



Curso de Java 8 para Web

Professor
Antonio Benedito Coimbra Sampaio Jr

 | Treinamentos

www.abctreinamentos.com.br

Primeira **Disciplina**

JAVA 8 - Fundamentos Teóricos e Orientação a Objetos

- **UNIDADE 1:** Introdução à Tecnologia Java
- **UNIDADE 2:** Introdução à Sintaxe Java
- **UNIDADE 3: Programação Orientada a Objetos em Java (Parte I)**
- **UNIDADE 4: Programação Orientada a Objetos em Java (Parte II)**

UNIDADE 3

PROGRAMAÇÃO ORIENTADA A OBJETOS EM JAVA (PARTE I)

Orientação a Objetos

Orientação a Objetos

- É um dos principais paradigmas de análise, projeto e construção de software. Surgiu na década de 60 e foi adotado por várias linguagens de programação, tais como: **Java, C++, Objective-C, Smalltalk, Delphi, Javascript, C#, Perl, Python, Ruby, PHP**, entre outras.
- **A Análise Orientada a Objetos:**
 - Determina o que o sistema deve fazer: Quais os atores envolvidos? Quais as atividades a serem realizadas?
 - Decompõe o sistema em objetos: Quais são? Quais tarefas cada objeto terá que fazer?
- **O Projeto Orientado a Objetos:**
 - Define como o sistema será implementado;
 - Modela os relacionamentos entre os objetos e atores (utiliza-se a linguagem de modelagem **UML**);
 - Utiliza e reutiliza abstrações como: **classes, objetos, atributos, métodos, interfaces, herança, polimorfismo, frameworks, etc.**

Análise 0.0 (1) x

Análise Procedural (2)

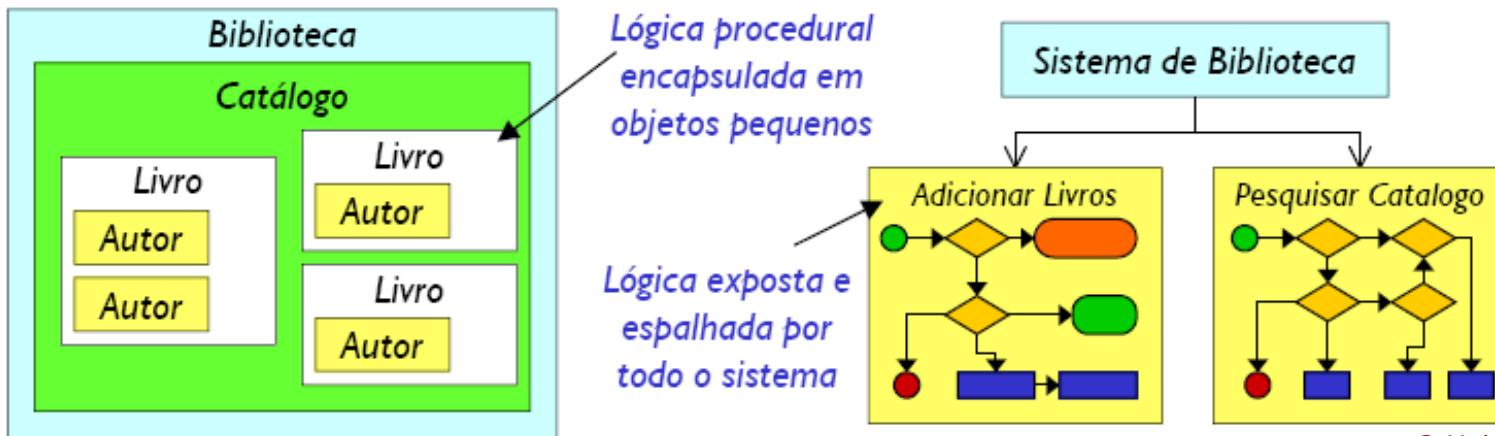


(1) Trabalha no **espaço do problema** (casos de uso simplificados em objetos)

- Abstrações mais simples e mais próximas do *mundo real*

(2) Trabalha no **espaço da solução** (casos de uso decompostos em procedimentos algorítmicos)

- Abstrações mais próximas do *mundo do computador*

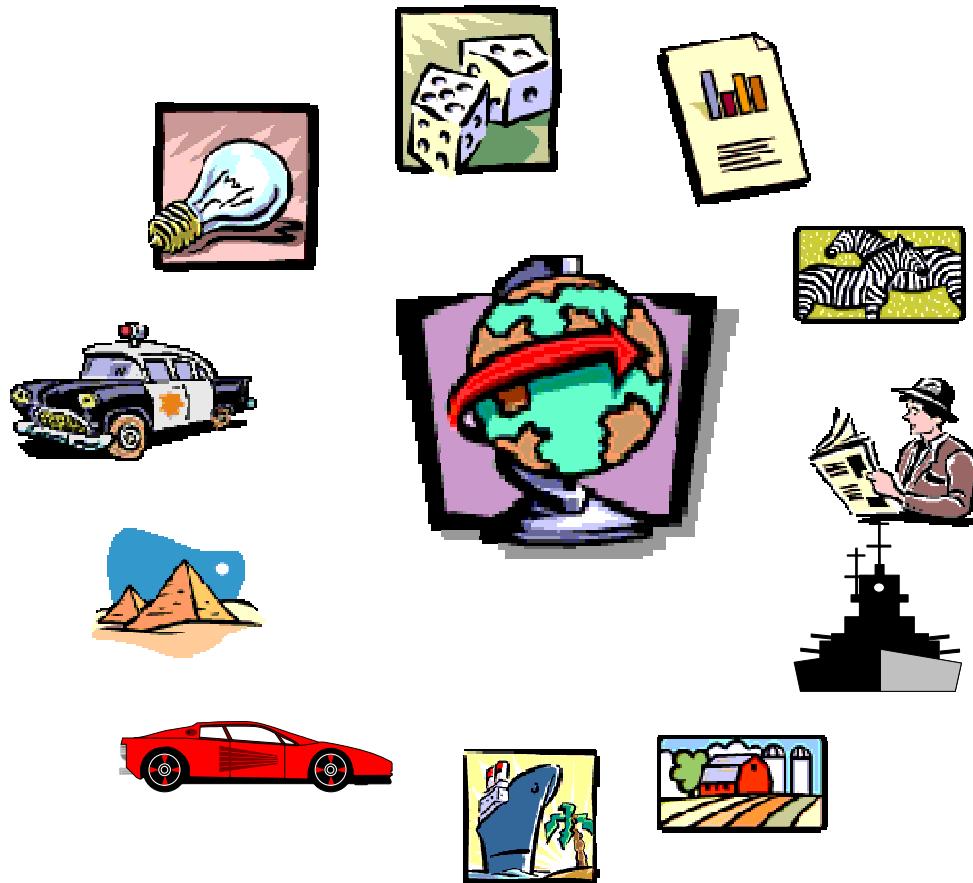


© Helder da Rocha

Vantagens da Análise O.O

- A Orientação a objetos ajudar a melhor organizar e escrever menos código, além de concentrar as responsabilidades nos pontos certos, flexibilizando a aplicação e encapsulando a lógica de negócios;
- Em Resumo, **os principais benefícios da Orientação a Objetos:**
 - 1. Facilidade em **projetar o software** desejado, visto a utilização de abstração (classes x objetos) de alto nível;
 - 2. Facilidade na **manutenção do software**, visto a simplicidade em testar, manter e depurar o código escrito;
 - 3. Facilidade em **reutilizar o software**, visto que as classes criadas podem ser reaproveitadas em novos códigos.

