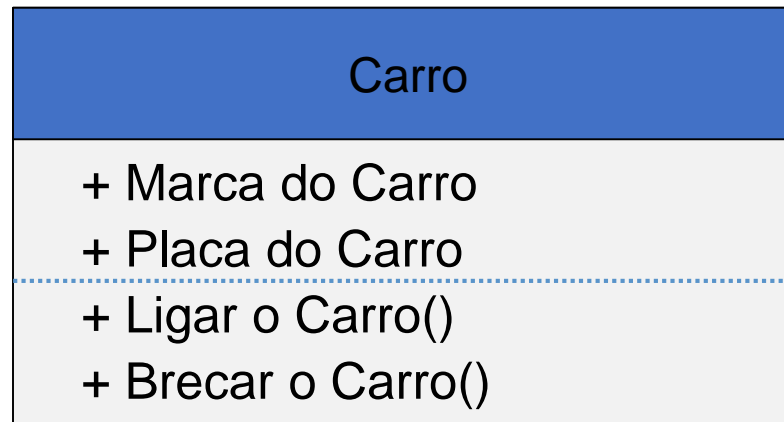
# Paradigma de Programação Orientada a Objetos – Classes e objetos

## Classes:

- Na Programação Orientada a Objetos (POO) toda a estrutura do programa é organizada em classes, que é uma declaração de tipo que encapsula constantes, variáveis e funções para a manipulação dessas variáveis (TUCKER; NOONAN, 2009).

As classes podem ser representadas semanticamente por um retângulo contendo três divisões, conforme a figura a seguir:

- 1) Nome da classe: exemplo de carro;
- 2) Propriedades ou atributos da classe: exemplo: marca do carro, placa do carro;
- 3) Os métodos da classe: ligar o carro() e breicar o carro().



Fonte: autoria própria.

# Paradigma de Programação Orientada a Objetos – Classes e objetos

## Objetos:

- Todo objeto é uma instância de uma classe. Cada objeto possui uma identidade única e uma existência própria;
- Os objetos são representados por determinada classe e diferenciam-se entre si pelos valores de seus atributos;
- São diferentes mesmo que as suas propriedades e comportamentos sejam iguais;
- Exemplo: o carro do Luiz possui um identificador único, e é uma instância da classe do carro.

### Classe

Carro
+ Marca do Carro + Placa do Carro
+ Ligar o Carro() + Brecar o Carro()

### Objeto

BFGHJK123456: Carro
+ Marca do Carro: Corsa + Placa do Carro: AAA1234
+ Ligar o Carro() + Brecar o Carro()

# Paradigma de Programação Orientada a Objetos – Classes e objetos

## Objetos:

- Na POO, o foco da programação está nos objetos que trocam as mensagens entre si, através da execução dos métodos que realizam os algoritmos, ou as atividades necessárias ou os serviços requisitados pelas mensagens.

Classe

Carro
+ Marca do Carro + Placa do Carro
+ Ligar o Carro() + Brecar o Carro()

Objeto

BFGHJK123456: Carro
+ Marca do Carro: Corsa + Placa do Carro: AAA1234
+ Ligar o Carro() + Brecar o Carro()

Fonte: autoria própria.



