Introdução ao Ambiente Java

Programação Orientada a Objetos

Prof. Tulio Alberton Ribeiro

Instituto Federal de Santa Catarina – IFSC
campus São José
tulio.alberton@ifsc.edu.br

31 de julho de 2014

Desenvolvimento de software

Princípio básico

O desenvolvimento de todo e qualquer software inicia-se com a **abstração do problema**. Ou seja, retirar do domínio do problema detalhes relevantes e representá-los na linguagem da solução.

Desenvolvimento de software

Princípio básico

O desenvolvimento de todo e qualquer software inicia-se com a **abstração do problema**. Ou seja, retirar do domínio do problema detalhes relevantes e representá-los na linguagem da solução.

- A evolução das linguagens de programação influenciaram a forma de como os problemas são atacados
- A tecnologia existente em cada época delimitou como os problemas eram atacados e ao longo dos tempos surgiram diversos **paradigmas de programação**

Desenvolvimento de software

Princípio básico

O desenvolvimento de todo e qualquer software inicia-se com a **abstração do problema**. Ou seja, retirar do domínio do problema detalhes relevantes e representá-los na linguagem da solução.

- A evolução das linguagens de programação influenciaram a forma de como os problemas são atacados
- A tecnologia existente em cada época delimitou como os problemas eram atacados e ao longo dos tempos surgiram diversos **paradigmas de programação**

Paradigma

Forma de como atacar um problema.

Paradigmas de programação

Programação sequencial

- A solução para o problema se dá através da execução sequencial de instruções, uma após a outra
- Faz uso de desvios incondicionais (`GOTO` e `JUMP`)
- Apresenta uma solução rápida para problemas de pequeno porte
- Não é ideal para problemas de grande porte
  - Dificuldade em organizar o código e o uso de desvios incondicionais pode-se tornar um transtorno
- Exemplos: Assembly, Basic

Programação estruturada

- Consiste em dividir o problema em partes menores e então apresentar soluções para essas pequenas partes
  - Dividir para conquistar!
- Está fundamentada sobre as 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
- Visa a reutilização de código
- Exemplos: Pascal, C

Paradigmas de programação

Programação orientada a objetos

- Surgiu da idéia que todo sistema de software funcionasse como um ser vivo
  - Cada célula do sistema poderia interagir com outras células, através do **envio de mensagens** e cada célula consistiria ainda em um sistema autônomo

Paradigmas de programação

Programação orientada a objetos

- Surgiu da idéia que todo sistema de software funcionasse como um ser vivo
  - Cada célula do sistema poderia interagir com outras células, através do **envio de mensagens** e cada célula consistiria ainda em um sistema autônomo

Programação Orientada a Objetos

Todo o sistema é visualizado como um conjunto de células interconectadas, denominadas **objetos**. Cada objeto possui uma tarefa específica e através da comunicação entre os objetos é possível realizar uma tarefa computacional completa.

Paradigmas de programação

Programação orientada a objetos

- Surgiu da idéia que todo sistema de software funcionasse como um ser vivo
  - Cada célula do sistema poderia interagir com outras células, através do **envio de mensagens** e cada célula consistiria ainda em um sistema autônomo

Programação Orientada a Objetos

Todo o sistema é visualizado como um conjunto de células interconectadas, denominadas **objetos**. Cada objeto possui uma tarefa específica e através da comunicação entre os objetos é possível realizar uma tarefa computacional completa.

- Tal paradigma é ideal para o desenvolvimento de software complexos
  - Extensão do projeto de forma fácil e simplificada
- Exemplos: Smalltalk, C++, Java, Python

Paradigmas de programação

- Em linguagens como a **C**, a forma de atacar o problema consiste em mapear a estrutura do problema para a estrutura do computador. Assim, deve-se criar uma associação entre a solução (modelo da máquina) e o modelo do problema
- A **Orientação a Objetos** tenta trazer o espaço da solução para o espaço do problema e assim representar ambos espaços como objetos
  - A ideia é fazer o funcionamento do programa se parecer com o mundo real
  - As abstrações correspondem aos objetos do domínio do problema
  - Facilidades na comunicação entre o desenvolvedor e o usuário final
  - Os objetos levantados passam a existir em todas as fases do desenvolvimento do software complexo
  - Fases posteriores de desenvolvimento adicionam novos objetos

Conceitos da Orientação a Objetos

A Orientação a Objetos fundamenta-se sobre 5 conceitos:

- Objetos
- Classes
- Mensagens
- Herança
- Polimorfismo

Objetos

Definição

Na orientação a objetos, um **objeto** é um item identificável, individual e com um papel bem definido dentro do domínio do problema