O Mundo é composto de Objetos!



O que é um Objeto?

DEFINIÇÕES

- Uma Abstração;
- Alguma coisa que faz sentido no domínio da aplicação.



UTILIDADES

- Facilita a compreensão;
- Oferece base real para implementação no computador.

Descrição de um Objeto

- Um objeto é representado por um conjunto de **atributos** (também conhecidos como propriedades) e por um conjunto de **métodos** (que definem o comportamento de um dado objeto):

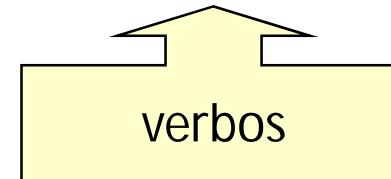
Atributos

Motor
Cor
Potência
Fabricante



Métodos

Acelerar
Retroceder
Parar
Abastecer



Exemplo de um Objeto

Atributos

Motor: V12

Cor: Azul

Potência: 600cv

Fabricante:

Ferrari



substantivos

Métodos

Acelerar

Retroceder

Parar

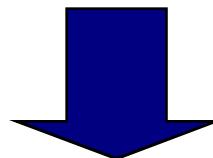
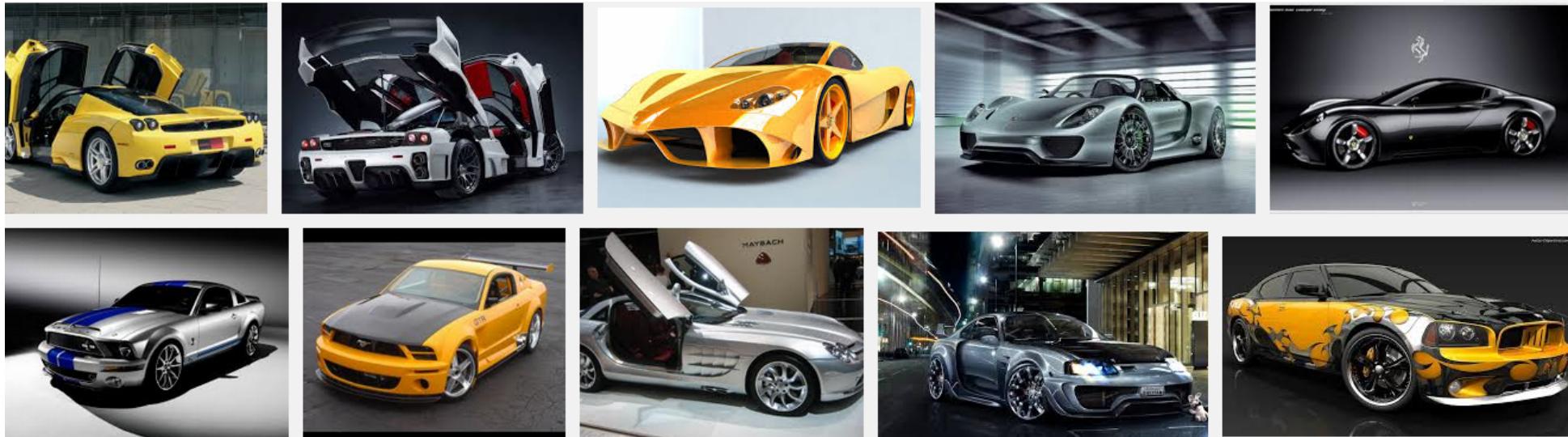
Abastecer

verbos

Classe de Objetos

DEFINIÇÃO

- O grupo de objetos que possuem os mesmos atributos e métodos diz-se que pertencem à mesma classe.

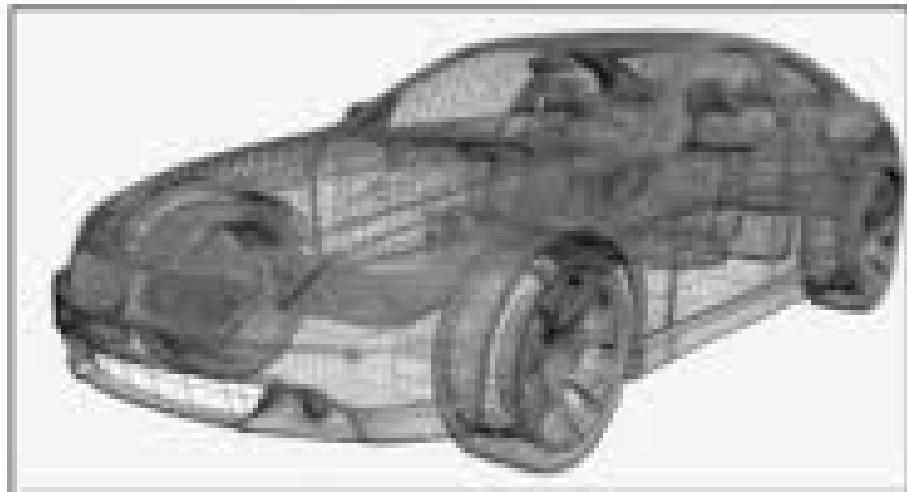


Classe Carro

Classe de Objetos

CLASSES X OBJETOS

- O grupo de objetos que possuem os mesmos atributos e métodos diz-se que pertencem à mesma classe.



Classe



Objeto

Classe de Objetos

CLASSES X OBJETOS

- Uma classe é um modelo ou protótipo que define as propriedades e métodos (comportamento) comuns a um conjunto de objetos;
- Classes são “moldes” que definem as variáveis e os métodos comuns a todos os objetos de um determinado tipo;
- No mundo real existem vários objetos do mesmo tipo. Por exemplo, o seu carro é um dos milhares que existem no mundo;
- Usando a terminologia de orientação a objetos, o objeto seu carro é uma instância da classe de objetos carro.

Exercícios

- 1) Uma casa está para uma planta arquitetônica assim como um objeto está para:
 - (a) um método
 - (b) uma propriedade
 - (c) uma classe
 - (d) um atributo
 - (e) uma herança
- 2) Para criar um sistema de informação que gerencie o aluguel de uma frota de carros, quais são as classes de objeto necessárias?

