

Introdução à

Programação Orientada a Objetos

<https://github.com/gplichoski/MinicursoPOOJava>

Conteúdo

- O que são modelos?
- O que é programação orientada a objetos?
- Classes, Campos e Métodos
- Objetos, Instâncias, Referências
- Encapsulamento e Modificadores de acesso
- Construtores
- Composição e Herança
- GUI Builder (modo design)

O que são modelos?

**Representações simplificadas
de objetos, pessoas, itens,
tarefas, processos, conceitos,
ideas ...**

● Modelo

RESTAURANTE CASEIRO					
Mesa 1	Mesa 2	Mesa 3	Mesa 4	Mesa 5	Mesa 6
refeição x 3	refeição x 1	refeição x 1	kg refeição x 2	kg refeição x 5	kg refeição x 2
sobremesa x 2	sobremesa x 1	sobremesa x 0	sobremesa x 0	sobremesa x 3	sobremesa x 0
refrig. 2L x 1	refrig. 2L x 1	refrig. 2L x 0	refrig. 2L x 0	refrig. 2L x 2	refrig. 2L x 0
água x 2	água x 0	água x 0	água x 2	água x 0	água x 0
cerveja x 0	cerveja x 0	cerveja x 0	cerveja x 0	cerveja x 5	cerveja x 0

Mesa
- refeição
- sobremesa
- refri. 2L
- água
- cerveja

● Modelo

- Dados
- Operações

Lampada

- estadoDaLampada
- acende()
- apaga()
- mostraEstado()

```
modelo Lampada //representa uma lâmpada em uso
inicio do modelo
    dado estadoLampada; //indica se está ligada ou não

        operação acende() //acende a lâmpada
        inicio
            estadoDaLampada = aceso;
        fim

        operação apaga() // apaga a lâmpada
        inicio
            estadoDaLampada = apagada;
        fim

        operação mostraEstado() // mostra o estado da lâmpada
        inicio
            se (estadoDaLampada == aceso)
                imprime "A lâmpada está acesa";
            senão
                imprime "A lâmpada está apagada";
            fim

    fim do modelo
```

O que é
programação
orientada a
objetos?

Programação Orientada a Objetos (POO)
é um paradigma de programação onde
se usam **classes** e **objetos**, criados a
partir de **modelos**, para representar e
processar dados usando programas de
computadores

modelo Lampada //representa uma lâmpada em uso

inicio do modelo

dado estadoLampada; //indica se está ligada ou não

operação acende() //acende a lâmpada

inicio

estadoDaLampada = aceso;

fim

operação apaga() // apaga a lâmpada

inicio

estadoDaLampada = apagada;

fim

operação mostraEstado() // mostra o estado da lâmpada

inicio

se (estadoDaLampada == aceso)

imprime "A lâmpada está acesa";

senão

imprime "A lâmpada está apagada";

fim

fim do modelo

Classe

Objeto



modelo Lampada //representa uma lâmpada em uso

inicio do modelo

dado estadoLampada; //indica se está ligada ou não

operação acende() //acende a lâmpada

inicio

estadoDaLampada = aceso;

fim

operação apaga() // apaga a lâmpada

inicio

estadoDaLampada = apagada;

fim

operação mostraEstado() // mostra o estado da lâmpada

inicio

se (estadoDaLampada == aceso)

imprime "A lâmpada está acesa";

senão

imprime "A lâmpada está apagada";

fim

fim do modelo

Classe

Objeto



Classes, Campos e Métodos

CLASSE PESSOA EM JAVA

```
1 package app.topico1;
2
3 /**
4  * Pessoa
5 */
6 public class Pessoa {
7
8     String nomeDaPessoa;
9
10    void criaPessoa(String nome) {
11        nomeDaPessoa = nome;
12    }
13
14    String apresentaPessoa() {
15        return "Olá, meu nome é " + nomeDaPessoa + "!";
16    }
17}
18
```

Campos são representados por tipos de dados nativos

- **boolean**
- **char, string**
- **byte, short, int, long**
- **float, double**

Campos são representados por tipos de dados nativos

- **boolean**
 - **possui dois valores, que podem ser considerados como 0 ou 1, falso ou verdadeiro.**

```
boolean aceso = true;
```

