

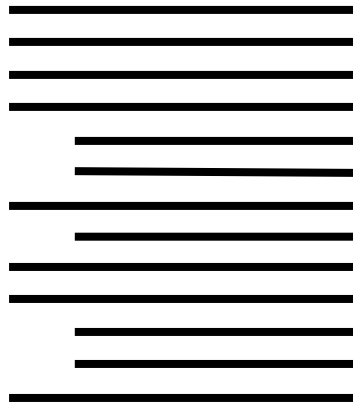
# Orientação a Objetos

Vanessa Braganholo  
vanessa@ic.uff.br

# Paradigma estruturado

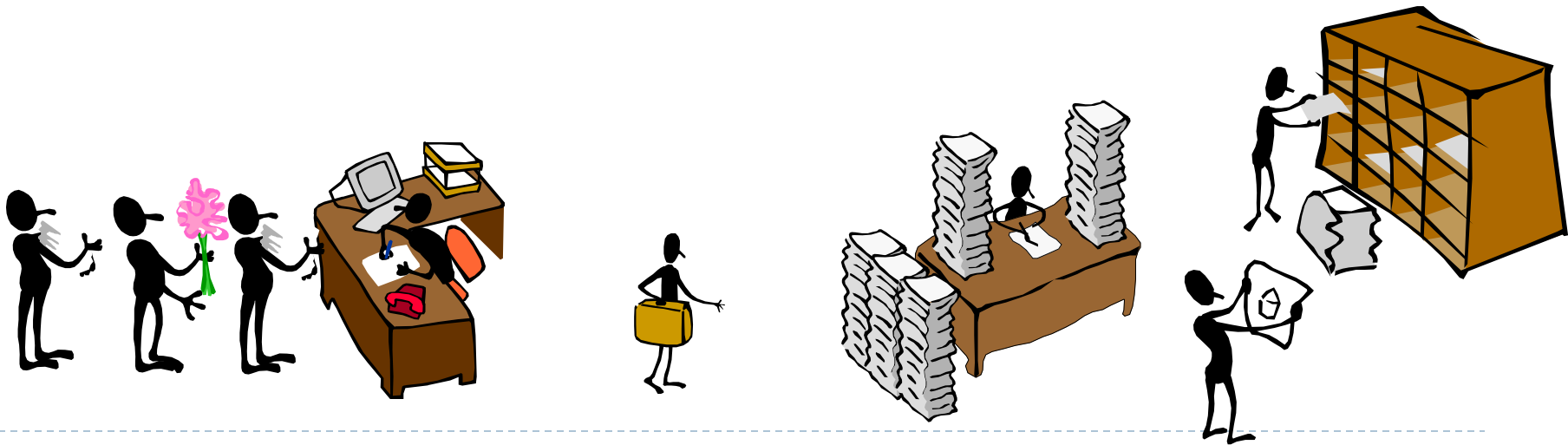
---

- ▶ 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?



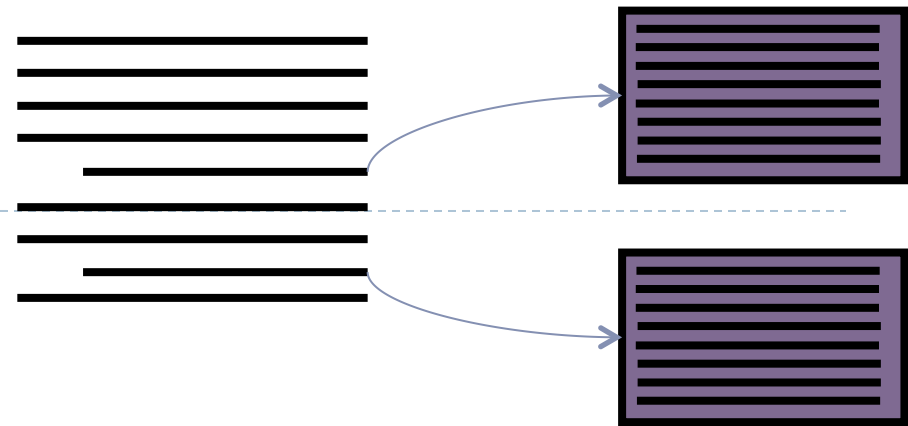
# Encapsulamento

- ▶ 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
- ▶ Em Java, essas ações complexas são denominadas métodos
  - ▶ Outras linguagens usam termos como procedimento, sub-rotina e função

# 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);
    }
}
```

Parecidas!



# 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();  
    }
```

Declaração  
do método

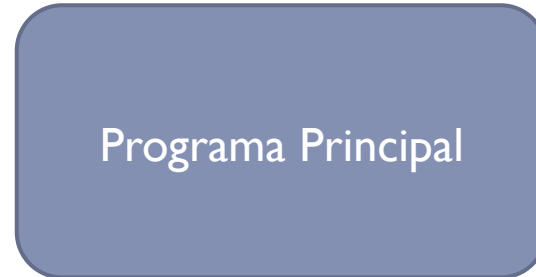
```
    public static void main(String[] args) {  
        double altura = leia("Entre com a sua altura em metros: ");  
        double massa = leia("Entre com a sua massa em kg: ");  
  
        double imc = massa / Math.pow(altura, 2);  
        System.out.println("Seu IMC é " + imc);  
    }  
}
```

