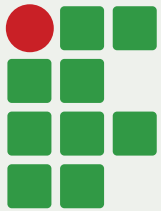
Linguagem de Programação 1

Introdução à Orientação a Objetos

Prof. Fábio José Rodrigues Pinheiro

Outubro de 2021



INSTITUTO FEDERAL
Catarinense
Campus Videira

Abstração do problema

- Princípio básico do desenvolvimento de software
- **Retirar do domínio do problema detalhes relevantes** e representá-los na linguagem da solução

Abstração do problema

- Princípio básico do desenvolvimento de software
- **Retirar do domínio do problema detalhes relevantes** e representá-los na linguagem da solução

Paradigma – forma de como atacar um problema

- A evolução das linguagens de programação influenciaram a forma como os problemas são atacados
- A tecnologia de cada época delimitou como os problemas eram atacados

Paradigmas de programação: sequencial

- Solução rápida para problemas de pequeno porte
- Faz uso de desvios incondicionais (`GOTO` e `JUMP`)
- Não é ideal para problemas de grande porte
  - Dificuldade em organizar o código
  - O uso de desvios incondicionais pode-se tornar um transtorno
- Exemplos: Assembly, Basic

Paradigmas de programação: procedural/estruturada

- Fundamentada sobre estruturas de **sequência**, **decisão** e **repetição**
  - Desvios condicionais são preferidos a desvios incondicionais
- A solução de cada pequena parte do problema é feita em **procedimentos** (ou **funções**) e a solução de todo problema consiste na invocação destes procedimentos
  - Dividir para conquistar!
- Visa a reutilização de código - basicamente de comportamento (funções e procedimentos)
- Exemplos: Pascal, C

Paradigmas de programação: orientada a objetos

- Ideal para o desenvolvimento de software complexo, porém traz complexidade desnecessária para projetos pequenos ou embarcados
  - Os benífícios/vantagens da OO tornam-se mais perceptíveis na construção de sistemas de médio e grande portes
- Todo o sistema é visualizado como um conjunto de células interconectadas, denominadas **objetos**
- **Cada objeto possui uma tarefa específica** e por meio da **troca de mensagens entre os objetos** é possível realizar uma tarefa computacional
- Visa a reutilização de código - de estado e comportamento
- Exemplos: Smalltalk, C++, Java, Python

Vantangens da Orientação a Objetos

- **reutilização**: o encapsulamento dos métodos e representação dos dados para a construção de classes facilitam o desenvolvimento de software reutilizável, auxiliando na produtividade de sistemas;
- **extensibilidade**: facilidade de estender o software devido a duas razões:
  - herança: novas classes são construídas a partir das que já existem;
  - as classes formam uma estrutura fracamente acoplada o que facilita alterações;
- **manutenibilidade**: a modularização natural em classes facilita a realização de alterações no software;
- **flexibilidade**: as classes delimitam-se em unidades naturais para a alocação de tarefas de desenvolvimento de software.

Conceitos da Orientação a Objetos

A Orientação a Objetos fundamenta-se sobre 5 conceitos:

- Objetos
- Classes
- Mensagens
- Herança
- Polimorfismo

Objetos: Definição

Um **objeto** é um item **identificável** e é composto por **estado** e por **comportamento**

Objetos: Definição

Um **objeto** é um item **identificável** e é composto por **estado** e por **comportamento**

Caneta para quadro branco

Cor: Preta

Tinta: 100%

Estado

Comportamento

Identidade

Regra de ouro da orientação a objetos

Identificar os estados e comportamentos de objetos do mundo real é um grande passo para se começar a pensar em termos de programação orientada a objetos (abstração)

- **Estado**
  - Representa as características do objeto
  - Ex: Um carro possui como características uma cor, modelo, velocidade atual
- **Comportamento**
  - Representa as operações (métodos) que este objeto é capaz de executar
  - Ex: Um carro pode trocar de marcha, acelerar, frear, etc.

Objetos no domínio do problema

Olhe ao redor e escolha dois objetos. Para estes responda:

- Quais os possíveis **estados** que este objeto pode assumir?
- Quais os possíveis **comportamentos** que este objeto pode ter?

Objetos no domínio do problema

Olhe ao redor e escolha dois objetos. Para estes responda:

- Quais os possíveis **estados** que este objeto pode assumir?
- Quais os possíveis **comportamentos** que este objeto pode ter?

- É possível notar diferentes níveis de complexidade de cada objeto
  - Por exemplo: lâmpada *vs* computador
- É possível notar que alguns objetos podem conter outros objetos
  - Um computador possui um disco rígido, este último por sua vez também é um objeto

Objetos de *software* são semelhantes aos objetos reais

Um objeto armazena seu estado em **atributos** e seu comportamento se dá por meio de **operações** (métodos)

Objetos de *software* são semelhantes aos objetos reais

Um objeto armazena seu estado em **atributos** e seu comportamento se dá por meio de **operações** (métodos)

- **Métodos de um objeto são invocados/chamados** para realizar uma computação e potencialmente para modificar os atributos deste objeto

Objetos de *software* são semelhantes aos objetos reais

Um objeto armazena seu estado em **atributos** e seu comportamento se dá por meio de **operações** (métodos)

- **Métodos de um objeto são invocados/chamados** para realizar uma computação e potencialmente para modificar os atributos deste objeto
  - **programador:** Qual a sua velocidade atual?
  - **objeto carro:** 20 km/hora
  - **programador:** Diminua a velocidade em 10%
  - **objeto carro:** Ok
  - **programador:** Qual a sua nova velocidade?
  - **objeto carro:** 18 km/hora

