Instituto de computação

Revisão de Orientação a Objetos

Prof. Lauro Eduardo Kozovits, D.Sc.

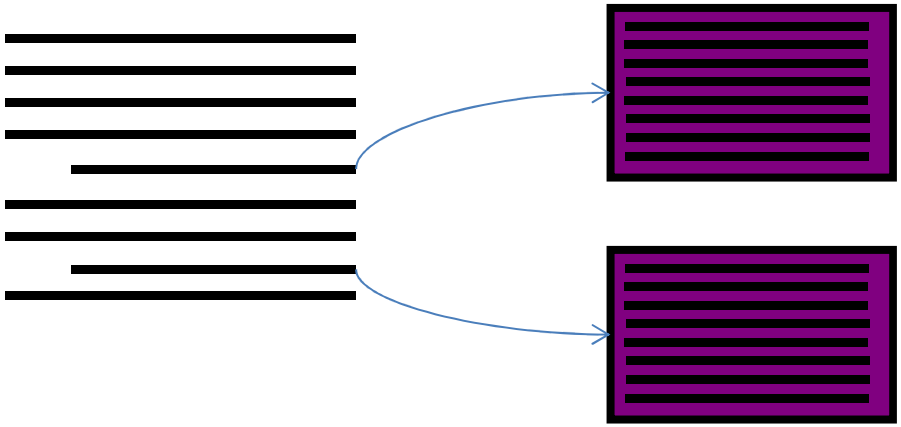
lauro@ic.uff.br

Utilizando material original do

Prof.Leonardo Gresta Paulino Murta

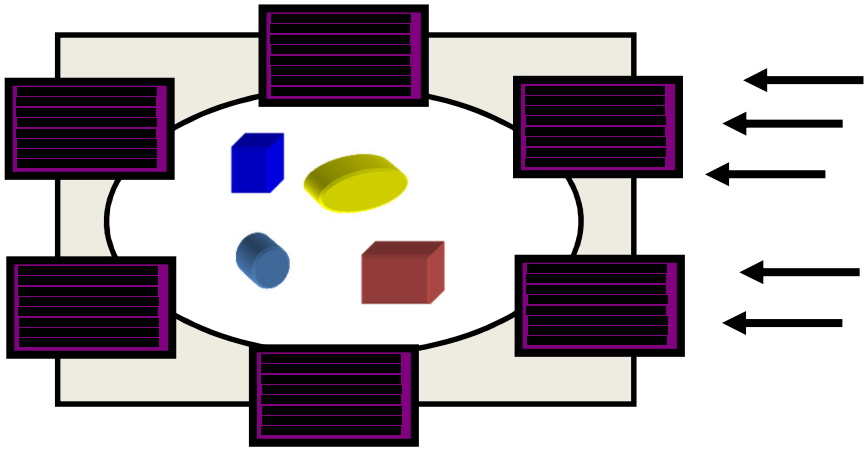
Paradigma procedimental

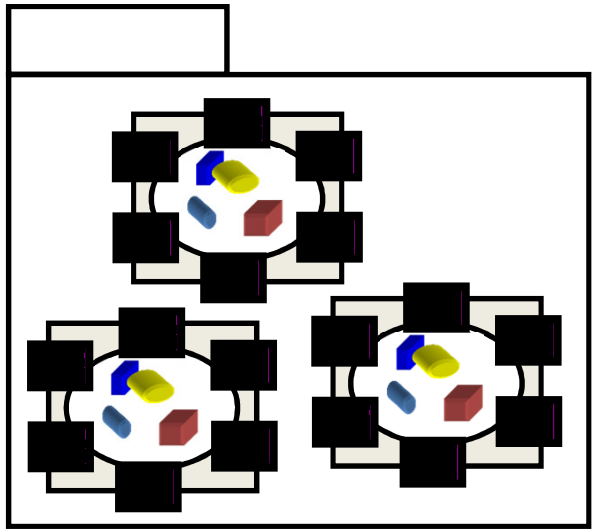
- Sinônimo: paradigma procedural
- Uso de subprogramação
  - Agrupamento de código permitindo a criação de ações complexas
  - Atribuição de um nome para essas ações complexas
  - Chamada a essas ações complexas de qualquer ponto do programa
- Essas ações complexas são denominadas procedimentos, sub-rotinas e funções



Paradigma orientado a objetos (OO)

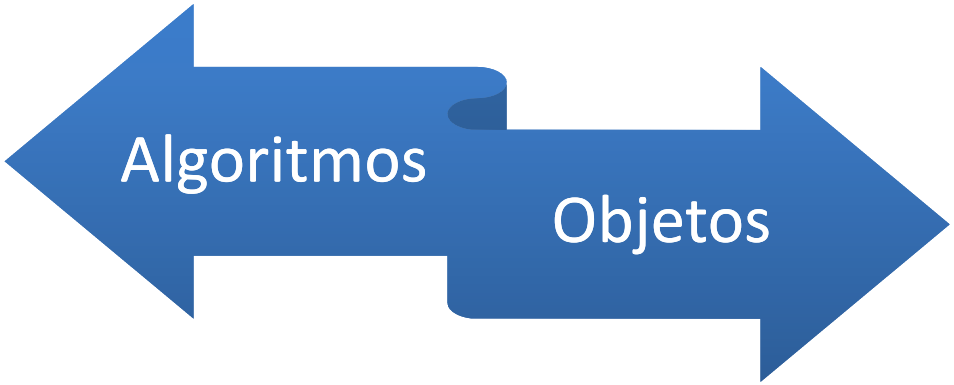
- Classes de objetos
  - Agrupamento de procedimentos e variáveis afins
- Pacotes de classes
  - Agrupamento de classes afins
  - Representam bibliotecas de apoio





Paradigma procedimental versus OO

- O **paradigma procedimental** organiza o programa em termos de **algoritmos**
- O **paradigma OO** organiza o programa em termos de **objetos**



~~Algoritmos~~Objetos

- Podemos criar programa pensando em termos de **objetos ao invés de algoritmos**?
- O mundo é composto de objetos
  - Uma loja tem produtos, pedidos, estoque, etc.
  - Um restaurante tem mesas, garçons, comidas, bebidas, etc.
  - Uma universidade tem professores, alunos, disciplinas, etc.
  - Uma rodoviária tem ônibus, passageiros, bagagens, etc.
- E se **criarmos programas** basicamente **criando objetos** equivalentes ao mundo real, e fazendo com que esses **objetos se comuniquem**?