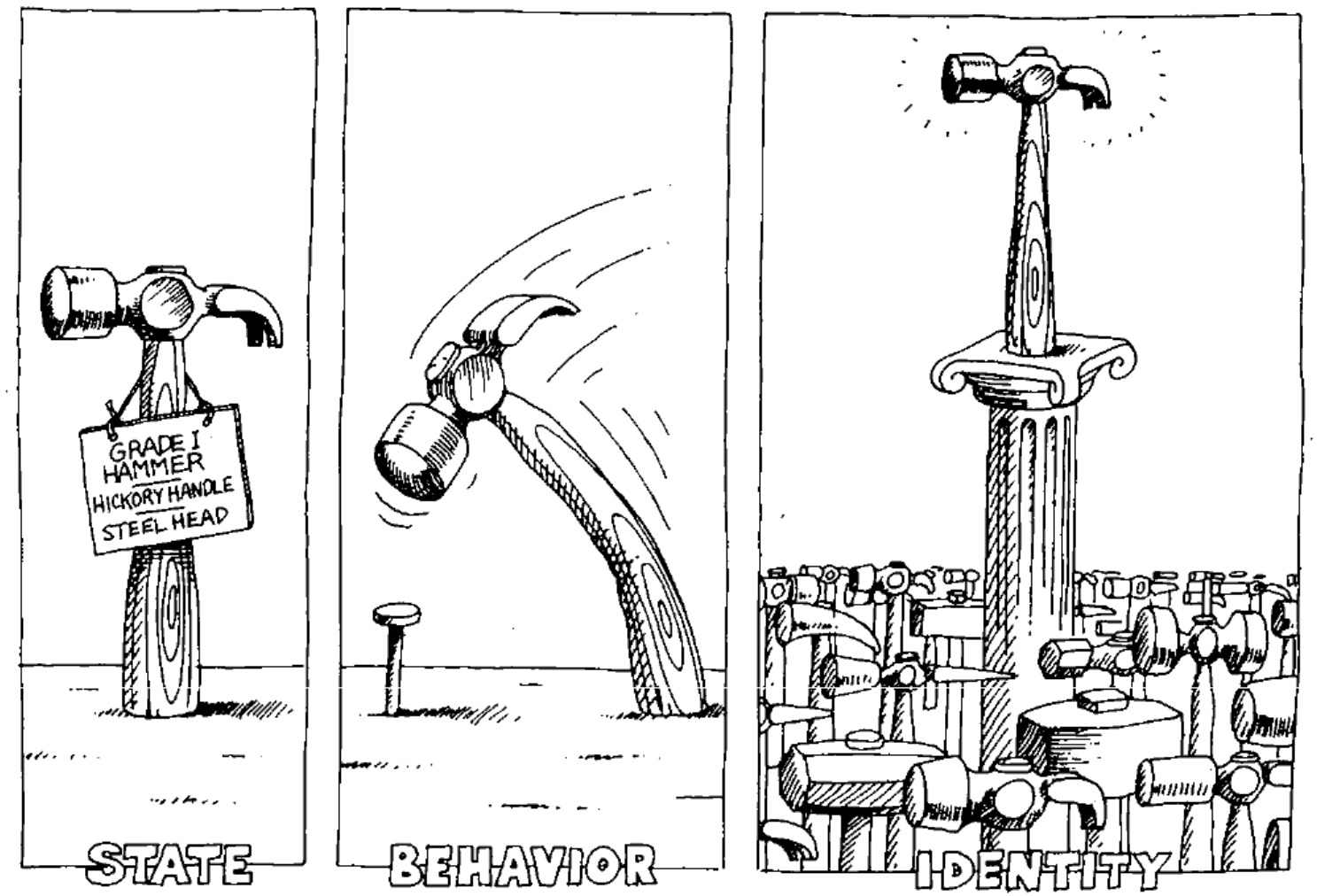
- É formado por um **estado** e um por um **comportamento**
- Em sistemas computacionais, os objetos geralmente são usados para modelar objetos que encontramos no dia-a-dia
  - Objetos reais: pessoa, cachorro, carro, caneta
  - Objetos abstratos: funcionário, professor, aluno

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**



Objetos

- **Estado**
  - O estado de um objeto representa as características deste
  - Um carro possui como características uma cor, modelo, potência, velocidade atual, marcha atual, etc.
- **Comportamento**
  - Representa as operações (métodos) que este objeto é capaz de executar
  - Um carro pode trocar de marcha, acelerar, frear, etc.

Objetos

- **Estado**
  - O estado de um objeto representa as características deste
  - Um carro possui como características uma cor, modelo, potência, velocidade atual, marcha atual, etc.
- **Comportamento**
  - Representa as operações (métodos) que este objeto é capaz de executar
  - Um carro pode trocar de marcha, acelerar, frear, etc.

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

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 em sistemas computacionais

Objetos de *software* são semelhantes aos objetos reais

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

Objetos em sistemas computacionais

Objetos de *software* são semelhantes aos objetos reais

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

- Em Java, os métodos de um objeto são invocados para realizar uma determinada computação e potencialmente para modificar os atributos deste objeto

Objetos em sistemas computacionais

Objetos de *software* são semelhantes aos objetos reais

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

- Em Java, os métodos de um objeto são invocados para realizar uma determinada 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

Encapsulamento: um dos princípios da orientação a objeto

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

Encapsulamento: um dos princípios da orientação a objeto

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

Exemplo: `System.out.println("Olá mundo");`

Mensagens são compostas por três partes

- **Objeto:** System.out
- **Nome do método:** println
- **Parâmetros:** "Olá mundo"

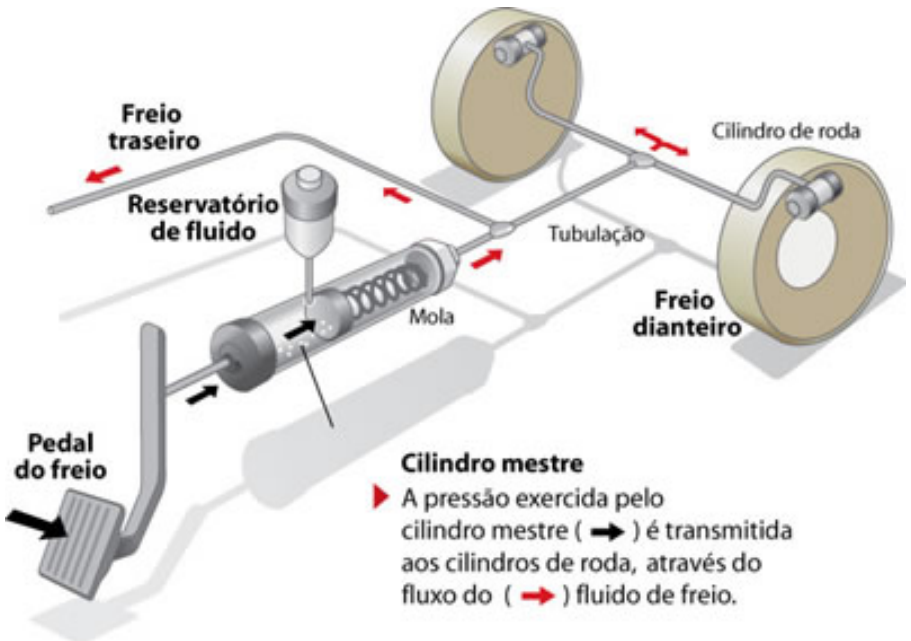
Encapsulamento

- O emissor das mensagens precisa saber quais operações o destinatário é capaz de realizar ou quais informações o destinatário pode fornecer
- 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**
  - Define as mensagens que ele está apto a receber e responder