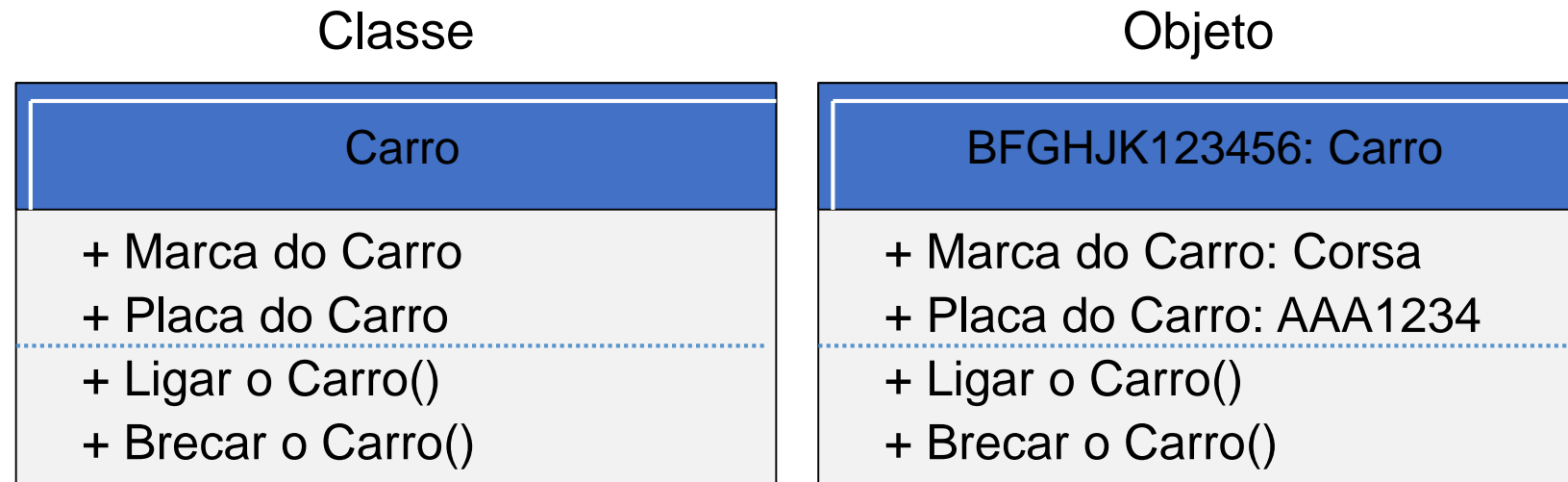
# Paradigma de Programação Orientada a Objetos – Atributos e métodos

## Atributos:

- Os atributos representados pela segunda divisão no nosso diagrama de classe, vide a figura a seguir;
- Este conjunto de propriedades define as informações ou os dados que representam ou caracterizam um objeto e armazenam os seus valores.

Exemplo de propriedades para um objeto carro:

- Ano-modelo do carro = 2015;
- Cor do carro = azul;
- Proprietário = Luiz Roberto.



# Paradigma de Programação Orientada a Objetos – Atributos e métodos

## Métodos:

- A terceira divisão do diagrama de classe, representam os métodos, também conhecido no jargão de TI como as assinaturas dos métodos das classes, como, por exemplo: Ligar o carro (), Breicar o carro ();
- Na POO, os métodos definem o comportamento do objeto; cada objeto possui o seu próprio conjunto de serviços ou operações que podem ser requisitados por outros objetos, por meio de troca de mensagens;
- Um objeto pode enviar uma mensagem ou uma chamada de um método a outro objeto, contendo a identificação dos objetos e, em determinados casos, os seus parâmetros.

Exemplo de métodos para um objeto carro:

- Ligar o carro ();
- Breicar o carro ();
- Acender os faróis ();
- Trocar de marcha ().

# Paradigma de Programação Orientada a Objetos – Encapsulamento

## Encapsulamento:

- O encapsulamento de dados permite restringir o acesso às variáveis e aos métodos da classe, ou, até mesmo, à própria classe;
- Para isso, é criada uma estrutura de restrição do acesso direto a alguns dos componentes de um objeto, por meio de métodos que podem ser utilizados por qualquer outra classe;
- Os detalhes de como o código foi implementado ficam ocultos ao usuário da classe, que passa a utilizar os serviços desta sem saber como isso ocorre internamente;
- Somente é apresentada para o programador uma lista das funcionalidades existentes.

Utiliza-se, como padrão, os métodos manipuladores:

- *Set* para atribuir um valor à propriedade da classe; e
- *Get* para recuperar um valor da propriedade da classe.
- Em geral, deve-se utilizar o modificador “*private*” para os atributos e “*public*” para os métodos.

# Paradigma de Programação Orientada a Objetos – Encapsulamento

## Encapsulamento:

- Observe na figura que somente é possível inserir os dados no objeto pessoa por meio dos métodos *getters* e *setters*.

```
public class Pessoa {  
    private String nome;  
    public String getNome() {  
        return nome;  
    }  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
}  
  
public static void main(String[] args)  
{  
    Pessoa pessoa = new Pessoa();  
    pessoa.setNome("Luiz Roberto");  
    System.out.println(funcionario.getN  
ome());  
}
```