Chamadas  
ao método

# Dividir para conquistar

---

- ▶ Antes: um programa gigante



- ▶ Depois: vários programas menores

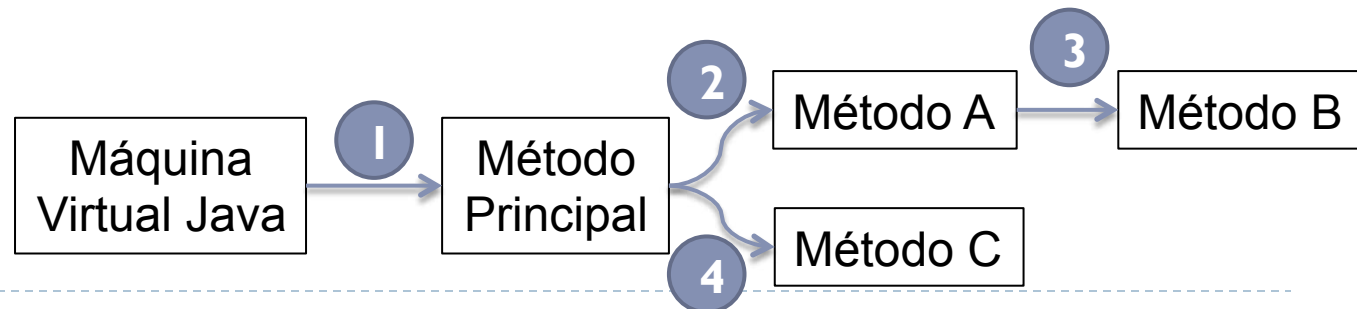
# Fluxo de execução

- ▶ 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

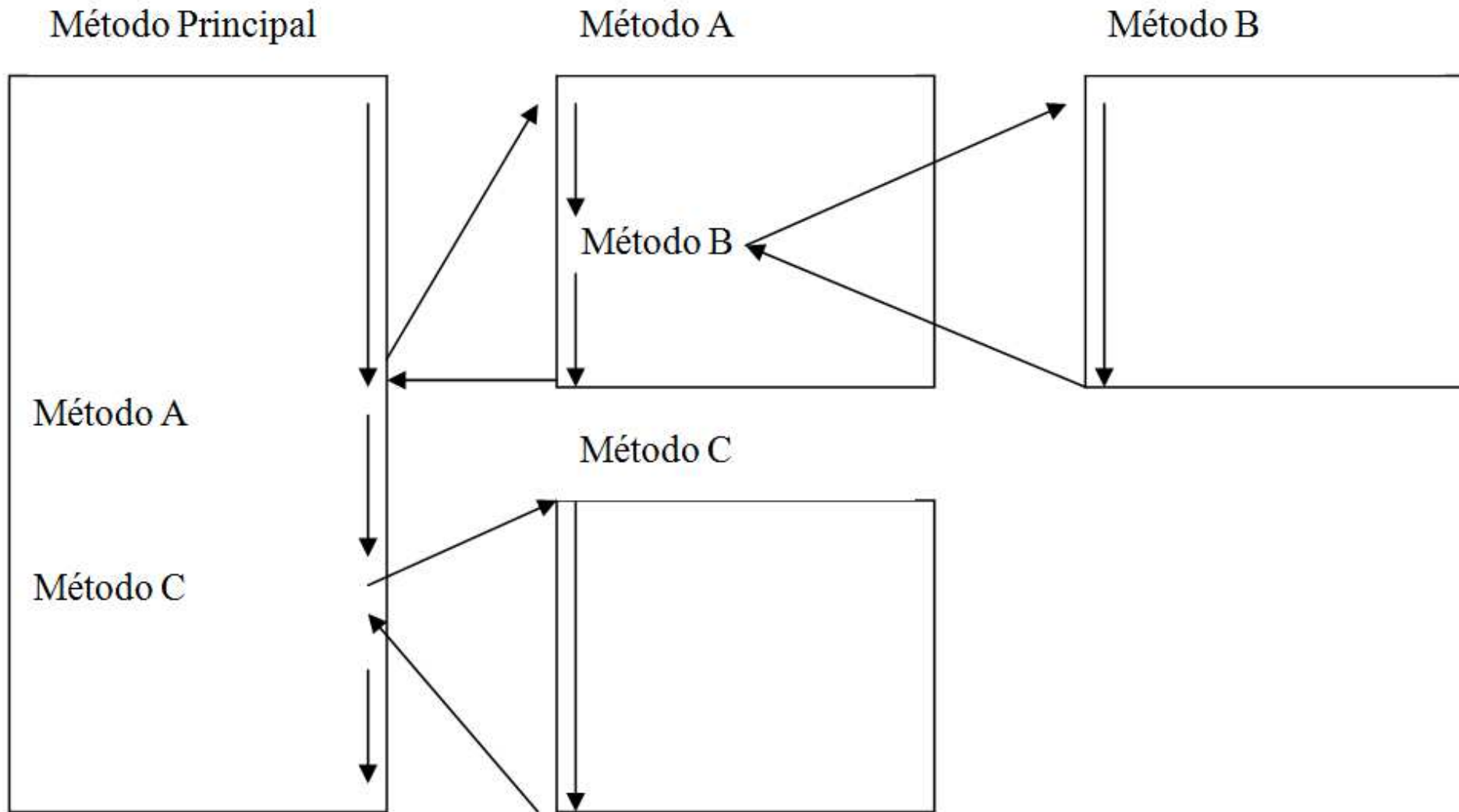
## Possível sequencia de chamadas





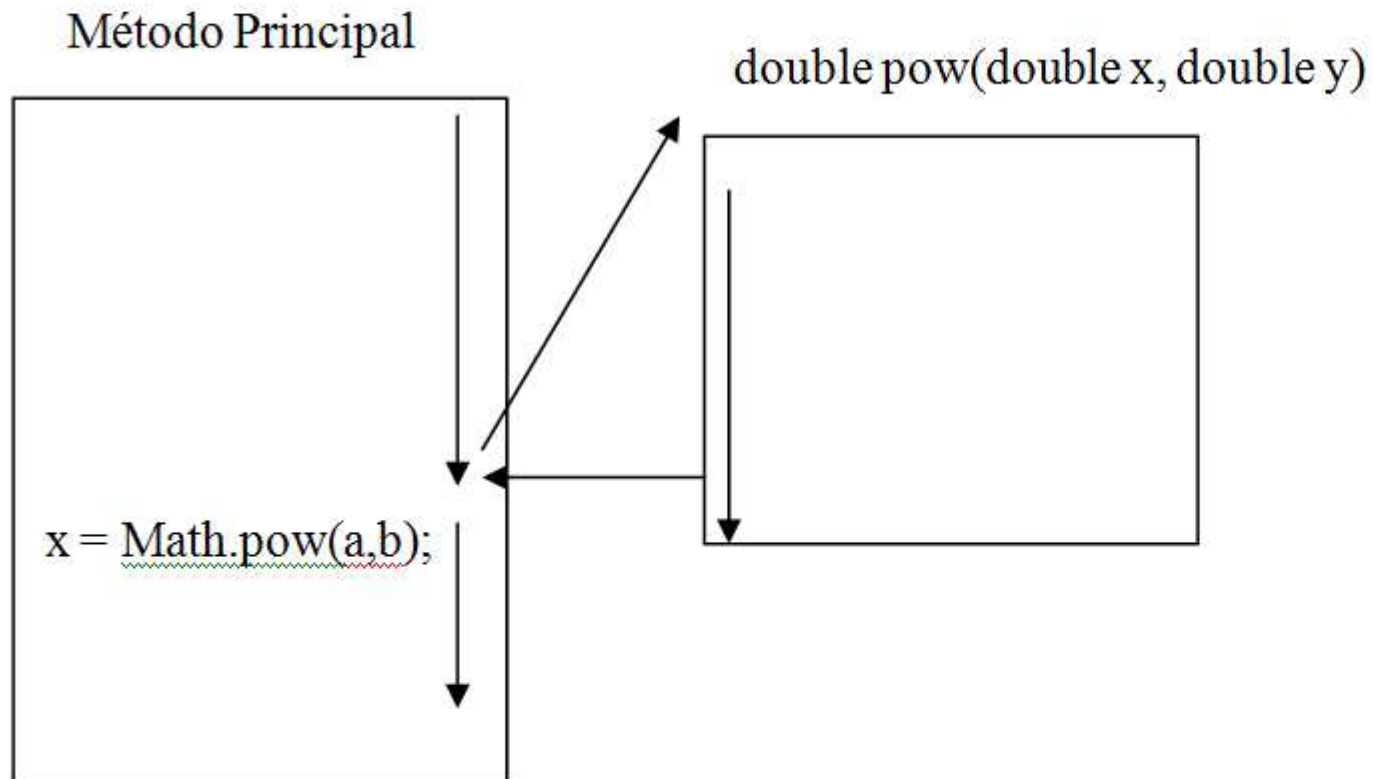
# Fluxo de execução

---



# Fluxo de execução

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