Vantagem do encapsulamento

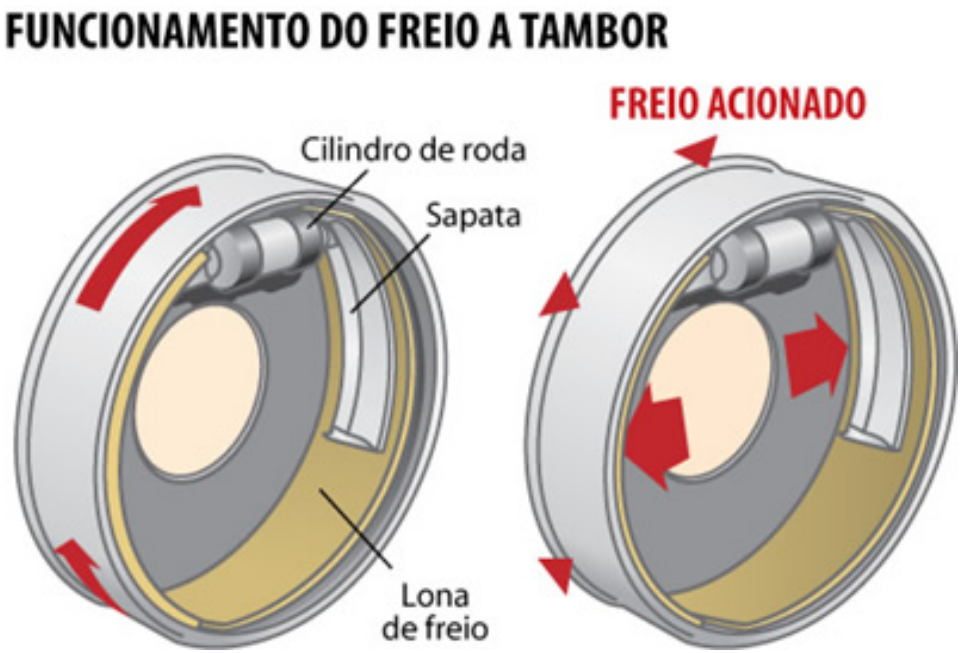
A implementação dentro de uma operação pode ser alterada sem que isso implique na alteração do código do objeto requisitante

Encapsulamento: Exemplo Sistema de freio hidráulico

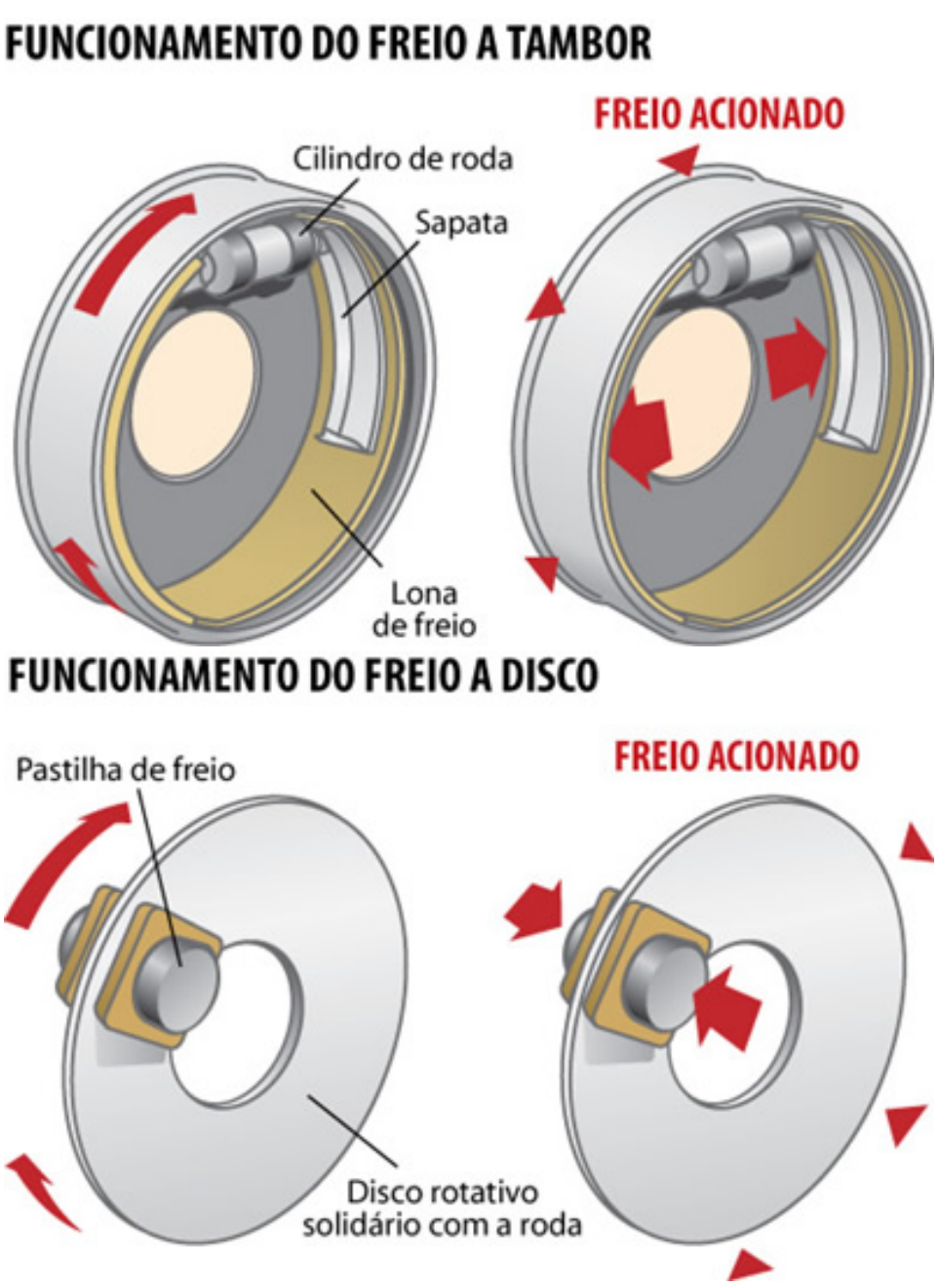


- Freios funcionam através de um sistema de pistões e mangueiras por onde circula o fluído de freio
- Ao pisar no pedal de freio, aciona-se o cilindro mestre que irá pressurizar o fluído.
- Esse fluído transmite a pressáo exercida no pedal até as rodas, acionando o freio.