Encapsulamento

Definição

Processo de esconder todos os detalhes de um objeto que não contribuem para as suas características essenciais. Ex: uma caixa preta

- A **interação entre objetos** se dá através da **troca de mensagens**
  - O **emissor da mensagem** não precisa conhecer como o **destinatário processará** a mensagem, ao emissor só importa receber a resposta
  - Essa troca de mensagens se dá através de **chamada de métodos**

Encapsulamento

Definição

Processo de esconder todos os detalhes de um objeto que não contribuem para as suas características essenciais. Ex: uma caixa preta

- A **interação entre objetos** se dá através da **troca de mensagens**
  - O **emissor da mensagem** não precisa conhecer como o **destinatário processará** a mensagem, ao emissor só importa receber a resposta
  - Essa troca de mensagens se dá através de **chamada de métodos**

Exemplo: `random.nextInt(100)`

Mensagens são compostas por três partes

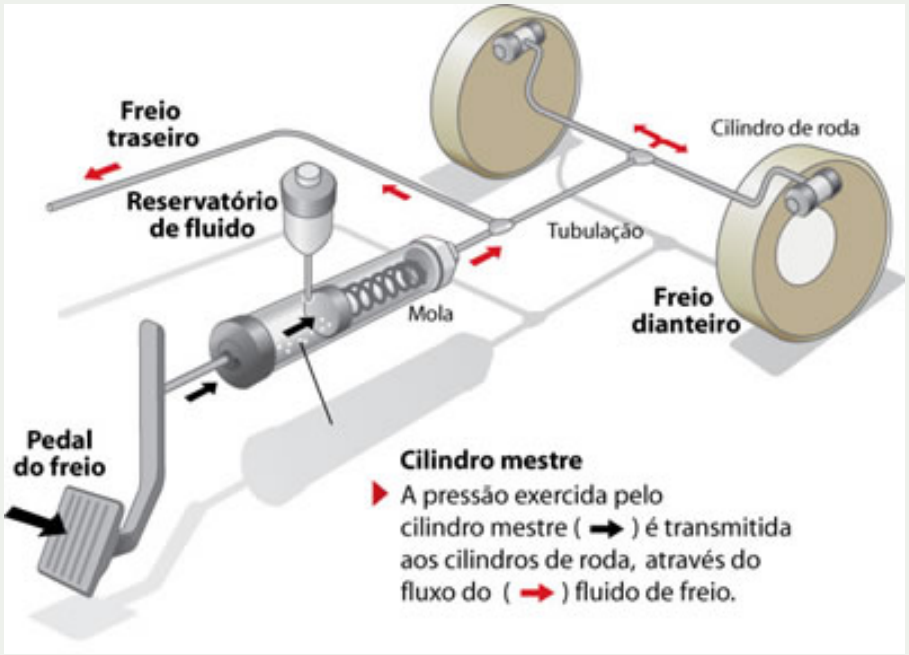
1. **Objeto:** `random do tipo/classe Random`
2. **Nome do método:** `nextInt`
3. **Parâmetros:** `10`

Vantagem do encapsulamento

A implementação dentro de uma operação pode ser alterada sem que isso implique na alteração do código do objeto requisitante

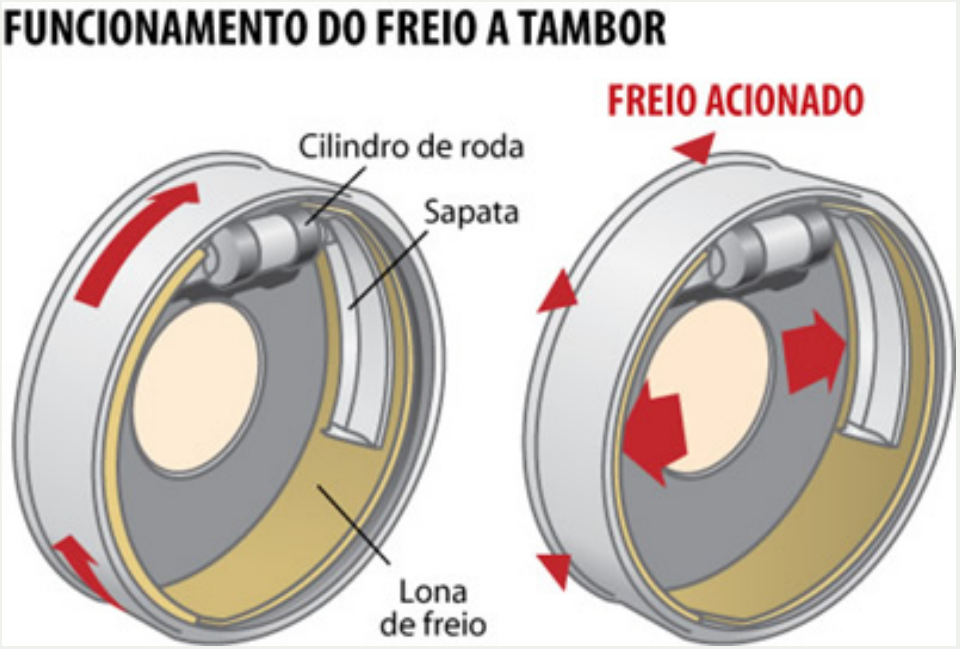
- A **interface** de um objeto corresponde ao que ele conhece e ao que ele sabe fazer, no entanto **sem descrever como ele conhece ou faz**
- O emissor das mensagens precisa saber quais operações o destinatário é capaz de realizar ou quais informações o destinatário pode fornecer
- Exemplos:
  - Mudar de canal da tv
  - Enviar uma mensagem via aplicativo
  - Salvar um documento em um editor