Orientação a Objetos em Java

Orientação a Objetos em JAVA

- Programas em Java provavelmente irão criar diversos objetos de diversos tipos de classes;
- Os objetos interagem entre si através da troca de mensagens;
- Após o objeto ter realizado o trabalho proposto, o mesmo é eliminado através da “coleta automática de lixo”;
- Para a construção de software em Java, faz-se uso:
 - Das classes já existentes no Java:
 - **String (java.lang)**
 - **List, Iterator, etc. (Collections API)**
 - **Frame (java.awt)**
 - **Outras.**
 - Das novas classes criadas pelo engenheiro de software.

Classe String

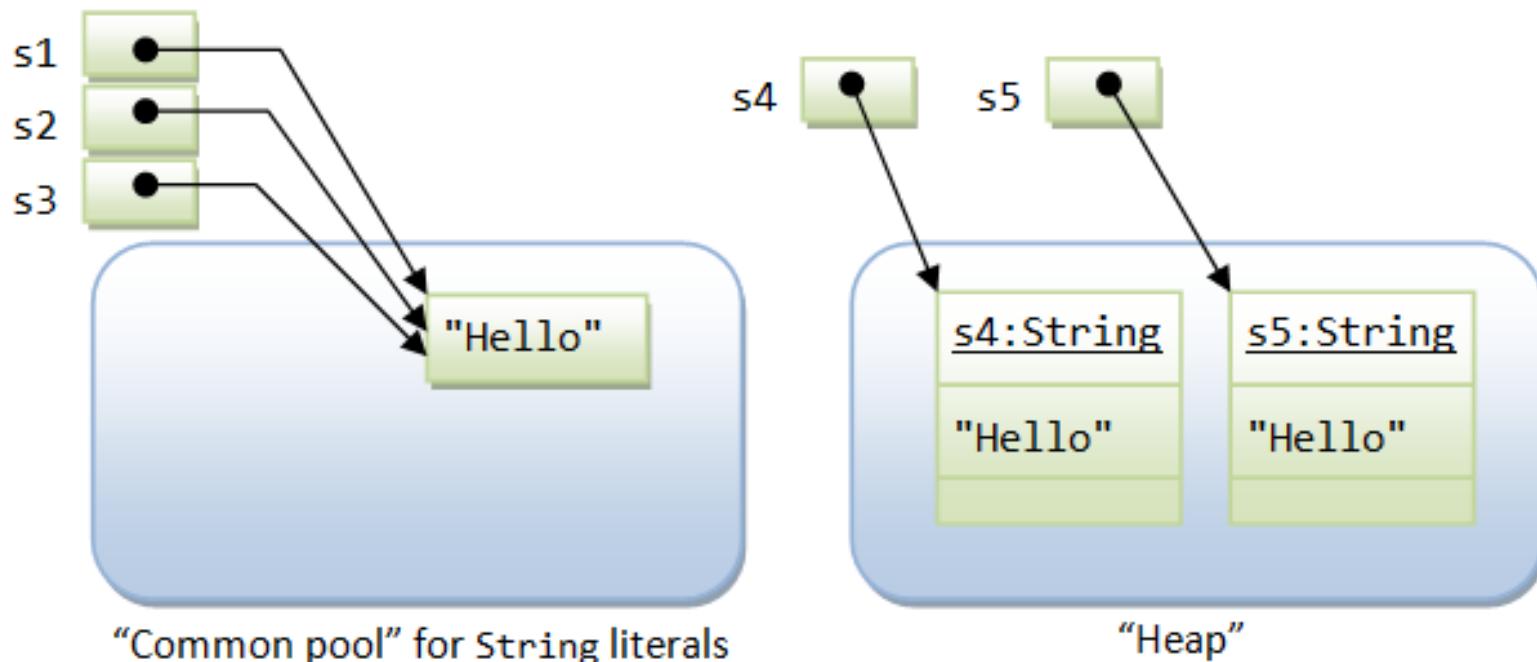
- A classe String está definida no pacote **java.lang** e representa uma seqüência de caracteres;
- A criação de um objeto do tipo String pode ser realizado com ou sem a palavra reservada **new**;

```
//construção implícita (String literal)
String str1 = "Java é show!";
//construção explícita (String object)
String nome = new String("Antonio");
```

- As Strings “literais” são armazenadas em uma área comum (common pool). Isso facilita o compartilhamento de armazenamento para Strings com o mesmo conteúdo. Já as Strings “objetos” são armazenadas em uma área específica de memória (**heap**), sem qualquer tipo de compartilhamento.

Classe String

```
String s1 = "Hello";           // String literal  
String s2 = "Hello";           // String literal  
String s3 = s1;                // same reference  
String s4 = new String("Hello"); // String object  
String s5 = new String("Hello"); // String object
```



© Chua Hock-Chuan

Classe String

```
s1 == s1;          // true, same pointer
s1 == s2;          // true, s1 and s1 share storage in common pool
s1 == s3;          // true, s3 is assigned same pointer as s1
s1.equals(s3);    // true, same contents
s1 == s4;          // false, different pointers
s1.equals(s4);    // true, same contents
s4 == s5;          // false, different pointers in heap
s4.equals(s5);    // true, same contents
```

- **Operações com Strings**

[© Chua Hock-Chuan](#)

- int length(), boolean equals(String another)
- int indexOf(String search), char charAt(int index)
- String substring(int fromIndex)
- String toLowerCase(), String toUpperCase()
- String trim(), String replace(char oldChar, char newChar)
- boolean matches(String regex)
- static String format(String formattingString, Object... args)

Classe String

- Operações com Strings

```
// Criação
String alo = "Alo Pessoal";
// Concatenação
String nova = alo + ", tudo Ok ?";
// Edição
String substr = alo.substring(0,3); // "Alo"
// Verificando tamanho
int tam = alo.length(); // tamanho = 11
```