# Vantagens

---

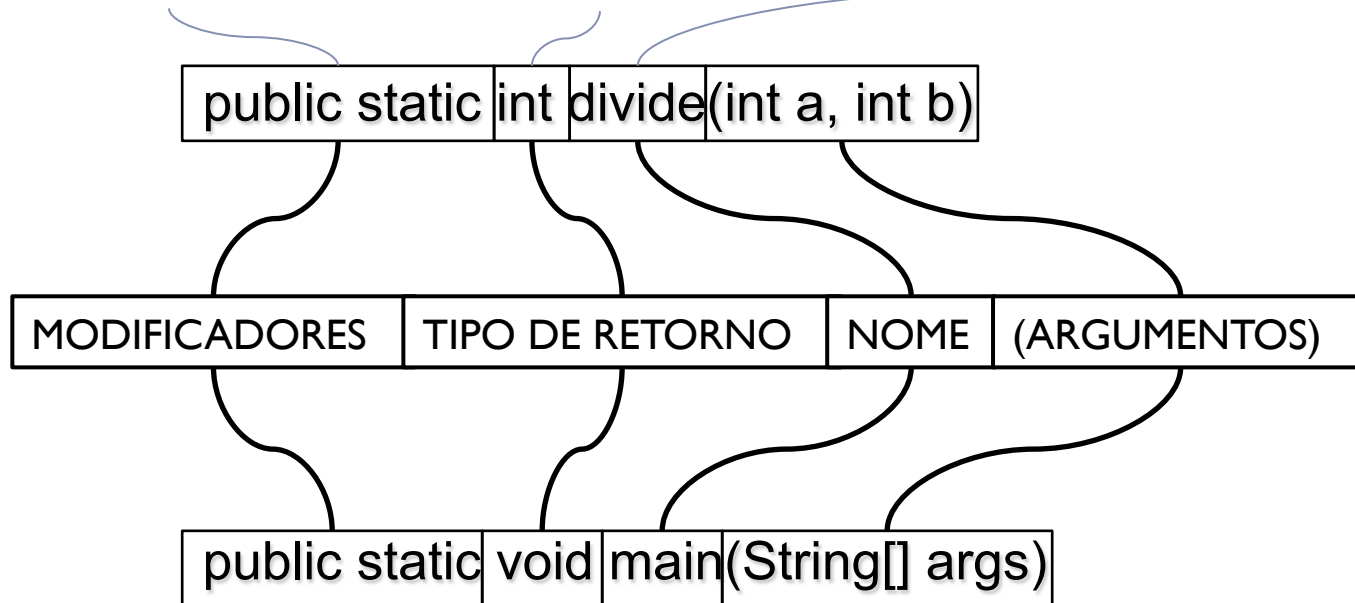
- ▶ **Economia de código**
  - ▶ Quanto mais repetição, mais economia
- ▶ **Facilidade na correção de defeitos**
  - ▶ Corrigir o defeito em um único local
- ▶ **Legibilidade do código**
  - ▶ Podemos dar nomes mais intuitivos a blocos de código
  - ▶ É como se criássemos nossos próprios comandos
- ▶ **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

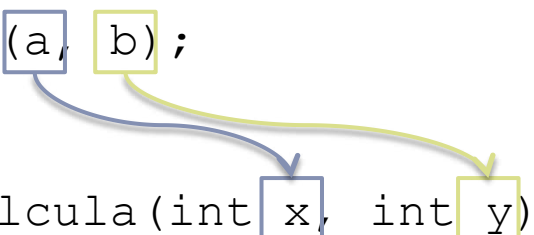