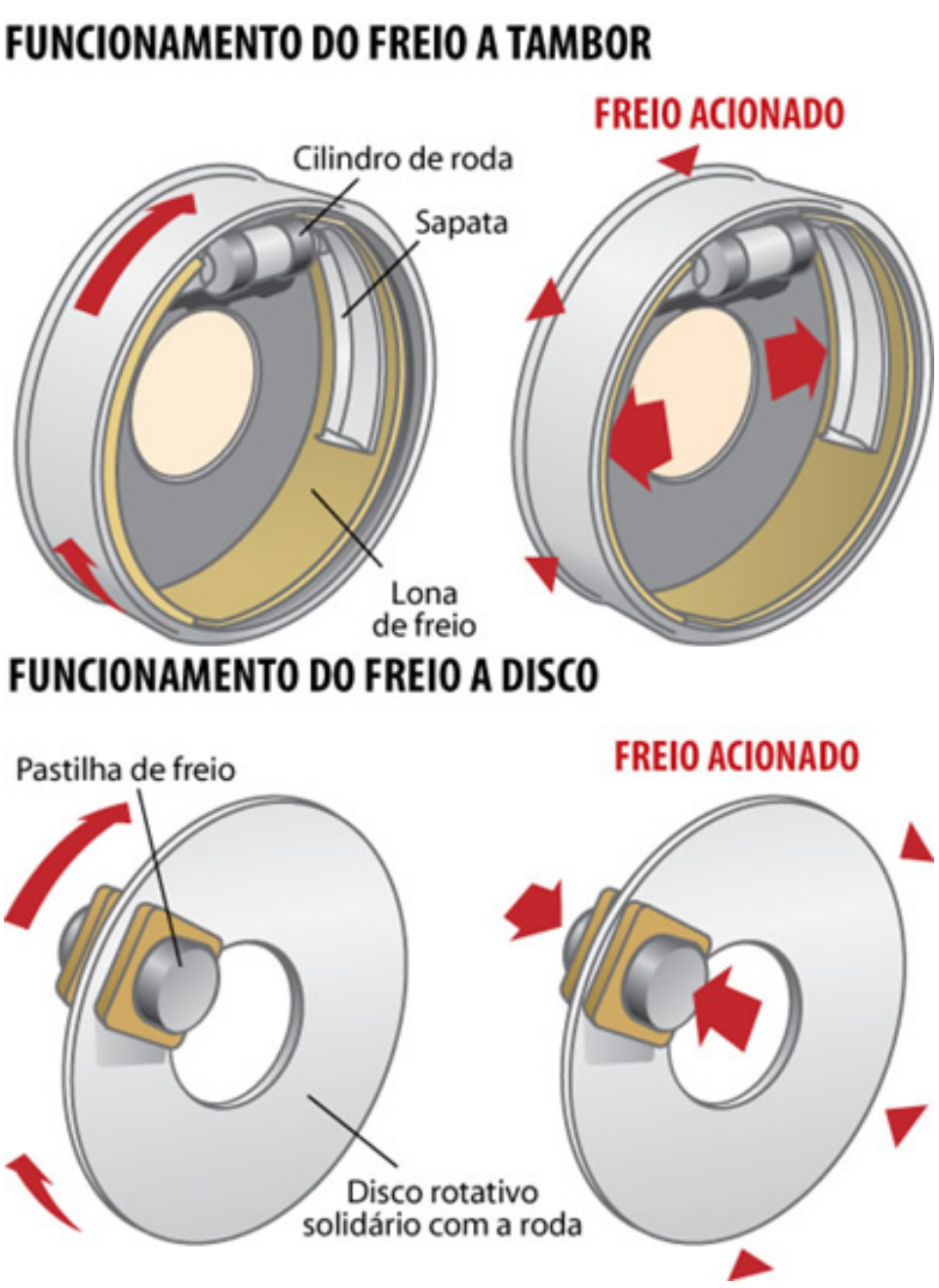
Encapsulamento: Exemplo Sistema de freio hidráulico



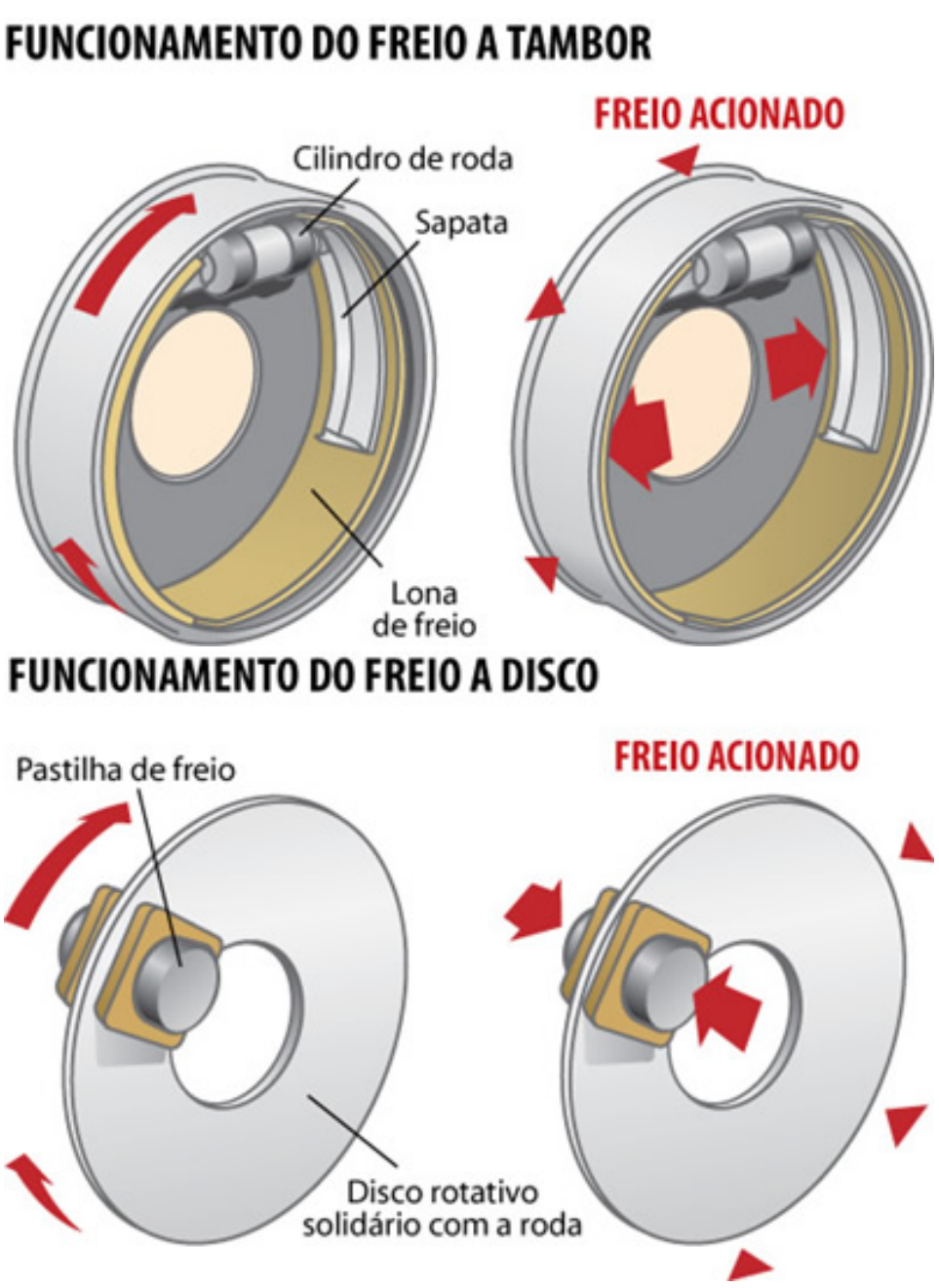
Encapsulamento: Exemplo Sistema de freio hidráulico