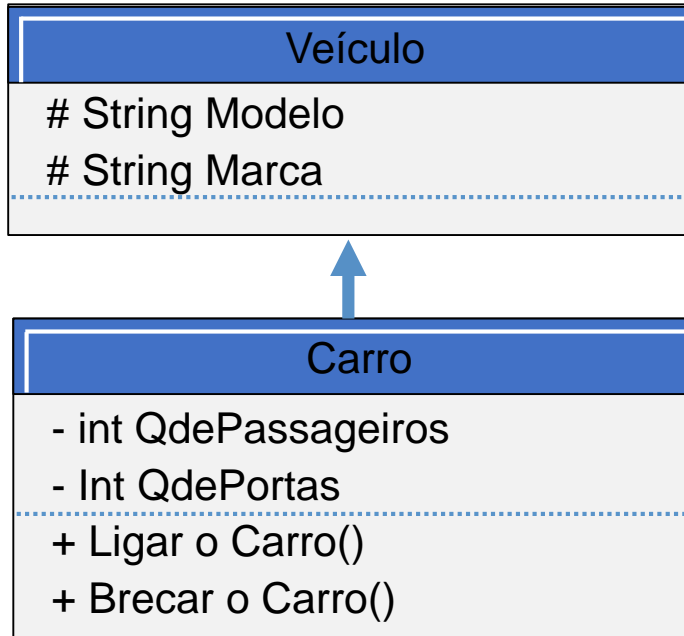
# Paradigma de Programação Orientada a Objetos – Herança

## Herança:

- Trata-se de um recurso no qual uma subclasse herda as propriedades (atributos) e os métodos da superclasse, também chamada de classe Pai ou classe Mãe, aumentando as possibilidades de reutilização de código, a diminuição do tempo e do custo de desenvolvimento;
- A subclasse ou classe Filha, além de herdar da classe Pai, também pode implementar os seus próprios atributos e métodos, ou, até mesmo, redefinir os métodos da classe Pai com novas implementações, promovendo as inúmeras possibilidades de reutilização;
- Uma vez criada, uma classe pode ser reutilizada no mesmo programa ou em outro.

# Paradigma de Programação Orientada a Objetos – Herança

## Herança:



```
public class Veiculo{
    public String marca;

    public Veiculo(String marca) {
        this.marca = marca;
    }
}

public class Carro extends Veiculo{
    public int qdePassageiros;
    public Carro(String marca, int qdePassageiros) {
        super(marca);
        this.qdePassageiros = qdePassageiros;
    }
}

public static void main(String[] args) {
    Carro carro = new Carro("Jeep", 5);
}
```

# Paradigma de Programação Orientada a Objetos – Polimorfismo

## Polimorfismo:

- É a capacidade de um objeto de assumir diferentes formas. Por meio do uso das técnicas de encapsulamento e herança, novas implementações podem ser adicionadas às classes;
- A partir do momento em que uma classe foi herdada, os métodos herdados desta classe podem ser redefinidos para um novo algoritmo permitindo a reutilização de código e, conseqüentemente, a redução no tempo e no custo do desenvolvimento;
- O polimorfismo permite implementar a generalização, e a herança, a especialização;
  - Cada classe pode assumir o mesmo comportamento da superclasse; desta maneira, uma classe da hierarquia pode assumir as diferentes formas (funcionalidades) de acordo com as classes de nível superior;
  - O uso do polimorfismo gera uma economia de código-fonte;
  - Na prática, o polimorfismo permite também que vários métodos podem existir com o mesmo nome, porém com as implementações diferentes (FURGERI, 2015).

# Paradigma de Programação Orientada a Objetos – Polimorfismo

## Polimorfismo:

```
public class Pessoa {  
    public void exibeNomeClasse(){  
        System.out.println("Super Classe  
        Pessoa");  
    }  
}
```

```
public class PessoaFisica extends Pessoa {  
    public void exibeNomeClasse(){  
        System.out.println("Classe Pessoa Física");  
    }  
}
```

```
public class PessoaJuridica extends Pessoa {  
    public void exibeNomeClasse(){  
        System.out.println("Classe Pessoa Jurídica");  
    }  
}
```

```
public static void main(String[] args) {  
    Pessoa pessoa = new PessoaFisica();  
    Pessoa pessoa = new PessoaJuridica();  
    ...  
    pessoa.exibeNomeClasse();  
}
```



# Paradigma de Programação Orientada a Objetos – Classe abstrata

## Classe abstrata:

- A classe abstrata não permite ser instanciada, ou seja, não podem ser criados os objetos a partir dela. Os seus métodos são apenas declarados e não são definidos, ou seja, não têm implementação;
- Uma subclasse, ao criar uma instância de uma classe abstrata, deve implementar, isto é, definir o algoritmo para todos os métodos abstratos herdados. Uma classe abstrata é utilizada como base para a elaboração de outras classes.