Mesma regra de declaração de variáveis, separando cada argumento por vírgula

# 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

```
z = calcula(a, b);
```



```
public static double calcula(int x, int y)
```

# 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));
    }
}
```

# Sobrescrita 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();  
}
```

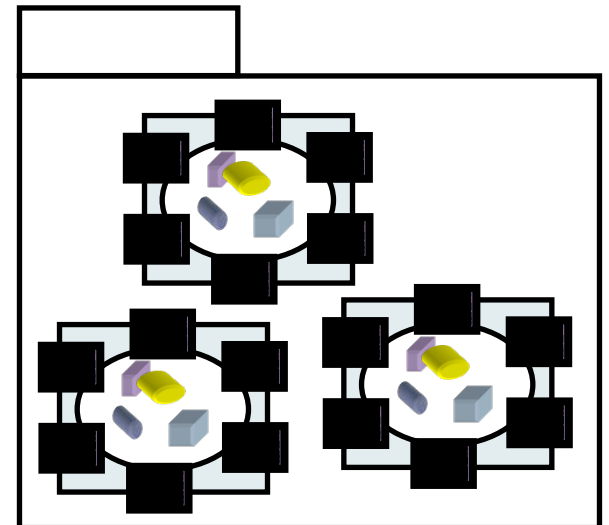
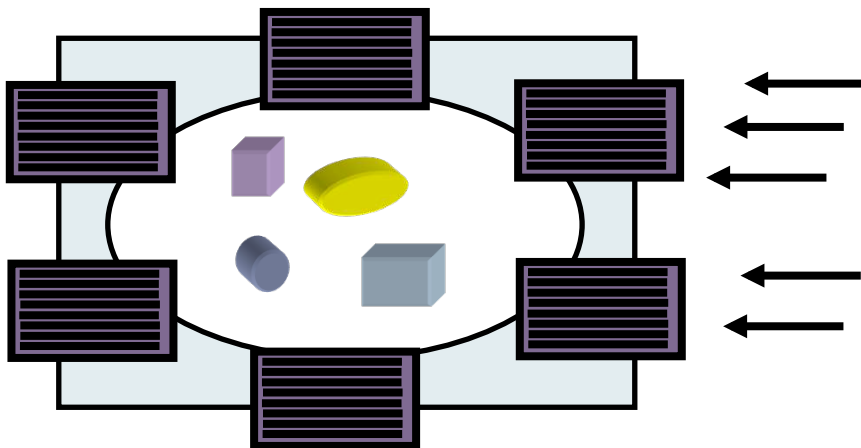
- ▶ Exemplo de chamada:

```
pulaLinha();
```