Campos são representados por tipos de dados nativos

- **char, string**
 - **para campos de caracteres ou palavras**

```
String nomeDaPessoa;
```

Campos são representados por tipos de dados nativos

- **char, string**
 - **para campos de caracteres ou palavras**

```
String nomeDaPessoa = "Marcos";
```

Campos são representados por tipos de dados nativos

- **byte, short, int, long**
 - Números inteiros

```
Integer numeroDeItens = 5;
```

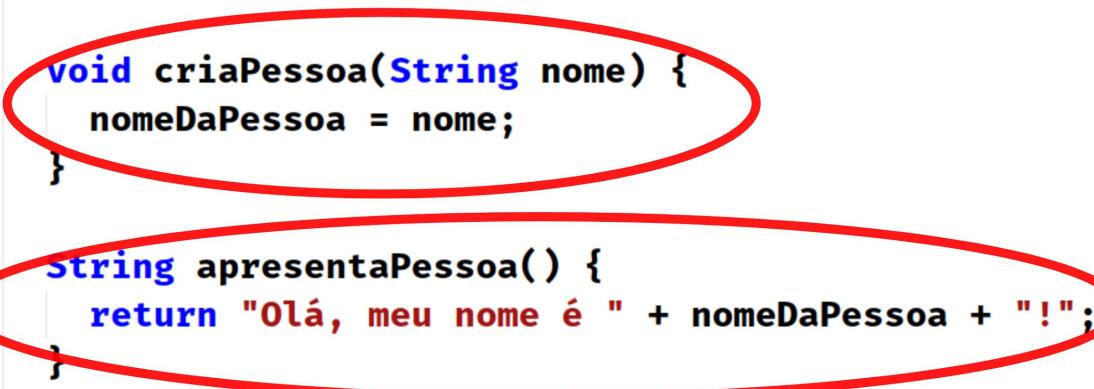
Campos são representados por tipos de dados nativos

- **float, double**
 - **números com ponto flutuante com precisão**

Double altura = 1.83;

MÉTODOS

```
1 package app.topico1;  
2  
3 /**  
4  * Pessoa  
5  */  
6 public class Pessoa {  
7  
8     String nomeDaPessoa;  
9  
10    void criaPessoa(String nome) {  
11        nomeDaPessoa = nome;  
12    }  
13  
14    String apresentaPessoa() {  
15        return "Olá, meu nome é " + nomeDaPessoa + "!";  
16    }  
17}  
18}
```

Two red ovals highlight specific sections of the Java code. The first oval encloses the entire body of the 'criaPessoa' method, from its declaration to its closing brace. The second oval encloses the entire body of the 'apresentaPessoa' method, also from its declaration to its closing brace.

MÉTODOS

```
void criaPessoa(String nome) {  
    nomeDaPessoa = nome;  
}  
  
String apresentaPessoa() {  
    return "Olá, meu nome é " + nomeDaPessoa + "!";  
}
```

MÉTODOS

- retorno de métodos

```
void criaPessoa(String nome) {  
    nomeDaPessoa = nome;  
}
```

```
String apresentaPessoa() {  
    return "Olá, meu nome é " + nomeDaPessoa + "!";  
}
```

MÉTODOS

- parâmetros

```
void criaPessoa(String nome) {  
    nomeDaPessoa = nome;  
}
```

```
String apresentaPessoa() {  
    return "Olá, meu nome é " + nomeDaPessoa + "!";  
}
```

Objetos, Instâncias e Referências

CLASSE PESSOA NO MÉTODO MAIN

```
public static void main(String[] args) {  
    Pessoa referenciaGuilherme = new Pessoa();  
    referenciaGuilherme.criaPessoa("Guilherme");  
    System.out.println(referenciaGuilherme.apresentaPessoa());  
}  
}
```

CLASSE PESSOA NO MÉTODO MAIN

```
public static void main(String[] args) {  
    Pessoa referenciaGuilherme = new Pessoa();  
    referenciaGuilherme.criaPessoa("Guilherme");  
    System.out.println(referenciaGuilherme.apresentaPessoa());  
}  
}
```