Encapsulamento: Exemplo Sistema de freio hidráulico

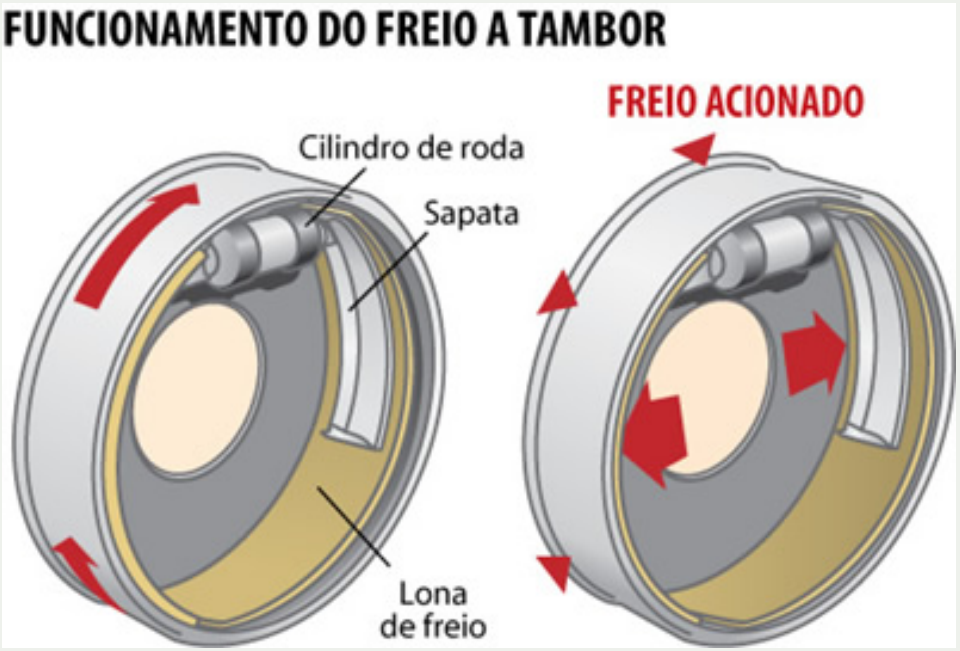


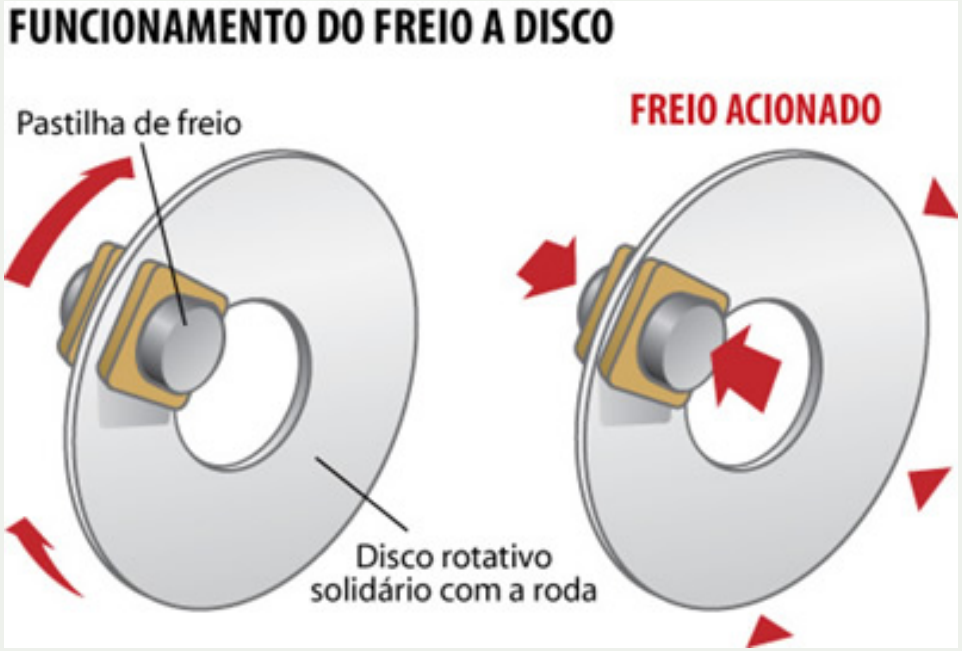
- Freios funcionam por meio de um sistema de pistões e mangueiras por onde circula o fluído de freio
- Ao pisar no pedal de freio, aciona-se o cilindro mestre que irá pressurizar o fluído.
- Esse fluído transmite a pressão exercida no pedal até as rodas, acionando o freio.