Objetos

- Definição
  - Um objeto é a **representação computacional de um elemento ou processo do mundo real**
  - Cada objeto possui suas **características** e seu **comportamento**

- Exemplos de Objetos

cadeira	*mesa*	*caneta*	*lápis*
carro	*piloto*	*venda*	*mercadoria*
cliente	*aula*	*programa*	*computador*
aluno	*avião*		

Características de objetos

- Definição
  - Uma característica descreve uma propriedade de um objeto, ou seja, algum elemento que descreva o objeto.
  - Cada característica é chamada de **atributo** e funciona como uma **variável** pertencente ao objeto

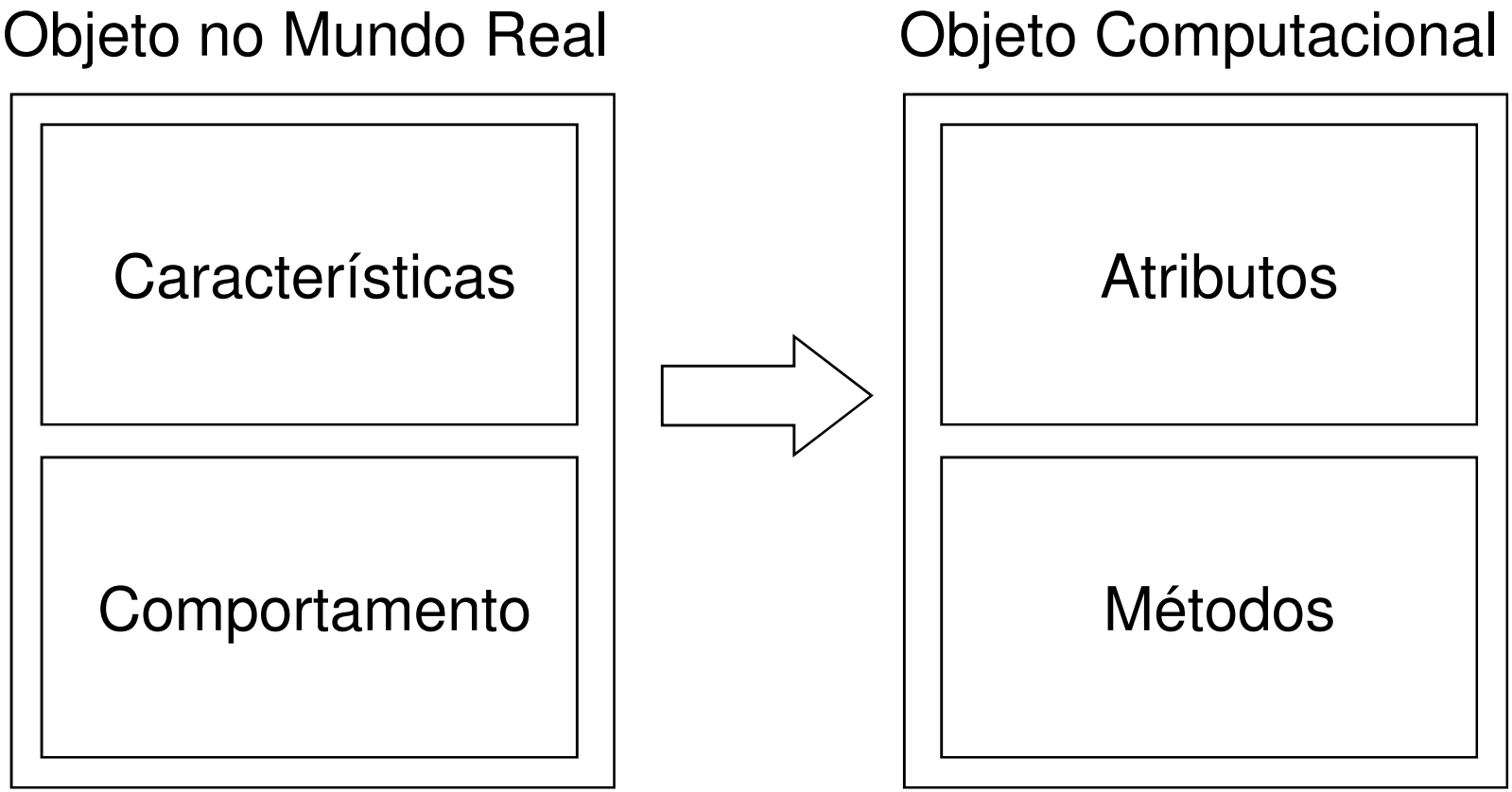
- Exemplo de características do objeto **carro**
  - Cor
  - Marca
  - Número de portas
  - Ano de fabricação
  - Tipo de combustível

Comportamento de objetos

- Definição
  - Um comportamento representa uma ação ou resposta de um objeto a uma ação do mundo real
  - Cada comportamento é chamado de **método** e funciona como um **procedimento/função** pertencente ao objeto

- Exemplos de comportamento para o objeto **carro**
  - Acelerar
  - Frear
  - Virar para direita
  - Virar para esquerda

Mapeamento de objetos



Paradigma procedimental versus OO (exemplo: Agenda)

Paradigma Procedimental

- Variáveis
  - Vetor de nomes
  - Vetor de endereços
  - Vetor de telefones
- Procedimentos
  - Listagem de todos os nomes
  - Listagem do endereço dado um nome
  - Listagem do telefone dado um nome
  - Adição de nome, endereço e telefone
  - Remoção de nome, endereço e telefone

Paradigma OO

- Objeto Agenda
  - Atributo
    - Vetor de Contatos
  - Métodos
    - Listagem de Contatos
    - Adição de um Contato
    - Remoção de um Contato
- Objeto Contato
  - Atributos
    - Nome
    - Endereço
    - Telefone
  - Métodos
    - Exibição de nome, endereço e telefone
    - Edição de nome, endereço e telefone

Paradigma OO
(exemplo: total da compra)