# Paradigma orientado a objetos (OO)

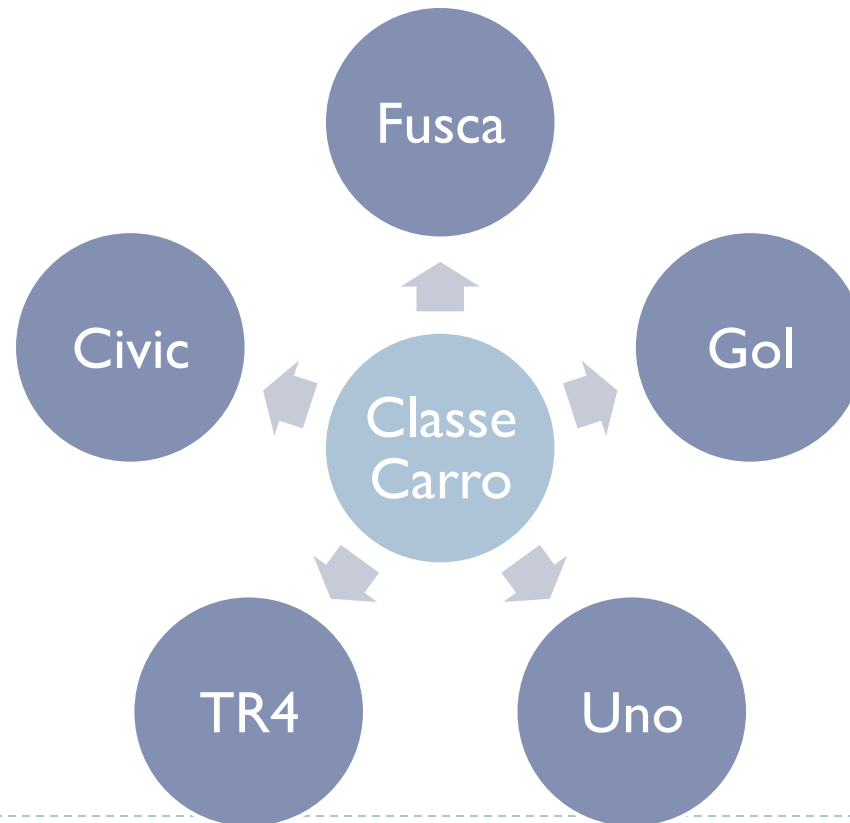
- ▶ Classes de objetos
  - ▶ Agrupamento de métodos afins
- ▶ Pacotes de classes
  - ▶ Agrupamento de classes afins
  - ▶ Representam bibliotecas de apoio



# Classes x Objetos

---

- ▶ Uma classe é como se fosse uma fôrma, capaz de produzir (instanciar) objetos com características distintas



# 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

<i>cadeira</i>		<i>mesa</i>		<i>caneta</i>	<i>lápis</i>
<i>carro</i>	<i>piloto</i>		<i>venda</i>		<i>mercadoria</i>
<i>cliente</i>		<i>aula</i>		<i>programa</i>	<i>computador</i>
<i>aluno</i>	<i>avião</i>				

# 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 do 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** do objeto

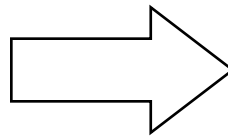
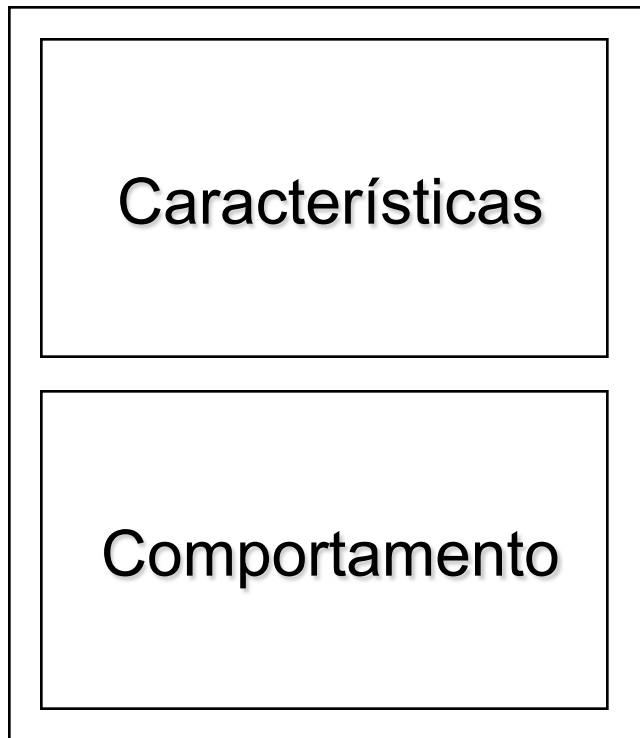
## ▶ Exemplos de comportamento para o objeto **carro**