Encapsulamento: Exemplo Sistema de freio hidráulico



Encapsulamento: Exemplo Sistema de freio hidráulico



Encapsulamento: Exemplo Sistema de freio hidráulico

Objeto: Fusca

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

```
7
8     System.out.println("Acionando o freio do Fusca");
9
10    fusca.frear();
11
12    System.out.println("Acionando o freio da Ferrari");
13
14    ferrari.frear();
```

Classes

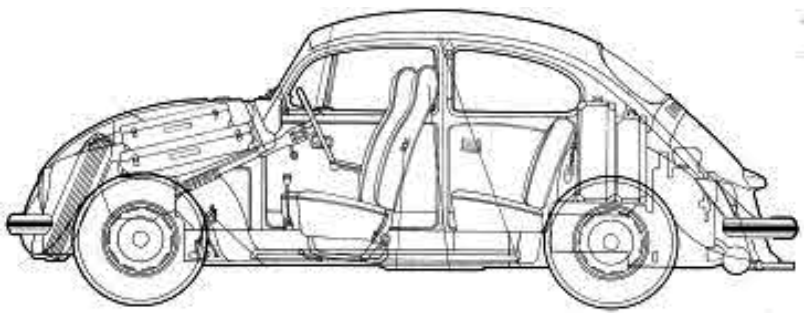
- **Classe** é uma planta (projeto) que indica como os **objetos** deverão ser construídos

Classes

- **Classe** é uma planta (projeto) que indica como os **objetos** deverão ser construídos
- Exemplo: Fusca
  - Cada carro é construído com base em um mesmo projeto de engenharia e por consequência todos carros possuirão os mesmos componentes

Classes

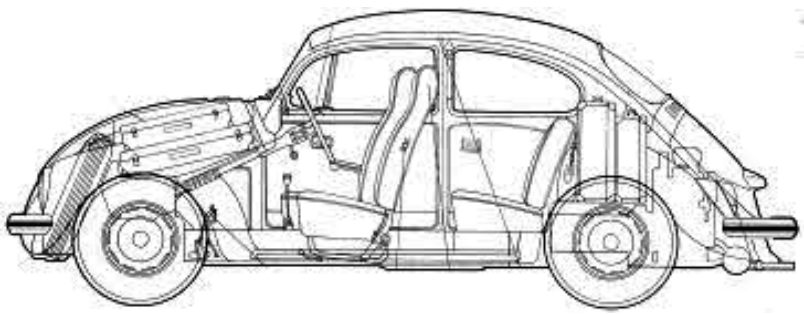
- **Classe** é uma planta (projeto) que indica como os **objetos** deverão ser construídos
- Exemplo: Fusca
  - Cada carro é construído com base em um mesmo projeto de engenharia e por consequência todos carros possuirão os mesmos componentes



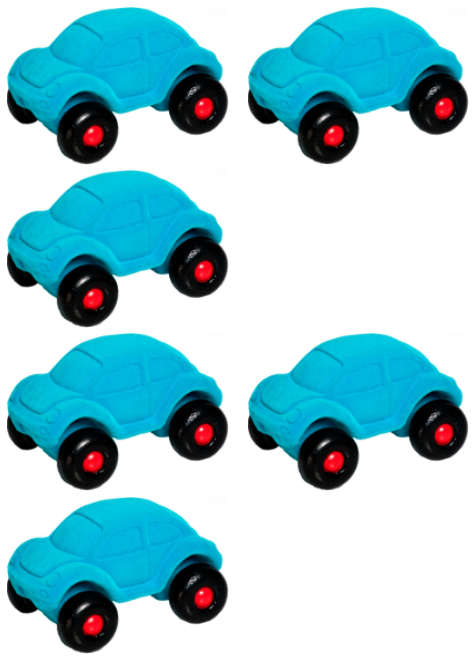
Classe

Classes

- **Classe** é uma planta (projeto) que indica como os **objetos** deverão ser construídos
- Exemplo: Fusca
  - Cada carro é construído com base em um mesmo projeto de engenharia e por consequência todos carros possuirão os mesmos componentes