Pedido: 12345
Cliente: João da Silva
Endereço: Rua dos Bobos, número zero

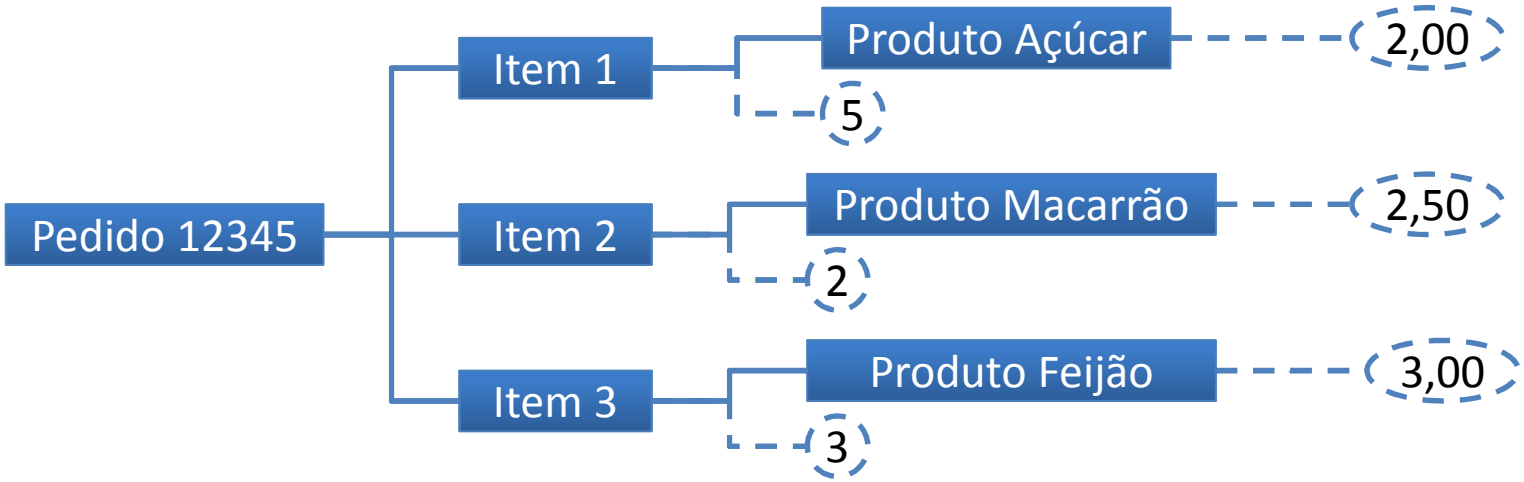
Item	Produto	Preço	Quantidade	Subtotal
1	Açúcar	R$ 2,00	5	R$ 10,00
2	Macarrão	R$ 2,50	2	R$ 5,00
3	Feijão	R$ 3,00	3	R$ 9,00
			TOTAL	R$ 24,00

Quais são os objetos participantes do cálculo do total da compra?

Paradigma OO
(exemplo: total da compra)

Pedido: 12345
Cliente: João da Silva
Endereço: Rua dos Bobos, número zero

Item	Produto	Preço	Quantidade	Subtotal
1	Açúcar	R$ 2,00	5	R$ 10,00
2	Macarrão	R$ 2,50	2	R$ 5,00
3	Feijão	R$ 3,00	3	R$ 9,00
			TOTAL	R$ 24,00

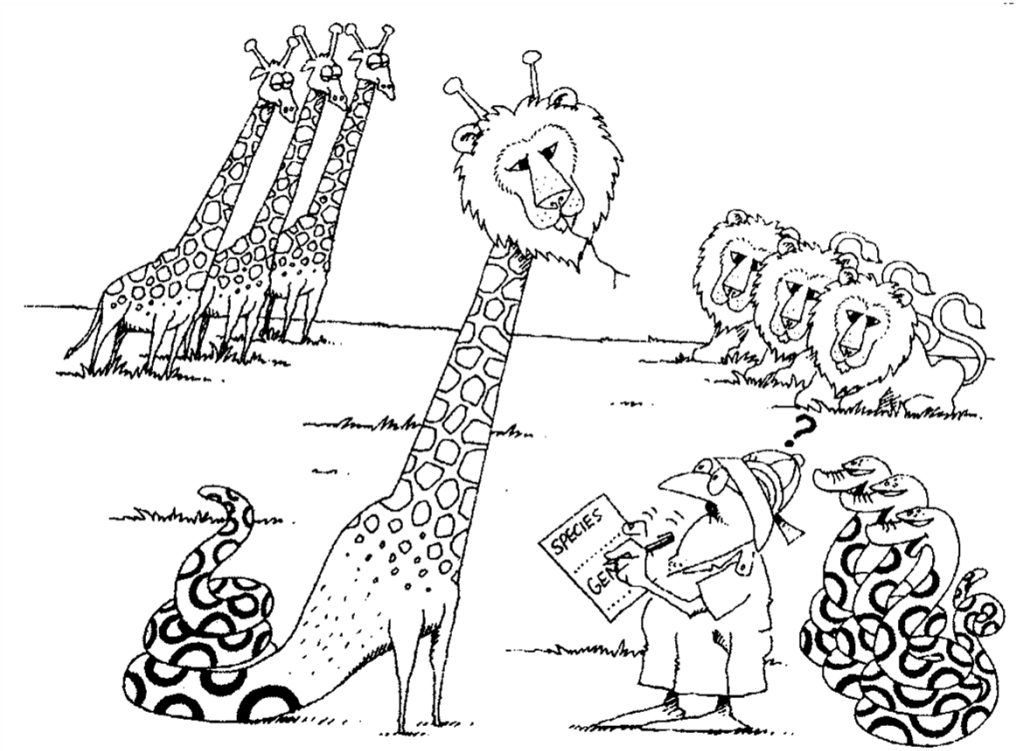


Paradigma OO
(exemplo: total da compra)

- Como obter o total da compra?
  1. O objeto **Caixa** pediria ao objeto **Pedido** seu valor total
  2. O objeto **Pedido**, por sua vez, percorreria todos os seus objetos **Item** perguntando o seu valor subtotal e somaria esses valores para responder ao objeto **Caixa**
  3. Cada objeto **Item** perguntaria ao objeto **Produto** o seu preço e multiplicaria esse preço pela quantidade que está sendo comprada, para responder ao objeto **Pedido**

Classes versus objetos

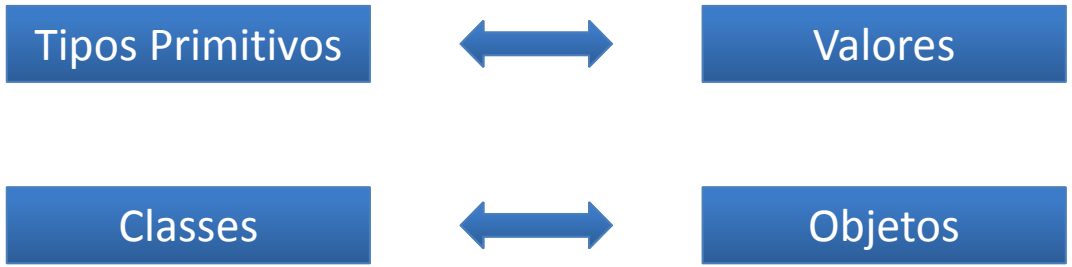
- A **Classe** é o **tipo do Objeto**



Fonte: livro “Object-Oriented Analysis and Design with Applications”

