Instituto de computação uff

Subprogramação e Orientação a Objetos

Leonardo Gresta Paulino Murta

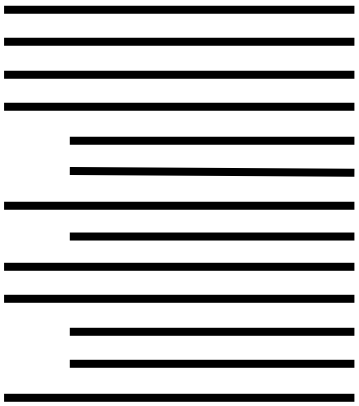
leomurta@ic.uff.br

Aula de hoje

- Estudaremos três estruturas de encapsulamento da Orientação a Objetos
  - Métodos
  - Classes
  - Pacotes

Paradigma estruturado

- Código mais fácil de ler, mas **ainda difícil para sistemas grandes** devido a repetição de código
  - Só usa sequência, repetição e decisão
- O que fazer se for necessário **repetir uma sequência de linhas de código** em diferentes locais?

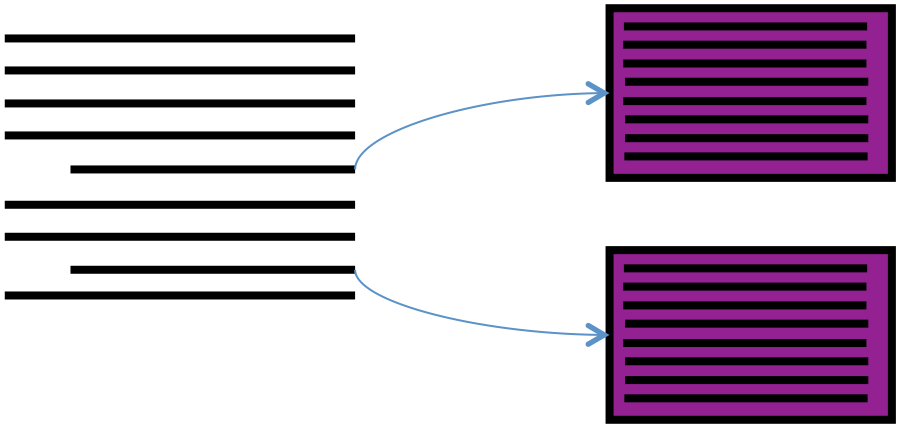


Encapsulamento

- Mecanismo utilizado para lidar com o aumento de complexidade
- Consiste em exibir “o que” pode ser feito sem informar “como” é feito
- Permite que a granularidade de abstração do sistema seja alterada, criando estruturas mais abstratas

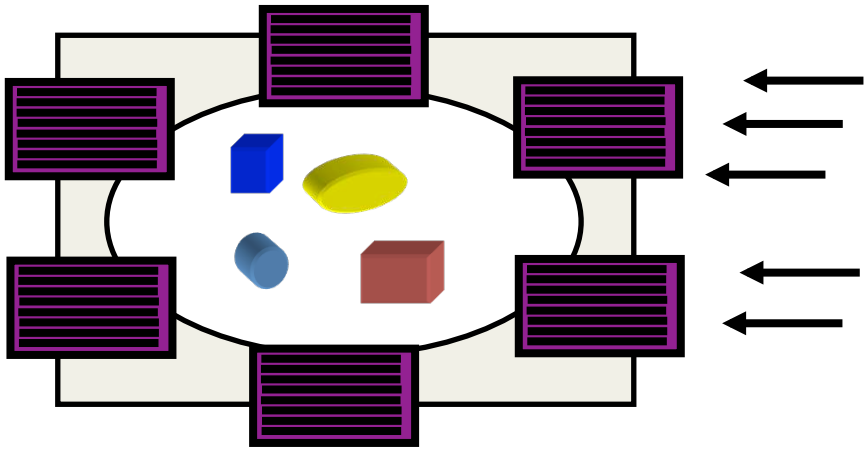
Paradigma procedimental

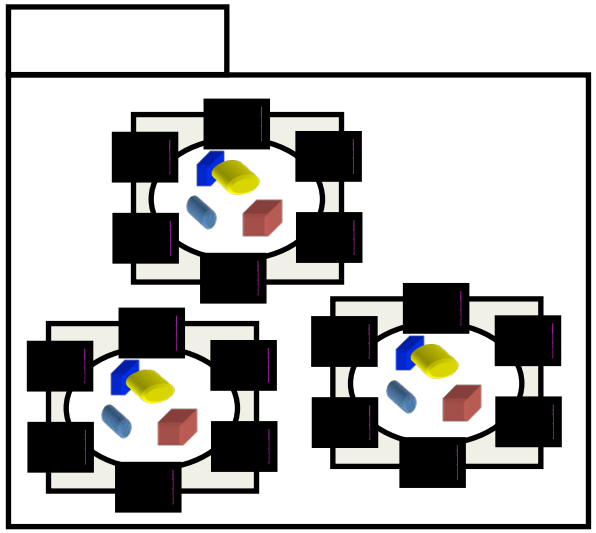
- Sinônimo: paradigma procedural
- Uso de subprogramação
  - Agrupamento de código permitindo a criação de ações complexas
  - Atribuição de um nome para essas ações complexas
  - Chamada a essas ações complexas de qualquer ponto do programa
- Essas ações complexas são denominadas procedimentos, sub-rotinas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





Parte I

MÉTODOS

Exemplo

```
import java.util.Scanner;
public class IMC {
  public static void main(String[] args) {
    Scanner teclado = new Scanner(System.in);

    System.out.print("Entre com a sua altura em metros: ");
    double altura = teclado.nextDouble();

    System.out.print("Entre com a sua massa em kg: ");
    double massa = teclado.nextDouble();

    double imc = massa / Math.pow(altura, 2);
    System.out.println("Seu IMC é " + imc);
  }
}
```

Parecidos!

Exemplo usando método

```
import java.util.Scanner;
public class IMC {
   public static double leia(String mensagem) {
      Scanner teclado = new Scanner(System.in);
      System.out.print(mensagem);
      return teclado.nextDouble();
   }

   public static void main(String[] args) {
      double altura = leia("Entre com a sua altura em metros: ");
      double massa = leia("Entre com a sua massa em kg: ");

      double imc = massa / Math.pow(altura, 2);
      System.out.println("Seu IMC é " + imc);
   }
}
```

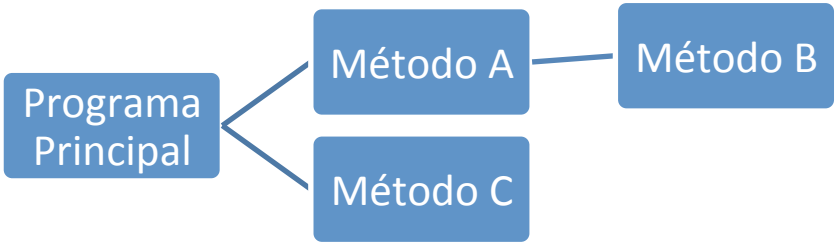
Declaração do método

Chamadas ao método

Dividir para conquistar

- Antes: um programa gigante

- Depois: vários programas menores



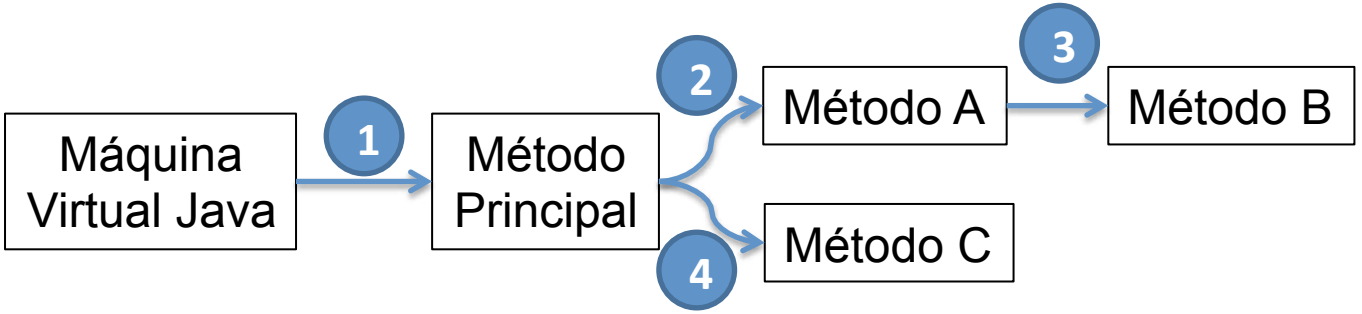
Fluxo de execução

- O programa tem início em um método principal (no caso do Java é o método *main*)
- O método principal chama outros métodos
- Estes métodos podem chamar outros métodos, sucessivamente
- Ao fim da execução de um método, o programa retorna para a instrução seguinte à da chamada ao método