CLASSE PESSOA NO MÉTODO MAIN

```
public static void main(String[] args) {  
    Pessoa referenciaGuilherme = new Pessoa();  
    referenciaGuilherme.criaPessoa("Guilherme");  
    System.out.println(referenciaGuilherme.apresentaPessoa());  
}  
}
```

CLASSE PESSOA NO MÉTODO MAIN

```
public static void main(String[] args) {  
    Pessoa referenciaGuilherme = new Pessoa();  
    referenciaGuilherme.criaPessoa("Guilherme");  
    System.out.println(referenciaGuilherme.apresentaPessoa());  
}  
}
```

CLASSE PESSOA NO MÉTODO MAIN

```
public static void main(String[] args) {  
    Pessoa referenciaGuilherme = new Pessoa();  
    referenciaGuilherme.criaPessoa("Guilherme");  
    System.out.println(referenciaGuilherme.apresentaPessoa());  
}  
}
```

CLASSE PESSOA NO MÉTODO MAIN

```
public static void main(String[] args) {  
    Pessoa referenciaGuilherme = new Pessoa();  
    referenciaGuilherme.criaPessoa("Guilherme");  
    System.out.println(referenciaGuilherme.apresentaPessoa());  
}  
}
```

// Criando referências à objetos

Encapsulamento e Modificadores de Acesso

- **Encapsulamento** significa separar o programa em partes, para:
 - **Controlar o acesso** aos atributos (campos) e métodos de uma classe
 - **Proteger os dados** manipulados dentro da classe

Modificadores de Acesso

- **public**
- **private**
- **protected**
- **default (padrão)**

Modificadores de Acesso

- **public**
 - pode ser acessada de qualquer lugar e por qualquer entidade

```
public String nome;
```

```
public String getName() {
```

Modificadores de Acesso

public

```
1 package app.topico1;
2
3 /**
4 * Pessoa
5 */
6 public class Pessoa {
7
8     public String nomeDaPessoa;
9
10    public void criaPessoa(String nome) {
11        nomeDaPessoa = nome;
12    }
13
14    public String apresentaPessoa() {
15        return "Olá, meu nome é " + nomeDaPessoa + "!";
16    }
17
18    public static void main(String[] args) {
19        Pessoa referenciaGuilherme = new Pessoa();
20        referenciaGuilherme.criaPessoa("Guilherme");
21
22        referenciaGuilherme.nomeDaPessoa = "Jonas";
23
24        System.out.println(referenciaGuilherme.apresentaPessoa());
25    }
26
27 }
```

DEBUG CONSOLE

Olá, meu nome é Jonas!

Modificadores de Acesso

- **Private**
 - não podem ser acessados ou usados por nenhuma outra classe

```
private String nome;
```

```
private String getName()
```

Modificadores de Acesso

private

```
1 package app._1_classes;
2
3 You, a few seconds ago | 1 author (You)
4 /**
5  * Pessoa
6 */
7 public class Pessoa {
8
9     private String nomeDaPessoa;
10
11    void criaPessoa(String nome) {
12        nomeDaPessoa = nome;
13    }
14
15    String apresentaPessoa() {
16        return "Olá, meu nome é " + nomeDaPessoa + "!";
17    }
18
19    Run | Debug
20    public static void main(String[] args) {
21        Pessoa referenciaGuilherme = new Pessoa();
22        referenciaGuilherme.criaPessoa("Guilherme");
23
24        referenciaGuilherme.nomeDaPessoa = "Jonas";
25
26        System.out.println(referenciaGuilherme.apresentaPessoa());
27    }
28 }
```

DEBUG CONSOLE

Olá, meu nome é Jonas!

Modificadores de Acesso

private

```
1 package app.topico1;
2
3 /**
4 * PessoaMain
5 */
6 public class PessoaMain {
7     Run | Debug
8     public static void main(String[] args) {
9         Pessoa referenciaGuilherme = new Pessoa();
10
11         referenciaGuilherme.criaPessoa("Guilherme");
12         referenciaGuilherme.nomeDaPessoa = "Jonas";
13
14         System.out.println(referenciaGuilherme.apresentaPessoa());
15     }
16 }
17
18
19
```

