

Encontro de Tecnologia da Informação

Conceitos de Orientação a Objetos Implementados em Java

Prof. MSc. Vinícius Camargo Andrade

vcandrade@utfpr.edu.br

Departamento Acadêmico de Informática
Universidade Tecnológica Federal do Paraná



Vinícius Camargo Andrade

Professor do Magistério Superior



/vcandrade



/Vinicius_Camargo_Andrade



/prof-vcandrade

Apresentação Pessoal

- *Nome?*
- *Curso?*
- *Período/Ano?*
- *Experiência com que Linguagem de Programação?*
- *Experiência com Orientação a Objeto?*

Paradigmas de Programação

Paradigmas de Programação

*“Paradigma de programação é a forma como o programador enxerga a **solução do problema**.”*

Paradigmas de Programação

É possível escrever programas de maneiras diferentes, a partir de diferentes visões da solução de um problema.

Paradigmas de Programação

*Linguagens de programação são escritas para
suportar um ou mais paradigmas.*

Paradigmas de Programação

Exemplos:

- *Lógico*
 - *Linguagem Prolog*
- *Funcional*
 - *Linguagem Lisp*
- *Orientado a Objetos,*
 - *Linguagem Java, C++, C#, Smalltalk*

Paradigma Orientado a Objetos

Paradigma Orientado a Objetos

Surgiu para solucionar problemas existentes no desenvolvimento de softwares complexos

- *Baixo custo de desenvolvimento e manutenção.*

Paradigma Orientado a Objetos

Inspiração: mundo real é formado por objetos.

Paradigma Orientado a Objetos

Ideia: representar os objetos do mundo real em um software.

- *Mais natural*
- *Facilidade em representar as suas funcionalidades*

Sistema Orientado a Objetos

Sistema Orientado a Objetos

Um sistema orientado a objetos é uma coleção de objetos que interagem entre si por meio de mensagens.

Sistema Orientado a Objetos

*O termo OO significa organizar o mundo real como uma coleção de objetos que incorporam **estrutura de dados e comportamento**.*

Objetivos da Orientação a Objetos

Objetivos da Orientação a Objetos

Natural: programas naturais são mais inteligíveis;

Objetivos da Orientação a Objetos

Reutilizável: classes e objetos podem ser reutilizados;

Objetivos da Orientação a Objetos

Manutenível: a natureza modular dos objetos facilita a detecção e correção de erros;

Objetivos da Orientação a Objetos

Extensível: a Programação Orientada a Objeto (POO) oferece vários recursos para estender código. Dentre eles, herança, polimorfismo, sobreposição, delegação, e alguns padrões de projeto;

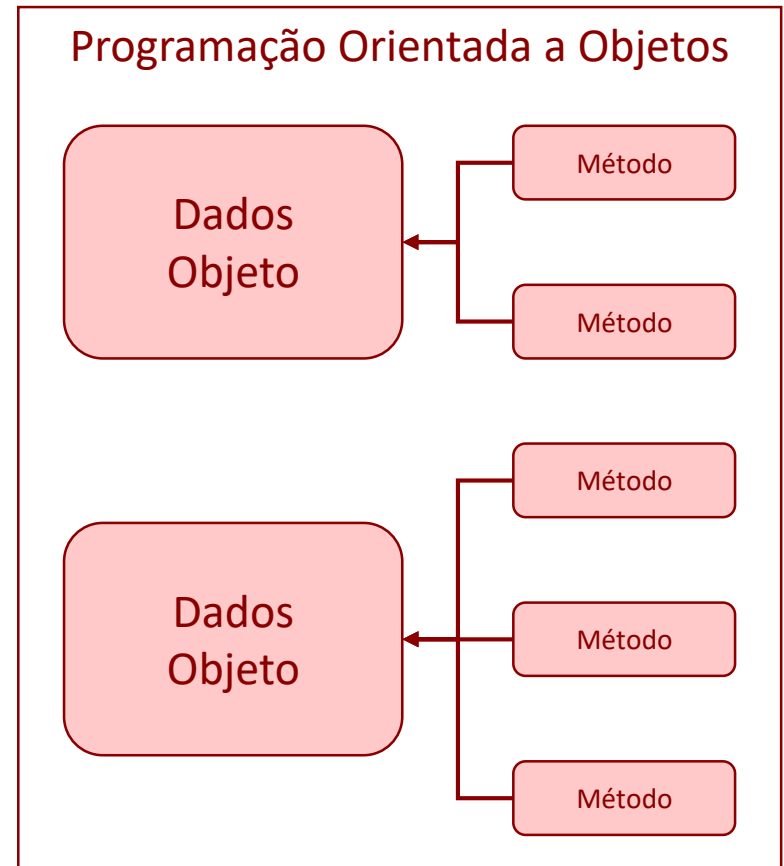
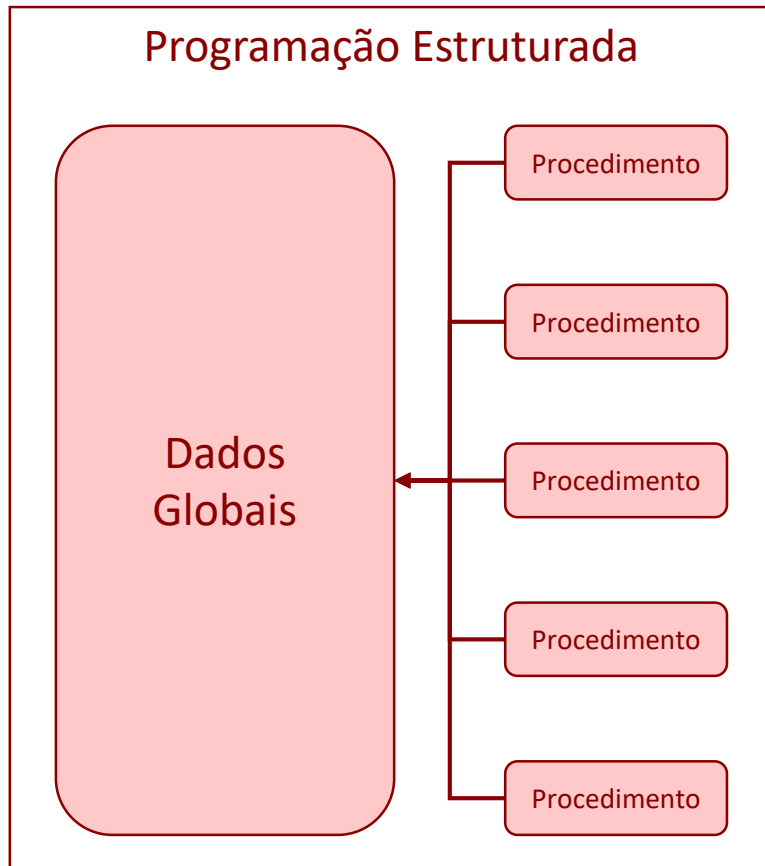
Vantagens da Programação Orientada a Objetos

Vantagens

- *Aumento de produtividade*
- *Reuso de código*
- *Redução das linhas de código programadas*
- *Separação de responsabilidades*
- *Divisão em módulos*
- *Maior flexibilidade do sistema*
- *Escalabilidade*
- *Facilidade na manutenção, dentre outras vantagens.*

Diferenças

Diferenças



Pilares da Orientação a Objetos

Pilares da Orientação a Objetos

Uma Linguagem Orientada a Objetos possui os seguintes tópicos:

- *Abstração;*
- *Encapsulamento;*
- *Herança;*
- *Polimorfismo.*



Pilares da Orientação a Objetos

Uma Linguagem Orientada a Objetos possui os seguintes tópicos:

- *Abstração;*
- *Encapsulamento;*
- *Herança;*
- *Polimorfismo.*



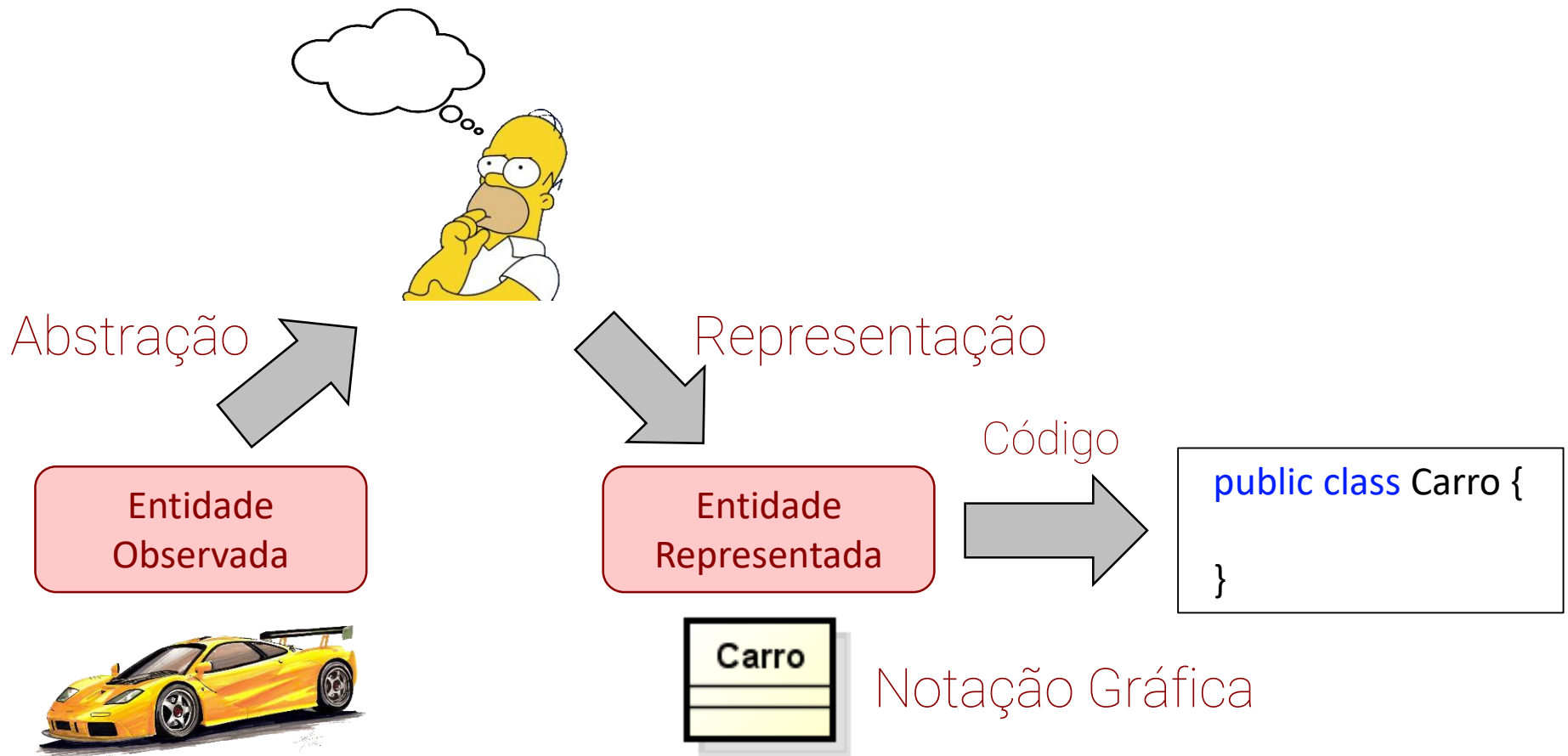
Abstração

Pilares da Orientação a Objetos

Abstração

Abstração é uma operação intelectual que consiste em isolar, um aspecto determinado de um estado de coisas relativamente complexo, a fim de simplificar a sua avaliação classificação ou para permitir a comunicação do mesmo.