- ▶ Acelerar
- ▶ Frear
- ▶ Virar para direita
- ▶ Virar para esquerda

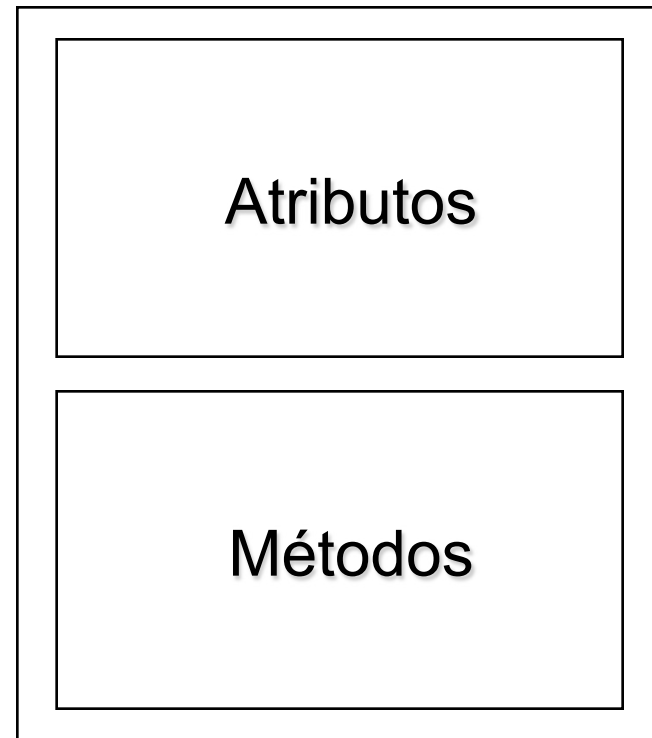
# Mapeamento de Objetos

---

## Objeto no Mundo Real



## Objeto Computacional

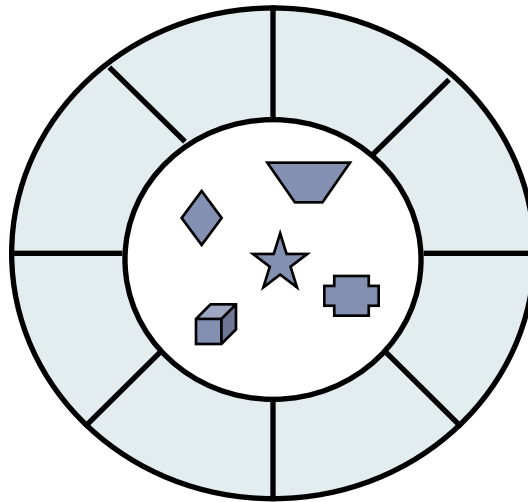


# Encapsulamento

---

## ▶ 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

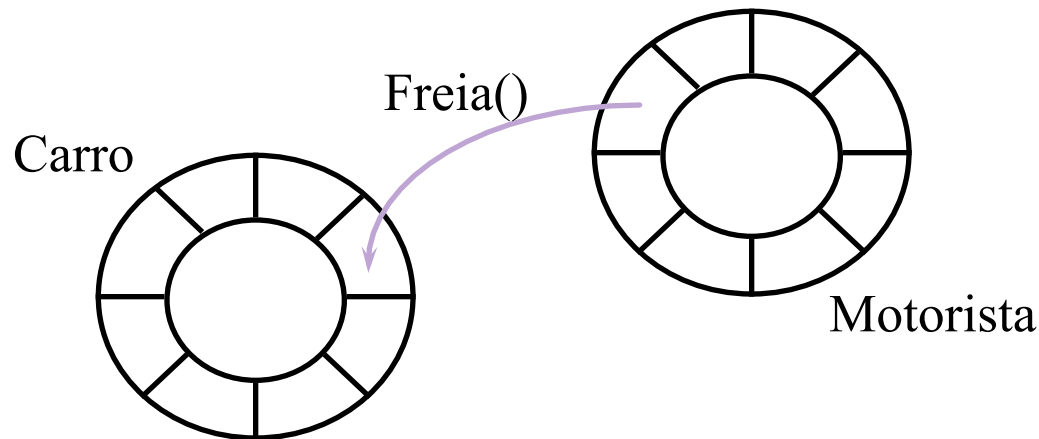


# Chamada de métodos

---

## ► Colaboração

- 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





# 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)

# Classe & Objetos

---

## Carro

Velocidade

Cor

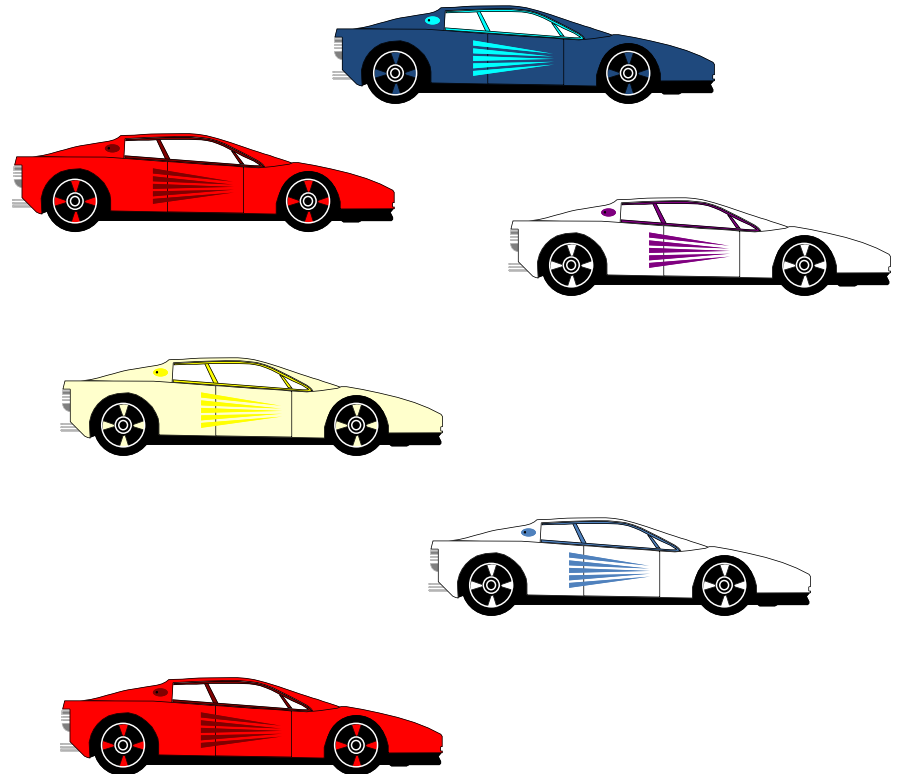
Cor Lateral

Freia