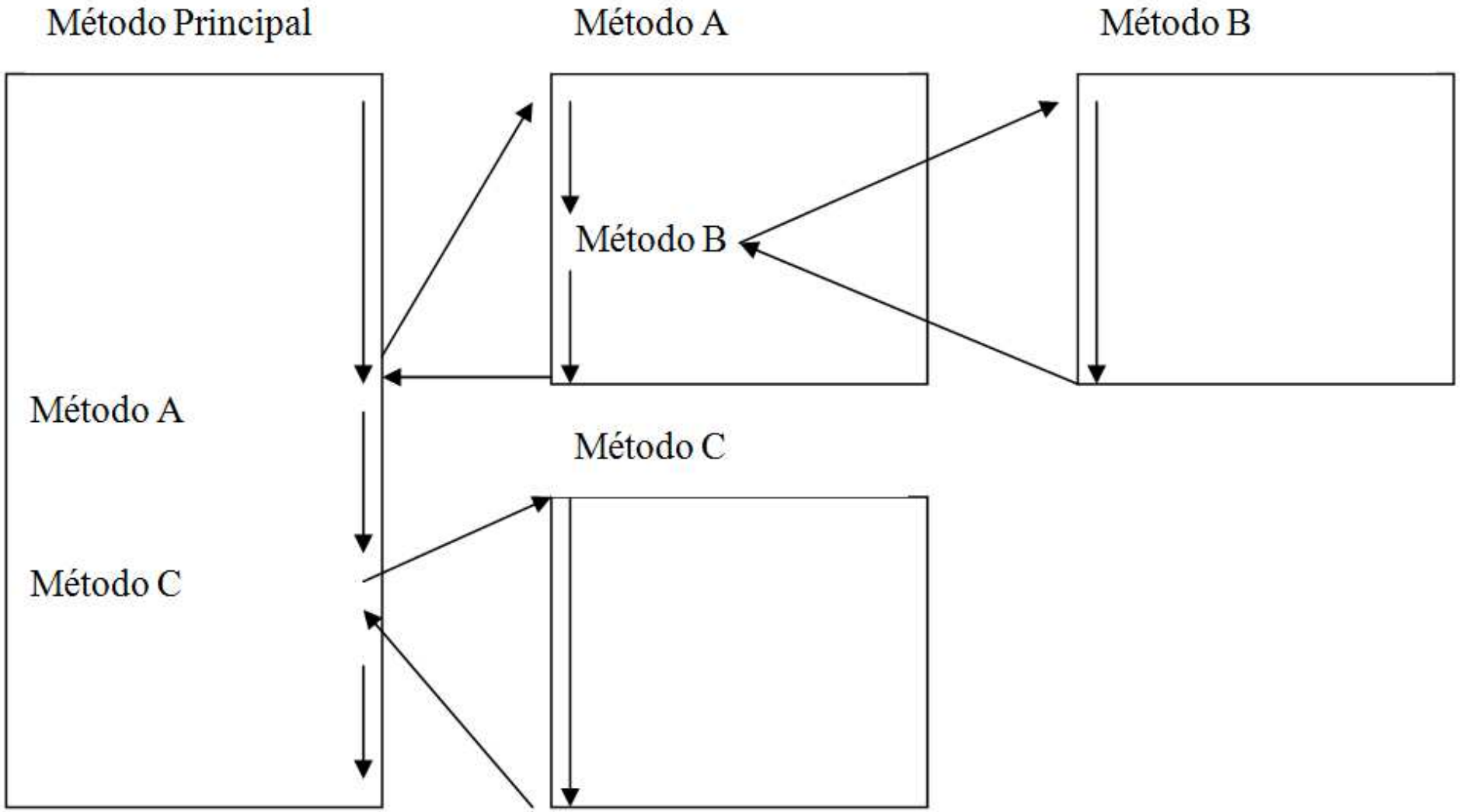
Programa

Método Principal
Método A
Método B
Método C

Possível sequencia de chamadas

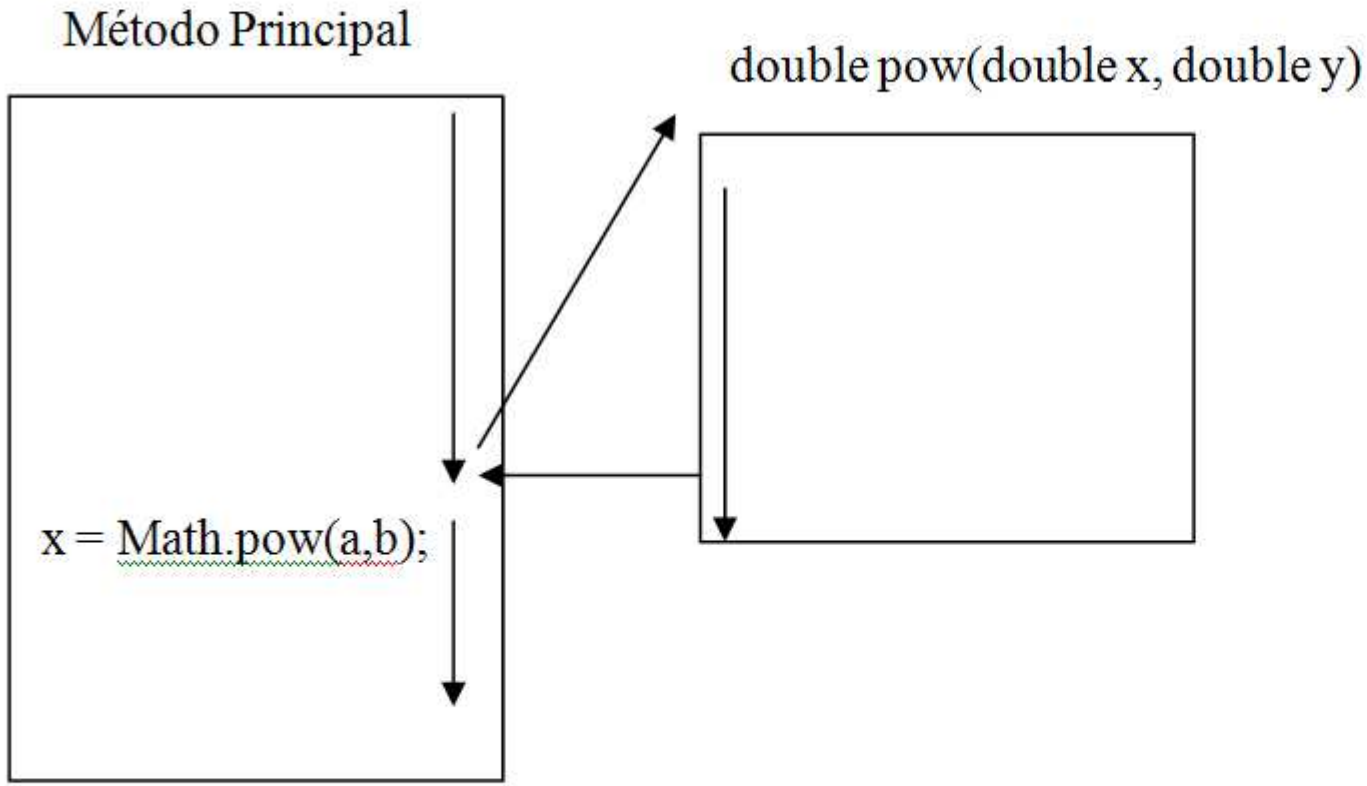


Fluxo de execução



Fluxo de execução

- É equivalente ao que acontece quando chamamos um método predefinido do Java



Vantagens

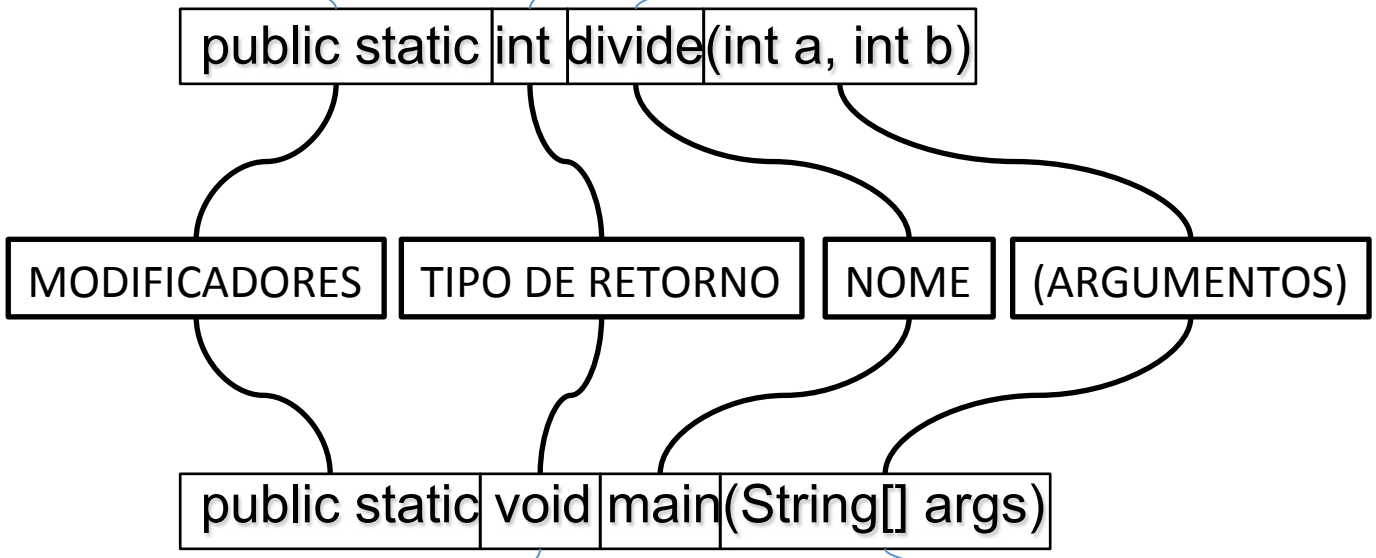
- Economia de código
  - Quanto mais repetição, mais economia
- Facilidade na correção de defeitos
  - Corrigir o defeito em um único local
- Legibilidade do código
  - Podemos dar nomes mais intuitivos a blocos de código
  - É como se criássemos nossos próprios comandos
- Melhor tratamento de complexidade
  - Estratégia de "dividir para conquistar" nos permite lidar melhor com a complexidade de programas grandes
  - Abordagem *top-down* ajuda a pensar!

Sintaxe de um método

Vamos usar esses modificadores por enquanto

Qualquer tipo da linguagem

Mesma regra de nome de variável

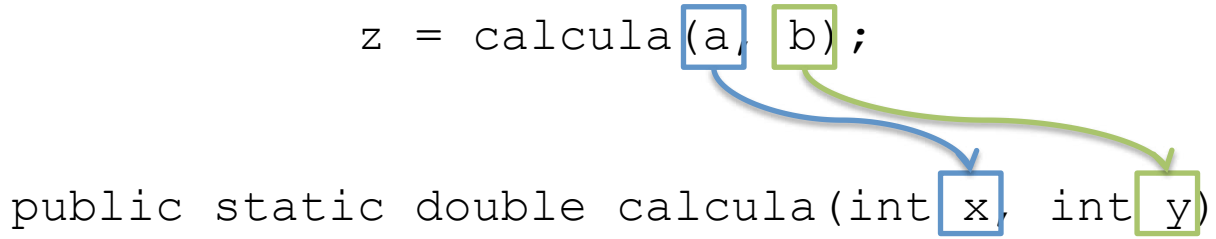


Significa que não tem retorno

Mesma regra de declaração de variáveis, separando por vírgula cada argumento

Acesso a variáveis

- Um método não consegue acessar as variáveis de outros métodos
  - Cada método pode criar as suas próprias variáveis locais
  - Os parâmetros para a execução de um método devem ser definidos como argumentos do método
- Passagem por valor
  - Java **copiará o valor** de cada argumento para a respectiva variável
  - Os nomes das variáveis podem ser diferentes