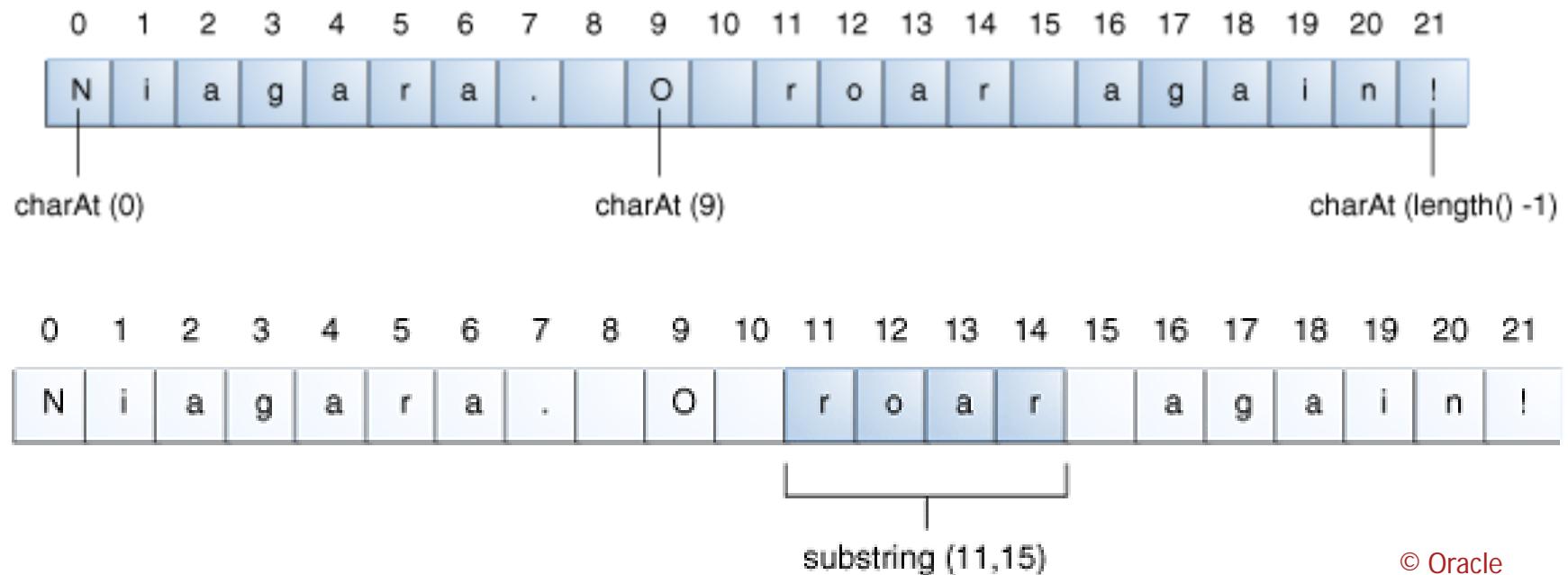
```
//A partir do Java 5
String.format("%.1f", 1.234); // "1.2"
System.out.printf("%.1f", 1.234);
```

Classe String

- Operações com Strings

```
String anotherPalindrome = "Niagara. O roar again!";
char aChar = anotherPalindrome.charAt(9);
```

```
String anotherPalindrome = "Niagara. O roar again!";
String roar = anotherPalindrome.substring(11, 15);
```



© Oracle

Classe String

- Conversão

Conversão String => Número

```
float a = (Float.valueOf(args[0])).floatValue();
float a = Float.parseFloat(args[0]);
```

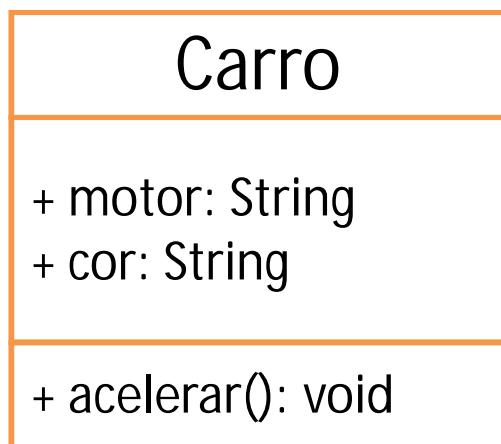
Conversão Número => String

```
String s = "" + i;
String s = Float.toString(i);
```

Nova Classe em JAVA

DECLARAÇÃO

```
class Carro {  
    //define os seus atributos  
    String motor, cor;    Atributos  
    ...  
    //define os seus métodos  
    void acelerar() {...} Método  
}
```



Atributos

Método

Representação em UML

Novo Objeto em JAVA

- Um objeto de uma classe é criado utilizando-se a palavra **new**.

```
Carro car1;  
// cria apenas a referência  
  
Carro car1 = new Carro();  
// aloca memória e atribui  
// endereço à referência
```

car1

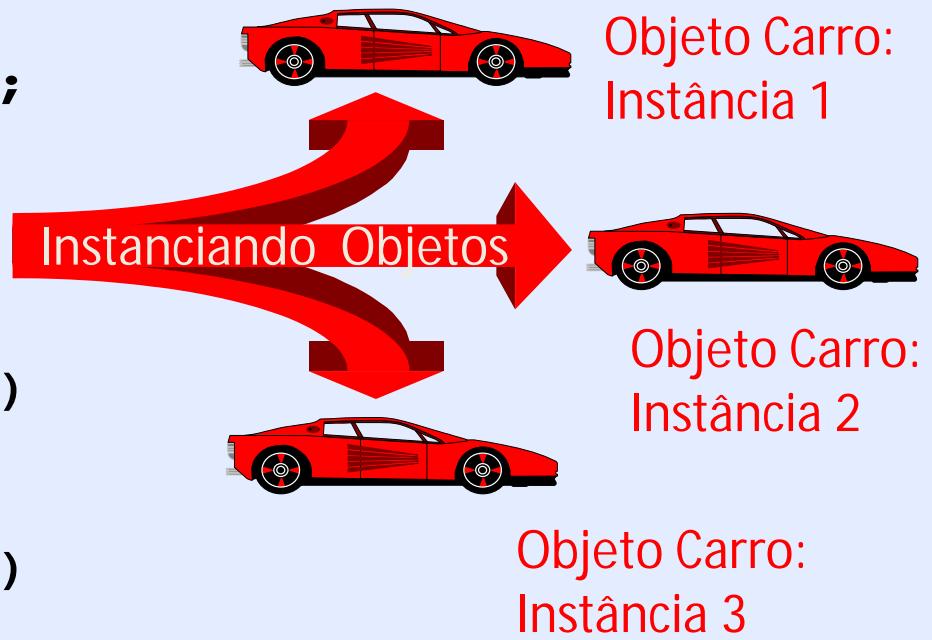
null

car1



Exemplo

```
class Aplicacao {  
    public static void main(String arg[])  
    {  
        Carro car1, car2, car3;  
        car1 = new Carro();  
        car2 = new Carro();  
        car3 = new Carro();  
  
        if (car1.TemGasolina())  
        { car1.ligar();}  
  
        if (car2.TemGasolina())  
        { car2.ligar();}  
    }  
}
```



- Para cada objeto do tipo 'Carro' criado, é alocado um espaço de memória específico.

Exercícios

- 1) Escreva as classes (**Cliente**, **Locacao** e **Carro**) do sistema de informação que gerencie o aluguel (**sisalucar**) de uma frota de carros.
- 2) Na classe principal do **sisalucar** (**SisalucarApp**) crie dois objetos do tipo **Cliente** e dois objetos do tipo **Carro**.