# Paradigma de Programação Orientada a Objetos – Classe abstrata

**Classe abstrata:**

```
public abstract class Pessoa {
```

```
// Atributos da classe
```

```
String nome;
```

```
// Método abstrato
```

```
public abstract void GravarPessoa();  
};
```

```
public class PessoaFisica extends Pessoa {
```

```
public void GravarPessoa() {
```

```
nome = "Luiz";
```

```
String CPF = "000.000.000-00";
```

```
System.out.println("Nome: " + nome + " CPF: " + CPF );}}
```

```
public class PessoaJuridica extends Pessoa {
```

```
public void GravarPessoa() {
```

```
nome = "Grupo Empresarial Luiz";
```

```
String CNPJ = "000.000.000/0001-00";
```

```
System.out.println("Nome: " + nome + " CNPJ: " + CNPJ);}}
```

```
...
```

```
PessoaFisica pf = new PessoaFisica();
```

```
pf.GravarPessoa();
```

```
PessoaJuridica pj = new PessoaJuridica();
```

```
pj.GravarPessoa();
```

# Paradigma de Programação Orientada a Objetos – Interface

## Interface:

- A interface define apenas a especificação de uma classe e não a maneira como vai ser implementada (codificada);
- Ela funciona de maneira semelhante às classes abstratas, mas não permite a implementação de nenhum método, pois apenas declara um ou mais métodos (comportamento) que a classe que implementa a interface deve ter (FURGERI, 2015);
- Uma classe implementa uma interface; ela não herda, como é feito na classe abstrata; então, uma classe pode implementar uma ou mais interfaces (SEBESTA, 2011);
  - Nos projetos de *software*, a interface é muito útil para que os desenvolvedores sejam obrigados a seguir um padrão de projeto;
  - Como, por exemplo, a de nomenclatura de métodos, pois as classes que implementam uma interface são obrigadas a implementar os métodos definidos na interface.

# Paradigma de Programação Orientada a Objetos – Interface

**Interface:**

```
public interface ICalculadora {  
public int Somar(int valor1, int valor2);  
public int Subtrair(int valor1, int valor2);  
}
```

```
public class Calculadora implements ICalculadora{  
public int Somar(int valor1, int valor2) {  
    return valor1 + valor2;  
}  
public int Subtrair(int valor1, int valor2) {  
    return valor1 - valor2;  
}  
} public static void main(String[] args) {  
    ICalculadora calculadora = new Calculadora();  
    int resultadoSoma = calculadora.Somar(2, 2);  
    System.out.println(resultadoSoma);  
    int resultadoSubtracao = calculadora.Subtrair(2, 2);  
    System.out.println(resultadoSubtracao);  
}
```

# Interatividade

Complete a frase. O polimorfismo, que é um dos pilares da Programação Orientada a Objetos:

- a) Ocorre quando uma classe tem um relacionamento com outra classe.
- b) Consiste em ocultar os atributos da classe, por meio da implementação dos métodos *set* e *get*.
- c) É a capacidade de um objeto de assumir diferentes formas. Cada classe pode assumir o mesmo comportamento da superclasse; desta maneira, uma classe da hierarquia pode assumir diferentes funcionalidades de acordo com as classes de nível superior.
- d) É um conceito utilizado para implementar as operações de uma classe de maneira pública.
- e) Permite utilizar as operações distintas de uma classe e os atributos de uma subclasse.

## Resposta

Complete a frase. O polimorfismo, que é um dos pilares da Programação Orientada a Objetos:

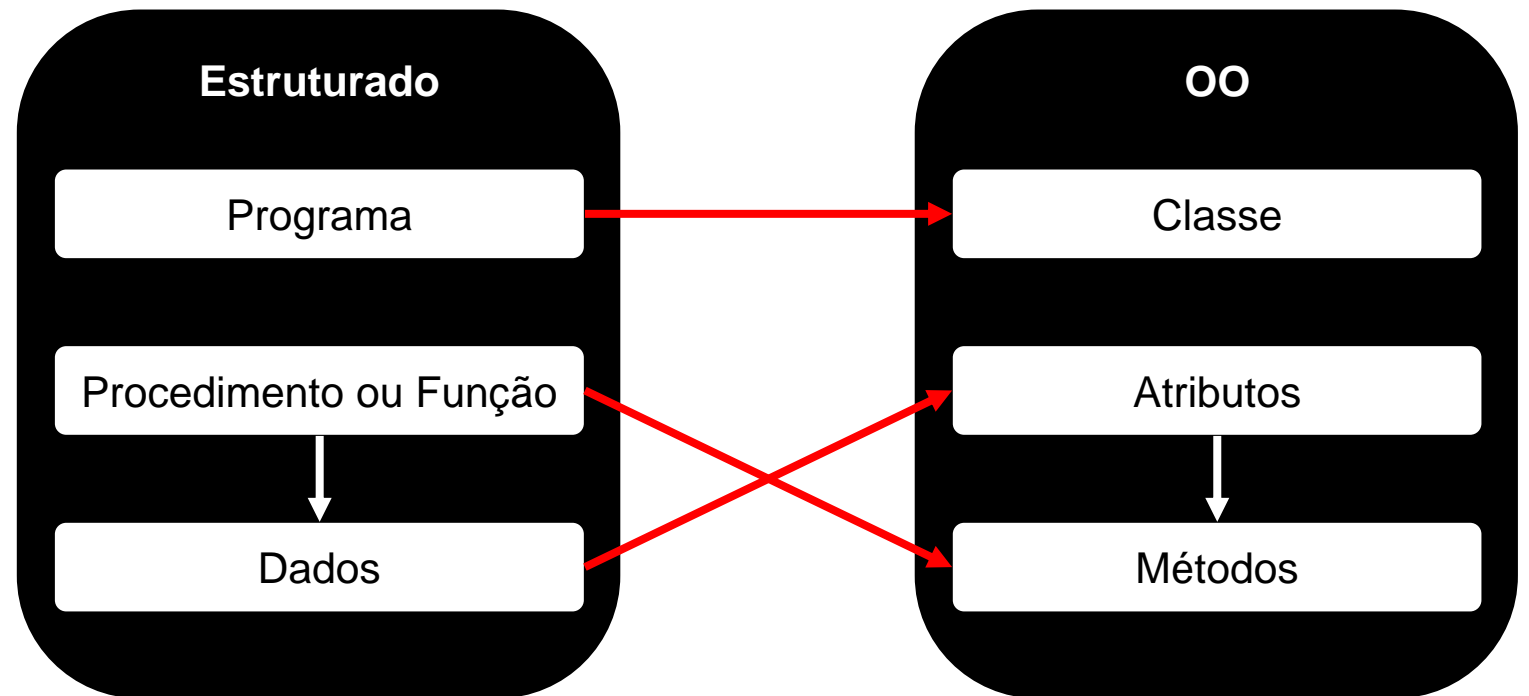
- a) Ocorre quando uma classe tem um relacionamento com outra classe.
- b) Consiste em ocultar os atributos da classe, por meio da implementação dos métodos *set* e *get*.
- c) É a capacidade de um objeto de assumir diferentes formas. Cada classe pode assumir o mesmo comportamento da superclasse; desta maneira, uma classe da hierarquia pode assumir diferentes funcionalidades de acordo com as classes de nível superior.
- d) É um conceito utilizado para implementar as operações de uma classe de maneira pública.
- e) Permite utilizar as operações distintas de uma classe e os atributos de uma subclasse.

# Programação Estruturada X Orientada a Objetos

- Um paradigma de programação é uma metodologia que fornece a visão que o desenvolvedor possui sobre a estrutura e a execução do programa.
- Para resolver um problema, podemos adotar uma entre as variadas metodologias de desenvolvimento de *software*; ao criar um programa, podemos adotar determinado paradigma de programação para desenvolvê-lo.
- Na programação orientada a objetos, os desenvolvedores podem criar uma abstração do programa, como uma coleção de objetos que trocam mensagens entre si.
  - Na programação estruturada, os desenvolvedores realizarão este processo de abstração do programa por meio da execução de procedimentos e funções executadas de modo empilhado.