Exemplo

```
public class Troca {
  public static void troca(int x, int y) {
    int aux = x;
    x = y;
    y = aux;
  }
  public static float media(int x, int y) {
    return (x + y) / 2f;
  }
  public static void main(String[] args) {
    int a = 5;
    int b = 7;
    troca(a, b);
    System.out.println("a: " + a + ", b: " + b);
    System.out.println("média: " + media(a,b));
  }
}
```

Sobrecarga de métodos

- Uma classe pode ter **dois ou mais métodos com o mesmo nome**, desde que os tipos de seus argumentos sejam distintos
- Isso é útil quando queremos implementar um método em função de outro
- Exemplo baseado na classe String:

```
public int indexOf(String substring) {
  return indexOf(substring, 0);
}
```

Métodos sem argumentos

- Não é necessário ter argumentos nos métodos
  - Nestes casos, é obrigatório ter () depois do nome do método
  - A chamada ao método também precisa conter ()
- Exemplo de declaração:

```
public static void pulaLinha() {
  System.out.println();
}
```

- Exemplo de chamada:

```
pulaLinha();
```

Exercício

- Faça uma calculadora que forneça as seguintes opções para o usuário, usando métodos sempre que possível
- A calculadora deve operar sempre sobre o valor corrente na memória

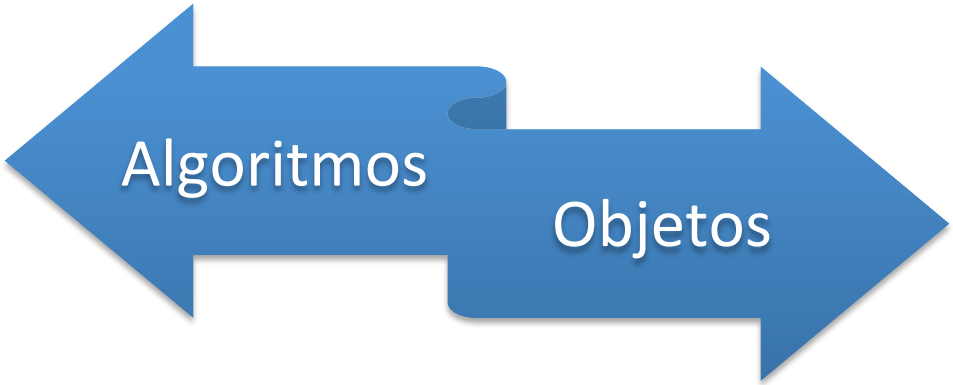
```
Estado da memória: 0
Opções:

(1) Somar
(2) Subtrair
(3) Multiplicar
(4) Dividir
(5) Limpar memória
(6) Sair do programa

Qual opção você deseja?
```