Atributos e Métodos

Atributos

- As propriedades dos objetos podem ser manipuladas diretamente pelo operador de ponto (.).

```
Carro car1 = new Carro();
car1.cor = "azul";
car1.fabricante = "ferrari";
```



```
System.out.println("Cor carro: " + car1.cor); //azul
System.out.println("Fabricante carro: " + car1.fabricante);
//ferrari
```

- Os valores das propriedades podem ser obtidos facilmente pelo operador de ponto (.)

Métodos

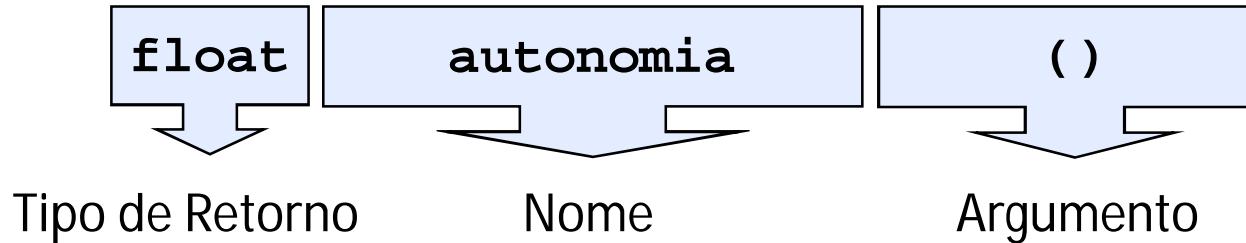
- Definem o comportamento da classe;
- Possuem sintaxe semelhante à sintaxe de definição das funções de um programa procedural;
- Determinam o comportamento da classe e a troca de mensagens com outras classes.

DECLARAÇÃO

```
class Carro {  
    String fabricante, cor;  
    int capacidadeTanque;  
    float consumo;  
    public float autonomia ( ) {  
        return capacidadeTanque * consumo;  
    }  
}
```

Métodos

ASSINATURA



A PALAVRA-CHAVE RETURN

- A palavra-chave **return** especifica o que será retornado após a chamada a um método. Se o método for **void**, não haverá o uso do **return**.

```
boolean método() {  
    if (condição) {  
        instrução;  
        return true;  
    }  
    resto do método  
    return false;  
}
```

Chamadas de Métodos

- A troca de mensagens entre os objetos é realizada através da chamada de métodos.

EXEMPLO

```
class Aplicacao
{
    public static void main (String args[])
    {
        Carro car1 = new Carro();
        ...
        System.out.println(car1.autonomia());
    }
}
```

Chamada do método

Método **MAIN(...)**

- O método **main()** é chamado (automaticamente) pelo interpretador Java;
 - Sempre deve possuir a seguinte assinatura:

Exercícios

- 1) No **sisalucar** criar os atributos e os métodos das classes listadas abaixo:
 - **Carro** (idCarro, placa, fabricante, modelo, ano, cor, valorDiaria)
 - **Cliente** (idCliente, cpf, nome, cnh)
 - **Locacao** (idLocacao, idCarro, idCliente, valorLocado, dataInicio, dataFim)
- 2) Na classe principal do **sisalucar** (**SisalucarApp**) crie um objeto do tipo **Cliente** e um objeto do tipo **Carro**, com os seus respectivos atributos.
- 3) Na classe principal do **sisalucar** (**SisalucarApp**) criar um objeto do tipo **Locacao** que faz a associação do objeto do tipo **Cliente** com o objeto do tipo **Carro**, criado no exercício anterior. Esta associação é feita via método **realizarLocacao(...)**.

Variáveis Locais e Varargs

Variáveis Locais

- Além das propriedades de um objeto, podem ser definidas variáveis, locais a um método ou a um bloco de operações;
- Essas variáveis existirão enquanto o procedimento (método, bloco de controle de execução) a que estiverem associadas for executado;
- Não podem ser usadas fora do procedimento onde foram criadas;
- Não pode ter modificadores de acesso (**private, public, package**).

Variáveis Locais

variáveis visíveis dentro da classe, apenas

novoRaio é variável local ao método mudaRaio

maxRaio é variável local ao método mudaRaio

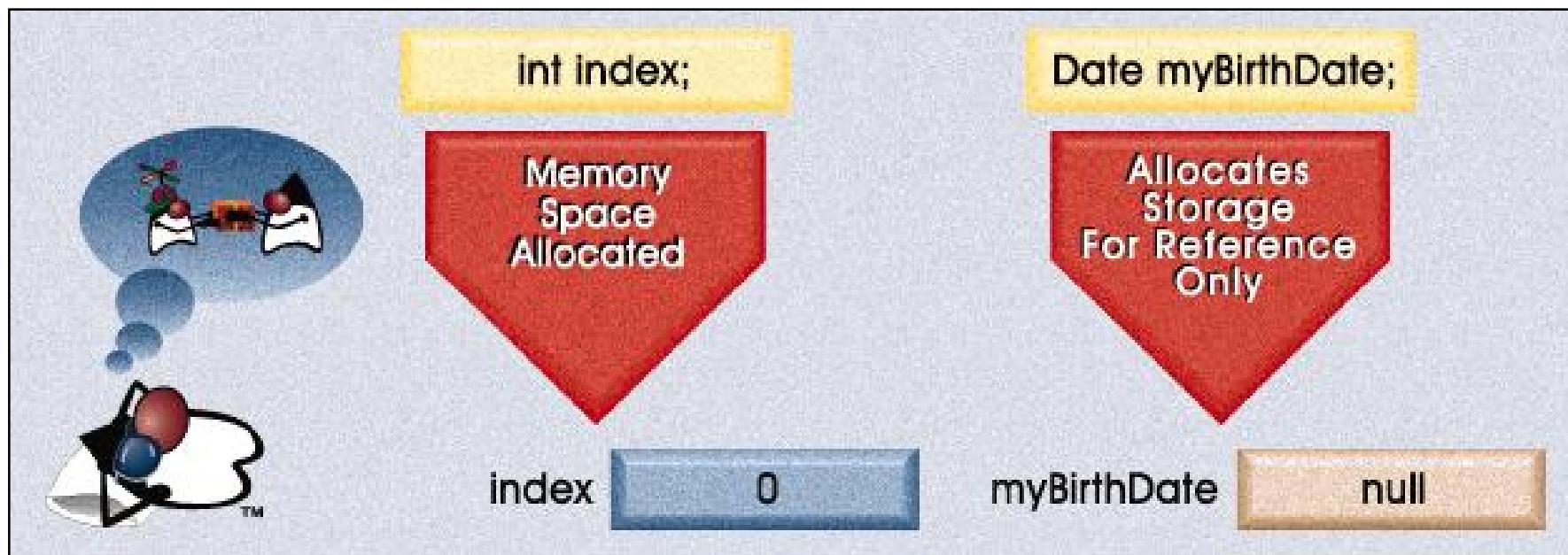
raio é variável de instância

inutil é variável local ao bloco if

```
public class Circulo {  
  
    private int raio;  
    private int x, y;  
  
    public double area() {  
        return Math.PI * raio * raio;  
    }  
  
    public void mudaRaio(int novoRaio) {  
        int maxRaio = 50;  
        if (novoRaio > maxRaio) {  
            raio = maxRaio;  
        }  
        if (novoRaio > 0) {  
            int inutil = 0;  
            raio = novoRaio;  
        }  
    }  
}
```

Tipos de Variáveis Locais

- As variáveis podem ser declaradas por um tipo primitivo (**int, float, char, etc.**) ou por um tipo de Classe (própria ou do Java);
- No primeiro caso (tipo primitivo), o espaço em memória é alocado como parte da operação. No segundo caso (tipo Classe), não há prévia alocação de memória, somente quando o objeto da classe for criado pela chamada da instrução **new**.