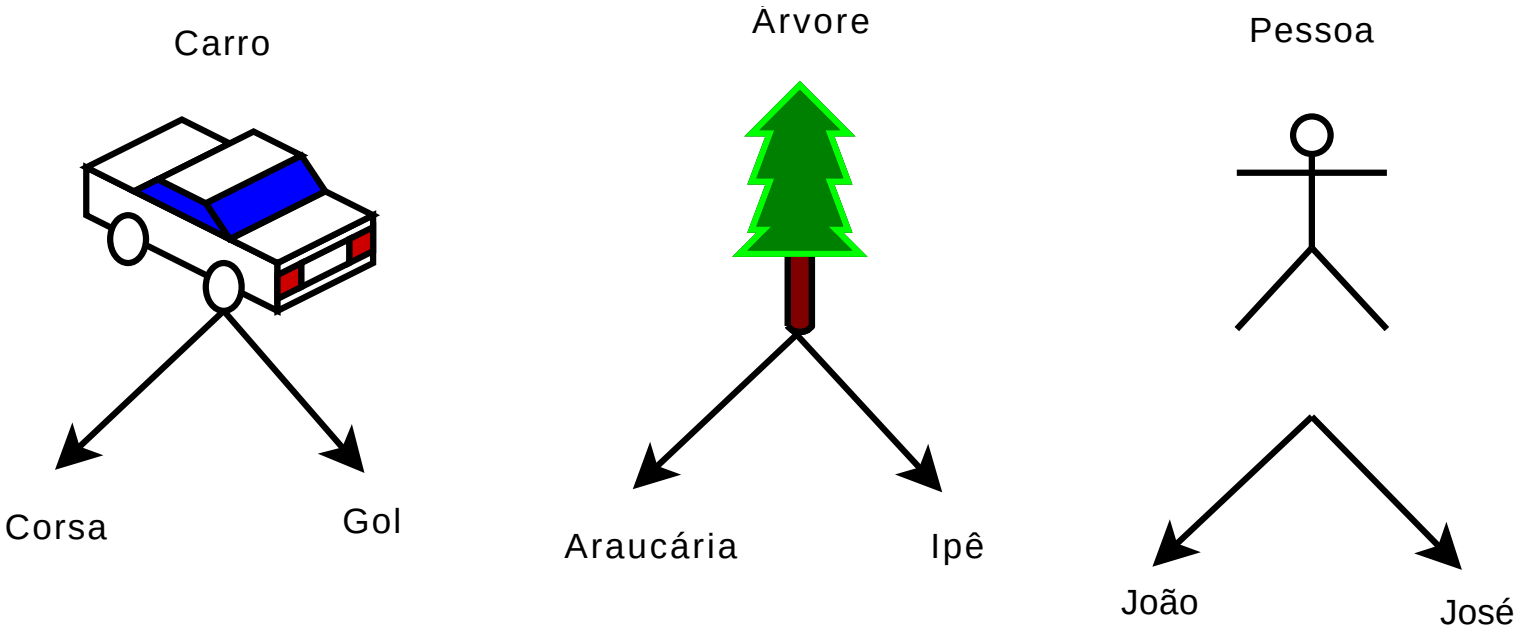


Classe



Objetos

Identifique as classes e os objetos



Uma classe em Java

```
14 public class Carro{
15     // Atributos
16     private double velocidade;
17     private String marca;
18     private String modelo;
19     // Métodos
20     public void acelerar(double intensidade){
21         ...
22     }
23     public void frear(double intensidade){
24         ...
25     }
26     public String obterMarca(){
27         return marca;
28     }
29     public void imprimirVelocidade(){
30         System.out.println("Velocidade: " + velocidade);
31     }
32 }
```

Exemplo de classe com declaração não ideal de atributos.

```
32 public class CarroNaoIdeal{
33    // Atributos
34    public float velocidade;
35
36    // Métodos
37    public void definirVelocidade(float v){
38       velocidade = v;
39    }
40    public void acelerar(float v){
41       // O carro só pode atingir 200km/h
42       if ((velocidade + v) <= 200){
43          velocidade += v;
44       }else{
45          velocidade = 200;
46       }
47    }
48 }
```

Como acessar os métodos da classe Carro?

```
48 public static void main(String args[]){
49    CarroNaoIdeal fusca = new CarroNaoIdeal();
50
51    // alterando a velocidade atraves dos metodos do objeto
52    fusca.definirVelocidade(150);// velocidade = 150
53    fusca.acelerar(400); // velocidade = 200
54
55    // alterando diretamente o valor do atributo
56    fusca.velocidade = 400;
57 }
```

Exemplo de classe com declaração ideal.

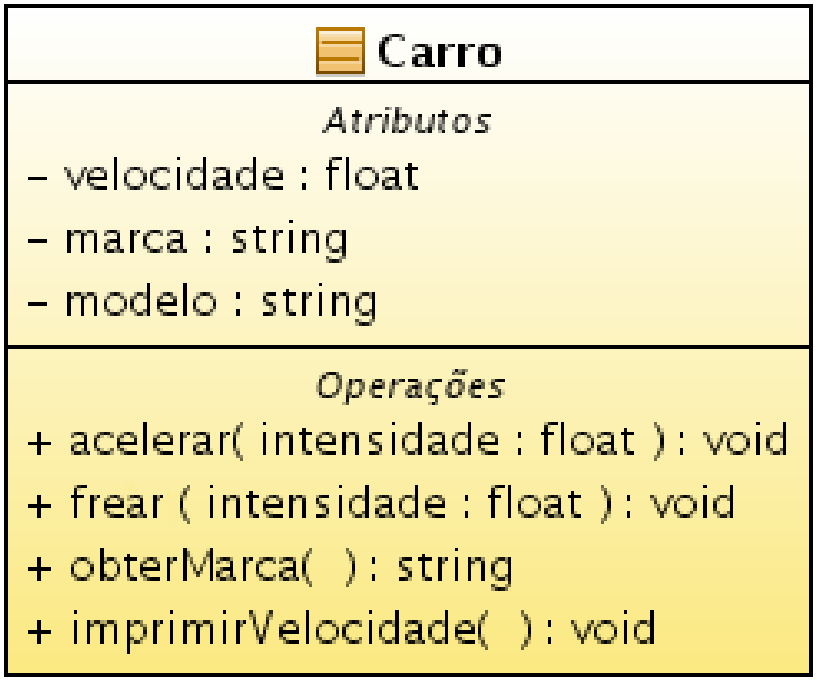
```
57 public class CarroIdeal{
58    // Atributos
59    private float velocidade;
60
61    // Métodos
62    public void definirVelocidade(float v){
63       velocidade = v;
64    }
65    public void acelerar(float v){
66       // O carro só pode atingir 200km/h
67       if ((velocidade + v) <= 200){
68          velocidade += v;
69       }else{
70          velocidade = 200;
71       }
72    }
73 }
```

Como acessar os métodos da classe Carro...