DEBUG CONSOLE

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
The field Pessoa.nomeDaPessoa is not visible
at app.topico1.PessoaMain.main(PessoaMain.java:13)
```

// Criando classe Carro

Carro

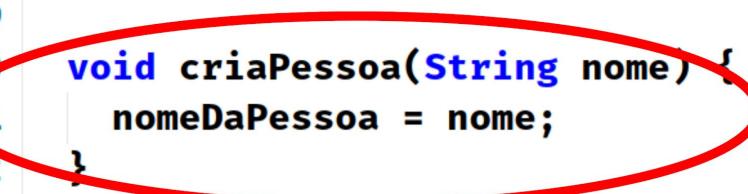
- marca
- modelo
- combustivel

- construirCarro(marca, modelo, combustivel)
- apresentarCarro()

Construtores

CONSTRUTORES

```
1 package app.topico1;  
2  
3 /**  
4  * Pessoa  
5 */  
6 public class Pessoa {  
7  
8     String nomeDaPessoa;  
9  
10    void criaPessoa(String nome) {  
11        nomeDaPessoa = nome;  
12    }  
13  
14    String apresentaPessoa() {  
15        return "Olá, meu nome é " + nomeDaPessoa + "!";  
16    }  
17 }  
18 }
```



CONSTRUTORES

```
1 package app.topico4;
2
3 /**
4 * PessoaConstrutor
5 */
6 public class PessoaConstrutor {
7
8     private String nome;
9     private Integer idade;
10
11    PessoaConstrutor(String nome, Integer idade) {
12        this.nome = nome;
13        this.idade = idade;
14    }
15
16    public String apresentaPessoa() {
17        return "Olá, meu nome é " + this.nome + ", eu tenho " + this.idade + " anos!";
18    }
19
20    public static void main(String[] args) {
21        PessoaConstrutor pessoaConstrutor = new PessoaConstrutor("Guilherme", 28);
22        System.out.println(pessoaConstrutor.apresentaPessoa());
23    }
24 }
```

CONSTRUTORES

```
1 package app.topico4;
2
3 /**
4  * PessoaConstrutor
5 */
6 public class PessoaConstrutor {
7
8     private String nome;
9     private Integer idade;
10
11    PessoaConstrutor(String nome, Integer idade) {
12        this.nome = nome;
13        this.idade = idade;
14    }
15
16    public String apresentaPessoa() {
17        return "Olá, meu nome é " + this.nome + ", eu tenho " + this.idade + " anos!";
18    }
19
20    Run | Debug
21    public static void main(String[] args) {
22        PessoaConstrutor pessoaConstrutor = new PessoaConstrutor("Guilherme", 28);
23        System.out.println(pessoaConstrutor.apresentaPessoa());
24    }
}
```

// Adicionar Construtor na classe Carro

Carro

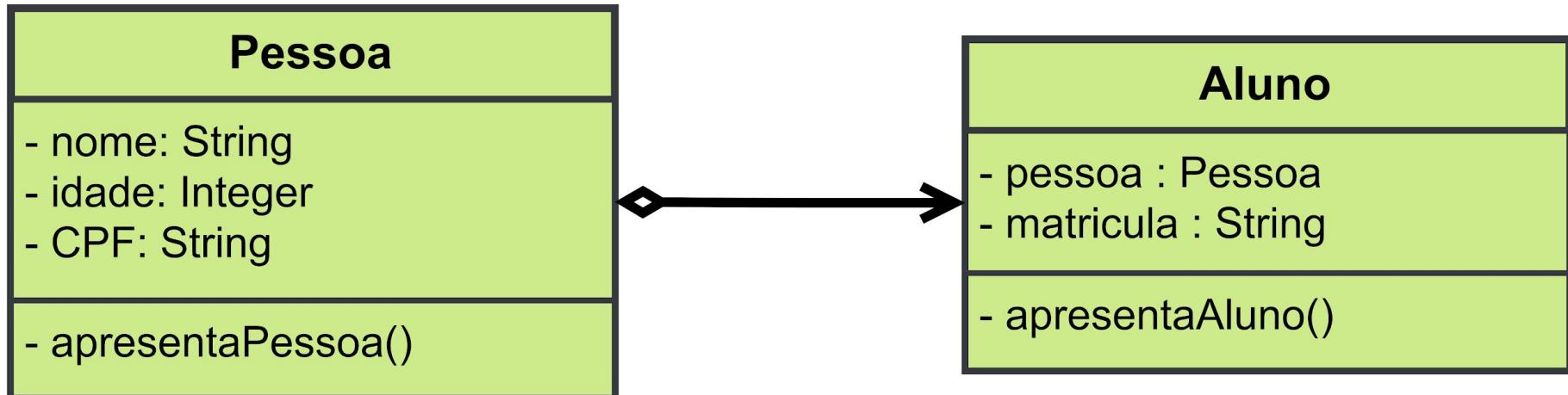
- marca
- modelo
- combustivel

- apresentarCarro()

Composição e Herança

COMPOSIÇÃO

- Quando classe A está contida em classe B



COMPOSIÇÃO

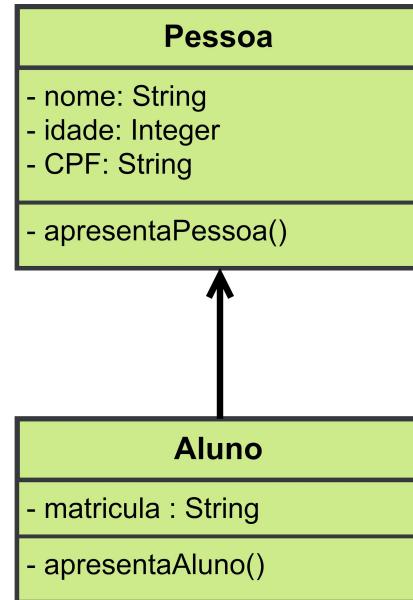
```
1 package app._3_composicao;
2
3 /**
4 * Pessoa
5 */
6 public class Pessoa {
7
8     private String nome;
9     private Integer idade;
10    private String cpf;
11
12    Pessoa(String nome, Integer idade, String cpf) {
13        this.nome = nome;
14        this.idade = idade;
15        this.cpf = cpf;
16    }
17
18    public String apresentaPessoa() {
19        String result = "Olá, meu nome é " + this.nome + "!\n";
20        result = result + "Eu tenho " + this.idade + " anos.\n";
21        result = result + "E meu CPF é " + this.cpf + ".";
22    }
23}
24}
```