Encapsulamento: Exemplo Sistema de freio hidráulico

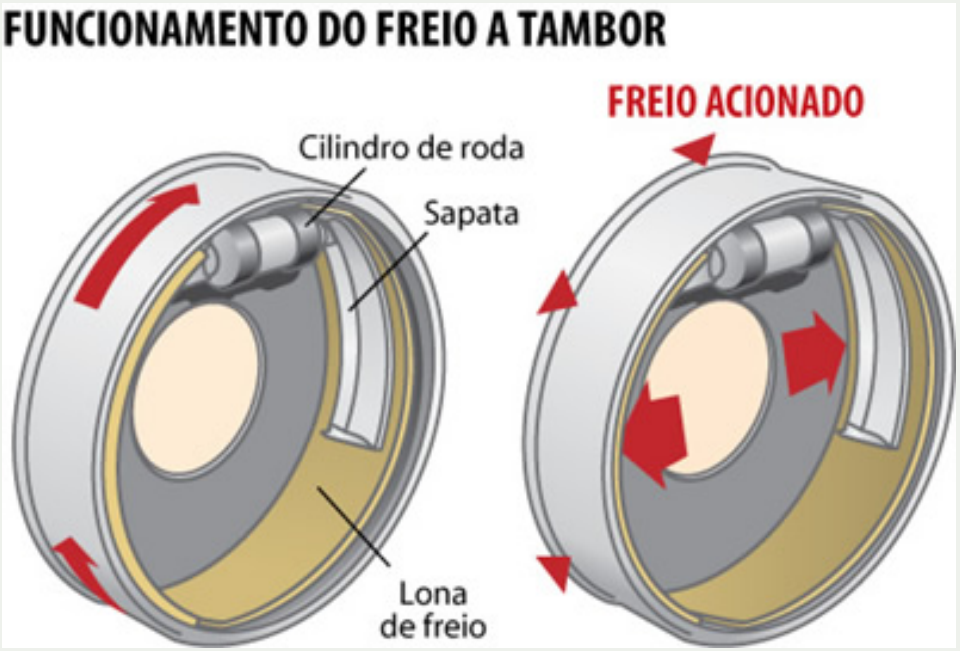


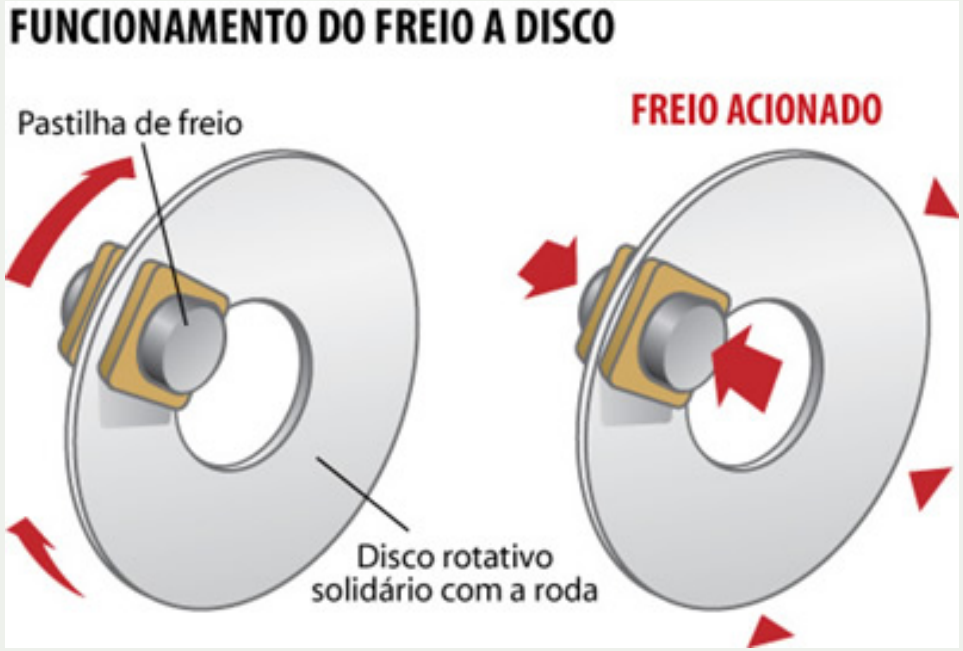
Encapsulamento: Exemplo Sistema de freio hidráulico





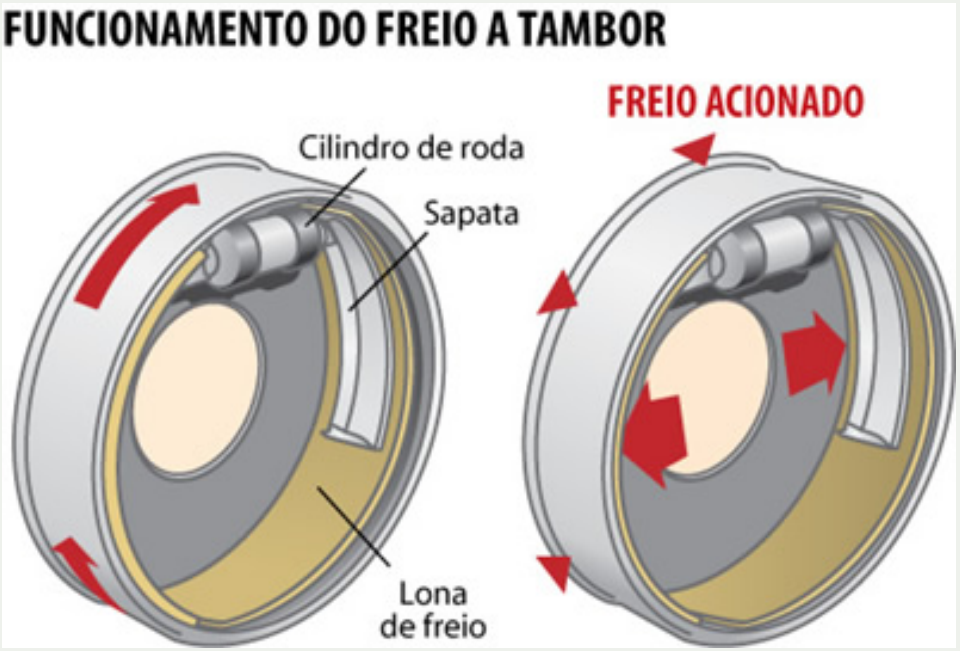
Encapsulamento: Exemplo Sistema de freio hidráulico