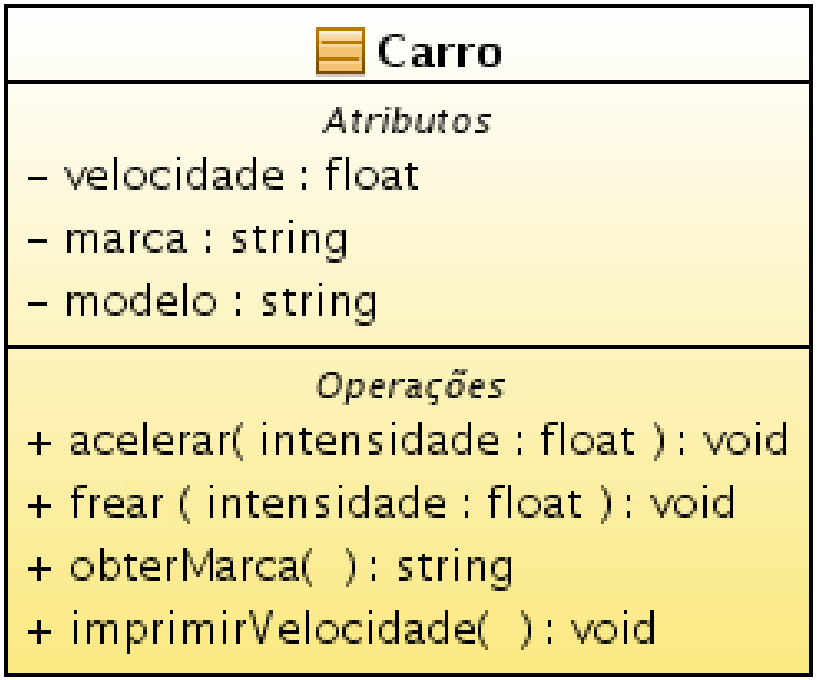
```
73 public static void main(String args[]){
74    CarroIdeal fusca = new CarroIdeal();
75
76    // Alterando a velocidade através dos métodos do objeto
77    fusca.definirVelocidade(150);// velocidade = 150
78    fusca.acelerar(400); // velocidade = 200
79
80    // alterando diretamente o valor do atributo
81    fusca.velocidade = 400; // ERRO ! não irá compilar
82 }
```

- Existe um problema nesse código, qual é?

Representação gráfica em UML da classe Carro



Representação gráfica em UML da classe Carro



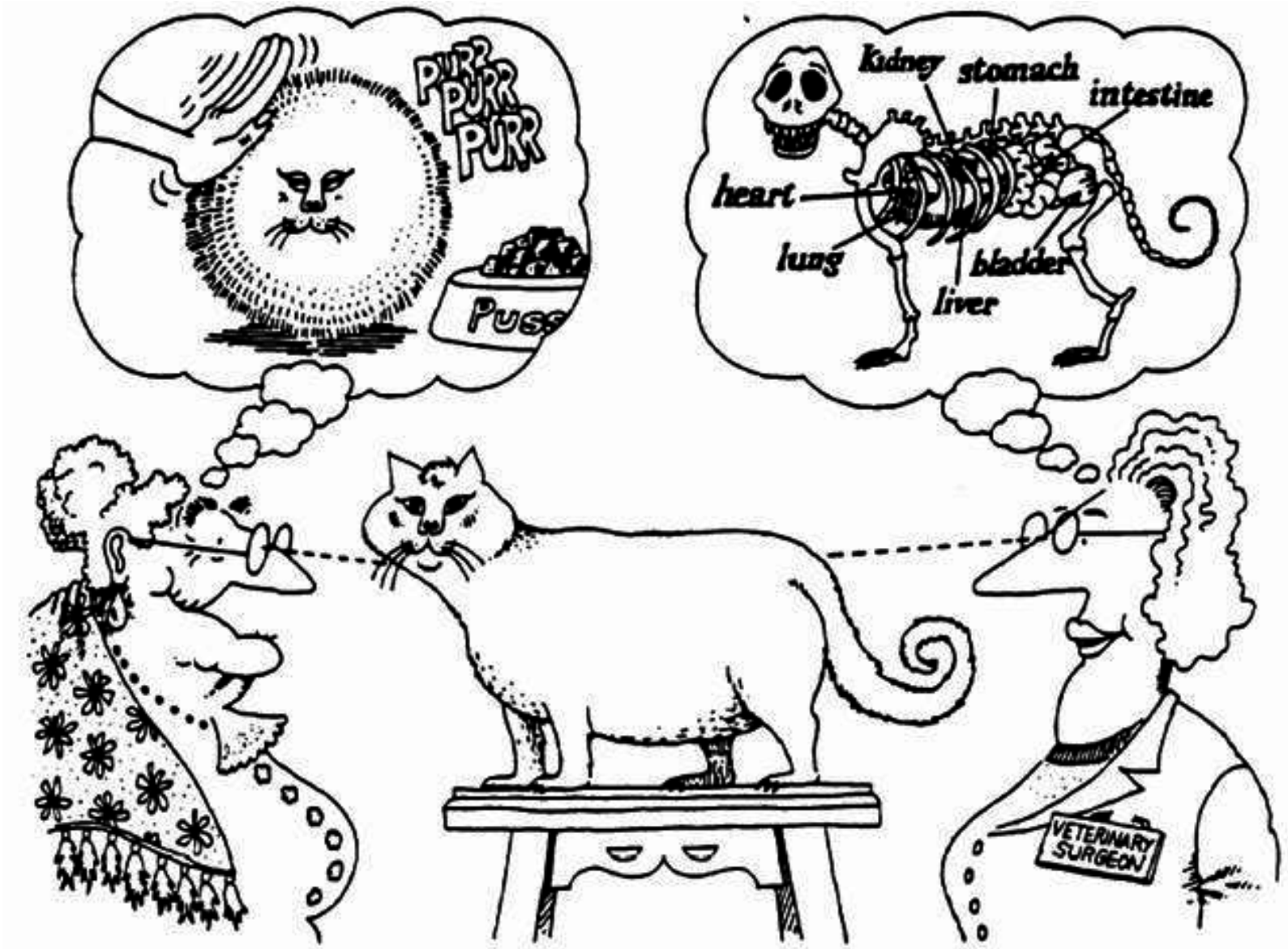
Linguagem de Modelagem Unificada – UML

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.