Parte II

ORIENTAÇÃO A OBJETOS

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

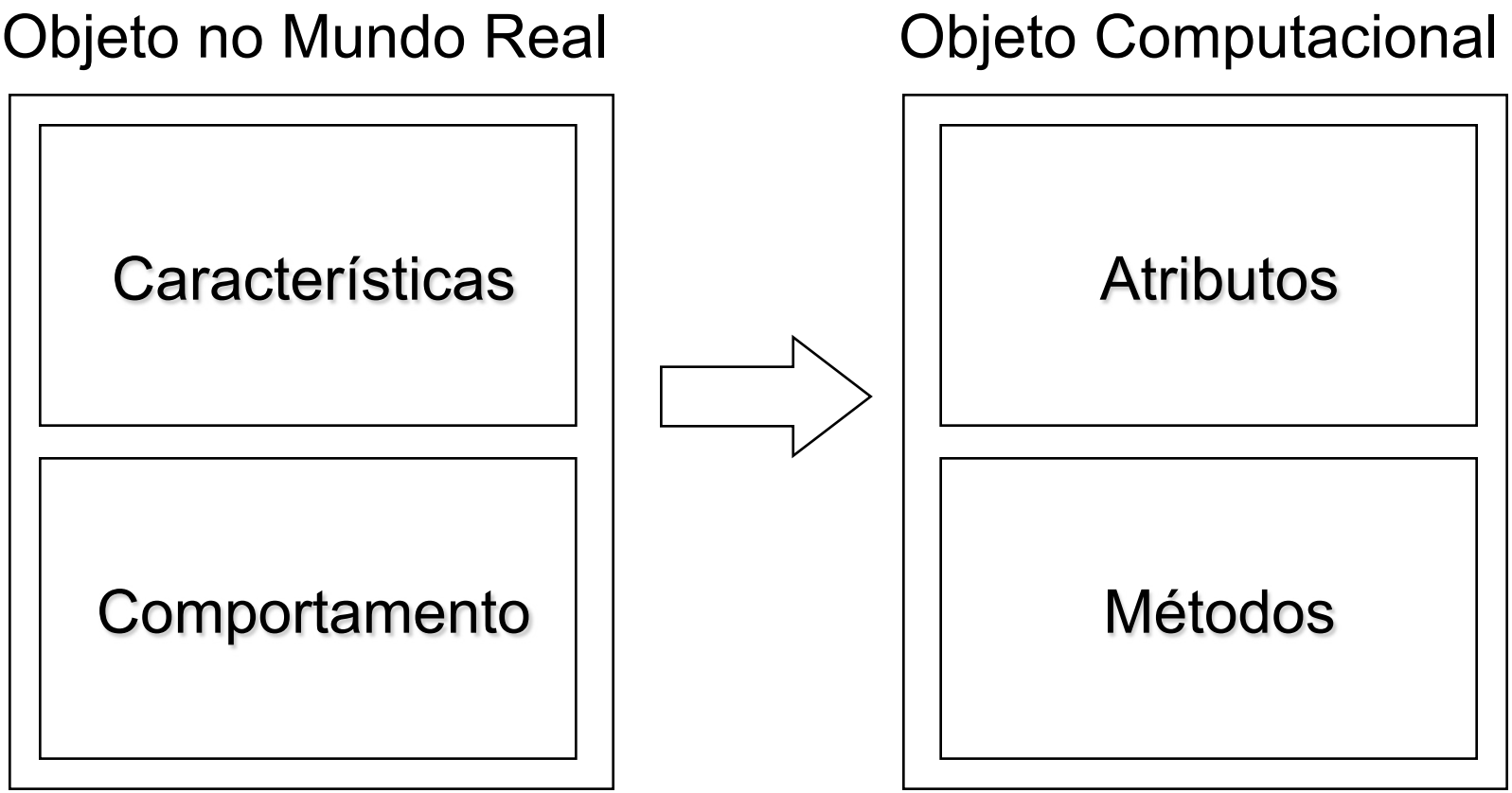
- Exemplo de características do objeto **carro**
  - Cor
  - Marca
  - Número de portas
  - Ano de fabricação
  - Tipo de combustível

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

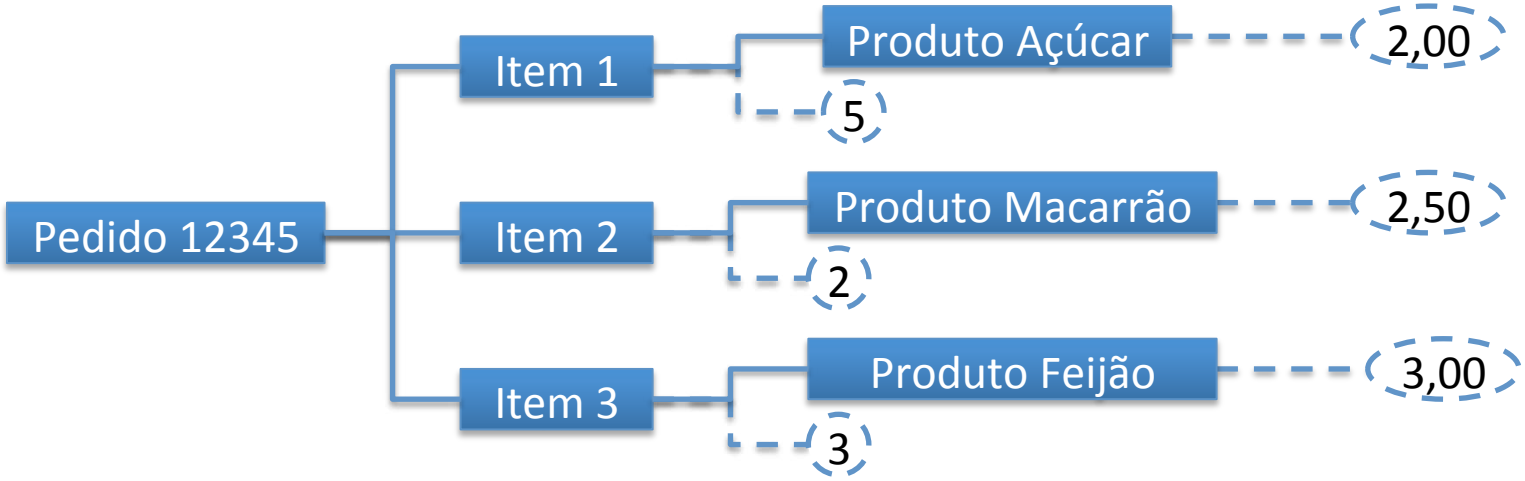
Item	Produto	Preço	Quantidade	Subtotal
1	Açúcar	R$ 2,00	5	R$ 10,00
2	Macarrão	R$ 2,50	2	R$ 5,00
3	Feijão	R$ 3,00	3	R$ 9,00
			TOTAL	R$ 24,00

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



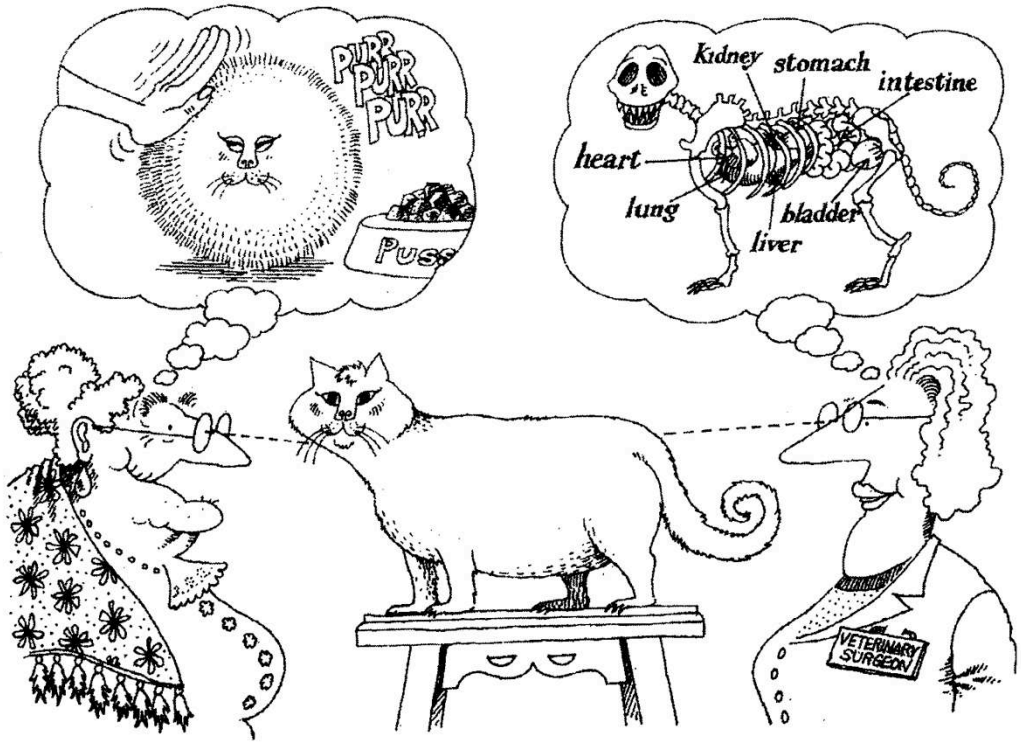
Paradigma OO
(Exemplo: total da compra)

