## **Classes e Objetos**

Desenvolver um programa em Java é uma tarefa que requer atenção. É fácil encontrarmos programas, aparentemente simples, que contenham mais de 10 mil linhas de código-fonte. É importante sabermos que existem técnicas que permitem melhorar a manutenibilidade (medida da facilidade de manutenção de um código – quanto mais fácil for aplicar mudanças em um código, melhor a manutenibilidade) e facilitar a expansão dos nossos programas. Neste tópico, conheceremos dois conceitos que nos ajudarão a produzir programas Java de forma mais organizada e que facilitarão a manutenção no futuro: o paradigma de [Orientação a Objetos](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=Orienta%C3%A7%C3%A3o%20a%20Objetos) e a modularização de atributos (organização de atributos dentro de objetos).

## **Programação Orientada a Objetos**

O paradigma de [Programação Orientada a Objetos (POO)](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=Programa%C3%A7%C3%A3o%20Orientada%20a%20Objetos%20(POO)) propõe a representação de entidades do mundo real dentro de programas, de forma a tornar o desenvolvimento de software mais claro e mais próximo da realidade. No mundo real, exemplos de entidades, como um gato, uma impressora, uma pessoa ou mesmo um carro, têm suas características próprias. Essas características podem ser atributos, ou seja, características da entidade em questão, como altura, peso, cor, idade, modelo, potência etc., ou podem ser comportamentos, como mover, comunicar, desenhar, comer entre outros. A proposta do POO é simples: representar as características das entidades do mundo real, sejam elas atributos ou comportamentos, usando uma linguagem de programação para que possamos incorporar essa entidade aos nossos programas.

O Paradigma de [Orientação a Objetos](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=Orienta%C3%A7%C3%A3o%20a%20Objetos) tem uma unidade fundamental: a [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe). Classes são os modelos das entidades do mundo real feitos em linguagem de programação. A partir das [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe)s, damos vida aos objetos. As [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe)s serão estudadas a seguir.

## [**Classe**](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=Classe)

A [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) é o modelo para criação de objetos. Pense em uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) como uma fôrma, um conjunto de características que uma entidade pode ter dentro de um programa . Um objeto é uma entidade criada a partir das características de uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe), ou seja, a partir da materialização do formato sugerido pela [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe).

Atente-se aos exemplos a seguir e perceba a diferença entre [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) e objeto.

* Um projeto de arquitetura de uma casa é uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe); a casa construída é o objeto;
* A lista de ingredientes de um bolo é uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe); o bolo pronto é um objeto.

Como podemos identificar o que são classes e objetos na Linguagem Java? Vamos aprender isso agora, mas, antes, é necessário entender como construir uma classe.

Suponha uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) denominada Pessoa. Nela, devemos colocar características que representem uma pessoa, ou seja, características que sejam comuns a várias pessoas, por exemplo, nome e idade. Em [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe)s, essas características são chamadas de atributos. Note que todas as pessoas têm atributos como nome e idade, mas note também que esses atributos variam de pessoa para pessoa (nem todo mundo tem o mesmo nome, por exemplo). As pessoas têm alguns comportamentos em comum, como comer, trabalhar e dormir. Em [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe)s, esses comportamentos (ações) são chamados de métodos.

A tabela a seguir está dividida em duas partes: o Mundo Real (à esquerda, em que listamos algumas das características citadas no slide anterior, comuns às pessoas) e Em Java (à direita, temos a representação dessas características na forma de Linguagem de Programação Java.

|  |  |
| --- | --- |
| **Mundo real** | **Em Java** |
| |  |  | | --- | --- | | **Pessoa** | | | Atributos | Ações | | Nome  Idade | Andar  Falar | | 01 [public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) class Pessoa {  02  [public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) String nome;  03  [public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) int idade;  04  [public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) void andar(){  05 System.out.println("estou caminhando");  06 }  07  [public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) void falar(){  08 System.out.println("bla, bla, bla");  09 }  10 } |

Tabela 1: Comparação do modelo de Pessoa do mundo real com Pessoa da Linguagem Java.

## **Objetos**

Objetos são instâncias de uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe), ou seja, são entidades "vivas" que criamos a partir das características definidas em uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe). Assim, um objeto possui todas as características (atributos e comportamentos) que estão descritas na [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) a partir da qual ele foi criado.