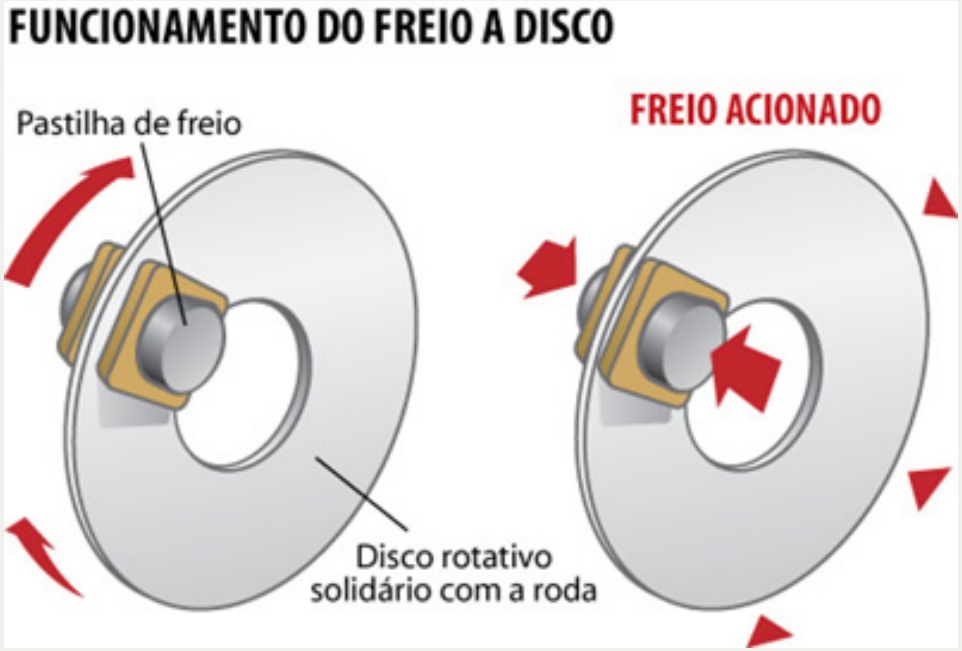




- Como você faz para frear um carro com o sistema de freio a tambor?

Encapsulamento: Exemplo Sistema de freio hidráulico





- Como você faz para frear um carro com o sistema de freio a tambor?
- Como você faz para frear um carro com o sistema de freio a disco?

Encapsulamento: Exemplo Sistema de freio hidráulico

Objeto: Fusca

- Para diminuir a velocidade do carro basta pressionar o pedal do freio
- Não é necessário entender como o mecanismo de freio funciona, mas ao acionar o freio o Fusca irá diminuir sua velocidade

Encapsulamento: Exemplo Sistema de freio hidráulico

Objeto: Fusca

- Para diminuir a velocidade do carro basta pressionar o pedal do freio
- Não é necessário entender como o mecanismo de freio funciona, mas ao acionar o freio o Fusca irá diminuir sua velocidade

```
1 System.out.println("Acionando o freio do Fusca");
2 fusca.frear();
3
4 System.out.println("Acionando o freio da Ferrari");
5 ferrari.frear();
```


Classes

- **Classe** é uma planta (modelo, forma, projeto) que indica como os **objetos** deverão ser construídos

Fusca

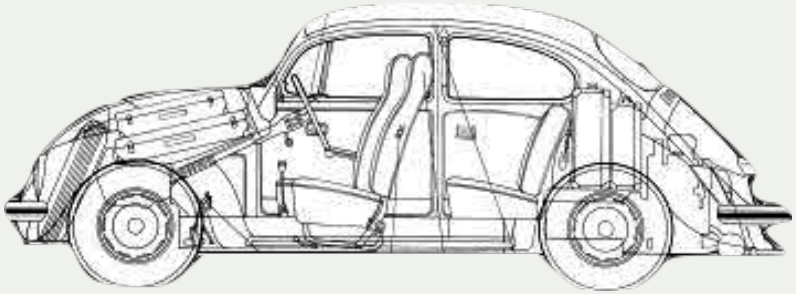
- Cada carro é construído com base em um mesmo projeto de engenharia e por consequência todos carros possuirão os mesmos componentes

Classes

- **Classe** é uma planta (modelo, forma, projeto) que indica como os **objetos** deverão ser construídos

Fusca

- Cada carro é construído com base em um mesmo projeto de engenharia e por consequência todos carros possuirão os mesmos componentes