Acelera

Vira para direita

Vira para esquerda



# 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;
```

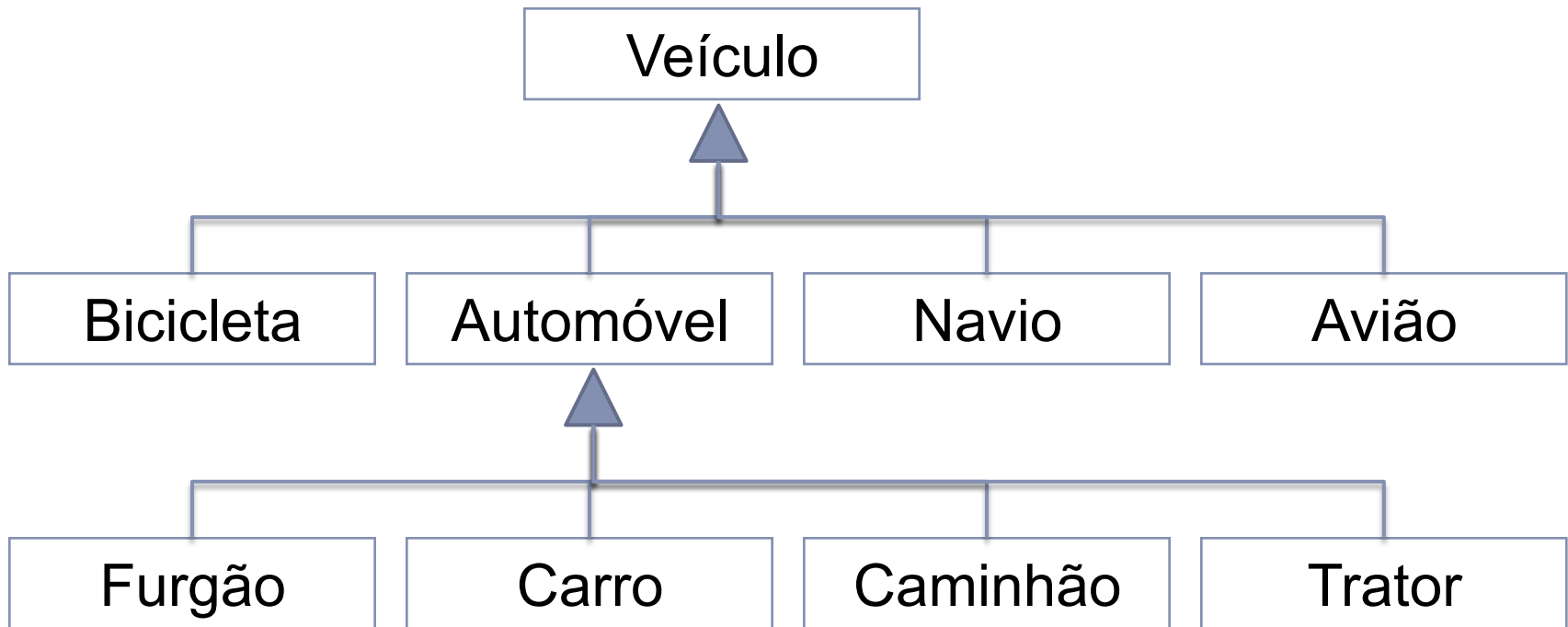
# Herança

---

- ▶ Classes são organizadas em estruturas hierárquicas
  - ▶ Uma classe pode **herdar características e comportamento de outras classes**
  - ▶ A classe que forneceu os elementos herdados é chamada de **superclasse**
  - ▶ A classe herdeira é chamada de **subclasse**
  - ▶ A subclasse **herda 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 tigran = new CarroInteligente(10);  
for (int i = 10; i > 0; i--) {  
    tigran.freia();  
}  
tigran.estaciona();
```