Modelagem Conceitual



Classe

Classe

*Classe é uma estrutura que abstrai um conjunto de objetos com **características** e **comportamentos** similares.*

Classe



Características:

Modelo: Ford GT

Marca: Ford

Ano: 2015

Placa: AAA-1234

Cor: Amarelo

Comportamentos:

Ligar

Desligar

Acelerar

Frear



Características:

Modelo: Fusca

Marca: Volkswagen

Ano: 1970

Placa: MNO-9876

Cor: Vermelho

Comportamentos:

Ligar

Desligar

Acelerar

Frear



Características:

Modelo: Chevette

Marca: Chevrolet

Ano: 1983

Placa: XYZ-5555

Cor: Laranja

Comportamentos:

Ligar

Desligar

Acelerar

Frear

Classe

Uma classe é composta por:

- *Identificação;*
- *Atributos;*
- *Métodos.*

Identificação

É o nome da classe.

Atributos

São condições individuais que diferenciam um objeto de outro e determinam a aparência, estado, ou outras qualidades de um objeto.



Modelo: Ford GT
Marca: Ford
Ano: 2015
Placa: AAA-1234
Cor: Amarelo



Modelo: Fusca
Marca: Volkswagen
Ano: 1970
Placa: MNO-9876
Cor: Vermelho



Modelo: Chevette
Marca: Chevrolet
Ano: 1983
Placa: XYZ-5555
Cor: Laranja

Métodos

São procedimentos que formam os comportamentos e serviços oferecidos por objetos de uma classe.



Ligar
Desligar
Acelerar
Frear

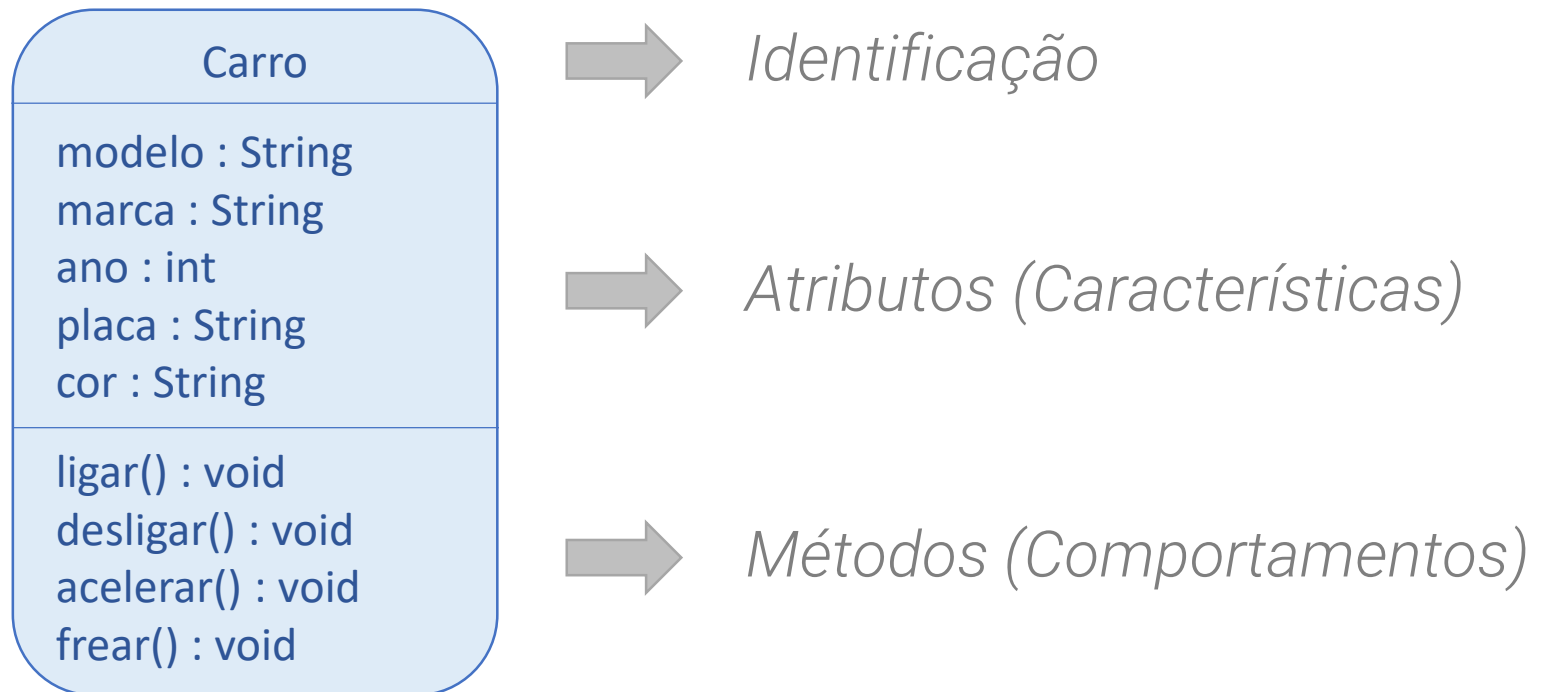


Ligar
Desligar
Acelerar
Frear

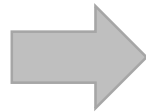
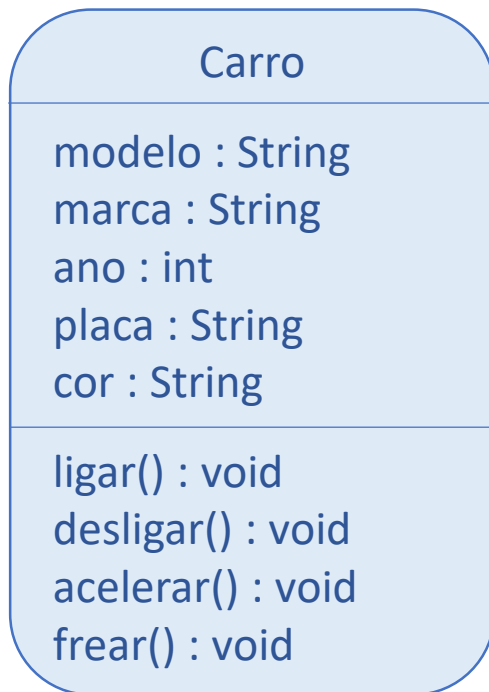


Ligar
Desligar
Acelerar
Frear

Entidade Representada



Implementação em Java (1)



```
public class Carro {  
  
    String modelo;  
    String marca;  
    int ano;  
    String placa;  
    String cor;  
  
    public void ligar() {  
  
    }  
  
    public void desligar() {  
  
    }  
  
    public void acelerar() {  
  
    }  
  
    public void frear() {  
  
    }  
  
}
```

Exercício 1

Exercício 1

- Crie a classe *Carro*.
- Declare os atributos: *modelo* (String), *marca* (String), *ano* (int), *placa* (String), *cor* (String), *ligado* (boolean) e *velocidade* (int).
- Declare os métodos: *ligar*, *desligar*, *acelerar* e *frear*. Todos os métodos *void*.

Objetos / Instâncias

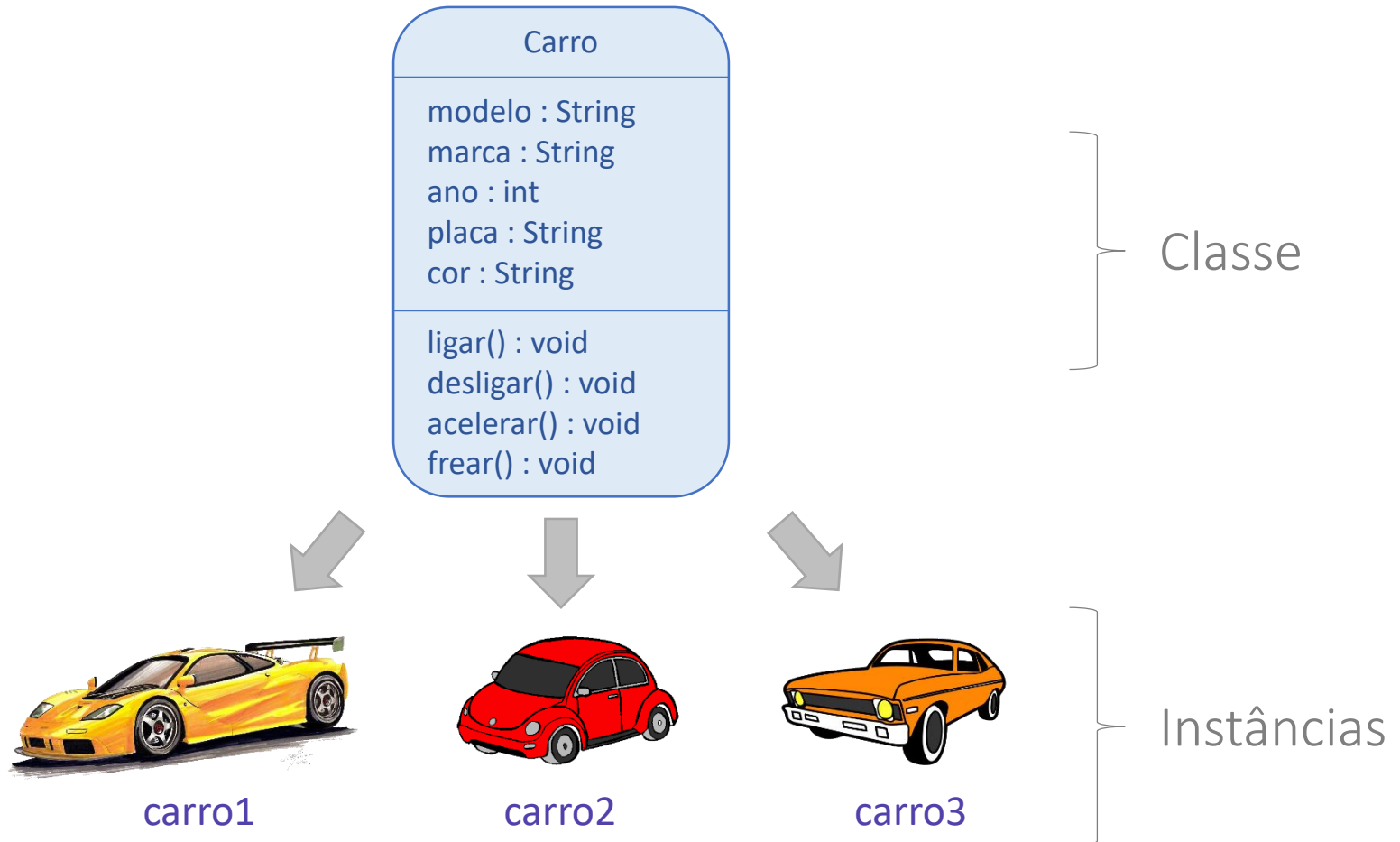
Instâncias

Instâncias são representações concretas de uma classe.