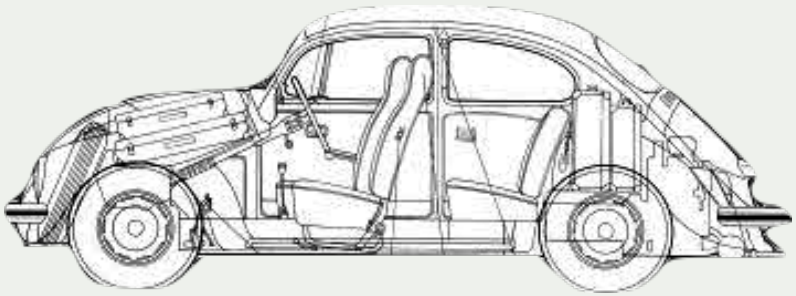
Classe

Classes

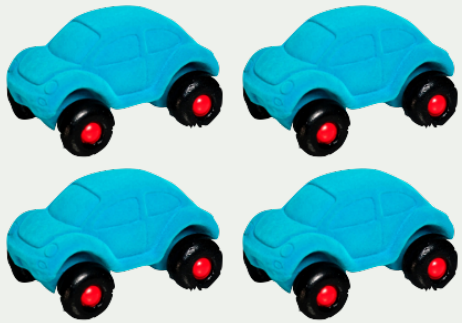
- **Classe** é uma planta (modelo, forma, projeto) que indica como os **objetos** deverão ser construídos

Fusca

- Cada carro é construído com base em um mesmo projeto de engenharia e por consequência todos carros possuirão os mesmos componentes