Exercício

- 1) Na classe **SisalucarApp** criar o método **gerarRelatorioLocacao(...)** que mostre todas as locações realizadas nos últimos cinco dias. Crie pelo menos uma variável local para calcular o total faturado nesse período.

VARARGS

ASSINATURA TRADICIONAL DE MÉTODO

```
public static void somar(double n1, double n2)
{
    double soma = n1 + n2;
    System.out.println(soma);
}
```

- No exemplo do método somar, se houver necessidade de incluir mais notas, por exemplo quatro, o mesmo método deverá ser alterado.

```
public static void somar(double n1, double n2,
                        double n3, double n4)
{
    double soma = n1 + n2 + n3 + n4;
    System.out.println(soma);
}
```

VARARGS

- Com o recurso do VARARGS (*Variable-Length Arguments*) é possível declarar um método sem especificar a quantidade de parâmetros exata, trazendo maior flexibilidade ao desenvolvedor.

```
public static void somar(double ...numeros)
{
    double soma = 0;
    for(double numero: numeros)
        soma = soma + numero;
    System.out.println(soma);
}
```

- Com esta nova assinatura, o método somar pode receber de 0 a N parâmetros (neste caso, números do tipo *double*).

VARARGS

SINTAXE

- Não é possível declarar novos parâmetros após o uso de varargs.
- EXEMPLO INVÁLIDO

```
void doIt(int a, int b, int ...v, int c){  
    //instruções  
}
```

ERRO

- EXEMPLO VÁLIDO

```
void doIt(int a, int b, int ...v) {  
    //instruções  
}
```

Passagem de Argumentos

TIPOS DE ARGUMENTOS

- **Objetos (Passagem por Referência)** - As referências originais são sujeitas a alterações dentro do método;
- **Tipos Primitivos (Passagem por Valor)** – As referências originais não são sujeitas a alterações dentro do método.

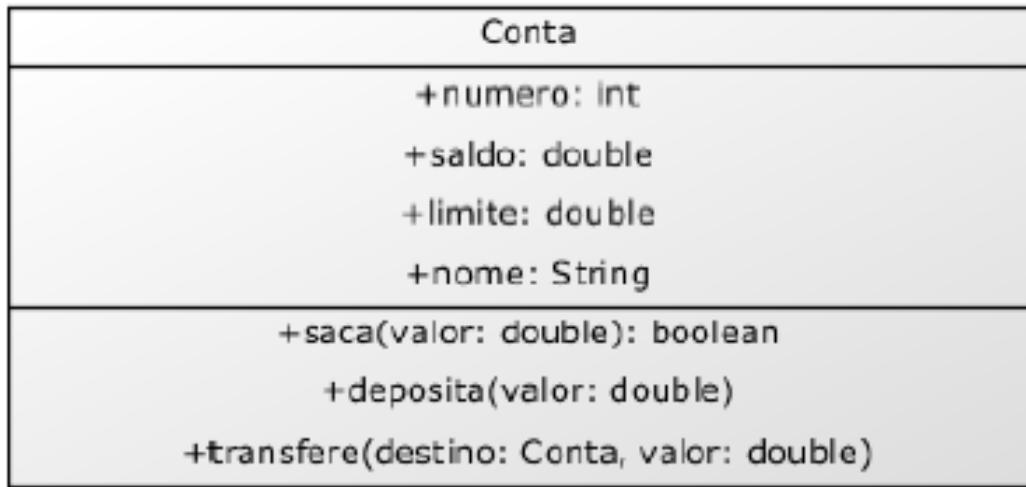
```
class Conta {  
    int cpf, numero;  
    float saldo;  
    public boolean transfere (Conta conta,  
                             float valor) {  
        ...  
        boolean resultado = "true";  
    }  
}
```

Objeto

Tipo

Exercícios

- 1) Alterar o método **gerarRelatorioLocacao(...)** para fazer uso de Varargs.
- 2) Crie a classe Conta conforme o diagrama abaixo:



- 3) Posteriormente, implemente o método transfere conforme o trecho de código abaixo:

```
conta1.transfere(conta2, 50);
```

A leitura deste código seria “Conta1 transfere para conta2 50 reais”.

Construtores

Construtores

- Utilizam-se construtores para inicializar o estado inicial dos objetos de uma determinada classe, isto é, atribuir valores aos seus atributos no momento de criação do objeto pela instrução **new**;
- Os Construtores possuem o mesmo nome da classe e não têm tipo de retorno;

```
class Carro {  
    float consumo;  
    ...  
    Carro(float consumo) {  
        this.consumo = consumo;  
    }  
    Carro() { ...  
    }  
}
```

Construtor

Construtor

Palavra Reservada 'THIS'

- Às vezes é necessário que o objeto se auto-referencie;
- Existe uma palavra reservada **this** que significa uma referência ao próprio objeto.
- Um dos principais usos do **this** é para resolver ambiguidade.

```
class Carro {  
    float consumo;  
    void acelerar(float consumo) {  
        this.consumo = consumo;  
    }  
}
```

Construtores

- Toda classe Java define (implicitamente) um construtor padrão “vazio”, isto é, não há atribuição de valores aos atributos de um determinado Objeto;

```
class Carro {  
    float consumo;  
    ...  
    Carro() { ... }  
}
```

Construtor

```
Carro carl = new Carro();  
//é chamado o construtor vazio
```

Construtores

- Quando se cria um novo construtor em uma determinada classe, o construtor padrão (vazio) deixa de existir e é substituído pelo novo criado pelo desenvolvedor;

```
class Carro {  
    ...  
    Carro(float consumo)  
    {  
        ...  
    }  
}
```

```
Carro car1 = new Carro(500);  
//é chamado o novo construtor
```

```
Carro car1 = new Carro();
```

ERRO

Construtores

- No Java é possível criar uma classe com vários construtores;

```
class Carro {  
    float consumo;  
    ...  
    Carro(float consumo) {  
        this.consumo = consumo;  
    }  
    Carro() { ...  
    }  
}
```

- A escolha de qual construtor será chamado vai depender dos argumentos que serão passados (ou não) no momento da criação do Objeto.