- 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**

Princípios do Paradigma OO

Abstração

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



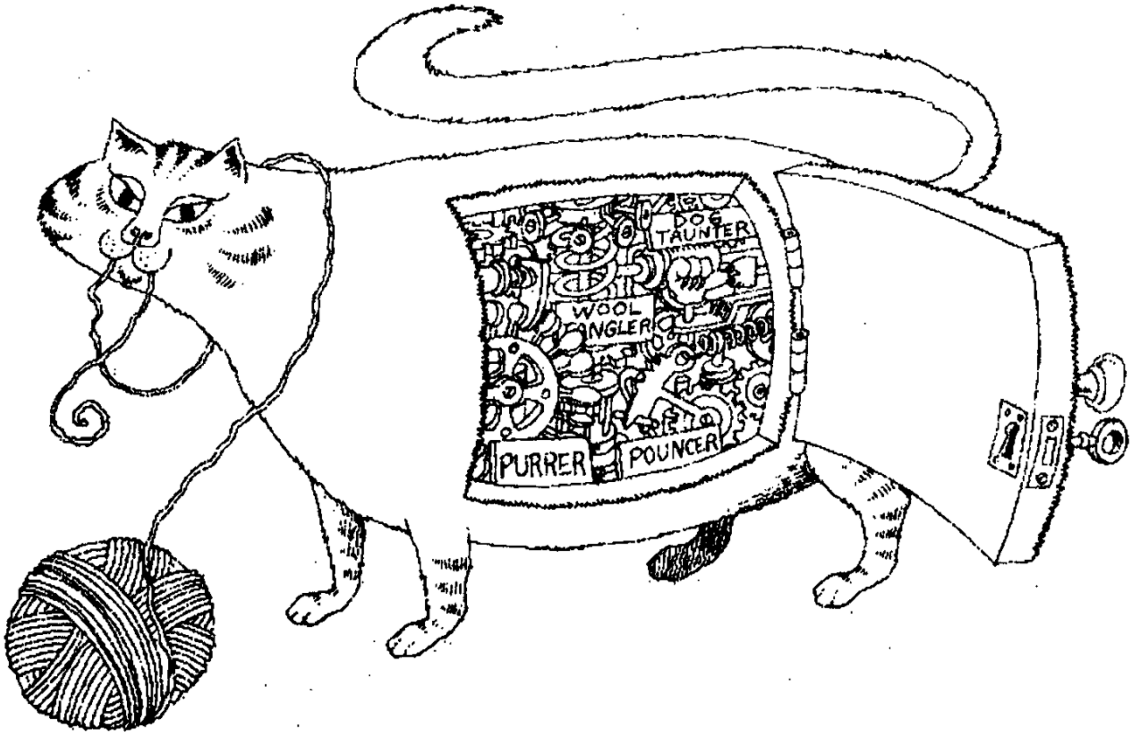
Fonte: livro “Object-Oriented Analysis and Design with Applications”

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

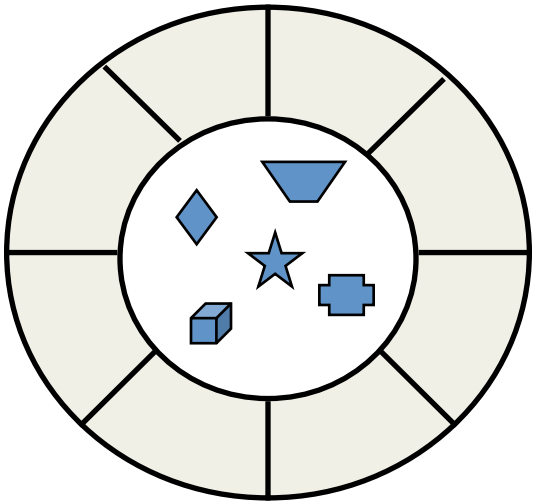
- O objeto deve esconder seus dados e os detalhes de sua implementação



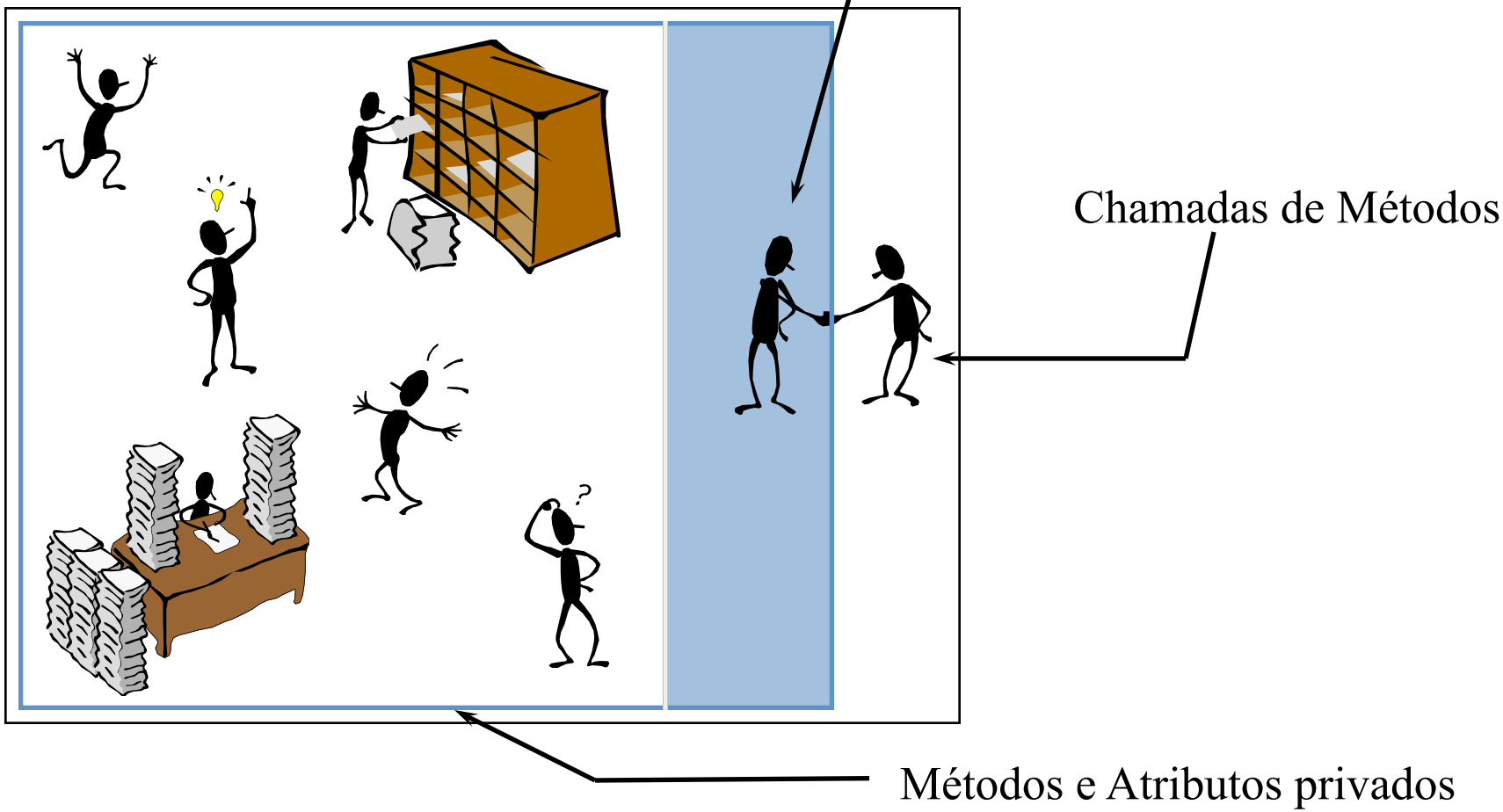
Fonte: livro “Object-Oriented Analysis and Design with Applications”