Instâncias

*Instâncias compartilham o mesmo conjunto de **atributos**, embora sejam diferentes quanto ao conteúdo.*

Instâncias



Implementação em Java (2)

Para instanciar um novo objeto em Java, a seguinte estrutura é utilizada:

```
Carro carro1 = new Carro();
```

```
Carro carro2 = new Carro();
```

```
Carro carro3 = new Carro();
```

Implementação em Java (2)

Carro: tipo do objeto que será declarado.

```
Carro carro1 = new Carro();  
Carro carro2 = new Carro();  
Carro carro3 = new Carro();
```

Implementação em Java (2)

carro1: nome do objeto que será instanciado.

```
Carro carro1 = new Carro();
```

```
Carro carro2 = new Carro();
```

```
Carro carro3 = new Carro();
```


Implementação em Java (2)

***new:** palavra reservada utilizada para alocar espaço em memória onde será armazenado o objeto.*

```
Carro carro1 = new Carro();
```

```
Carro carro2 = new Carro();
```

```
Carro carro3 = new Carro();
```

Implementação em Java (2)

Carro(): método construtor da classe que será instanciada.

```
Carro carro1 = new Carro();
```

```
Carro carro2 = new Carro();
```

```
Carro carro3 = new Carro();
```

Exercício 2

Exercício 2

- Crie uma classe *CarroTeste*
- Declare o método *main*, nesta classe.
- Instancie três carros: dê os nomes de *carro1*, *carro2* e *carro3*.

Métodos Construtores

Método Construtor

*Um objeto também deve ser inicializado, e sua inicialização se dá por meio de um método, chamado **método construtor**, que é executado para preparar os dados do objeto.*

Método Construtor

Características:

- Deve possuir o *mesmo nome* da classe;
- *Não* possui valor de retorno definido.

Método Construtor

Finalidade:

- *Inicializar os valores das variáveis-membro da classe, para o objeto criado.*

Ponteiro de Auto Referência

this

- *Ponteiro de **auto referência** para o objeto;*
- *Todas as linguagens Orientadas a Objetos possuem essa referência (algumas utilizam **self**);*
- *Diferencia os **atributos** de **variáveis locais**.*

```
public class Carro {

    String modelo;
    String marca;
    int ano;
    String placa;
    String cor;
    boolean ligado;
    int velocidade;

    public Carro(String modelo, String marca, int ano, String placa, String cor) {

        this.modelo = modelo;
        this.marca = marca;
        this.ano = ano;
        this.placa = placa;
        this.cor = cor;
        this.ligado = false;
        this.velocidade = 0;
    }

    public void ligar() {

    }

    public void desligar() {

    }

    public void acelerar() {

    }

    public void frear() {

    }

}
```

```
public class Carro {  
  
    String modelo;  
    String marca;  
    int ano;  
    String placa;  
    String cor;  
    boolean ligado;  
    int velocidade;  
  
    public Carro(String modelo, String marca, int ano, String placa, String cor) {  
  
        this.modelo = modelo;  
        this.marca = marca;  
        this.ano = ano;  
        this.placa = placa;  
        this.cor = cor;  
        this.ligado = false;  
        this.velocidade = 0;  
    }  
  
    public void ligar() {  
  
    }  
  
    public void desligar() {  
  
    }  
  
    public void acelerar() {  
  
    }  
  
    public void frear() {  
  
    }  
}
```

```
public class Carro {

    String modelo;
    String marca;
    int ano;
    String placa;
    String cor;
    boolean ligado;
    int velocidade;

    public Carro(String modelo, String marca, int ano, String placa, String cor) {

        this.modelo = modelo;
        this.marca = marca;
        this.ano = ano;
        this.placa = placa;
        this.cor = cor;
        this.ligado = false;
        this.velocidade = 0;
    }

    public void ligar() {

    }

    public void desligar() {

    }

    public void acelerar() {

    }

    public void frear() {

    }

}
```

```
public class Carro {  
  
    String modelo;  
    String marca;  
    int ano;  
    String placa;  
    String cor;  
    boolean ligado;  
    int velocidade;  
  
    public Carro(String modelo, String marca, int ano, String placa, String cor) {  
  
        this.modelo = modelo;  
        this.marca = marca;  
        this.ano = ano;  
        this.placa = placa;  
        this.cor = cor;  
        this.ligado = false;  
        this.velocidade = 0;  
    }  
  
    public void ligar() {  
  
    }  
  
    public void desligar() {  
  
    }  
  
    public void acelerar() {  
  
    }  
  
    public void frear() {  
  
    }  
}
```

A red arrow points from the 'String modelo;' line to the 'this.modelo = modelo;' line. A red box highlights the 'this.modelo = modelo;' line, and another red arrow points from this box to the 'this.marca = marca;' line. This diagram illustrates the flow of variable assignment within the Carro constructor.

```
public class Carro {

    String modelo;
    String marca;
    int ano;
    String placa;
    String cor;
    boolean ligado;
    int velocidade;

    public Carro(String modelo, String marca, int ano, String placa, String cor) {

        this.modelo = modelo;
        this.marca = marca;
        this.ano = ano;
        this.placa = placa;
        this.cor = cor;
        this.ligado = false;
        this.velocidade = 0;
    }

    public void ligar() {

    }

    public void desligar() {

    }

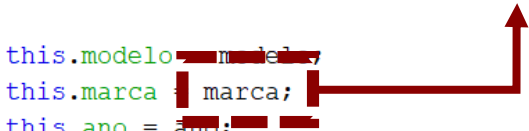
    public void acelerar() {

    }

    public void frear() {

    }

}
```



Implementação em Java (3)

```
public class CarroTeste {  
    public static void main(String[] args) {  
        Carro carro1 = new Carro("Ford GT", "Ford", 2015, "AAA-1234", "Amarelo");  
        Carro carro2 = new Carro("Fusca", "Volkswagen", 1970, "MNO-9876", "Vermelho");  
        Carro carro3 = new Carro("Chevette", "Chevrolet", 1983, "XYZ-5555", "Laranja");  
    }  
}
```

Exercício 3

Exercício 3

- *Implemente na classe **Carro** um método construtor que recebe os parâmetros: **modelo** (String), **marca** (String), **ano** (int), **placa** (String) e **cor** (String);*
- *Ainda no método construtor, atribua cada variável ao seu respectivo atributo;*
- *Para o atributo **ligado**, inicie-o com o valor **false**;*
- *Para o atributo **velocidade**, inicie-o com o valor **0**;*

Exercício 3

*Na classe **CarroTeste**, instancie três carros passando por parâmetro para o método construtor os valores dos atributos necessários.*

Pilares da Orientação a Objetos

Uma Linguagem Orientada a Objetos possui os seguintes tópicos:

- *Abstração;*
- *Encapsulamento;*
- *Herança;*
- *Polimorfismo.*



Pilares da Orientação a Objetos

Uma Linguagem Orientada a Objetos possui os seguintes tópicos:

- *Abstração;*
- *Encapsulamento;*
- *Herança;*
- *Polimorfismo.*



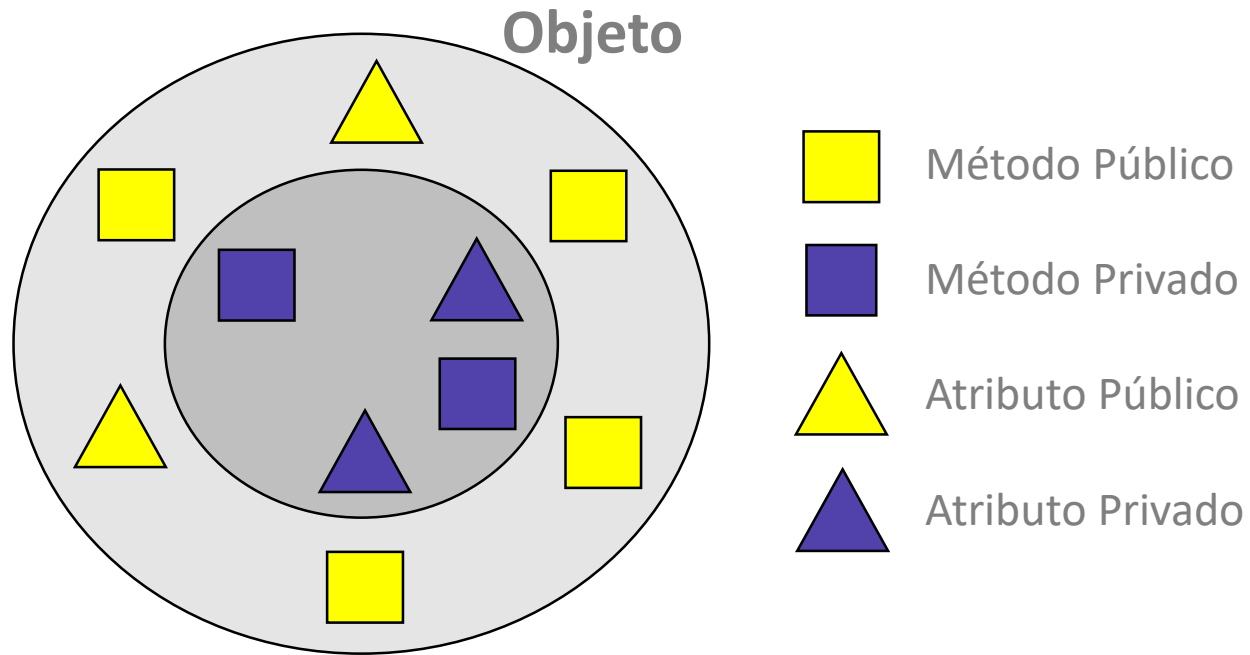
Encapsulamento de Dados

Encapsulamento de Dados

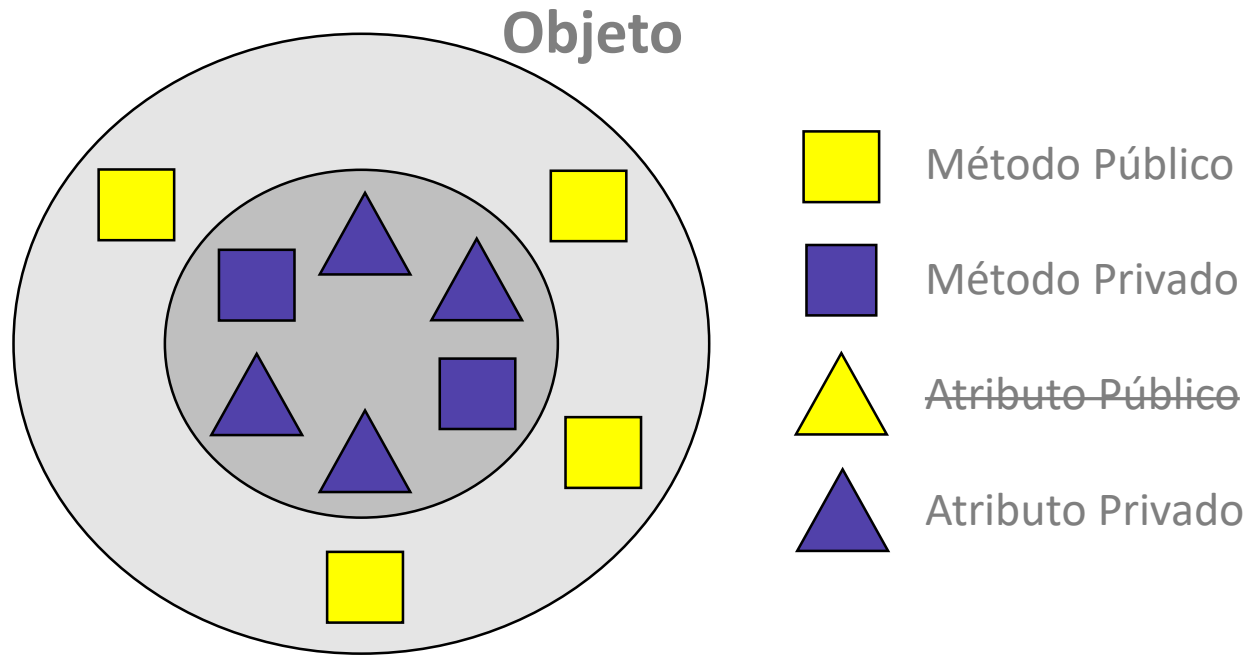
Técnica de esconder a estrutura de um objeto atrás de um conjunto de operações sobre ele, sendo acessível somente por meio destas operações



Encapsulamento de Dados



Encapsulamento de Dados



Modificadores de Acesso

Encapsulamento de Dados

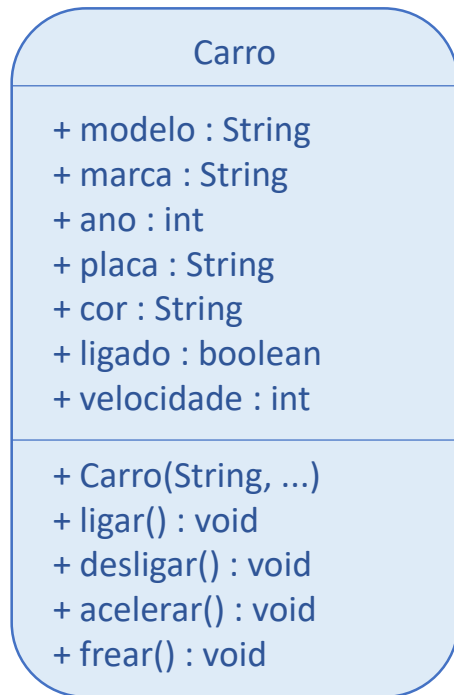
Modificadores de Acesso

- *public*
- *protected*
- *private*

public

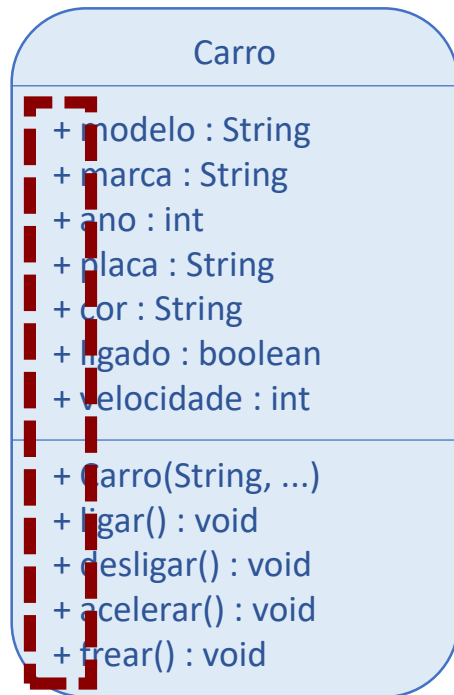
- *Atributos e métodos são **sempre acessíveis** em todos os métodos de todas as classes.*
- *Este é o nível **menos rígido** de encapsulamento, que equivale a não ocultar.*
- *Identificado por **+***

public



```
public class Carro {  
  
    public String modelo;  
    public String marca;  
    public int ano;  
    public String placa;  
    public String cor;  
    public boolean ligado;  
    public int velocidade;  
  
    public Carro(String modelo, String marca, in  
  
        this.modelo = modelo;  
        this.marca = marca;  
        this.ano = ano;  
        this.placa = placa;  
        this.cor = cor;
```

public

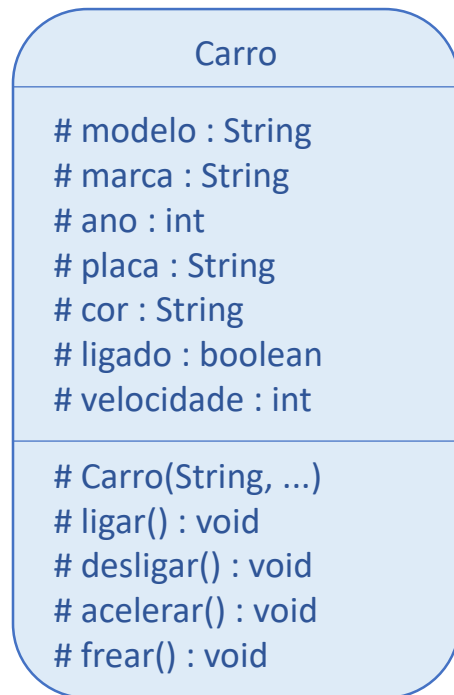


```
public class Carro {  
    public String modelo;  
    public String marca;  
    public int ano;  
    public String placa;  
    public String cor;  
    public boolean ligado;  
    public int velocidade;  
  
    public Carro(String modelo, String marca, in  
  
        this.modelo = modelo;  
        this.marca = marca;  
        this.ano = ano;  
        this.placa = placa;  
        this.cor = cor;
```

protected

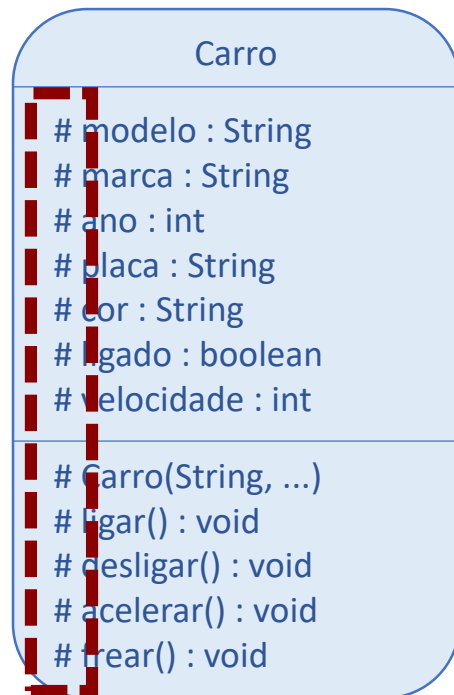
- *Atributos e métodos são acessíveis nos métodos da própria classe e suas subclasses e dentro do mesmo pacote.*
- *Identificados por #*

protected



```
public class Carro {  
  
    protected String modelo;  
    protected String marca;  
    protected int ano;  
    protected String placa;  
    protected String cor;  
    protected boolean ligado;  
    protected int velocidade;  
  
    protected Carro(String modelo, String marca, in  
  
        this.modelo = modelo;  
        this.marca = marca;  
        this.ano = ano;  
        this.placa = placa;  
        this.cor = cor;  
        this.ligado = false;  
        this.velocidade = 0;  
    }  
}
```

protected

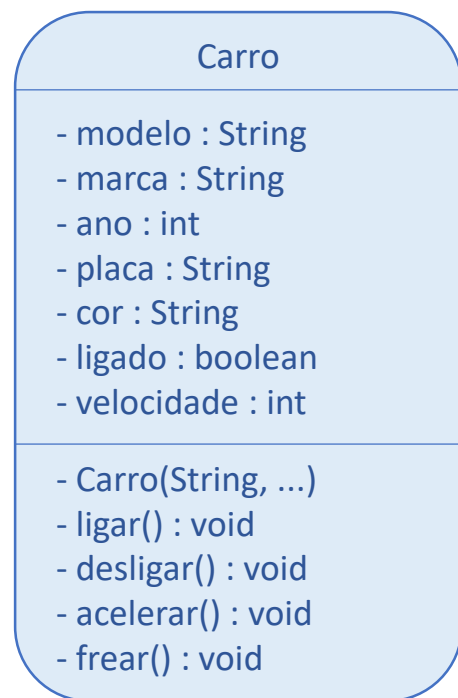


```
public class Carro {  
    protected String modelo;  
    protected String marca;  
    protected int ano;  
    protected String placa;  
    protected String cor;  
    protected boolean ligado;  
    protected int velocidade;  
  
    protected Carro(String modelo, String marca, in  
  
        this.modelo = modelo;  
        this.marca = marca;  
        this.ano = ano;  
        this.placa = placa;  
        this.cor = cor;  
        this.ligado = false;  
        this.velocidade = 0;  
    }  
}
```


private

- *atributos e métodos são **acessíveis** somente nos métodos (todos) da própria classe.*
- *Este é o nível **mais rígido** de encapsulamento.*
- *Identificado por –*

private



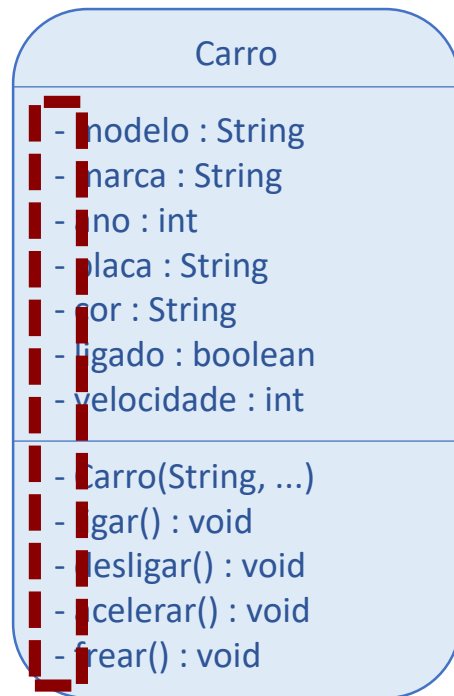
```
public class Carro {

    private String modelo;
    private String marca;
    private int ano;
    private String placa;
    private String cor;
    private boolean ligado;
    private int velocidade;

    private Carro(String modelo, String marca, in

        this.modelo = modelo;
        this.marca = marca;
        this.ano = ano;
        this.placa = placa;
        this.cor = cor;
```

private



```
public class Carro {  
    private String modelo;  
    private String marca;  
    private int ano;  
    private String placa;  
    private String cor;  
    private boolean ligado;  
    private int velocidade;  
  
    private Carro(String modelo, String marca, in  
        this.modelo = modelo;  
        this.marca = marca;  
        this.ano = ano;  
        this.placa = placa;  
        this.cor = cor;
```