```
1 package app._3_composicao;
2
3 /**
4 * Aluno
5 */
6 public class Aluno {
7
8     private Pessoa pessoa;
9     private String matricula;
10
11    Aluno(String nome, Integer idade, String cpf, String matricula) {
12        pessoa = new Pessoa(nome, idade, cpf);
13        this.matricula = matricula;
14    }
15
16    public String apresentaAluno() {
17        String result = pessoa.apresentaPessoa();
18        result = result + "\nE minha matrícula é " + this.matricula + "!";
19    }
20}
21}
```

HERANÇA

- **uma classe (classe filha) que tem os mesmos atributos de outra (classe mãe), mais alguns atributos distintos**



HERANÇA

```
1 package app._4_heranca;
2
3 /**
4  * Pessoa
5  */
6 public class Pessoa {
7
8     private String nome;
9     private Integer idade;
10    private String cpf;
11
12    Pessoa(String nome, Integer idade, String cpf) {
13        this.nome = nome;
14        this.idade = idade;
15        this.cpf = cpf;
16    }
17
18    public String apresentaPessoa() {
19        String result = "Olá, meu nome é " + this.nome + "!\n";
20        result = result + "Eu tenho " + this.idade + " anos.\n";
21        result = result + "E meu CPF é " + this.cpf + ".";
22        return result;
23    }
24 }
```



```
1 package app._4_heranca;
2
3 /**
4  * Aluno
5  */
6 public class Aluno extends Pessoa {
7
8     private String matricula;
9
10    Aluno(String nome, Integer idade, String cpf, String matricula) {
11        super(nome, idade, cpf);
12        this.matricula = matricula;
13    }
14
15    public String apresentaAluno() {
16        String result = super.apresentaPessoa();
17        result = result + "\nE minha matrícula é " + this.matricula + "!";
18        return result;
19    }
20 }
```

// GUI Builder (modo design)

1. Registro Acadêmico
2. Apresentar Classe Carro em componente JFrame

<https://forms.gle/rHW5fihfSabwZUCq9>

MAGRATHEA LABS

www.magrathealabs.com