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



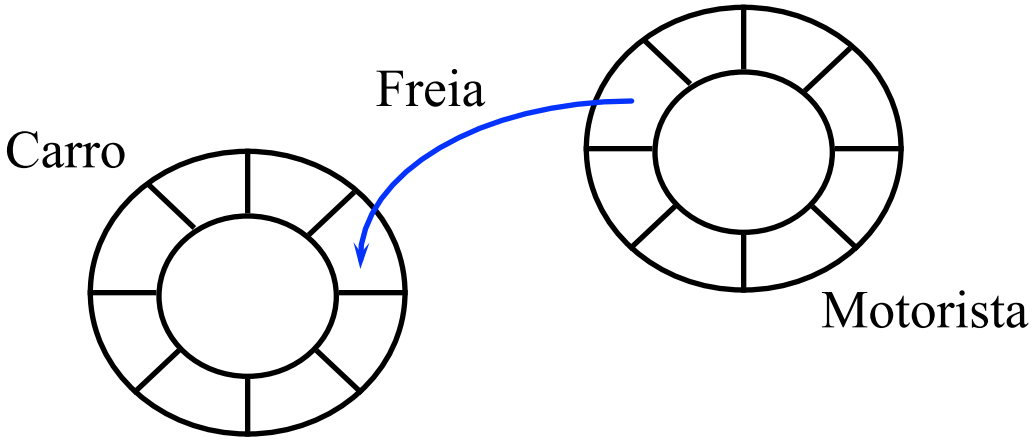
Modularidade

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

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

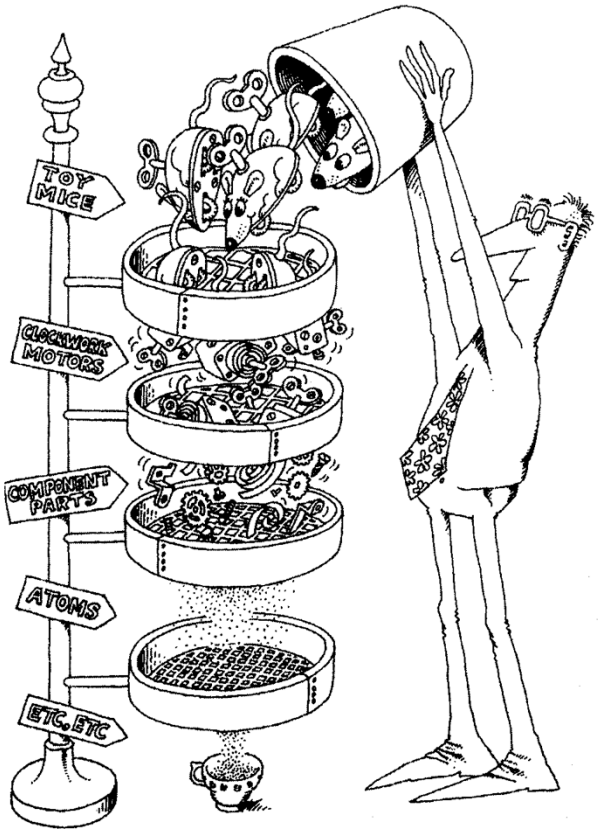
Modularidade

- 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



Hierarquia

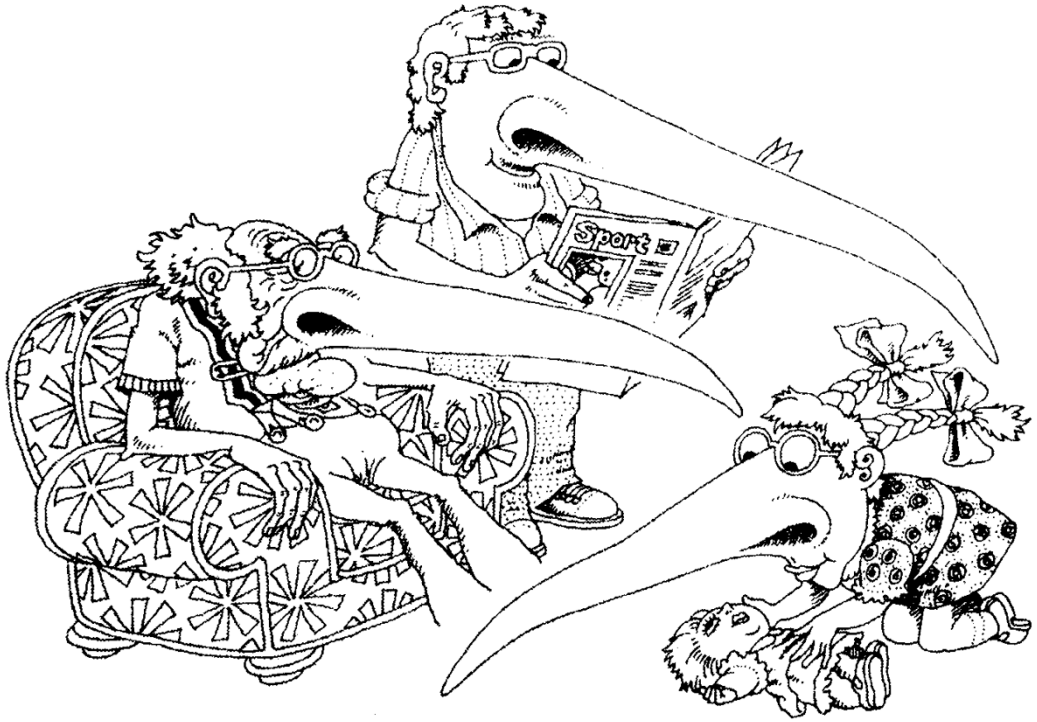
- Os objetos devem ser organizados no sistema de forma hierárquica



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

Hierarquia

- Objetos herdam atributos e métodos dos seus ancestrais na hierarquia



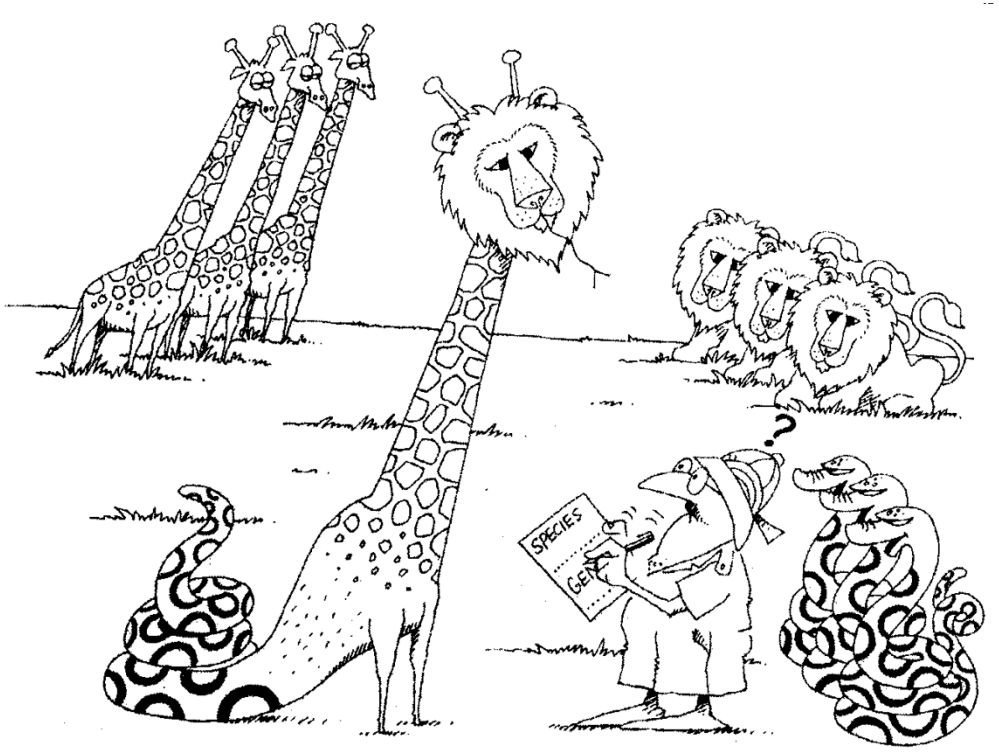
Fonte: livro “Object-Oriented Analysis and Design with Applications”