Métodos de Acesso

Encapsulamento de Dados

Métodos de Acesso

Os atributos de um objeto só podem ser acessados por métodos da mesma classe

Métodos de Acesso

- *Getters*
- *Setters*

Getters

Métodos utilizados para obter o valor de uma variável

```
public class Carro {  
  
    private String modelo;  
    private int ano;  
  
    public String getModelo() {  
        return modelo;  
    }  
  
    public int getAno() {  
        return ano;  
    }  
}
```

Setters

Métodos utilizados para atribuir um valor a uma variável

```
public class Carro {  
  
    private String modelo;  
    private int ano;  
  
    public void setModelo(String modelo) {  
        this.modelo = modelo;  
    }  
  
    public void setAno(int ano) {  
        this.ano = ano;  
    }  
}
```


Implementação em Java (4)

```
public class Carro {  
  
    private String modelo;  
    private int ano;  
  
    public String getModelo() {  
        return modelo;  
    }  
  
    public void setModelo(String modelo) {  
  
        this.modelo = modelo;  
    }  
  
    public int getAno() {  
        return ano;  
    }  
  
    public void setAno(int ano) {  
        this.ano = ano;  
    }  
}
```

Exercício 4

Exercício 4

- Na classe *Carro*, deixe os atributos com visibilidade *private* e crie métodos de acesso (*setters* e *getters*) para cada um.

Associações

Associações

As associações são os relacionamentos das classes entre si.

Associações

Permite que as classes compartilhem informações entre si e colaborem para a execução dos processos executados pelo sistema.

Associações

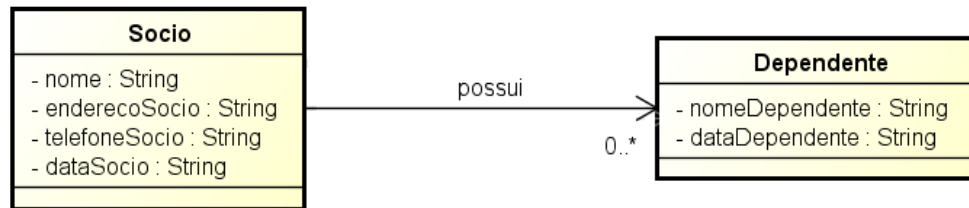
Uma associação descreve um vínculo que ocorre normalmente entre os objetos de uma ou mais classes.

Navegabilidade

Navegabilidade

Representada por uma seta em uma das extremidades da associação, identificando o sentido em que as informações são transmitidas entre os objetos das classes envolvidas.

Navegabilidade



```
public class Socio {

    private String nome;
    private String enderecoSocio;
    private String telefoneSocio;
    private String dataSocio;
    private Dependente [] dependente;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

```
public class Dependente {

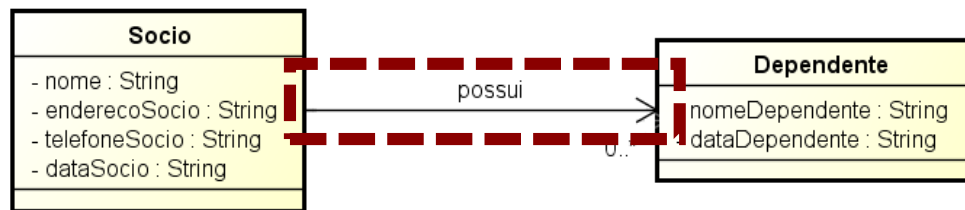
    private String nomeDependente;
    private String dataDependente;

    public String getNomeDependente() {
        return nomeDependente;
    }

    public void setNomeDependente(String nomeDependente) {
        this.nomeDependente = nomeDependente;
    }

    public String getDataDependente() {
        return dataDependente;
    }
}
```

Navegabilidade



```
public class Socio {

    private String nome;
    private String enderecoSocio;
    private String telefoneSocio;
    private String dataSocio;
    private Dependente [] dependente;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

```
public class Dependente {

    private String nomeDependente;
    private String dataDependente;

    public String getNomeDependente() {
        return nomeDependente;
    }

    public void setNomeDependente(String nomeDependente) {
        this.nomeDependente = nomeDependente;
    }

    public String getDataDependente() {
        return dataDependente;
    }
}
```

Multiplicidade de Associações

Multiplicidade de Associações

Especifica quantas instâncias de uma classe pode se relacionar com cada instância de outra classe.

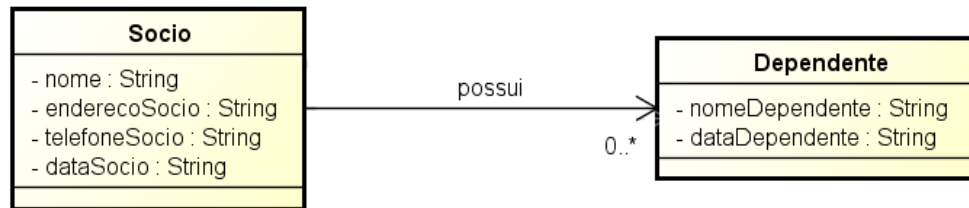
Multiplicidade de Associações

Limita o número na relação entre os objetos.

Multiplicidade de Associações

Geralmente é “um” ou “muitos”, mas pode ser um conjunto finito.

Multiplicidade de Associações



```
public class Socio {

    private String nome;
    private String enderecoSocio;
    private String telefoneSocio;
    private String dataSocio;
    private Dependente [] dependente;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

```
public class Dependente {

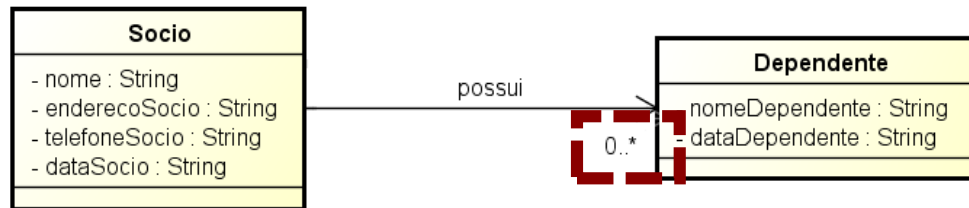
    private String nomeDependente;
    private String dataDependente;

    public String getNomeDependente() {
        return nomeDependente;
    }

    public void setNomeDependente(String nomeDependente) {
        this.nomeDependente = nomeDependente;
    }

    public String getDataDependente() {
        return dataDependente;
    }
}
```


Multiplicidade de Associações



```
public class Socio {

    private String nome;
    private String enderecoSocio;
    private String telefoneSocio;
    private String dataSocio;
    private Dependente [] dependente;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

```
public class Dependente {

    private String nomeDependente;
    private String dataDependente;

    public String getNomeDependente() {
        return nomeDependente;
    }

    public void setNomeDependente(String nomeDependente) {
        this.nomeDependente = nomeDependente;
    }

    public String getDataDependente() {
        return dataDependente;
    }
}
```

Associações

Tipos de Associações:

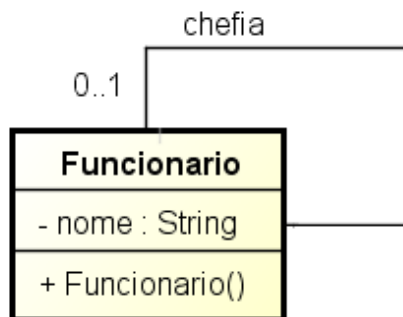
- *Unária ou Reflexiva;*
- *Binária;*
- *Agregação;*
- *Composição;*
- *Generalização/Especialização (Herança);*
- *Classe Associativa;*
- *Dependência;*
- *Realização.*

Associação Unária/Reflexiva

Associações

Associação Unária ou Reflexiva

Ocorre quando existe um relacionamento de um objeto de uma classe com objetos da mesma classe.



```
public class Funcionario {  
  
    private String nome;  
    private Funcionario funcionario;  
  
    public Funcionario() {  
  
    }  
  
    // métodos da classe  
}
```

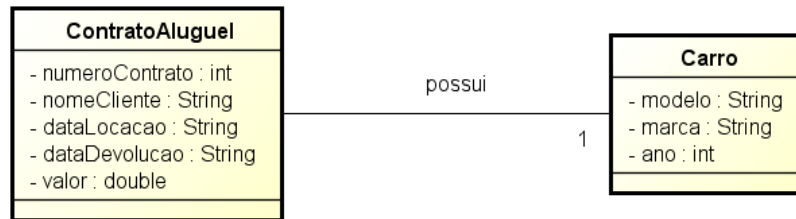
Associação Binária

Associações

Associação Binária

Ocorre quando são identificados relacionamento entre objetos de duas classes distintas.