Construtores

EXEMPLO

```
Carro car1 = new Carro();
//é chamado o construtor vazio
Carro car2 = new Carro(500);
//é chamado o construtor que
//recebe parâmetro
```

Exercício

- 1) No **sisalucar** criar pelo menos um construtor para as classes listadas abaixo:
 - **Carro** (idCarro, placa, fabricante, modelo, ano, cor, valorDiaria)
 - **Cliente** (idCliente, cpf, nome, cnh)
 - **Locacao** (idLocacao, idCarro, idCliente, valorLocado, dataInicio, dataFim)

Membros de Classe

Membros de Classe

- Como já foi visto, uma classe é composta de atributos e métodos (também conhecidos como seus membros);
- No Java, é possível que esses membros sejam de objeto (é o comum, já estudado até aqui) e que também sejam de classe (a ser estudado agora);
- Os **Membros de Classe ou “Estáticos”** são aqueles utilizados para fazer referência a uma determinada classe, sem haver qualquer relação com os objetos que serão criados da mesma. Neste caso, não se replicam quando novos objetos são criados.
- Todo membro de classe (atributo e método) deve usar o qualificador **static**.

Membros de Classe

QUANDO UTILIZAR?

- Considere o trecho de código abaixo. Pergunta-se: é possível saber o total de contas criadas pela classe “Contas”?

```
class Conta {  
    private int totalDeContas;  
    //...  
  
    Conta() {  
        this.totalDeContas = this.totalDeContas + 1;  
    }  
}
```

- Com o código acima, o valor de '**totalDeContas**' será sempre 1! Por quê?
- Como o referido atributo é 'de objeto', toda vez que um objeto 'Conta' for criado, o valor de sua variável '**totalDeContas**' será 1! Qual a solução?
- Adicionar a palavra **static** na definição do referido atributo.

Membros de Classe

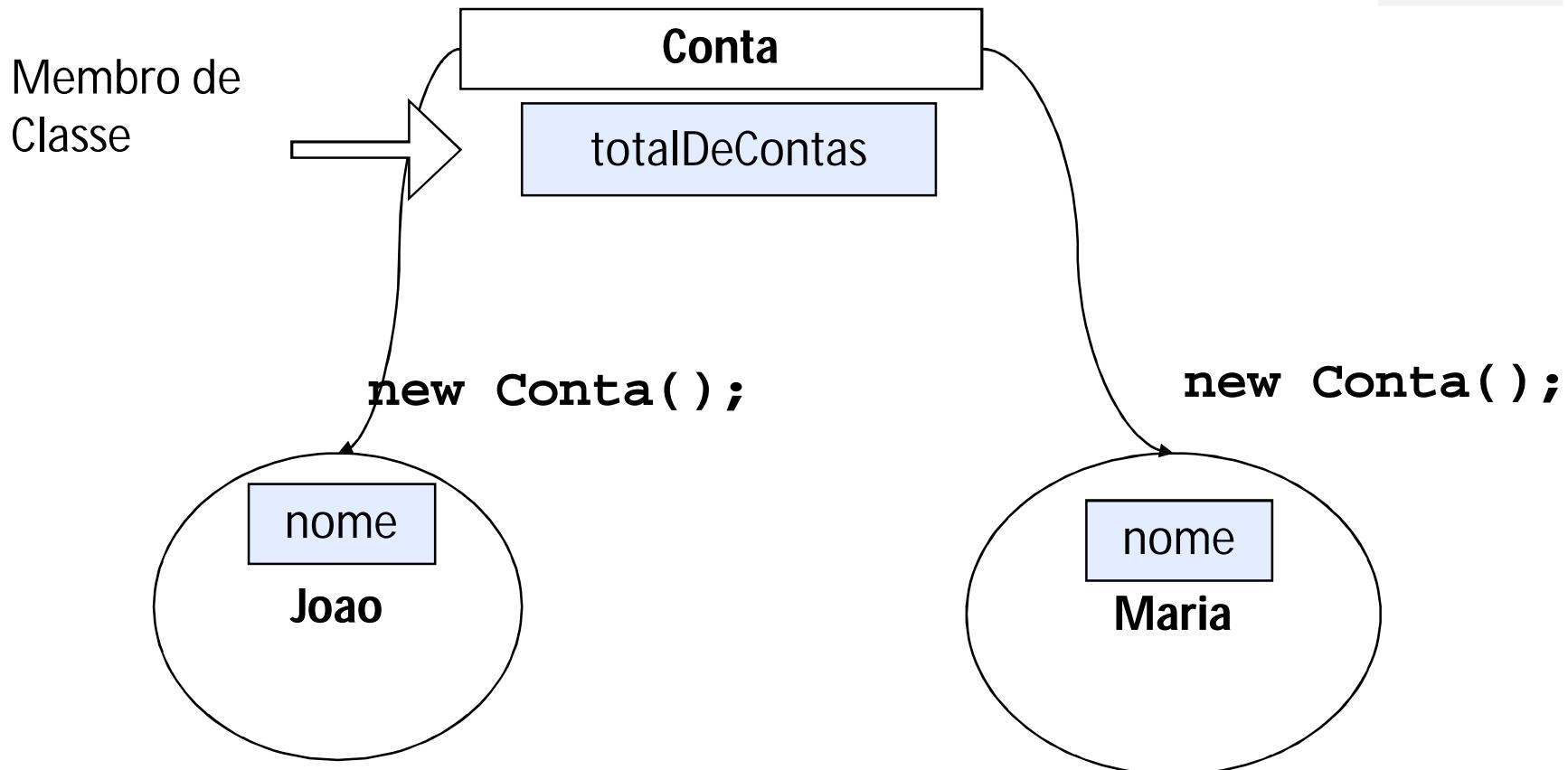
SOLUÇÃO

```
class Conta {  
    private static int totalDeContas;  
    //...  
  
    Conta() {  
        Conta.totalDeContas = Conta.totalDeContas + 1;  
    }  
}
```

- Com o código acima, o atributo '**totalDeContas**' passa a ser único e o seu conteúdo é único para a classe 'Conta';
- Para ter acesso a um atributo estático, utiliza-se a sintaxe:
<<Classe.membro_estático>>
- Exemplo: **Conta.totalDeContas**

Membros de Classe

SOLUÇÃO



Procedimentos de Inicialização

- Usados para inicializar objetos ou classes.

```
class Example {  
    ...  
    static {  
        ...  
    }  
}
```

Membros de Classe (Resumo)

```

public class Casa {
    private Porta porta;
    private int numero;
    public java.awt.Color cor;

    public Casa() {
        porta = new Porta();
        numero = ++contagem * 10;
    }

    public void abrePorta() {
        porta.abre();
    }

    public static String arquiteto = "Zé";
    private static int contagem = 0;

    static {
        if ( condição ) {
            arquiteto = "Og";
        }
    }
}

```

Atributos de instância: cada objeto poderá armazenar valores diferentes nessas variáveis.