Parte III

CLASSES E INTERFACES

Classes versus Objetos

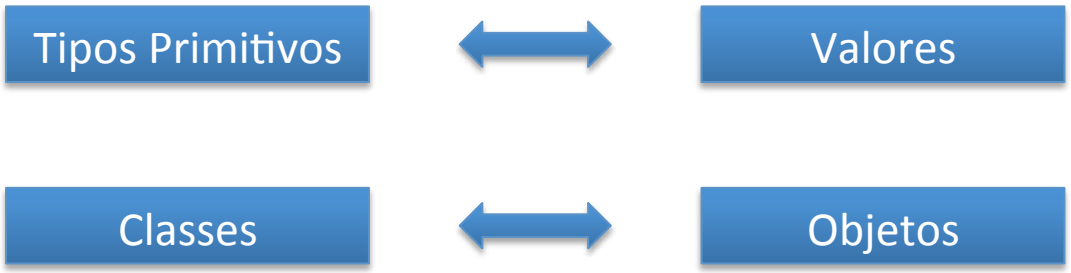
- A **Classe** é o **tipo do Objeto**



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

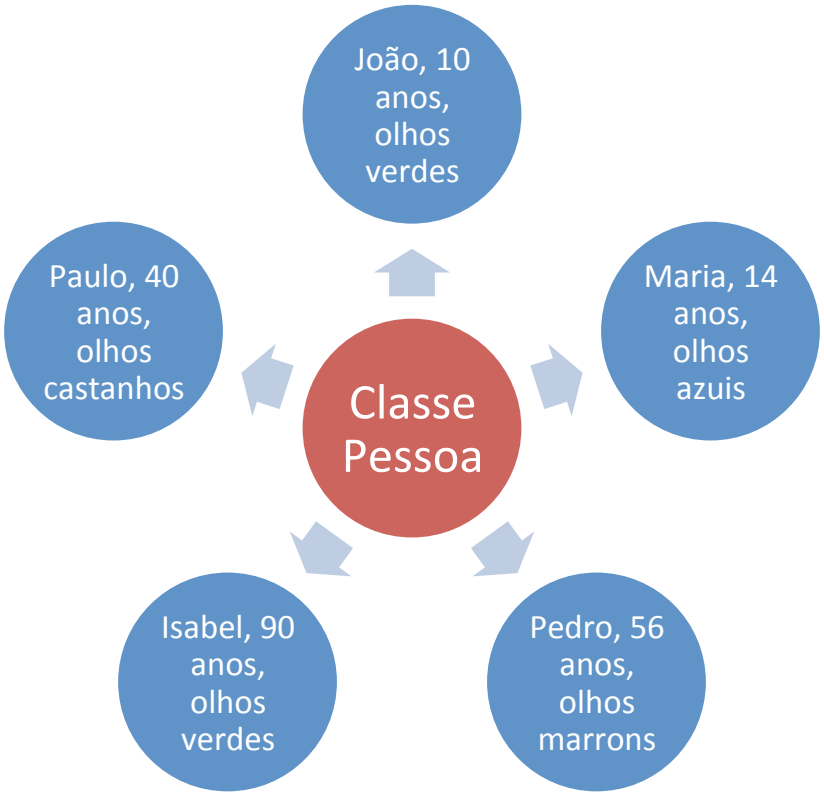
Classes versus Objetos

- 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
  - Em Java, **cada objeto pertence a uma única classe**
  - 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)

Criação de objetos

- A classe é responsável pela criação de seus objetos via método construtor
  - Mesmo nome da classe
  - Sem tipo de retorno

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

Criação de objetos

- Objetos devem ser instanciados antes de utilizados
  - O comando ***new*** instancia um objeto, chama o seu construtor
- Exemplo:

```
Carro fusca = new Carro(10);
Carro bmw = new Carro(15);
fusca.freia();
bmw.acelera();
fusca = bmw;
```

Qual a velocidade de cada carro em cada momento?

O que acontece aqui?

Criação de objetos

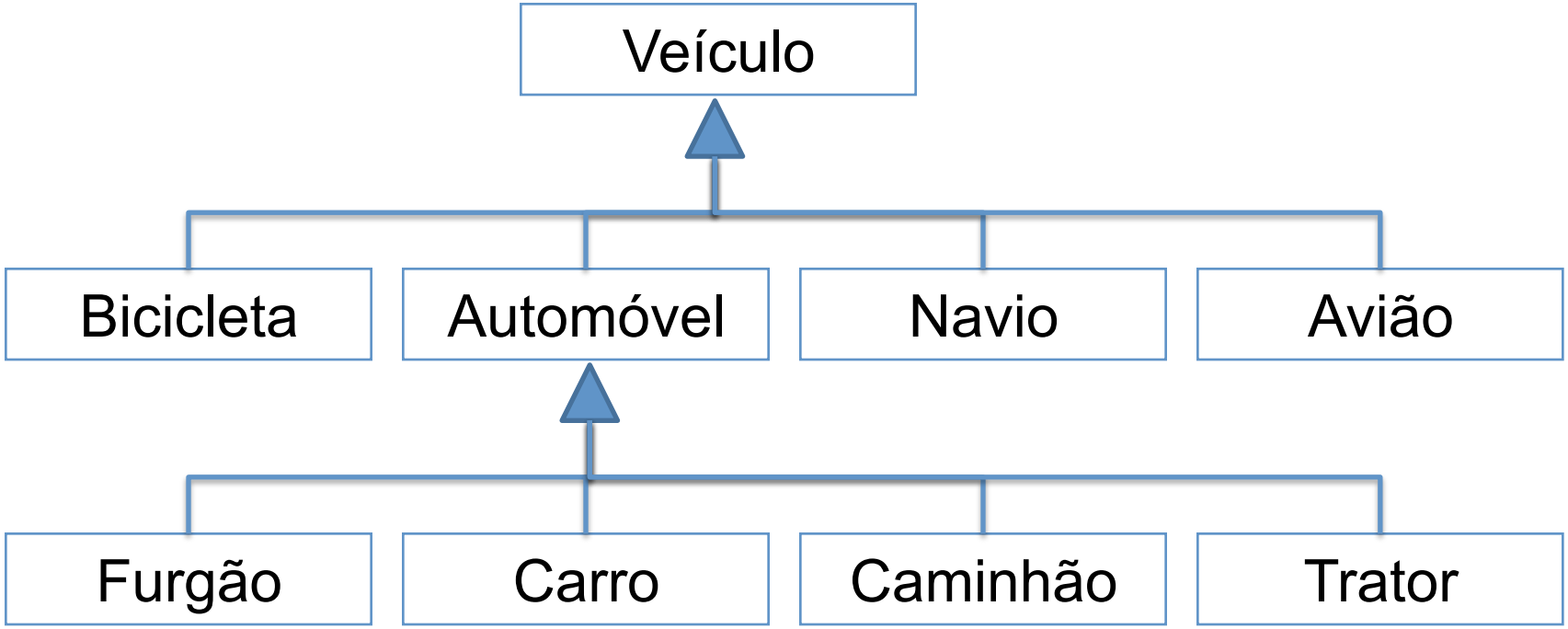
- Valor *null*:
  - Utilizado para representar um objeto não inicializado
  - Quando um método retorna um objeto, ele pode retornar *null* para indicar, por exemplo, que o objeto não foi encontrado
  - É possível atribuir *null* para descartar um objeto previamente instanciado
- Exemplo:

```
Carro fusca = new Carro(10);
fusca.acelera();
fusca = null;
```