Associação Binária



```
public class ContratoAluguel {

    private int numeroContrato;
    private String nomeCliente;
    private String dataLocacao;
    private String dataDevolucao;
    private double valor;
    private Carro carro;

    public int getNumeroContrato() {
        return numeroContrato;
    }

    public void setNumeroContrato(int numeroContrato) {
        this.numeroContrato = numeroContrato;
    }
}
```

```
public class Carro {

    private String modelo;
    private String marca;
    private int ano;

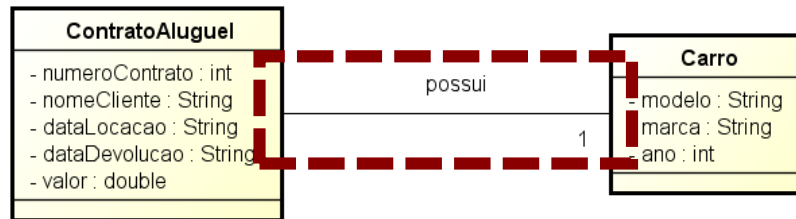
    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }

    public int getAno() {

```

Associação Binária



```
public class ContratoAluguel {

    private int numeroContrato;
    private String nomeCliente;
    private String dataLocacao;
    private String dataDevolucao;
    private double valor;
    private Carro carro;

    public int getNumeroContrato() {
        return numeroContrato;
    }

    public void setNumeroContrato(int numeroContrato) {
        this.numeroContrato = numeroContrato;
    }
}
```

```
public class Carro {

    private String modelo;
    private String marca;
    private int ano;

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }

    public int getAno() {
    }
```


Agregação

Associações

Agregação

É um tipo de associação em que se tenta demonstrar que as informações de um objeto precisam ser complementadas pelas informações contidas em um ou mais objetos de outra classe.

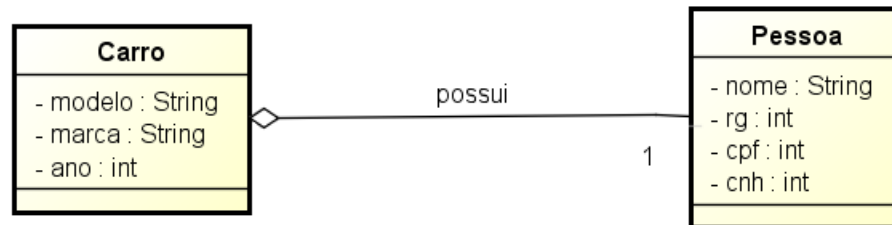
Agregação

Este tipo de associação tenta demonstrar um relação todo/parte entre os objetos associados.

Agregação

Isso significa que a parte de um tipo A está contida em um tipo B, quando esse tem relação de agregação entre eles, porém, essa mesma parte A não existe somente para compor B, essa parte pode agregar outros tipos.

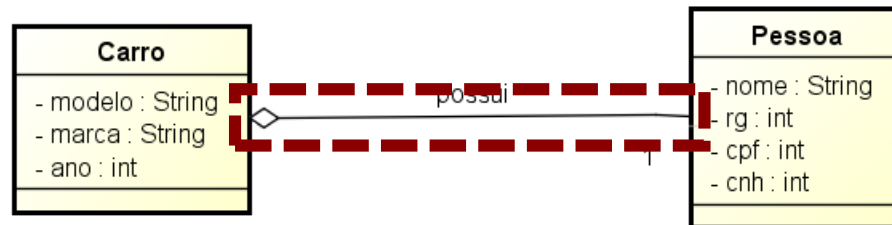
Agregação



```
public class Carro {  
  
    private String modelo;  
    private String marca;  
    private int ano;  
    private Pessoa proprietario;  
  
    public String getModelo() {  
        return modelo;  
    }  
  
    public void setModelo(String modelo) {  
        this.modelo = modelo;  
    }  
}
```

```
public class Pessoa {  
  
    private String nome;  
    private int rg;  
    private int cpf;  
    private int cnh;  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
}
```

Agregação



```
public class Carro {  
  
    private String modelo;  
    private String marca;  
    private int ano;  
    private Pessoa proprietario;  
  
    public String getModelo() {  
        return modelo;  
    }  
  
    public void setModelo(String modelo) {  
        this.modelo = modelo;  
    }  
}
```

```
public class Pessoa {  
  
    private String nome;  
    private int rg;  
    private int cpf;  
    private int cnh;  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
}
```

Composição

Associações

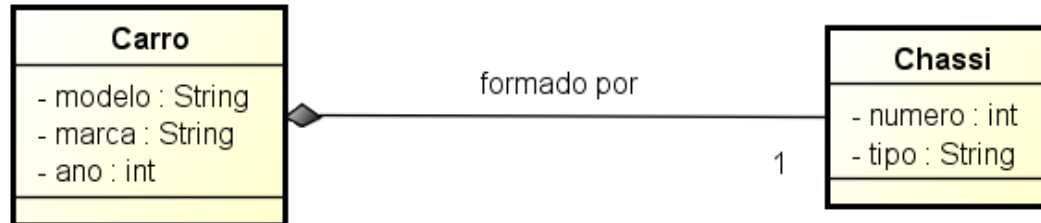
Composição

Associação do tipo composição é uma variação da agregação, porém é apresentado um vínculo mais forte entre os objeto-todo e os objetos-parte, procurando demonstrar que os objetos-parte tem de estar associados a um único objeto-todo.

Composição

Em uma composição os objetos-parte não podem ser destruídos por um objeto diferente do objeto-todo ao qual estão relacionados.

Composição



```
public class Carro {

    private String modelo;
    private String marca;
    private int ano;
    private Chassi chassi;

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }

}
```

```
public class Chassi {

    private int numero;
    private String tipo;

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

}
```

Composição



```
public class Carro {

    private String modelo;
    private String marca;
    private int ano;
    private Chassi chassi;

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }

}
```

```
public class Chassi {

    private int numero;
    private String tipo;

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

}
```

Pilares da Orientação a Objetos

Uma Linguagem Orientada a Objetos possui os seguintes tópicos:

- *Abstração;*
- *Encapsulamento;*
- *Herança;*
- *Polimorfismo.*



Pilares da Orientação a Objetos

Uma Linguagem Orientada a Objetos possui os seguintes tópicos:

- *Abstração;*
- *Encapsulamento;*
- *Herança;*
- *Polimorfismo.*



Herança

Herança

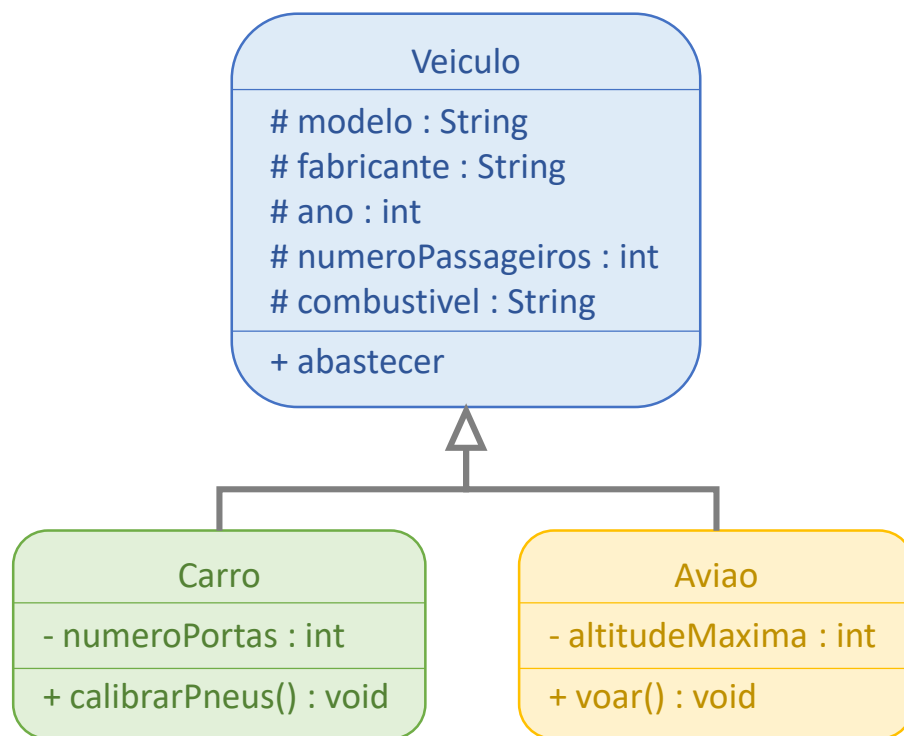
Herança é um princípio de orientação a objetos, que permite que classes compartilhem atributos e métodos

Herança

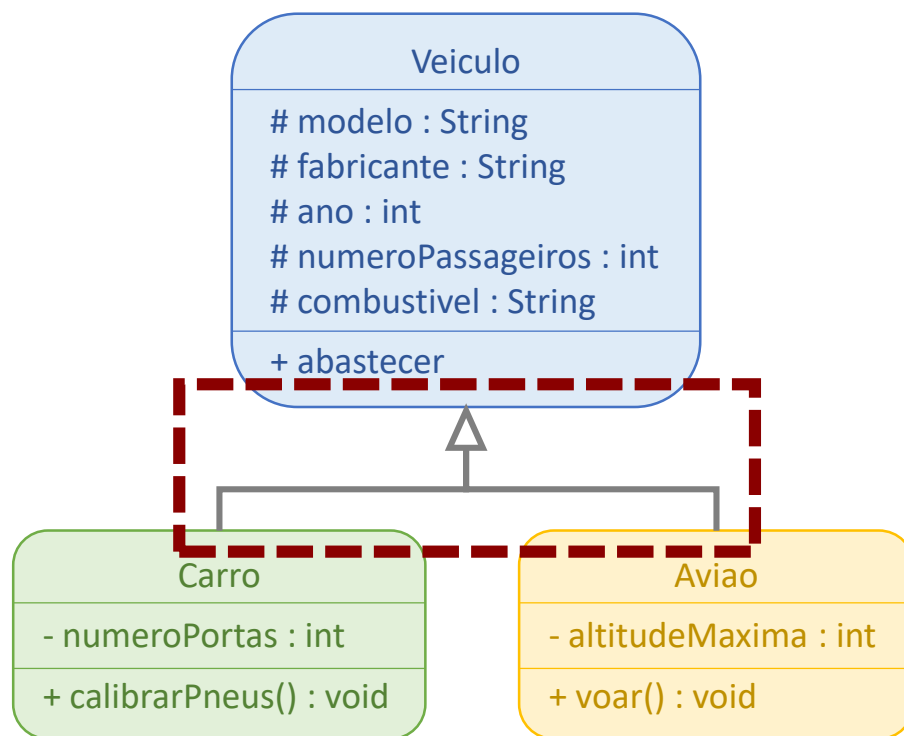
Uma subclasse pode:

- *Utilizar métodos da classe pai;*
- *Executar construtores da classe pai;*
- *Sobrepôr (anular) métodos da classe pai de forma que objetos da classe derivada (subclasse) o utilizem de forma diferente.*
- *Implementar novos códigos nos métodos da subclasse aproveitando o código escrito na classe pai.*