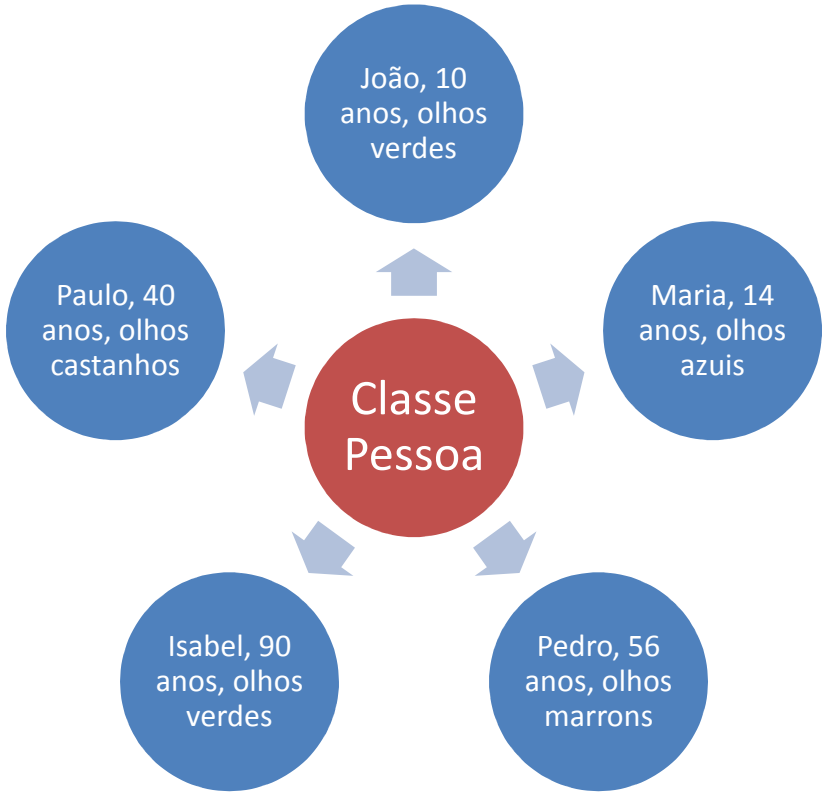
Classes versus objetos

- Valores têm tipos primitivos
  - 123 é um valor inteiro
  - True é um valor booleano
  - 12,3 é um valor real
- Objetos pertencem a classes
  - João, Pedro e Paulo são da classe Pessoa
  - Fusca e Ferrari são da classe Carro
  - Flamengo e Fluminense são da classe Time



Classes versus objetos

- Uma **classe é uma fôrma**, capaz de **produzir objetos**
- Os **programadores criam classes**, as **classes instanciam objetos**



Classes

- A classe descreve as características e comportamento de um conjunto de objetos
  - O objeto possuirá os atributos e métodos definidos na classe
  - O objeto é chamado de instância de sua classe
  - A classe é o bloco básico para a construção de programas OO

Exemplo de classe

```
public class Carro {
  private int velocidade;

  public void acelera() {
    velocidade++;
  }

  public void freia() {
    velocidade--;
  }
}
```

Atributos (características) são variáveis globais acessíveis por todos os métodos da classe

Métodos (comportamentos)

Exercício

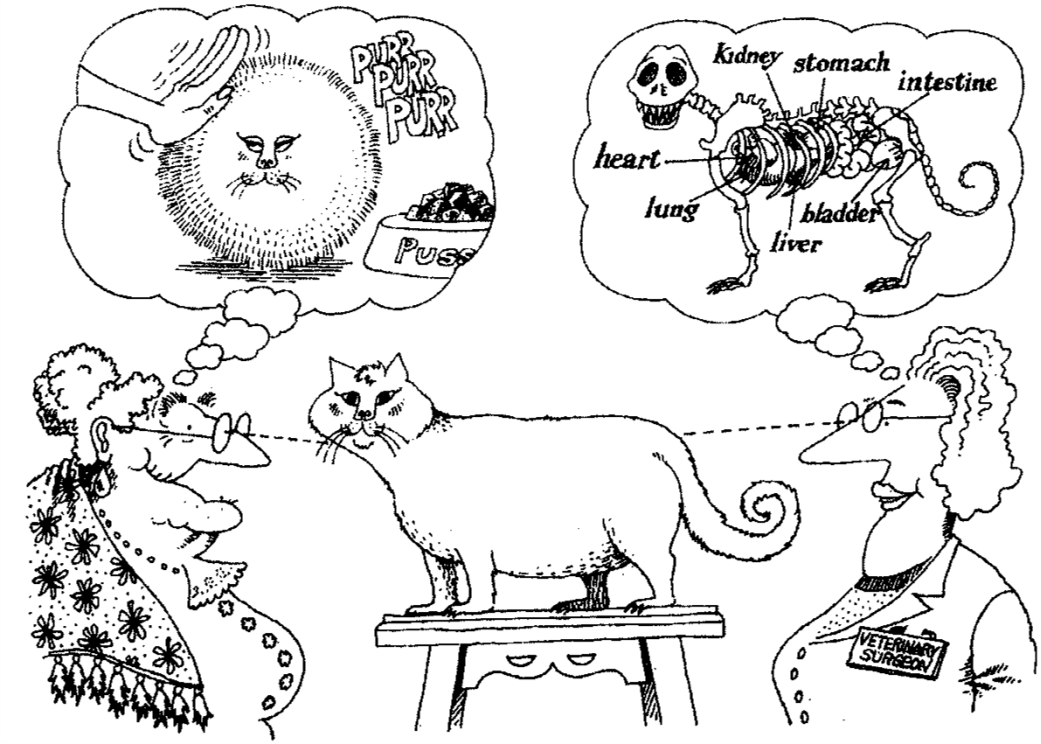
- Identifique as classes para a seguinte especificação:

"O supermercado vende diferentes tipos de produtos. Cada produto tem um preço e uma quantidade em estoque. Um pedido de um cliente é composto de itens, onde cada item especifica o produto que o cliente deseja e a respectiva quantidade. Esse pedido pode ser pago em dinheiro, cheque ou cartão."

Princípios do paradigma OO

Abstração

- A representação computacional do objeto real deve se concentrar nas características que são relevantes para o problema