Vamos utilizar a [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Pessoa do exemplo anterior como base para compreender o conceito de objetos. Vamos considerar duas pessoas: José, de 32 anos, e Lara, de 25 anos. Criaremos dois objetos para representar essas duas pessoas. Para tornar o nosso exemplo mais interessante, vamos adicionar um método chamado Apresentar à [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Pessoa. Esse método tem como função fazer uma apresentação da pessoa, na qual ela cumprimenta e diz seu nome, como fazemos ao conhecer alguém. O código do método Apresentar deve ser assim:

[public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) void apresentar(){

System.out.println("Olá, meu nome é " + nome);

}

Perceba que o método Apresentar utiliza o atributo Nome da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Pessoa. Como cada objeto do tipo Pessoa deverá possuir um valor para o atributo Nome, cada objeto executará o método Apresentar de forma única, exibindo seu nome no fim da mensagem. Um objeto do tipo Pessoa, que, por exemplo, representa uma pessoa chamada Maria, deverá produzir a mensagem "Olá, meu nome é Maria". Outro objeto que represente, por exemplo, uma pessoa chamada Rafael deverá produzir a mensagem "Olá, meu nome é Rafael", e assim por diante.

**OBS:**

Você já deve ter notado que algumas instruções Java possuem um ou mais pontos, como em System.out.println(), lara.apresentar(), etc.. Em Java, o ponto significa que o termo após cada ponto pertence, ou está em uma hierarquia abaixo do termo anterior a este ponto. Por exemplo: em System.out.println(), temos a [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) System. que possui um objeto chamado ou, que acessamos o método println() a partir deste objeto. Em lara.apresentar(), fazemos a chamada do método apresentar() a partir do objeto lara. Então, sempre que você precisar chamar um membro (um objeto, ou um método, por exemplo) que pertence a um outro membro, você deve usar o ponto para fazer isso.

## **Criando Classes - Resumo**

Classes têm como finalidade representar entidades do mundo real nos nossos programas. Ao criarmos uma classe, devemos tentar listar os atributos da entidade alvo (características como nome, idade, peso, cor, marca, modelo, descrição etc.) bem como seus comportamentos (ações que essa entidade normalmente apresenta). Escolher corretamente o tipo de dados de cada atributo é essencial para que a classe represente bem sua entidade alvo. Escolhas ruins (como String para armazenar valores de ponto flutuante ou tipos numéricos para armazenar valores booleanos) podem resultar em necessidade de modificar a classe no futuro. Já comportamentos que têm a forma de métodos em nossas classes podem ser completamente genéricos ou utilizar os atributos para gerar um resultado único, como percebemos no exemplo do método Apresentar da classe Pessoa.

## **Objetos - Resumo**

Objetos são a concretização de uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe), ou seja, são uma instância viva da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) em que são baseados. Um objeto é vivo no sentido que pode possuir valores para cada atributo previsto na sua [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) e por permitir a execução das ações (métodos) a partir deles. Os nomes das variáveis que armazenam um objeto não têm relação direta com o seu conteúdo, ou seja, um objeto da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Pessoa não precisa ter o nome igual ao valor do atributo nome dessa [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe). De fato, objetos podem ser nomeados de forma genérica com nomes como pessoa01, temp, aux etc. Objetos são criados com comandos compostos da palavra-chave “new” e um [método construtor](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=m%C3%A9todo%20construtor). Conheceremos mais sobre métodos construtores adiante.

Outro aspecto importante é o local onde usamos nossos objetos. Normalmente, não criamos métodos main em classes que visam apenas representar entidades, como a [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Pessoa que estudamos aqui. Em vez disso, criamos uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) apenas para isso, evitando misturar modelo de entidades com código executável. Esse cuidado não é de execução obrigatória, mas ajuda a manter o código organizado.

## **Vamos testar os conhecimentos?**

Considere a [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Funcionario no quadro a seguir. Ela possui dois atributos, nome e salário, e dois métodos, aplicarBonusAoSalario e exibirDadosDoFuncionario. O método aplicarBonusAoSalario recebe como parâmetro um valor a ser adicionado ao salário do funcionário e o método exibirDadosDoFuncionario mostra seus dados.