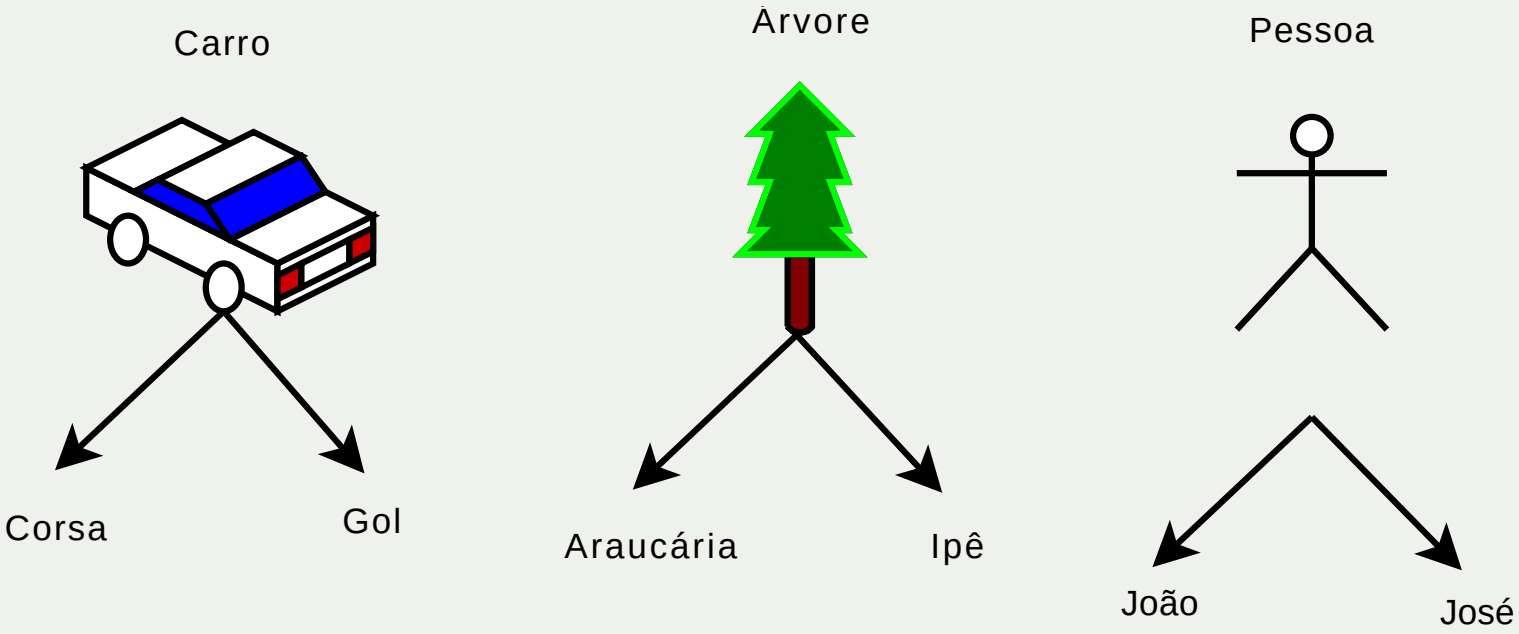


Classe



Objetos

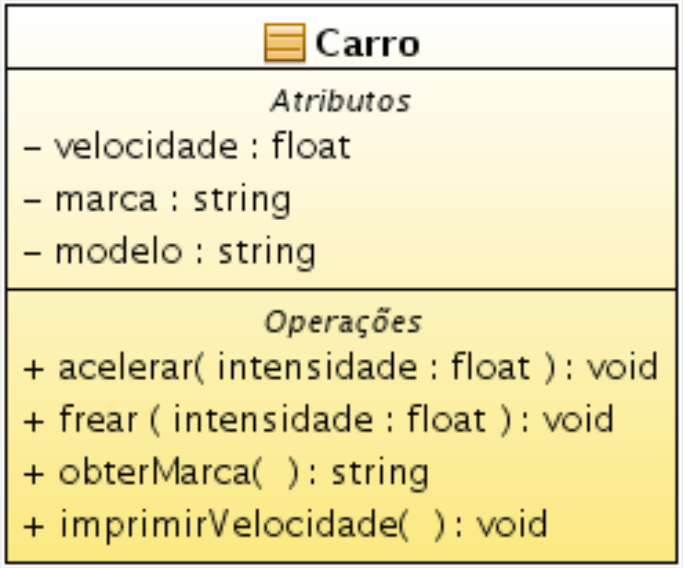
Identifique as classes e os objetos



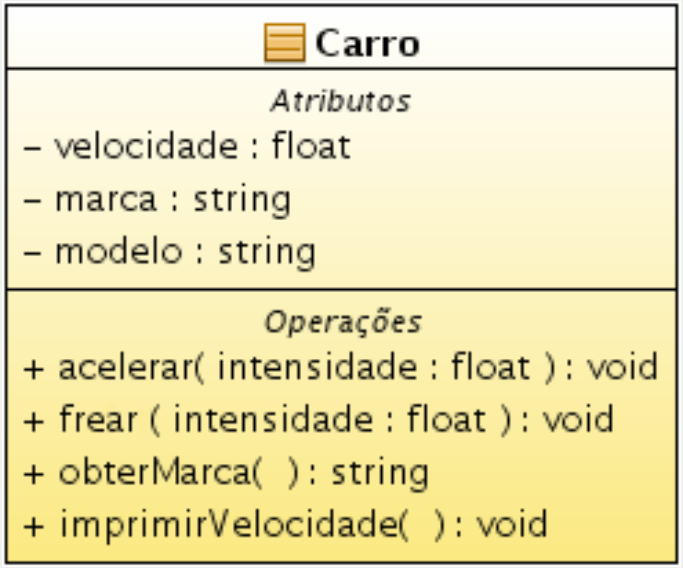
Uma classe em Java

```
1  public class Carro{
2    // atributos
3    private double velocidade;
4    private String marca;
5    private String modelo;
6    // metodos
7    public void acelerar(double intensidade){
8      ...
9     }
10   public void frear(double intensidade){
11     ...
12    }
13   public String obterMarca(){
14       return marca;
15    }
16   public void imprimirVelocidade(){
17       System.out.println("Velocidade: " + velocidade);
18    }
19 }
```


Representação gráfica em UML da classe Carro



Representação gráfica em UML da classe Carro



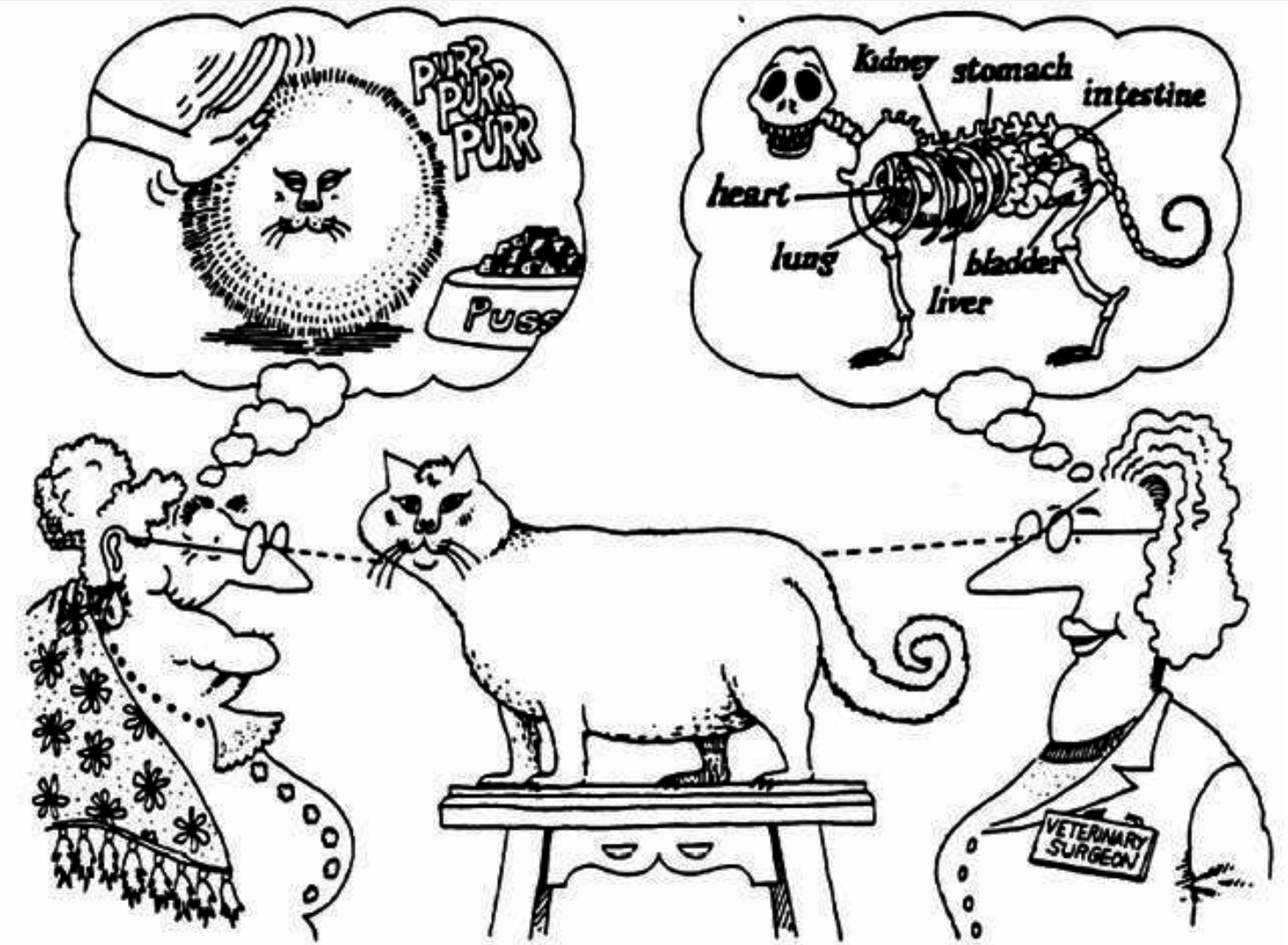
Linguagem de modelagem unificada – UML

Uma linguagem **padrão** para a modelagem de sistemas, amplamente utilizada tanto pela indústria do *software* quanto por instituições acadêmicas.