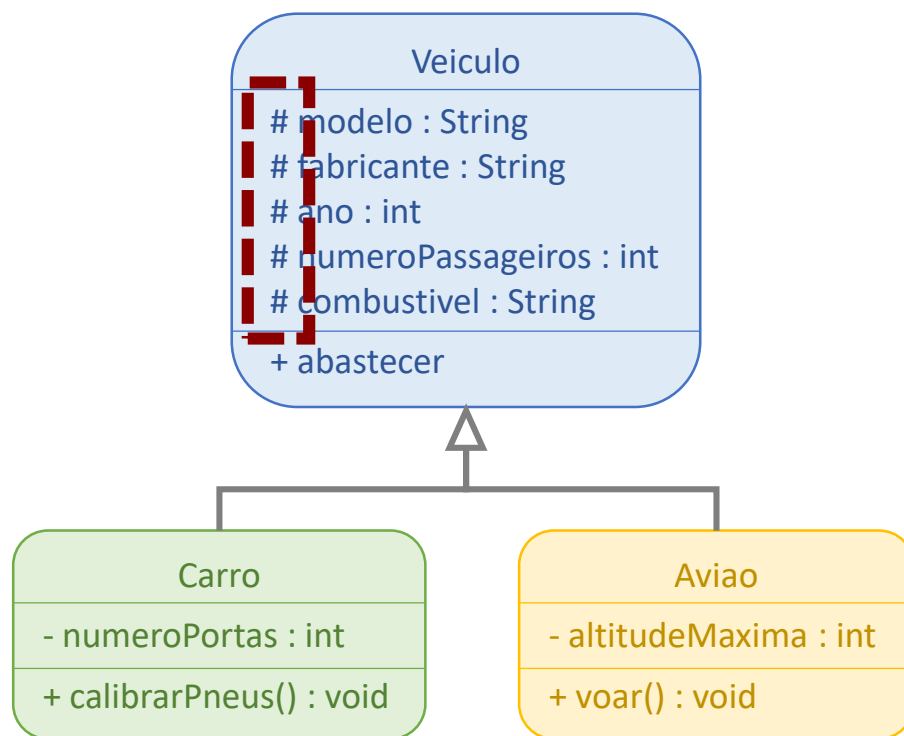
Herança



Herança



Herança



Implementação em Java (5)

*Na linguagem **Java**, para que uma classe herde as características da outra, usa-se a palavra-chave **extends** na assinatura da classe.*

```
public class Veiculo {  
  
    protected String modelo;  
    protected String fabricante;  
    protected int ano;  
    protected int numeroPassageiros;  
    protected String combustivel;  
  
    public void abastecer() {  
  
    }  
  
    public String getModelo() {  
        return modelo;  
    }  
}
```

```
public class Carro extends Veiculo {  
  
    private int numeroPortas;  
  
    public int getNumeroPortas() {  
        return numeroPortas;  
    }  
  
    public void setNumeroPortas(int numeroPortas) {  
        this.numeroPortas = numeroPortas;  
    }  
}
```

```
public class Aviao extends Veiculo {  
  
    private int altitudeMaxima;  
  
    public int getAltitudeMaxima() {  
        return altitudeMaxima;  
    }  
  
    public void setAltitudeMaxima(int altitudeMaxima) {  
        this.altitudeMaxima = altitudeMaxima;  
    }  
}
```

```
public class Veiculo {  
  
    protected String modelo;  
    protected String fabricante;  
    protected int ano;  
    protected int numeroPassageiros;  
    protected String combustivel;  
  
    public void abastecer() {  
  
    }  
  
    public String getModelo() {  
        return modelo;  
    }  
}
```

```
public class Carro extends Veiculo {  
  
    private int numeroPortas;  
  
    public int getNumeroPortas() {  
        return numeroPortas;  
    }  
  
    public void setNumeroPortas(int numeroPortas) {  
        this.numeroPortas = numeroPortas;  
    }  
}
```

```
public class Aviao extends Veiculo {  
  
    private int altitudeMaxima;  
  
    public int getAltitudeMaxima() {  
        return altitudeMaxima;  
    }  
  
    public void setAltitudeMaxima(int altitudeMaxima) {  
        this.altitudeMaxima = altitudeMaxima;  
    }  
}
```

```
public class Veiculo {  
    protected String modelo;  
    protected String fabricante;  
    protected int ano;  
    protected int numeroPassageiros;  
    protected String combustivel;  
  
    public void abastecer() {  
  
    }  
  
    public String getModelo() {  
        return modelo;  
    }  
}
```

```
public class Carro extends Veiculo {  
  
    private int numeroPortas;  
  
    public int getNumeroPortas() {  
        return numeroPortas;  
    }  
  
    public void setNumeroPortas(int numeroPortas) {  
        this.numeroPortas = numeroPortas;  
    }  
}
```

```
public class Aviao extends Veiculo {  
  
    private int altitudeMaxima;  
  
    public int getAltitudeMaxima() {  
        return altitudeMaxima;  
    }  
  
    public void setAltitudeMaxima(int altitudeMaxima) {  
        this.altitudeMaxima = altitudeMaxima;  
    }  
}
```

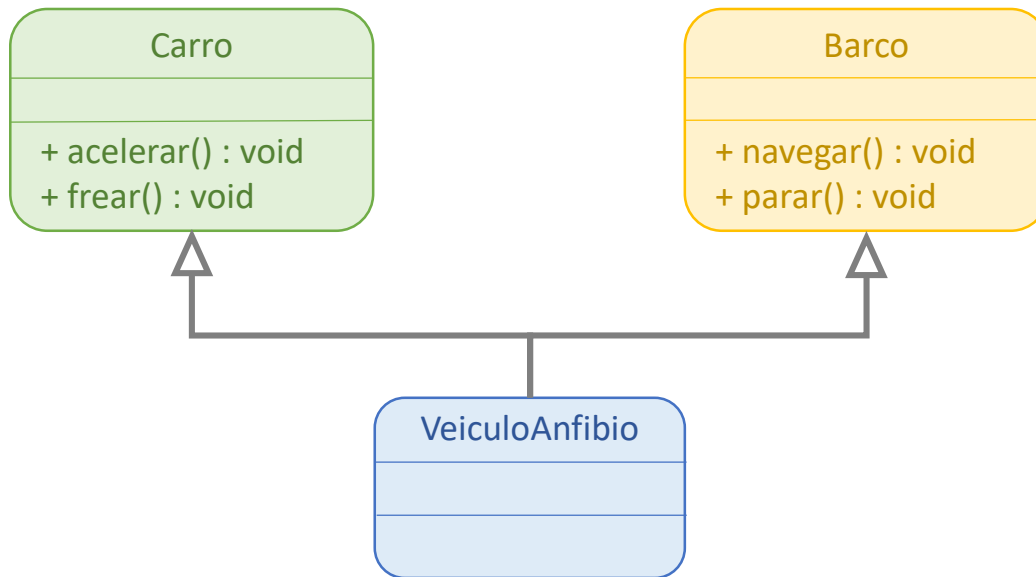
Herança Múltipla

Herança

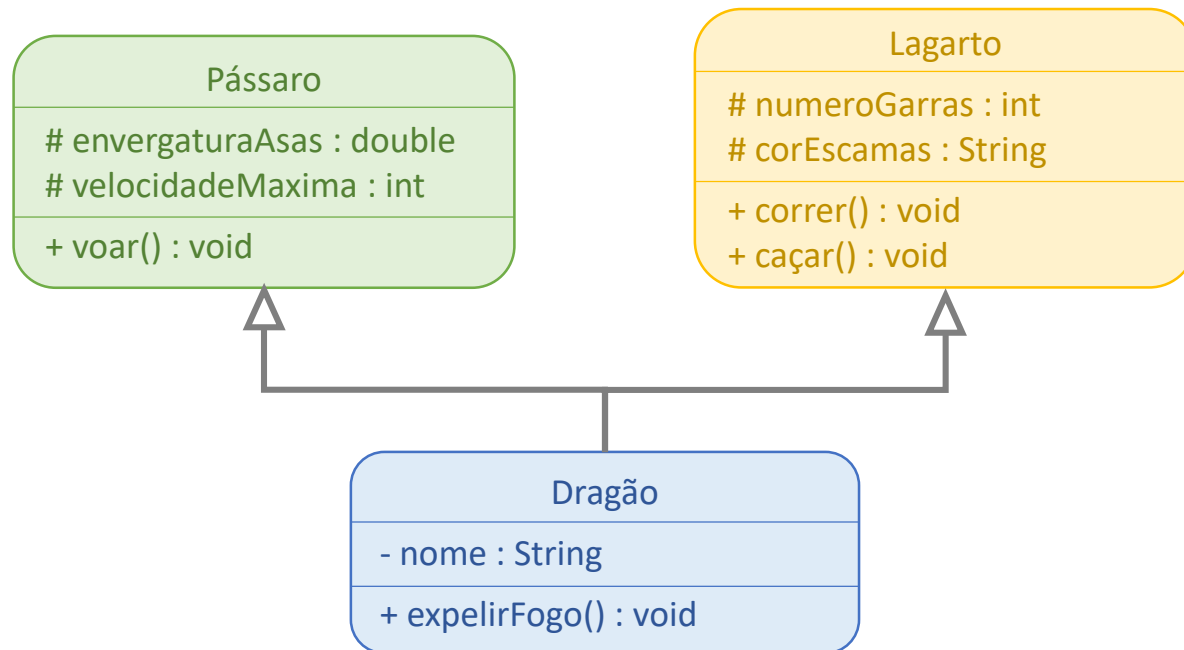
Herança Múltipla

Criação de uma subclasse a partir de mais de uma classe, herdando as características de todas elas.

Herança Múltipla



Herança Múltipla



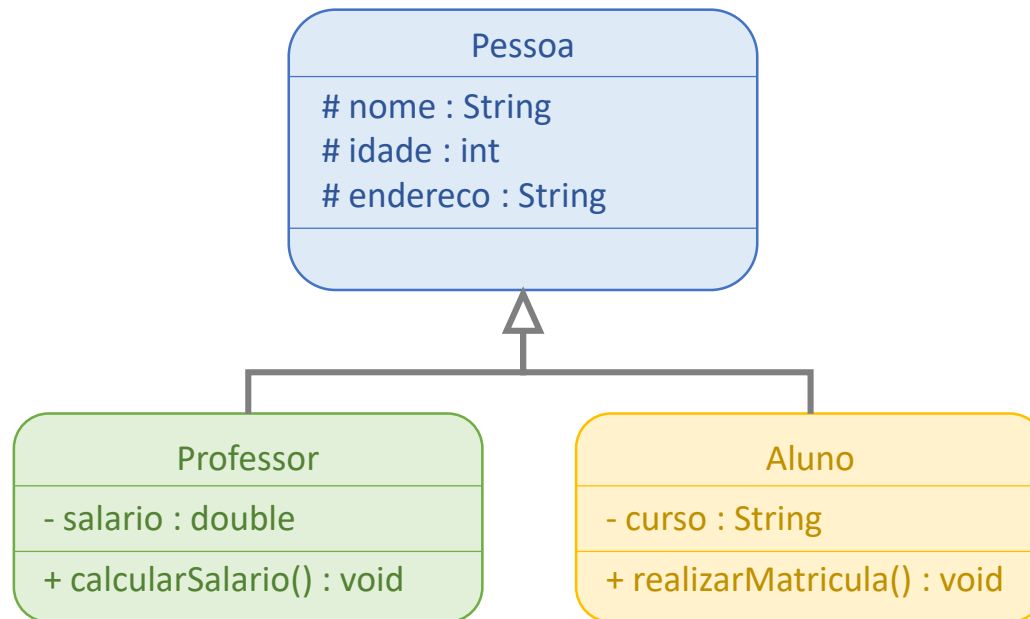
Implementação em Java

*Java **não** implementa herança múltipla!*

Exercício 5

Exercício 5

Codifique o exemplo do modelo abaixo:



Generalização

Herança

Generalização

Generalização é o ato de tornar um objeto geral (abstrair), agrupar características comuns para objetos dentro de um mesmo contexto.