Primeiro exercício: Um contador

Primeiro exercício: Um contador

- A classe `Contador` possui um único **atributo**
  - `valorAtual`
- Quais `métodos` a classe Contador pode prover?
  - Atribuir um valor ao contador

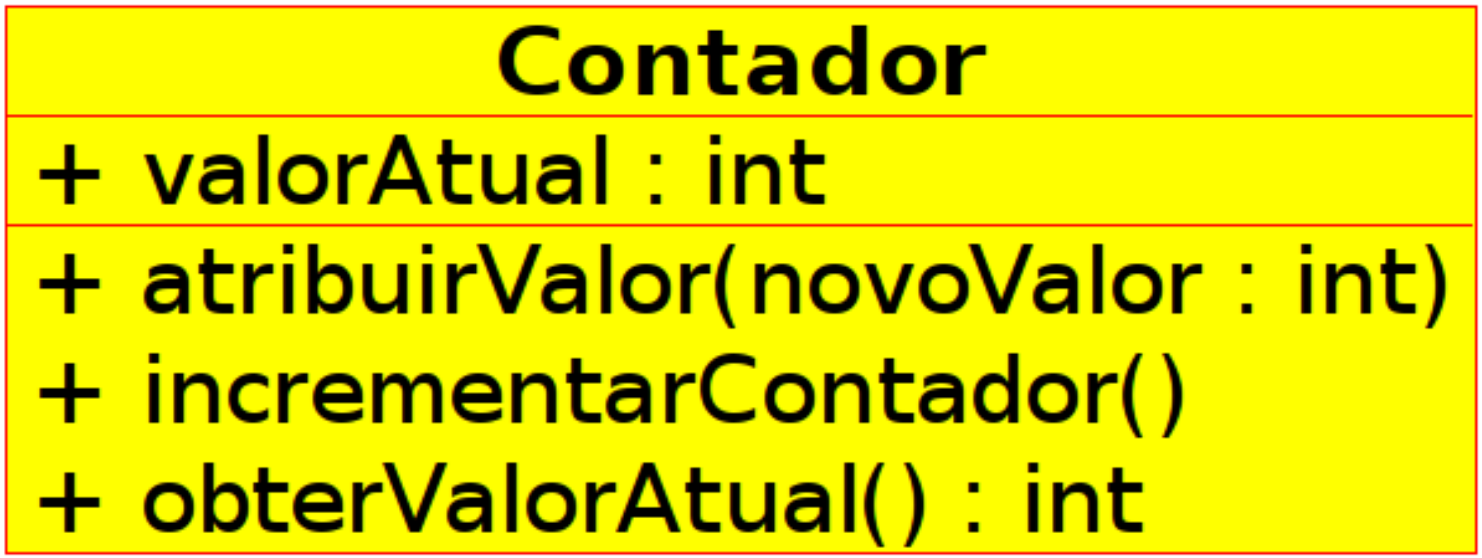
Primeiro exercício: Um contador

- A classe `Contador` possui um único **atributo**
  - `valorAtual`
- Quais `métodos` a classe Contador pode prover?
  - Atribuir um valor ao contador
  - Incrementar o contador

Primeiro exercício: Um contador

- A classe `Contador` possui um único **atributo**
  - `valorAtual`
- Quais `métodos` a classe Contador pode prover?
  - Atribuir um valor ao contador
  - Incrementar o contador
  - Obter o atual valor do contador

Representação gráfica em UML da classe Contador



Segundo exercício: Uma bicicleta

Segundo exercício: Uma bicicleta

- A classe `Bicicleta` possui muitos **atributos**:
  - 1 - `Pense em um contexto e colete informações sobre quais atributos uma bicicleta pode ter.`
  - 2 - Desenhe uma classe em UML para representar este objeto, de acordo com as informações que levantou no item anterior
  - 3 - Implemente em Java a classe para bicicleta e um aplicativo Java (classe Java com método `main`). Crie um objeto da classe Bicicleta e invoque alguns de seus métodos.

Como fazer? Exemplo para Classe Bicicleta

```
82  public class Bicicleta{
83     // Atributos
84     private double velocidade;
85     private String cor;
86     private int marchaAtual;
87     // Métodos
88     public void frearDianteiro(double intensidade){
89         velocidade -= intensidade; }
90     public void definirVelocidade(double v){
91         velocidade = v; }
92     public void acelerar(double v){
93         velocidade += v; }
94     public String obterCor(){
95         return cor;  }
96     public void trocarMarchaParaLeve(int v){
97         marchaAtual -= v; }
98     public void mostrarVelocidade(){
99         System.out.println("Velocidade: " + velocidade); }
100 }
```

Como fazer? Exemplo de como instanciar e utilizar os métodos

```
100 package bicicleta;
101 public class BicicletaInstanciacao{
102     public static void main(String args[]){
103         Bicicleta mb = new Bicicleta();
104         // Alterando a velocidade através dos metodos do objeto
105         mb.definirVelocidade(15);// velocidade = 15
106         System.out.println( "Def. vel: " + mb.qualVelocidade());
107 
108         mb.acelerar(1.1);// velocidade = 16.1
109         System.out.println( "Acelerei: " + mb.qualVelocidade());
110 
111         mb.acelerar(3); // velocidade = 19.1
112         System.out.println("Acelerei: " + mb.qualVelocidade());
113 
114         // alterando diretamente o valor do atributo
115         mb.velocidade = 20; // problemas, porque?
116     }
117 }
```

Leitura recomendada

Caelum Ensino e Soluções em Java
Apostila Caelum FJ-11 Java e Orientação a Objetos
`http://www.sj.ifsc.edu.br/~mello/livros/java`

- Capítulos 4 e 5