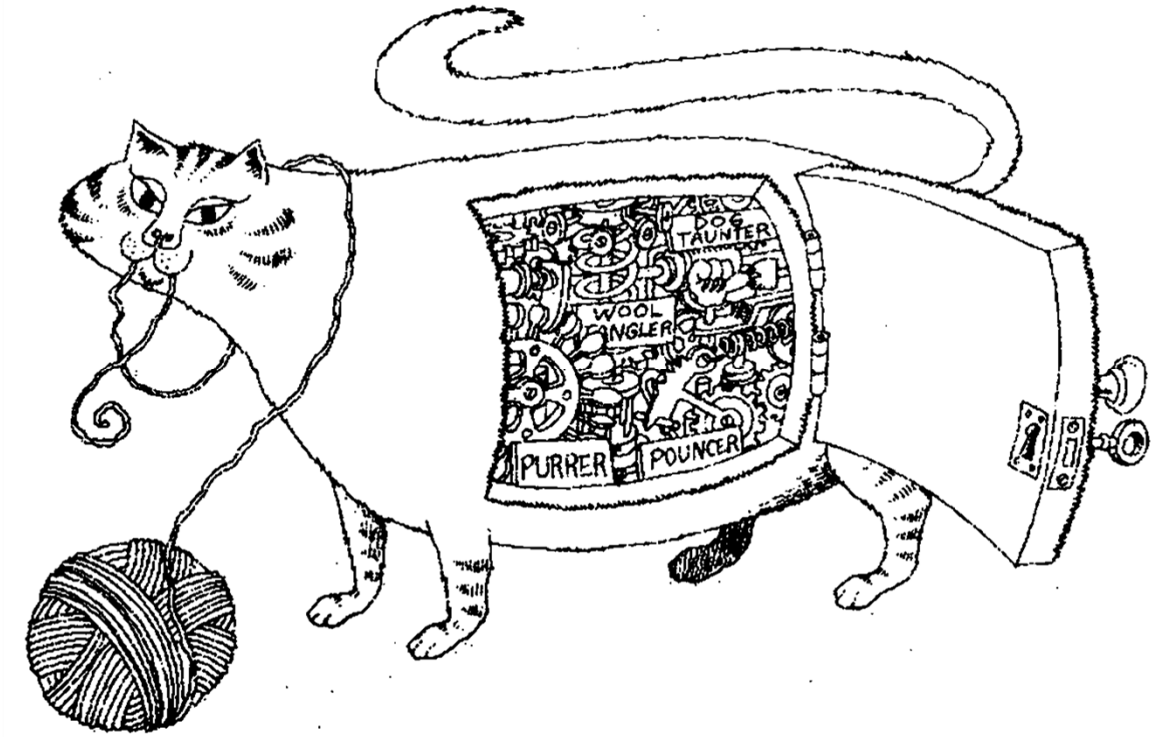
Fonte: livro “Object-Oriented Analysis and Design with Applications”

Abstração

- São criados somente os atributos e métodos necessários para o problema em mãos
- Quais seriam os atributos e métodos para o objeto Carro em cada uma das situações seguintes?
  - Sistema de uma locadora de carros
  - Sistema de uma revendedora de carros
  - Sistema de uma oficina mecânica
  - Sistema do DETRAN

Encapsulamento

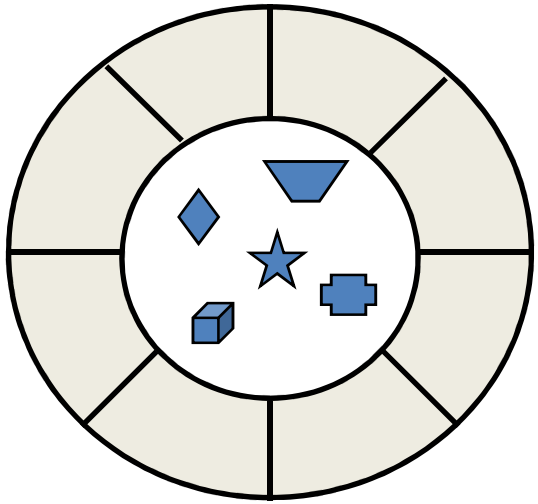
- O objeto deve esconder seus dados e os detalhes de sua implementação



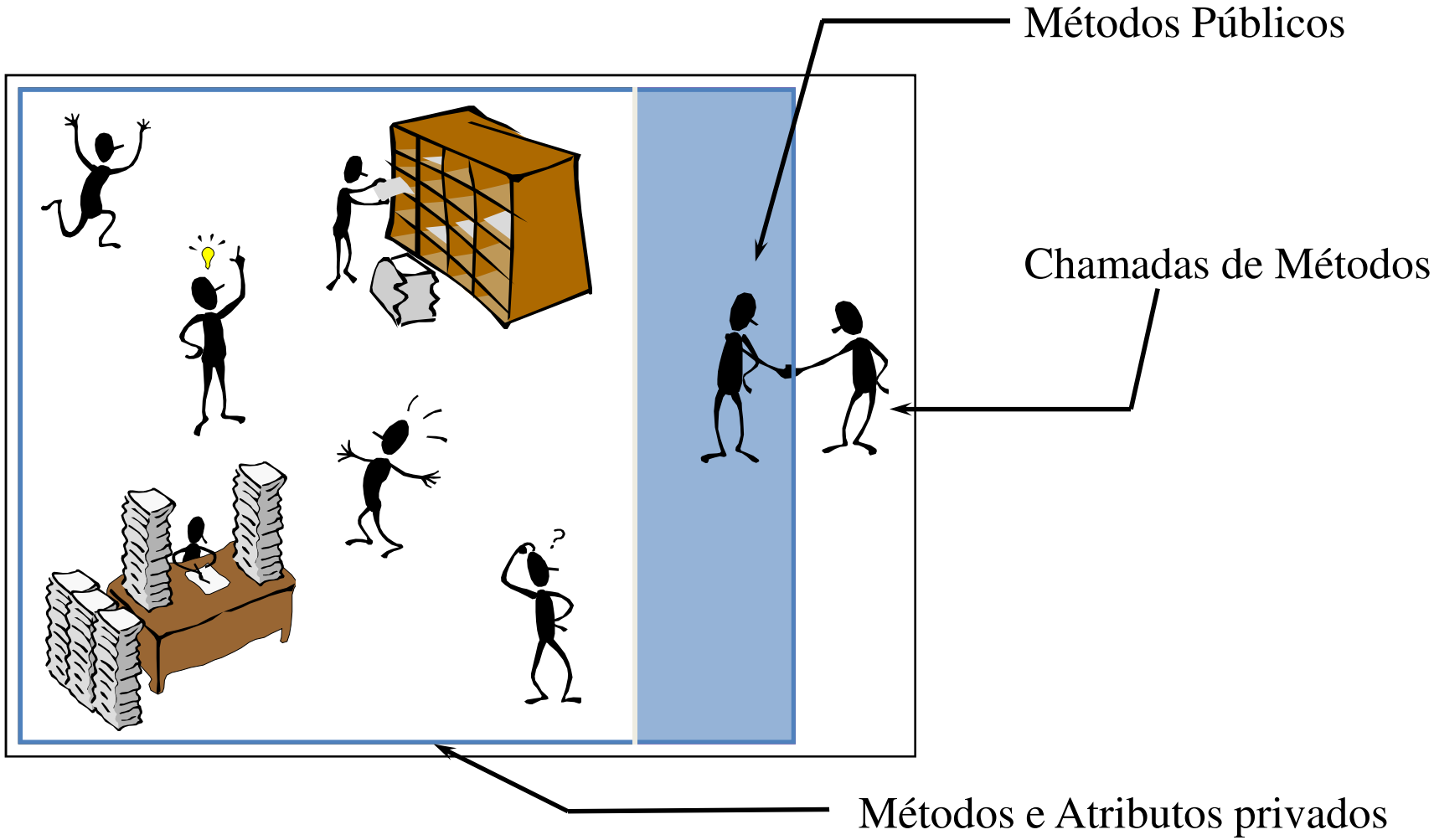
Fonte: livro “Object-Oriented Analysis and Design with Applications”