Procedimento de inicialização de objetos (Construtor): código é executado após a criação de cada novo objeto. Cada objeto terá um número diferente.

Método de instância: só é possível chamá-lo se for através de um objeto.

Atributos estáticos: não é preciso criar objetos para usá-los. Todos os objetos os compartilham.

Procedimento de inicialização estático: código é executado uma única vez, quando a classe é carregada. O arquiteto será um só para todas as casas: ou Zé ou Og.

© Helder da Rocha

Exercício

- 1) Na classe principal do **sisalucar** (**SisalucarApp**) criar o atributo de classe **frotaCarros**. Alterar também o método **gerarRelatorioLocacao(...)** para um membro de classe.

Atributos Constantes, Enumerações e Métodos Constantes

Atributos Constantes

- São variáveis cujo valor só pode ser definido uma única vez. São declarados com o uso da palavra reservada **final**.

```
class Teste {  
    // Atributos constantes  
    final int ESQUERDA = 0;  
    final int DIREITA = 0;  
    ...  
}
```

Atributos Constantes

- Utilizados para determinar valores que não podem ser alterados;

```
class Teste {  
    // Atributos constantes  
    public static final int INVERNO = 0;  
    public static final int PRIMAVERA = 1;  
    public static final int VERAO = 2;  
    public static final int OUTONO= 3;  
}
```

- As constantes geralmente são utilizadas em métodos.

```
setEstacao(int estacao){ ... }
```

Atributos Constantes

DESVANTAGENS

- (1) **Não é typesafe**: como a constante `estacao` é um inteiro, o seguinte código é possível:

```
setEstacao(4); //estação não existente
```
- (2) **Não tem namespace**: mistura-se com outras constantes existentes na classe;
- (3) **Requer recompilação do código** se novas constantes forem adicionadas.

SOLUÇÃO: ENUMERATIONS

ENUMERATIONS

- São classes Java que definem constantes;
- Não podem ser instanciadas usando a palavra **new**.

```
public enum Conceito
{
    RUIM, REGULAR, BOM, EXCELENTE
}
```

- Os únicos valores aceitos por um tipo **enum** são os valores previamente definidos.

```
Conceito c = Conceito.EXCELENTE;
if (c == Conceito.REGULAR)
...
switch (c)
{
    case RUIM:
        // ...
}
```

ENUMERATIONS

- Essas classes podem possuir construtores, métodos e atributos de instância.

```
public enum Estacao {  
    INVERNO, PRIMAVERA, VERAO, OUTONO;  
}  
  
public class TesteEstacao {  
    public static void main(String[] args) {  
        for (Estacao estacao: Estacao.values())  
            System.out.println(estacao);  
    }  
}
```

ENUMERATIONS

- Definindo valores às Constantes:

```
public enum Estacao {  
    INVERNO(1), PRIMAVERA(2), VERAO(3), OUTONO(4);  
    private int valor;  
  
    Estacao(int valor){  
        this.valor = valor;  
    }  
    public int getValor(){  
        return this.valor;  
    }  
}
```

- As constantes são sempre declaradas no início.

MÉTODOS CONSTANTES

- São aqueles que não podem ser modificados no processo de herança.

```
class Teste {  
    // Método constante  
    private final static void main (String args[]) {  
        ...  
    }  
    ...  
}
```

Exercício

- 1) Na classe Carro criar um atributo de classe **Revendedor** do tipo **enumeration** com as seguintes propriedades de um revendedor de carros (cnpj, endereço, fabricante).

RESUMO

BOAS PRÁTICAS AO ESCREVER CLASSES

- Use e abuse dos espaços;
- A ordem dos membros não é importante, mas seguir convenções melhora a legibilidade do código:
 - **Mantenha os membros do mesmo tipo juntos** (não misture métodos estáticos com métodos de instância);
 - **Declare as variáveis antes ou depois dos métodos** (não misture métodos, construtores e variáveis);
 - **Mantenha os seus construtores juntos, de preferência bem no início;**
- Se for necessário definir blocos **static**, defina apenas um, e coloque-o no início ou no final da classe.

TÓPICOS APRESENTADOS

- Nesta aula nós estudamos:
 - Orientação a Objetos
 - Orientação a Objetos em Java
 - Atributos e Métodos
 - Variáveis Locais e Varargs
 - Construtores
 - Membros de Classe
 - Atributos Constantes, Enumerações e Métodos Constantes

ATIVIDADES PARA SE APROFUNDAR

- 1) Escrever uma classe Java para realizar as seguintes operações:
 - A) Criar a String nome (“O Curso de Java”) e a String avaliacao (“é ótimo!”);
 - B) Imprimir em uma única linha: “O Curso de Java é ótimo!”
 - C) Extrair apenas a palavra “Java” fazendo uso do método substring(...).
- 2) Escreva uma classe Java para ler uma String e dividi-la em tokens, aplicando-se algum tipo de caractere ou caracteres como um delimitador. Por exemplo, o texto “02/20/67” poderia ser dividido em três tokens – 02, 20 e 67 – usando o caracter de barra (“/”) como delimitador.

ATIVIDADES PARA SE APROFUNDAR

- 3) Qual o erro do código abaixo?

```
public class SomethingIsWrong {  
    public static void main(String[] args) {  
        Rectangle myRect;  
        myRect.width = 40;  
        myRect.height = 50;  
        System.out.println("myRect's area is " + myRect.area());  
    }  
}
```

- 4) Relacione classes, atributos e métodos para a seguinte situação:
"Uma pequena loja deseja manter informações sobre clientes, produtos e pedidos de produtos feitos pelos clientes, as informações necessárias para atender um pedido são: nome e endereço do cliente que fez o pedido, relação das descrições e preços dos produtos solicitados e data e endereço de entrega dos produtos."

ATIVIDADES PARA SE APROFUNDAR

- **5) Dado o código abaixo, responda ao que se pede:**

```
public class IdentifyMyParts {  
    public static int x = 7;  
    public int y = 3;  
}
```

- **A) Quais as variáveis de classe e de instância?**
- **B) Quais os resultados abaixo?**

```
IdentifyMyParts a = new IdentifyMyParts();  
IdentifyMyParts b = new IdentifyMyParts();  
a.y = 5;  
b.y = 6;  
a.x = 1;  
b.x = 2;  
System.out.println("a.y = " + a.y);  
System.out.println("b.y = " + b.y);  
System.out.println("a.x = " + a.x);  
System.out.println("b.x = " + b.x);
```