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

Exemplo de herança



Exemplo de herança

- Declaração:

```
public class CarroInteligente extends Carro {
  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?

Exercício

- Identifique as classes e implemente um programa 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."

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

Interfaces

- A definição de uma interface é similar a de uma classe
  - Utilizamos a palavra reservada *interface*
  - A palavra reservada deve ser seguida do nome da interface
  - Uma interface pode herdar de outras interfaces (*extends*)
  - A interface possui apenas métodos abstratos e constantes

```
public interface Taxavel
{
        int ANO_INICIO = 1996;
        double calculaTaxa ();
}
```

Classes Abstratas

- Se uma classe possui algum método sem implementação (abstratos), o modificador ***abstract*** deve preceder sua declaração

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

}
```

Exemplo

```
public class Ferrari extends Carro implements Taxavel
{
        <atributos da Ferrari>
        <métodos da Ferrari>
        <métodos redefinidos de Carro>
        <métodos da interface Taxavel>
}
```

Parte III

PACOTES

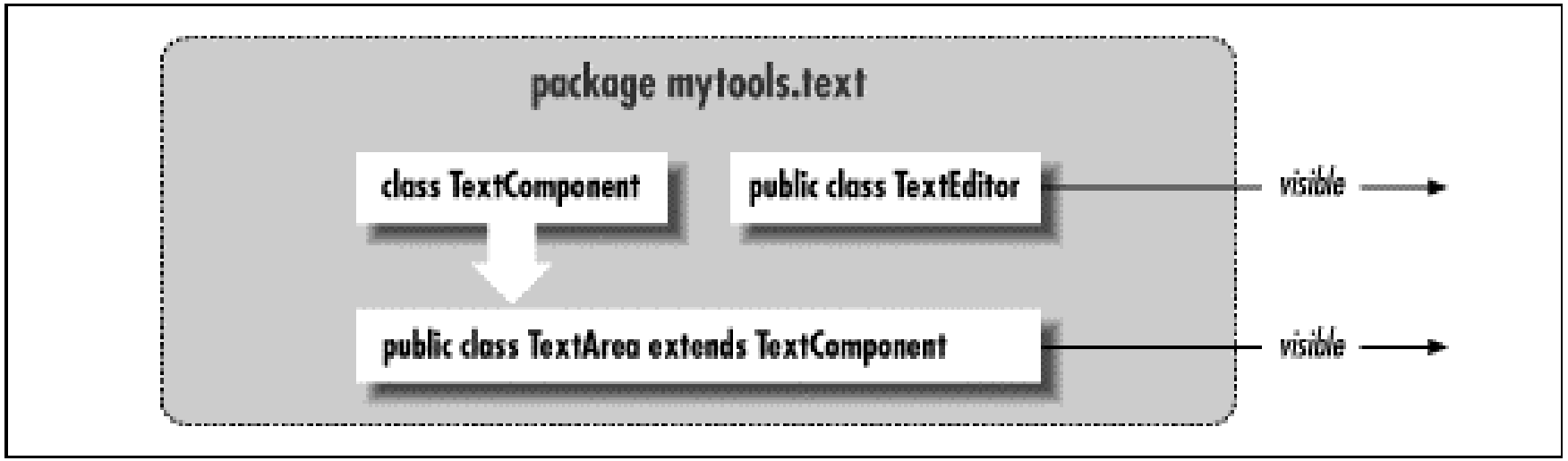
Pacotes

- Utilizados para agregar classes relacionadas
- O pacote de uma classe é indicado na primeira linha da classe
  - Declaração *package*
- Se uma classe não declara seu pacote, o interpretador assume que a classe pertence a um pacote *default*

```
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
- A importação se realiza através da palavra-chave *import*, seguida do nome da classe desejada
- As importações são apresentadas antes da declaração da classe mas depois da declaração do pacote

```
package br.uff.ic.prog1;

import java.util.Scanner;

public class Fisica {
   ...
}
```

Regra de ouro para classes e pacotes

- Classes devem ser mapeadas em arquivos com o mesmo nome
  - Classe **Pessoa**
  - Arquivo **Pessoa.java**
- Pacotes devem ser mapeados em diretórios
  - Pacote **br.uff.ic**
  - Diretório **br\uff\ic**
- Se o nome completo da classe é **br.uff.ic.Pessoa**
  - Deve haver **br\uff\ic\Pessoa.java**

Retornando aos métodos

- Modificadores
  - Estamos até agora usando somente *public static*
  - O que significam esses modificadores?
  - Quais outros modificadores existem?
- Passagem de parâmetros
  - O que acontece quando passamos objetos nos argumentos de um método?

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

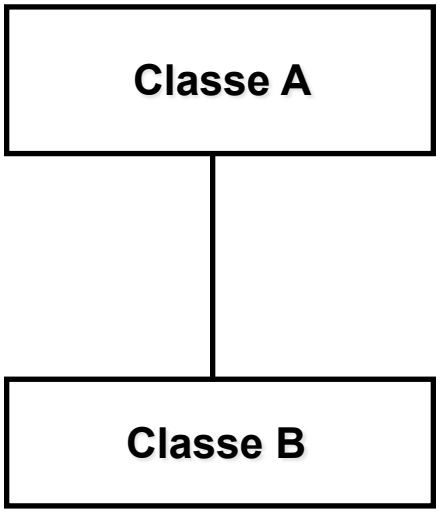
Passagem por valor vs. passagem por referência

- Algumas linguagens permitem passagem de argumentos por referência
  - Não é o caso de Java, que sempre faz passagem por valor
- Diferenças
  - Passagem por valor = cópia dos valores para outra posição de memória
  - Passagem por referência = reuso da posição de memória
- Quando é passado um objeto por valor...
  - Mudanças nos atributos dos objetos são vistas de fora
  - Instanciações de novos objetos nas variáveis não são vistas de fora

Outras Classes

- Relações 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 Atributos



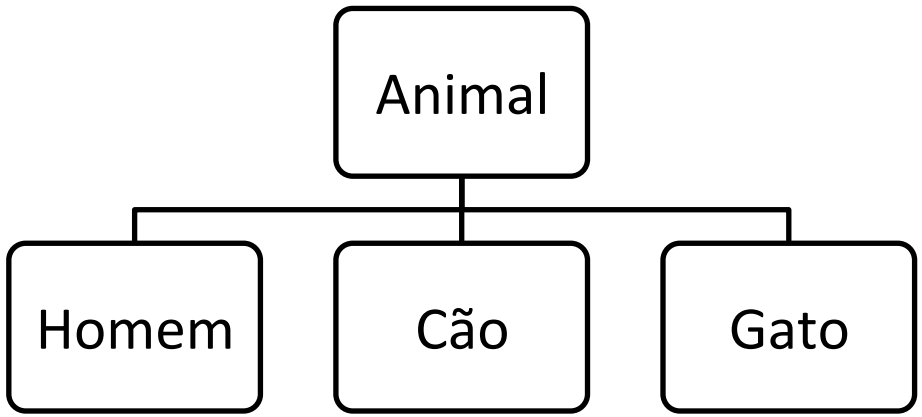

```
class A
{
        private B       b;
        …
}

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

Polimorfismo

- Uma subclasse pode redefinir 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, o Java saberá qual implementação deve ser usada

Exercício

- Faça uma classe Animal com um método abstrato “fala”
- Faça as classes Homem, Cão e Gato, herdando de animal, redefinindo o método “fala” para retornar “Oi”, “Au au” e “Miau”, respectivamente
- Crie um vetor de 10 Animais e instancie Homens, Cães e Gatos nesse vetor
- Faça um loop por todos os animais do vetor, pedindo para eles falarem



Instituto de computação uff

Subprogramação e Orientação a Objetos

Leonardo Gresta Paulino Murta

leomurta@ic.uff.br