Encapsulamento

- Atributos e Métodos
  - Os métodos formam uma “cerca” em torno dos atributos
  - Os atributos não devem ser manipulados diretamente
  - Os atributos somente devem ser alterados ou consultados através dos métodos do objeto



Encapsulamento



Encapsulamento

- Em uma classe Quadrado, quais métodos devem ser públicos e quais devem ser privados?
  - Transladar
  - Ampliar
  - Mover ponto
  - Girar
  - Adicionar ponto
  - Calcular área

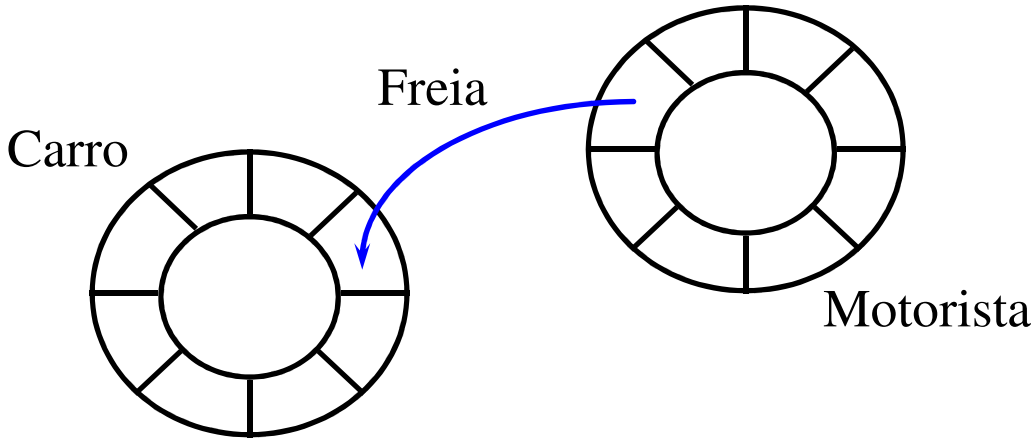
Modularidade

- Um sistema deve ser decomposto em um conjunto altamente coeso e fracamente acoplado de objetos

Fonte: livro “Object-Oriented Analysis and Design with Applications”

Modularidade

- Um programa OO é um conjunto de objetos que colaboram entre si para a solução de um problema
- Objetos colaboram através de chamadas de métodos uns dos outros

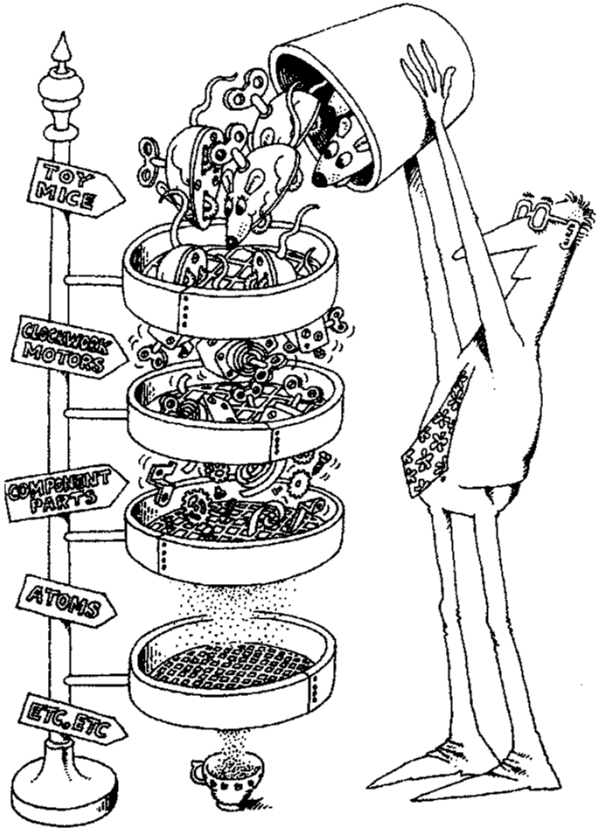


Modularidade

- Em um sistema acadêmico, há conceitos aluno, professor, disciplina, turma e inscrição. Onde colocar cada um dos métodos a seguir:
  - Exibição do histórico do aluno
  - Cálculo da média do aluno em uma turma
  - Obtenção do horário de uma aula
  - Descrição da ementa de uma disciplina
  - Cálculo do CR de um aluno

Hierarquia

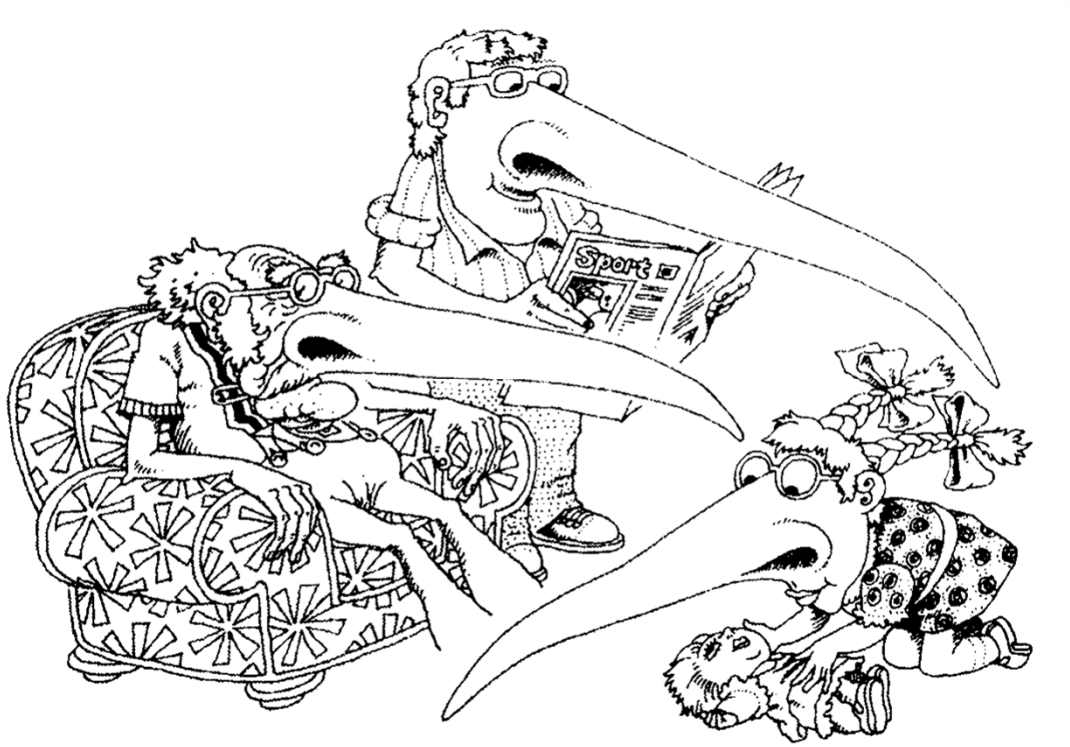
- Os objetos devem ser organizados no sistema de forma hierárquica