# 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*

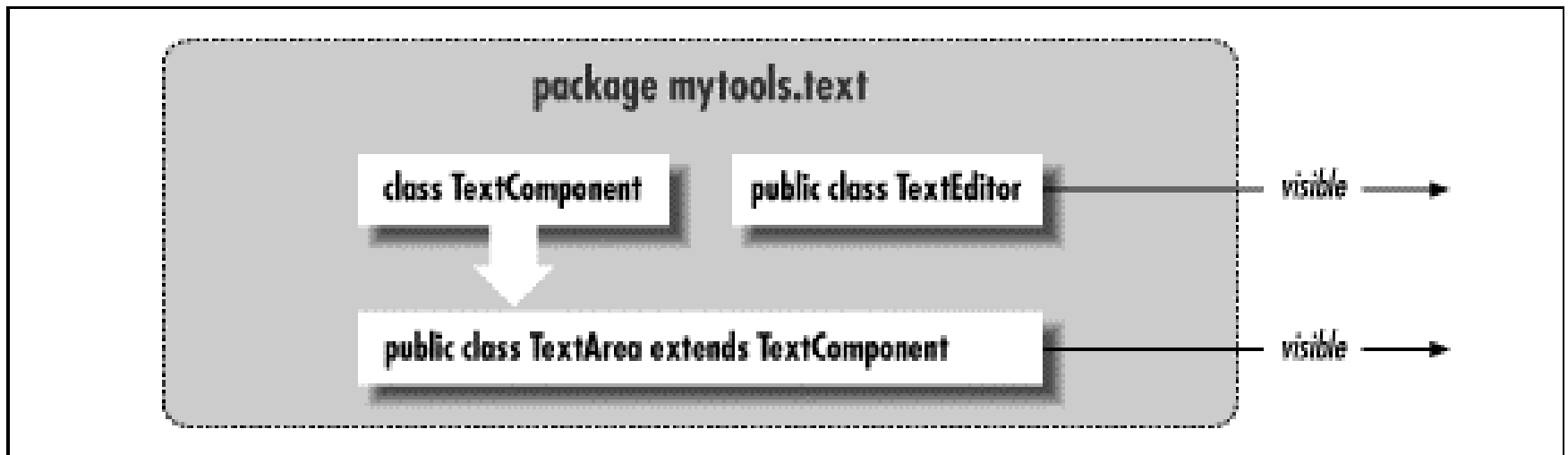
```
package br.uff.ic.prog1;
```

```
public class Fisica {  
    ...  
}
```

# 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 **Fisica**
  - ▶ Arquivo **Fisica.java**
- ▶ Pacotes devem ser mapeados em diretórios
  - ▶ Pacote **br.uff.ic.progI**
  - ▶ Diretório **br\uff\ic\progI**
- ▶ Se o nome completo da classe é **br.uff.ic.progI.Fisica**
  - ▶ Deve haver **br\uff\ic\progI\Fisica.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 (atributo ou classe):
  - ▶ 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
  - ▶ 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 (Pascal, por exemplo)
  - ▶ 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 mesma 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

# Exercício

---

- ▶ Faça um programa que, dado uma figura geométrica que pode ser uma circunferência, triângulo ou retângulo, calcule a área e o perímetro da figura
- ▶ O programa deve primeiro perguntar qual o tipo da figura:
  - ▶ (1) circunferência
  - ▶ (2) triângulo
  - ▶ (3) retângulo
- ▶ Dependendo do tipo de figura, ler o (1) tamanho do raio da circunferência; (2) tamanho de cada um dos lados do triângulo; (3) tamanho dos dois lados retângulo
- ▶ Usar uma classe Figura, e uma classe para cada tipo de figura, que herda de Figura
- ▶ Onde os métodos `calculaArea()` e `calculaPerimetro()` devem estar definidos?

# Referências

---

- ▶ Slides de Leonardo Murta

# Subprogramação

Vanessa Braganholo  
vanessa@ic.uff.br