Especialização

Herança

Especialização

Especialização nada mais é do que a parte que “especializa” o objeto vindo de uma Generalização, trazer características próprias para o objeto.

Pilares da Orientação a Objetos

Uma Linguagem Orientada a Objetos possui os seguintes tópicos:

- *Abstração;*
- *Encapsulamento;*
- *Herança;*
- *Polimorfismo.*



Pilares da Orientação a Objetos

Uma Linguagem Orientada a Objetos possui os seguintes tópicos:

- *Abstração;*
- *Encapsulamento;*
- *Herança;*
- *Polimorfismo.*



Polimorfismo

Polimorfismo

Polimorfismo é a habilidade pela qual uma única operação ou nome de atributo pode ser definido em mais de uma classe e assumir implementações diferentes em cada uma dessas classes.

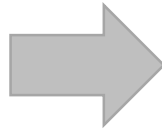
Polimorfismo

É uma das principais habilidades da orientação a objetos que consiste na operações se adequarem automaticamente aos objetos aos quais estão sendo aplicadas.

Exemplo



imprimir()



JATO DE TINTA



LASER



Vantagens

Projeto e implementação de sistemas que são facilmente extensíveis;

- *Novas classes podem ser adicionadas a partes gerais do programa com pouca ou nenhuma modificação;*
- *Por meio da utilização do polimorfismo é possível trazer clareza ao código, diminuir linhas do mesmo e ainda inserir aplicações flexíveis;*

Sobreposição

Polimorfismo

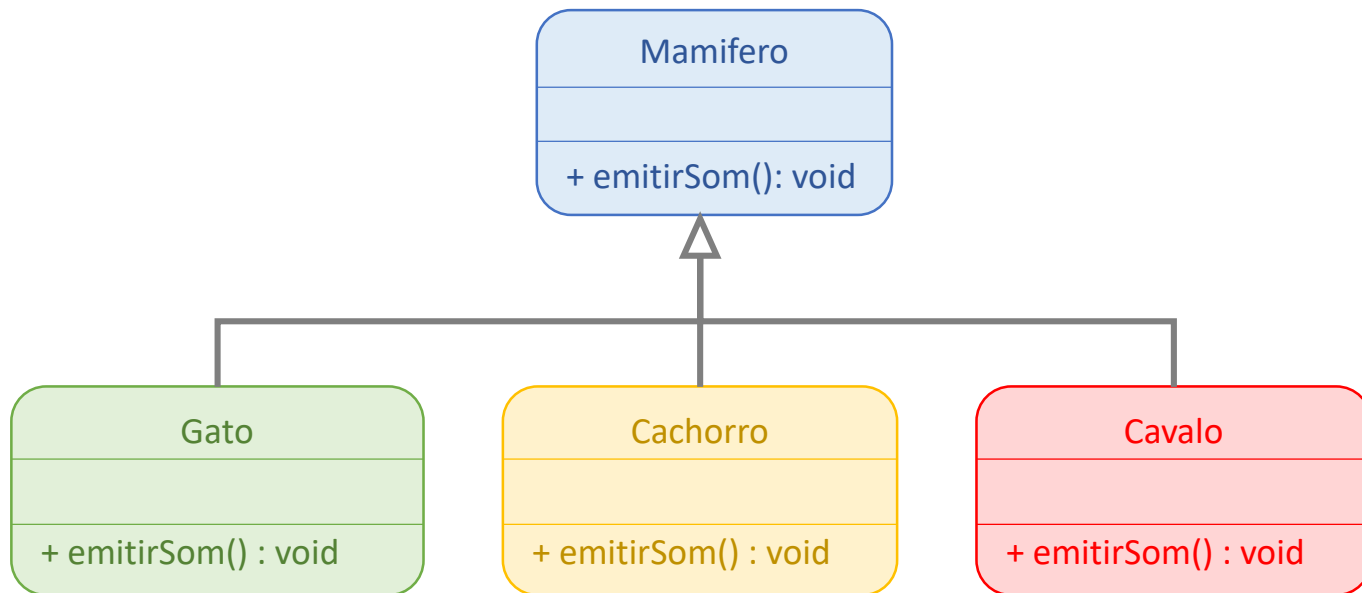
Sobreposição

*É a **redefinição** na classe filha de um método definido na classe pai.*

Sobreposição

*Esta operação **anula** o método da classe pai*

Sobreposição



Implementação em Java (6)

```
public class Mamifero {  
  
    public void emitirSom() {  
  
    }  
  
}
```

```
public class Gato extends Mamifero {  
  
    @Override  
    public void emitirSom() {  
  
        System.out.println("Miar");  
  
    }  
  
}
```

```
public class Cachorro extends Mamifero {  
  
    @Override  
    public void emitirSom() {  
  
        System.out.println("Latir");  
  
    }  
  
}
```

```
public class Cavalo extends Mamifero {  
  
    @Override  
    public void emitirSom() {  
  
        System.out.println("Relinchar");  
  
    }  
  
}
```

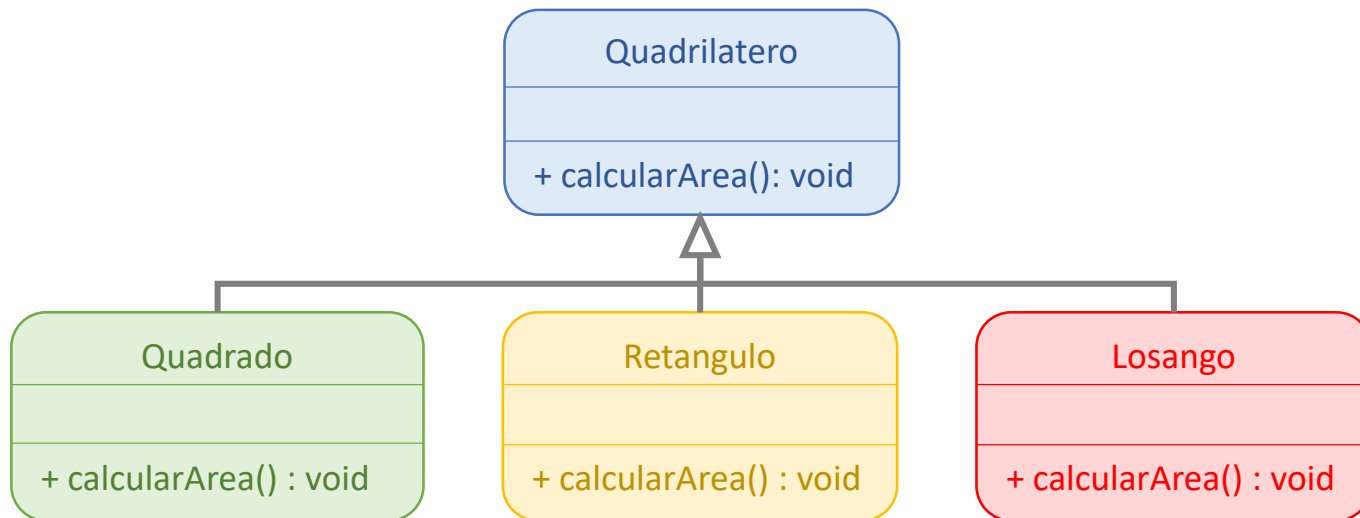
Exercício 6

Exercício 6

O Quadrado, Retângulo e Losango são definidos como Quadriláteros, porém cada um possui uma fórmula específica para calcular sua respectiva área.

*Codifique o **cálculo da área** de cada quadrilátero de acordo com o modelo do próximo slide.*

Exercício 6



Sobrecarga

Sobreposição

Sobrecarga

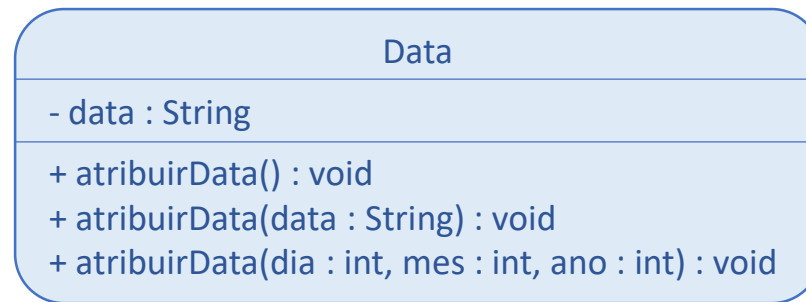
Quando um identificador é utilizado em diversas outras operações (métodos) em uma mesma classe ocorre uma sobrecarga.

Sobrecarga

Consistem em criarmos o mesmo método com possibilidades de entradas (parâmetros) diferentes em termos de:

- *Tipos*
- *Quantidade de parâmetros*
- *Posições dos tipos*

Exemplo



Implementação em Java (7)

```
public class Data {  
  
    private String data;  
  
    public void atribuirData() {  
  
        Date hoje = new Date();  
        SimpleDateFormat df = new SimpleDateFormat("dd/MM/yyyy");  
        this.data = df.format(hoje);  
    }  
  
    public void atribuirData(String data) {  
  
        this.data = data;  
    }  
  
    public void atribuirData(int dia, int mes, int ano) {  
  
        this.data = dia + "/" + mes + "/" + ano;  
    }  
}
```

Exercício 7

Exercício 7

- *Crie uma classe Calculadora.*
- *Implemente o método `calcularMedia()`, recebendo `dois` valores double por parâmetro.*
- *Implemente o método `calcularMedia()`, recebendo `três` valores double por parâmetro.*
- *Implemente o método `calcularMedia()`, recebendo `quatro` valores double por parâmetro.*

Obrigado

 vcandrade@utfpr.edu.br

 /vcandrade

 /Vinicius_Camargo_Andrade

 /prof-vcandrade