Fonte: livro “Object-Oriented Analysis and Design with Applications”

Hierarquia

- Objetos herdam atributos e métodos dos seus ancestrais na hierarquia



Fonte: livro “Object-Oriented Analysis and Design with Applications”

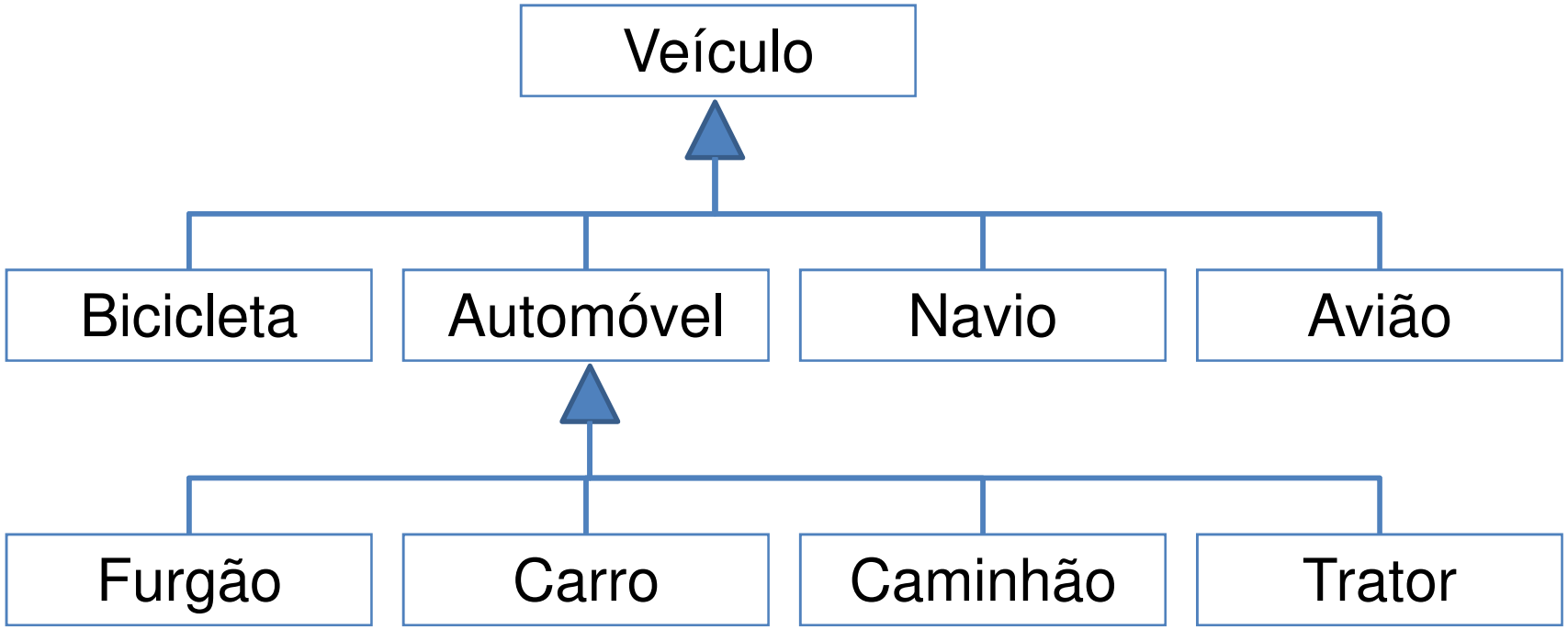
Herança

- Para viabilizar a hierarquia entre objetos, as classes são organizadas em estruturas hierárquicas
  - A classe que forneceu os elementos herdados é chamada de **superclasse**
  - A classe herdeira é chamada de **subclasse**
  - A subclasse pode **herdar os métodos e atributos** de suas superclasses
  - A subclasse pode **definir novos atributos e métodos** específicos

Polimorfismo

- Uma subclasse pode redefinir (sobrescrever) um método herdado
  - Este mecanismo é chamado de **polimorfismo**
  - O polimorfismo se realiza através da recodificação de um ou mais métodos herdados por uma subclasse
  - Em tempo de execução, a implementação mais específica será usada

Exemplo de herança



Teste da Leitura: "subclasse é um superclasse"
Ex.: Carro é um Automóvel; Trator é um Veículo; ...

Exemplo de herança (relembrando a classe Carro)

```
public class Carro {
  private int velocidade;

  public Carro(int velocidadeInicial) {
    velocidade = velocidadeInicial;
  }

  public void acelera() {
    velocidade++;
  }

  public void freia() {
    velocidade--;
  }
}
```

Exemplo de herança (criando um carro inteligente)

- Declaração:

```
public class CarroInteligente extends Carro {
  public CarroInteligente(int velocidadeInicial) {
    super(velocidadeInicial);
  }

  public void estaciona() {
    // código mágico para estacionar sozinho
  }
}
```

- Uso:

```
CarroInteligente tiguan = new CarroInteligente(10);
for (int i = 10; i > 0; i--) {
  tiguan.freia();
}
tiguan.estaciona();
```

De onde veio isso?

Exemplo de polimorfismo (criando um carro de corrida)

- Declaração:

```
public class CarroCorrida extends Carro {
  public CarroCorrida(int velocidadeInicial) {
    super(velocidadeInicial);
  }

  public void acelera() {
    velocidade+=5;
  }
}
```

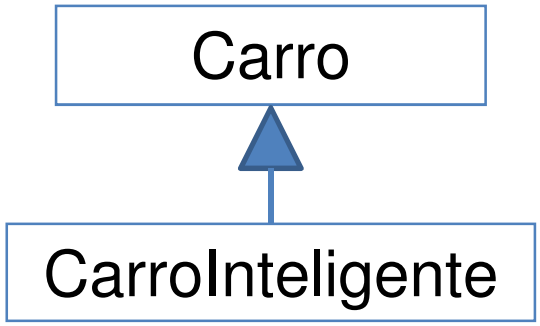
- Uso:

```
CarroCorrida f1 = new CarroCorrida(10);
f1.acelera();
```

Qual a velocidade agora?

Compatibilidade de tipos

- Qualquer **subclasse** é compatível com a sua **superclasse**
  - Contudo, a reciproca não é verdadeira



✓
```
Carro c = new CarroInteligente(20);
c.acelera();
c.freia();
```

✗
```
CarroInteligente c = new Carro(20);
c.acelera();
c.freia();
c.estaciona();
```

Exercício de herança

- Em um sistema de loja, há 3 tipos de usuário: gerente, funcionário e cliente. Todo usuário tem nome e senha. O cliente possui, além do nome e senha, outros dados cadastrais. O funcionário possui métodos relacionados a venda de produtos. O gerente pode fazer tudo que o funcionário pode e também fechamento do caixa. Como é a hierarquia de herança desse sistema no que se refere a controle de usuários?