# Programação Estruturada X Orientada a Objetos

- A POO permite a modularização e a separação da aplicação e dos objetos em camadas.
- Um programa OO pode ter um ou mais objetos, e esses objetos podem ser criados (instanciados) ou destruídos em tempo de execução.
- Na programação estruturada, a aplicação possui uma estrutura em que as funcionalidades (procedimentos e funções) e os dados (as variáveis) são organizados em um único bloco.



Fonte: autoria própria.



# Programação Estruturada X Orientada a Objetos

- O pensamento de um desenvolvedor OO deve se voltar para a própria forma de se olhar a estrutura de um programa.
- Enquanto que, na programação estruturada, o desenvolvedor se preocupa em dividir o programa em módulos com os procedimentos e as funções; na programação OO esta modularização é realizada para a resolução do problema, por meio das classes.

A tabela a seguir mostra a diferença de conceitos entre os paradigmas orientado a objeto e estruturado:

Fonte: autoria própria.

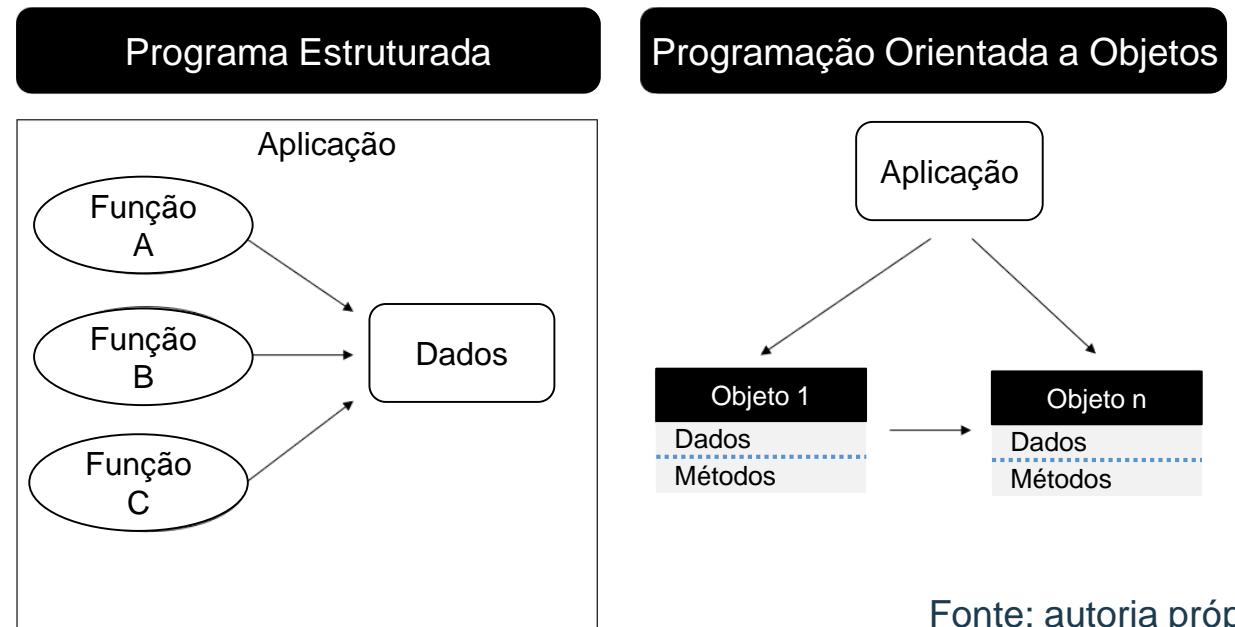
Programação OO	Programação estruturada
Atributos ou propriedades;	Variáveis;
Métodos das classes;	Procedimentos ou funções;
Troca de mensagens entre os objetos;	Chamadas aos procedimentos e às funções.
Polimorfismo;	
Herança;	
Criação de objetos como instâncias da classe.	

# Programação Estruturada X Orientada a Objetos

- Os dados na programação estruturada são acessados na memória do computador, por meio dos procedimentos e das funções, de forma totalmente dividida.
- No paradigma orientado a objeto, os dados são encapsulados dentro dos objetos e são protegidos pelos métodos desses objetos.
- O acesso aos atributos ou dados são realizados apenas pelos métodos implementados dentro da própria classe, aplicando-se o conceito de encapsulamento.