[public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) class Funcionario{

[public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) static void main(String[] args) {

String nome;

double salario;

/\* Adiciona um valor ao salário base do funcionário \*/

[public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) void aplicarBonusAoSalario(double valor){

salario += valor;

}

[public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) void exibirDadosFuncionário(){

System.out.println("Funcionario" +nome);

System.out.println("Salário Atual: R$" +salario);

}

}

}

Agora, considere os trechos soltos da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) TesteFuncionario a seguir. Reorganize as linhas de código do método main de forma a criar um objeto do tipo “funcionário” cujo atributo nome seja "Leonardo", tenha salário igual a R$ 1500 e que receberá um bônus de R$ 350. Depois, os dados desse funcionário devem ser exibidos no console.

[public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) class TesteFuncionario{

[public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) static void main(String[] args) {

funcionario01.exibirDadosFuncionario();

funcionario01.salario = 1500;

funcionario01.aplicarBonusAoSalario(350).

Funcionario funcionario01 = new Funcionario();

funcionario01.nome = "Leonardo";

}

}

## **Encapsulamento**

No tópico anterior, estudamos sobre a construção da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Pessoa. Nela, definimos dois atributos: nome e idade. Esses atributos têm visibilidade padrão, também chamada de visibilidade [default](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=default). Isso significa que esses atributos podem ser acessados por objetos de [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe)s que pertençam ao mesmo pacote da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Pessoa ou de sub[classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe)s de Pessoa. Sub[classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe)s são [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe)s "filhas", geradas com base em uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) e que aproveitam atributos e métodos de sua [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) "pai". Estudaremos mais sobre a relação entre [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe)s nas próximas aulas. Já os métodos Apresentar, Trabalhar e Comer estavam marcados com a palavra-chave [public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public), o que significa que esses métodos são visíveis (acessíveis) para qualquer objeto fora da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Pessoa.

Pacotes são como pastas que contêm uma ou mais classes. Utilizamos pacotes para proteger atributos de nossas classes ou mesmo para organizá-las por funcionalidade dentro de um programa, já que um mesmo programa pode conter diversas classes. Estudaremos mais sobre pacotes em aulas mais adiante.

Porém, por questões de segurança ou organização, é preferível limitar o acesso a métodos e atributos de uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe). Essa limitação é feita através de palavras-chave chamadas de modificadores de acesso. A partir desse ponto, para facilitar a descrição, chamaremos atributos e métodos de membros. Existem quatro níveis de visibilidade (acesso) para membros de uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe), resumidos a seguir:

* [default](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=default): qualquer objeto do mesmo pacote e quaisquer subclasses de outros pacotes podem acessar o membro. Não deve ser usada a palavra-chave [default](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=default) antes do membro para definir esse nível de visibilidade, mas simplesmente para omitir qualquer um dos modificadores de acesso a seguir;
* [public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public): qualquer objeto pode acessar o membro;
* [protected](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=protected): o membro é acessível apenas por objetos do mesmo pacote;
* [private](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=private): o membro é acessível apenas internamente (dentro da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe)).

## **Uso de Getters e Setters**

Com exceção do modificador [public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public), todos os demais níveis de visibilidade criam algum tipo de limitação no acesso aos membros da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe). Assim, temos membros de segurança contra acessos indesejados, mas perdemos a flexibilidade de usá-los em qualquer lugar. Para complementar o trabalho dos modificadores de acesso sobre atributos e devolver o acesso a eles de forma regrada, existem os métodos Getters e Setters. Suas funcionalidades podem ser resumidas da seguinte maneira:

Getters são métodos que permitem a leitura do valor de atributos não públicos. São métodos públicos e seu nome é formado por Get, seguido do nome do atributo que se deseja acessar. Não recebem parâmetros e, simplesmente, devolvem o valor do atributo alvo através da palavra-chave return.

Setters são métodos que permitem a modificação de valor de atributos não públicos. Eles são métodos públicos e seu nome é formado por Set, seguido do nome do atributo que se deseja modificar. Recebem apenas um parâmetro, que deve ser do mesmo tipo do atributo alvo. O atributo alvo deve receber o valor do parâmetro dentro do método. Métodos Set não têm retorno, ou seja, são do tipo void.