Interfaces

- Tipo especial de classe, que não tem implementação
  - Uma interface define um protocolo
  - Classes podem implementar uma ou mais interfaces

- Uma interface é um contrato assinado por uma classe
  - A interface define as responsabilidades da classe
  - As responsabilidades são mapeadas em métodos
  - A classe que implementa a interface implementa os métodos
  - A interface contém somente assinatura de métodos e constantes

Programação orientada a interfaces


```
public interface Stack {
  public Object pop();
  public void push(Object o);
  public int size();
}
```

Programação orientada a interfaces

```
adicionaPedido(Stack s) {
  Pedido p = new Pedido(...);
  s.push(p);
}
```

Desconhece a implementação

```
public interface Stack {
  public Object pop();
  public void push(Object o);
  public int size();
}
```

Programação orientada a interfaces


```
public interface Stack {
  public Object pop();
  public void push(Object o);
  public int size();
}
```

Desconhece o uso

```
public class MyStack implements Stack {
  public void push(Object o) {
  ...
```

Classes abstratas

- Uma classe que possui algum método sem implementação (abstratos)
- Classes abstratas não podem ter instâncias

```
abstract class Carro
{
    <atributos da classe Carro>
    <métodos comuns da classe Carro>
    <métodos abstratos da classe Carro>

}
```

Exemplo de Classe Abstrata

```
public abstract class Pagamento {
   ...
   public abstract void paga(...);
}
```

```
public class PagamentoVisa extends Pagamento {
   ...
   public void paga(...) {
      // código de pagamento com cartão Visa
   }
}
```

Pacotes

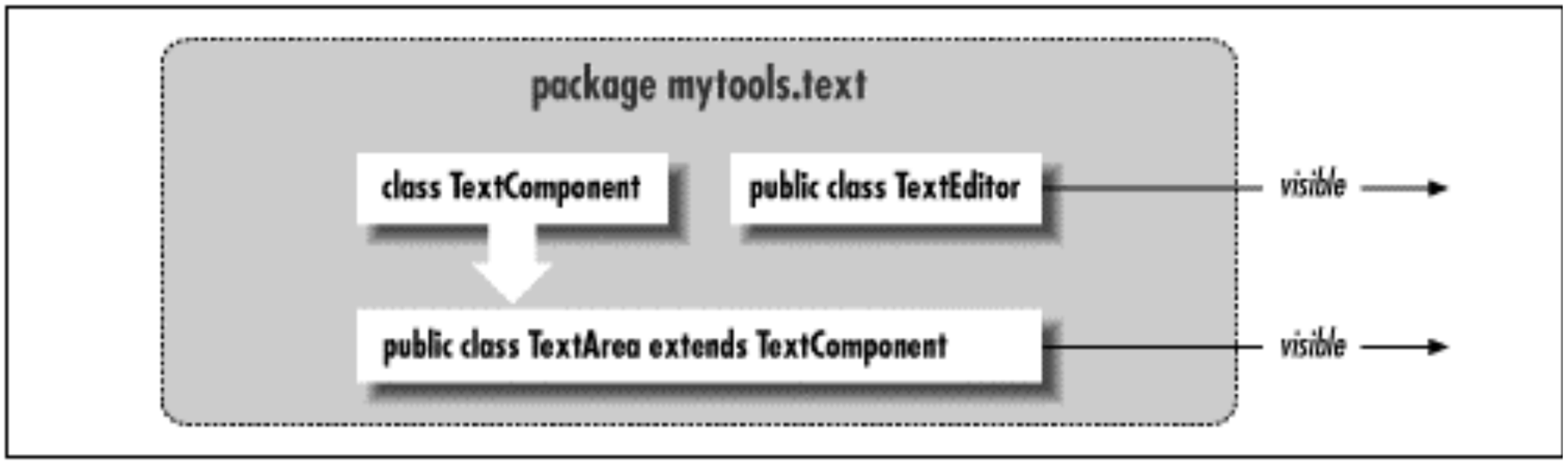
- Utilizados para agregar classes relacionadas

```
package br.uff.ic;

public class Pessoa {
   ...
}
```

Pacotes

- Modificadores permitem que determinadas classes sejam visíveis apenas para outras classes do mesmo pacote



Pacotes

- Sempre que for usar uma classe de outro pacote, é necessário importar

```
package br.uff.ic;

import java.util.List;

public class Pessoa {
   ...
}
```

Modificador de visibilidade

- Indica quem pode acessar o método (ou atributo):
  - O modificador ***private*** indica que o método pode ser chamado apenas por outros métodos da própria classe
  - A ausência de modificador é conhecida como ***package***, e indica que o método pode ser chamado somente por classes do mesmo pacote
  - O modificador ***protected*** indica que o método pode ser chamado somente por classes do mesmo pacote ou subclasses;
  - O modificador ***public*** indica que o método pode ser chamado por qualquer outra classe

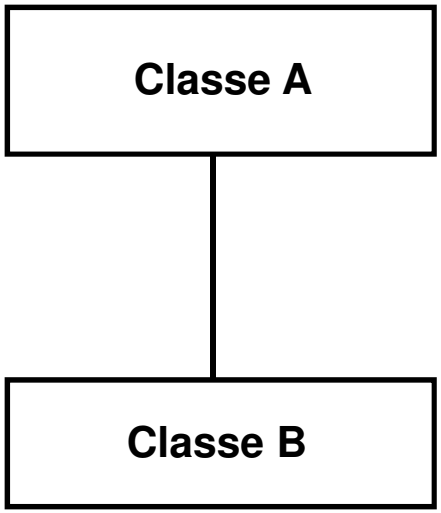
Modificador de escopo

- Indica a quem pertence o método (ou atributo)
  - Ao objeto (instância)
  - À classe como um todo
- Métodos estáticos (*static*) pertencem à classe como um todo
  - Podem ser chamados diretamente na classe, sem a necessidade de instanciar objetos
  - Só podem manipular atributos estáticos

Relacionamento entre classes

- Outras classes podem ser utilizadas como tipos dos atributos de uma determinada classe
- Neste caso, o atributo representa uma relação entre as duas classes
- O desenvolvedor deve definir a visibilidade da relação, ou seja, quais classes conhecem a relação

Objetos como atributos




```
class A
{
        private B       b;
        …
}

class B
{
        private A       a;
        …
}
```

Instituto de computação

Revisão de Orientação a Objetos

Prof. Lauro Eduardo Kozovits, D.Sc.

lauro@ic.uff.br

Utilizando material original do

Prof.Leonardo Gresta Paulino Murta