Abstração

- Trata-se do processo mental que nós seres humanos **nos atemos aos aspectos mais relevantes** de alguma coisa, ao mesmo tempo que **ignoramos os aspectos menos importantes**
- Isso nos permite gerenciar a complexidade de um objeto, ao mesmo tempo que concentramos nossa atenção nas características essenciais do mesmo
- Note que **abstração é dependente do contexto** sobre o qual este algo é analisado
  - O que é importante em um contexto pode não ser importante em outro

Abstração



Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

Abstração: Exemplo

- **Sistema para revenda de carros**
  - Necessita de um sistema para controlar os carros do estoque
  - Quais são as características essenciais para um carro?

- **Jogo de Fórmula 1**
  - Um usuário deseja controlar seu carro no jogo
  - Quais são as características essenciais para um carro?

Abstração: Exemplo

- **Sistema para revenda de carros**
  - Necessita de um sistema para controlar os carros do estoque
  - Quais são as características essenciais para um carro?
    - Atributos: código, marca, modelo, ano, preço
    - Métodos: obterCódigo, obterModelo, definirPreço, etc.
- **Jogo de Fórmula 1**
  - Um usuário deseja controlar seu carro no jogo
  - Quais são as características essenciais para um carro?
    - Atributos: código, cor, velocidadeAtual, velocidadeMaxima
    - Métodos: frear, acelerar, trocarPneus, etc.

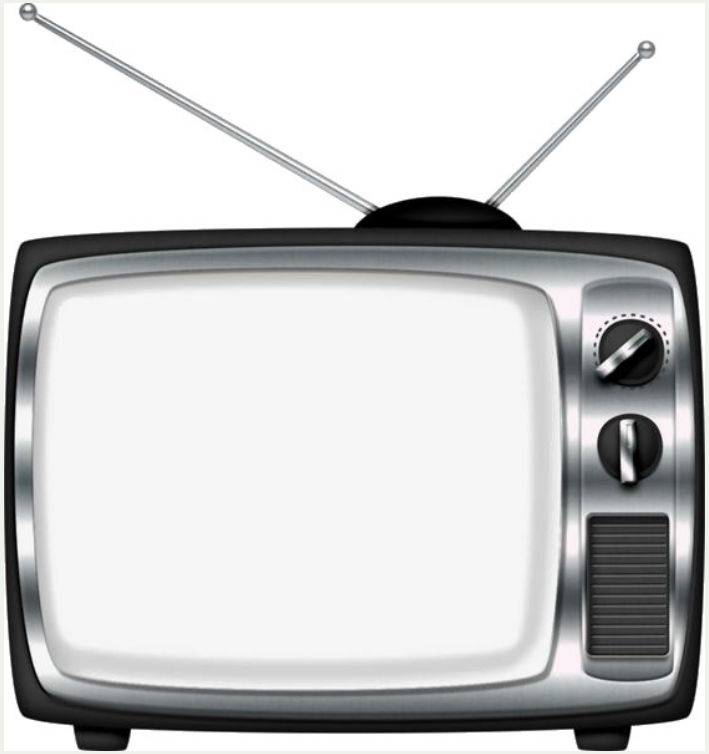
Codificando em Java

Exercício: Um contador

Exercício: Um contador

- A classe `Contador` possui um único **atributo**
  - `valorAtual`
- A classe provê **métodos** para:
  - Atribuir um valor ao contador
  - Incrementar o contador
  - Obter o atual valor do contador

Exercício: Televisão



- Pense em um contexto e realize o processo de abstração para coletar as informações essenciais
- Represente esse objeto em UML
- Implemente em Java esse objeto, além de uma outra classe de onde serão invocados alguns métodos do objeto modelado

Exercício: Buzz Lightyear

- Capacete retrátil
- 6 frases
- Dispara Laser
- Braço articulado para golpes
- Abre as asas

Faça uma classe para representar o Buzz e uma outra classe onde seja possível instanciar até 3 e interagir com cada uma deles.

Exercício: Buzz Lightyear

- Todos fazem as mesmas coisas, mas cada um é um objeto diferente

Exercício: Pessoas em diferentes contextos

1. Criar uma classe para representar uma Pessoa usada em um sistema de cadastro de clientes de uma loja varejista
2. Criar uma classe para representar uma Pessoa em um mundo virtual (Ex: *Second Life, Roblox, etc*)

Crie um programa Java (classe Java com método `main`) e crie 1 objeto para cada uma das classes acima e envie algumas mensagens para estes objetos

Leitura complementar

Caelum Ensino e Soluções em Java
Apostila Caelum FJ-11 Java e Orientação a Objetos
Pasta na página da disciplina - SIGAA

- Capítulos 4 e 5