[public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) class Aluno{

[private](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=private) String nome;

[private](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=private) float media;

[public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) String getNome() {

return nome;

}

[public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) void setNome(String nome) {

this.nome = nome;

}

[public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) float getMedia() {

return media;

}

[public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) void setMedia(float media) {

if(media >= 0 && media <= 10){

this.media = media;

}

}

}

Perceba que, no método setMedia, injetamos uma regra para manter a consistência do valor de média do aluno, que não pode ser menor que zero nem maior que dez. Assim, estamos restringindo a possibilidade de modificação da média para valores absurdos. Os demais métodos Get e Set apenas seguem o modelo padrão. É uma prática comum definirmos todos os atributos de uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) como privados (usando [private](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=private)) e criarmos métodos Get e Set para cada um deles.

**OBS:**

Em Java, temos algumas cláusulas que podem ter vários efeitos, dependendo de onde ela for aplicada. Por exemplo: static indica que um método ou objeto pode ser utilizado sem a necessidade de instanciação de uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe); [public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) indica que um membro pode ser utilizado em qualquer escopo etc.

A cláusula this é utilizada para referenciar o próprio objeto instanciado da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe). No exemplo anterior, temos a [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Aluno que pode ser instanciada por vários objetos. Imagine os objetos aluno1, aluno2 e aluno3 sendo instâncias da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Aluno. Então, para o objeto aluno1, this significa o objeto aluno1, para aluno2, this significa aluno2 e, para aluno3, this significa aluno3. A partir da chamada desta cláusula, os métodos, ou atributos, são chamados, evitando uma confusão entre objetos. Assim, para aluno1, a chamada this.nome referencia o atributo nome do objeto aluno1, e assim sucessivamente.

## [**Método Construtor**](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=M%C3%A9todo%20Construtor)

Os Métodos Construtores são aqueles que permitem a instanciação de objetos de uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) quando executados em conjunto com a palavra-chave “new”. Esses métodos devem ter o mesmo nome da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) à qual pertencem e não possuem um tipo de retorno explícito, ou seja, escrito na sua assinatura. Implicitamente, o tipo de retorno de um [método construtor](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=m%C3%A9todo%20construtor) é o mesmo tipo da sua [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe). Uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) pode ter diversos construtores, desde que eles não possuam os mesmos tipos de parâmetros na mesma sequência. Mesmo que não seja definido um [método construtor](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=m%C3%A9todo%20construtor) para uma [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe), a linguagem Java cria um [método construtor](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=m%C3%A9todo%20construtor) interno chamado de [default](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=default), com corpo vazio e que não recebe argumentos. Esse método [default](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=default) inicializa todos os atributos da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) com valores padrão (como zero para atributos numéricos, falso para atributos booleanos etc.) e nulo para objetos dentro da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe).

No exemplo a seguir, temos a [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Automovel, que possui dois construtores.

No exemplo da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Automovel, temos dois métodos construtores. Perceba que eles são diferentes, pois possuem uma lista diferente de parâmetros. É importante que você compreenda que não é o nome dos atributos que faz a diferença, mas sim o seu tipo. Isso significa que, se tentássemos criar um novo construtor que recebesse como parâmetro apenas um String (como o da linha 8) ou uma sequência de String > inteiro > String (como o da linha 12), independentemente dos nomes dos atributos, eles seriam considerados duplicados e a compilação não seria executada com sucesso.  
   
Outro detalhe importante é a palavra-chave “this”. Usamos “this” para que o compilador Java saiba a qual atributo estamos nos referenciando quando existirem dois deles com o mesmo nome. Perceba que, no [método Construtor](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=m%C3%A9todo%20Construtor), da linha 8, é recebido um parâmetro do tipo String, chamado Nome. Mas a [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Automovel já possui um atributo com esse mesmo nome e com esse mesmo tipo. Usamos “this.nome” para referenciar ao atributo da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) e apenas “nome” para referenciar o parâmetro do método.  
   
Podemos adicionar os métodos Get e Set para a [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Automovel. Confira, na tela a seguir, como devem ficar.

Lembre-se de que o construtor padrão só existe de forma automática quando a [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) não possui nenhum construtor em seu código, por exemplo, para a [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Pessoa do início da aula. No caso da [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) Automovel, você seria obrigado a usar um dos dois construtores presentes nela ou, se preferir, usar o construtor padrão e adicioná-lo à [classe](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=classe) explicitamente da seguinte forma: [public](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=public) Automovel(){ }.