# Programação Estruturada X Orientada a Objetos

- Na programação estruturada, os procedimentos realizam as operações com os dados compartilhados.
- Os dados são acessados, diretamente referenciados, pelos nomes das variáveis.
- Na programação OO, as operações são realizadas por meio de um conjunto de objetos que interagem entre si, através de “trocas de mensagens” ou da execução dos métodos existentes em cada objeto.
- Um serviço é solicitado de um objeto a outro através de uma mensagem.



Fonte: autoria própria.

# Programação Estruturada X Orientada a Objetos

## Vantagens da POO:

- Permite a melhor organização do código: a POO permite elaborar um relacionamento entre os diversos componentes, estabelecendo uma troca de mensagens ou comunicação entre eles;
- Contribui para o reaproveitamento de código: por meio da aplicação de alguns conceitos, como: herança, polimorfismo e interface facilita a reutilização de código, devido à simplicidade de se herdar os atributos e os comportamentos de outros objetos.

# Programação Estruturada X Orientada a Objetos

## Vantagens da POO:

- Manutenção do código: na POO, se o desenvolvedor codificar seguindo os padrões de construção, qualquer programador que conheça os padrões pode, facilmente, encontrar os problemas, utilizar o código escrito ou, até mesmo, melhorá-lo, se assim se fizer necessário;
- *Design Patterns*: Facade, Singleton, MVC, DAO etc.

# Programação Estruturada X Orientada a Objetos

## Desvantagens da POO:

- Pode não possuir o mesmo desempenho de códigos similares desenvolvidos em programação estruturada;
- Os conceitos da POO são de difícil compreensão, se comparados aos conceitos da programação estruturada.

# Interatividade

Como são tratados os dados no paradigma OO (POO) e no paradigma estruturado (PE)?

- a) No PE, os dados são encapsulados dentro dos objetos e são protegidos pelos métodos desses objetos. No POO, os dados são acessados na memória do computador, por meio dos procedimentos e das funções.
- b) No PE, o acesso aos dados é realizado apenas pelos métodos implementados dentro da própria classe. No POO, os dados são tratados por meio de escrita de funções ou dos procedimentos que operam sobre os dados.
- c) No PE, os procedimentos realizam as operações com os dados compartilhados. No POO, as operações com os dados são realizadas por meio de um conjunto de objetos que interagem entre si através de “trocas de mensagens”.
  - d) No POO, os procedimentos realizam as operações com os dados compartilhados. No PE, as operações são realizadas por meio de um conjunto de objetos.
  - e) No PE, os dados são encapsulados dentro das variáveis. No POO, os dados são compartilhados pelas variáveis.

# Resposta

Como são tratados os dados no paradigma OO (POO) e no paradigma estruturado (PE)?

- a) No PE, os dados são encapsulados dentro dos objetos e são protegidos pelos métodos desses objetos. No POO, os dados são acessados na memória do computador, por meio dos procedimentos e das funções.
- b) No PE, o acesso aos dados é realizado apenas pelos métodos implementados dentro da própria classe. No POO, os dados são tratados por meio de escrita de funções ou dos procedimentos que operam sobre os dados.
- c) No PE, os procedimentos realizam as operações com os dados compartilhados. No POO, as operações com os dados são realizadas por meio de um conjunto de objetos que interagem entre si através de “trocas de mensagens”.
- d) No POO, os procedimentos realizam as operações com os dados compartilhados. No PE, as operações são realizadas por meio de um conjunto de objetos.
- e) No PE, os dados são encapsulados dentro das variáveis. No POO, os dados são compartilhados pelas variáveis.



# Referências

- DAURICIO, J. S. *Algoritmos e Lógica de Programação*. Belo Horizonte: Editora e Distribuidora Educacional S.A., 2015.
- ETZION, O.; NIBLETT, P. *Event Processing in Action*. Stamford: Manning Publications Co., 2011.
- FURGERI, S. *Java 8 – Ensino Didático: desenvolvimento e implementação de aplicações*. São Paulo: Érica, 2015.
- SEBESTA, R. W. *Conceitos de Linguagens de Programação*. 9. ed. Porto Alegre: Bookman, 2011.
  - TUCKER, A. B.; NOONAN, R. E.; *Linguagens de Programação – Princípios e Paradigmas*. 2. ed. São Paulo: McGraw Hill, 2009.
  - VAREJÃO, F. M. *Linguagens de Programação: Conceitos e Técnicas*. Rio de Janeiro: Elsevier (*Campus*), 2004.

**ATÉ A PRÓXIMA!**