

Java Standard Edition (JSE)

Capítulo 04. Orientação a Objetos: Classes, Objetos e métodos



Esp. Márcio Palheta

MSN: marcio.palheta@hotmail.com



Agenda

- Revisão da aula anterior;
- Motivação – Trabalho em equipe e regras;
- Orientação a objetos;
- Classes, atributos e métodos;
- Relacionamento entre classes;
- Arrays dinâmicos;
- Exercícios de fixação



Revisão

- Declaração e atribuição de variáveis;
- Casting e comparação de variáveis;
- Declaração, criação e inicialização de arrays;
- Laços de repetição for e while;
- Dúvidas ?

Motivação – A importância da equipe



- “A verdade é que não há nada de digno em ser superior a outra pessoa. A única nobreza genuína é ser superior a seu antigo eu”.(Whitney M. Young JR)
- Um dia você precisou de alguém;
- Jogue com o time. Siga as regras;
- Vídeos:
 - 04.01 Equipe;
 - 04.02 Siga regras - Sinto de segurança



Novos recursos a aprender

- Dizer o que é e para que serve orientação a objetos;
- Conceituar classes, atributos e comportamentos;
- Entender o significado de variáveis e objetos na memória.
- Criação e manipulação de arrays dinâmicos;



Orientação a Objetos - OO

- Hei, já estamos usando OO;
- A OO, em alguns trechos utiliza os conceitos de programação estruturada;
- Em OO, os problemas são resolvidos pensando em interações entre diferentes objetos;
- Os dados e operações formam um único conjunto (objeto);
- O objeto guarda(encapsula) a lógica de negócios;



Orientação a Objetos - OO

- Benefícios da abordagem orientada a objetos:
 - **Modularidade:** Uma vez criado, um objeto pode ser passado por todo o sistema;
 - **Encapsulamento:** Detalhes de implementação ficam ocultos externamente ao objeto;
 - **Reuso:** Uma vez criado, um objeto pode ser utilizado em outros programas;
 - **Manutenibilidade:** Manutenção é realizada em pontos específicos do seu programa (objetos).



Objetos

- São “coisas” do mundo real que possuem dados(**atributos**) e realizam ações(**métodos**);
- O estado de um objeto pode variar de acordo com a execução do programa (tempo);
- São oriundos(instâncias) de classes;



Objetos Exemplos

Objeto	Estado	Comportamento
Pessoa	Nome, idade, RG	Falar, andar, respirar
Cachorro	Nome, raça	Latir, correr
Conta bancária	Agência, número	Creditar, debitar
Carro	Cor, marca, modelo	Acelerar, frear, abastecer;



Classe

- É a especificação para um determinado tipo de objeto;
- Cada instância da classe é um objeto;
- Todo objeto respeita às regras definidas na classe à qual pertence;



Exemplo de classe e objeto

Documento	Documento01	Documento02
Foto:	Img01.jpg	Img02.jpg
Código:	123456	654321
Nome:	Joao	Maria
Nascimento:	10/05/1980	30/06/1990



Classe - exemplo

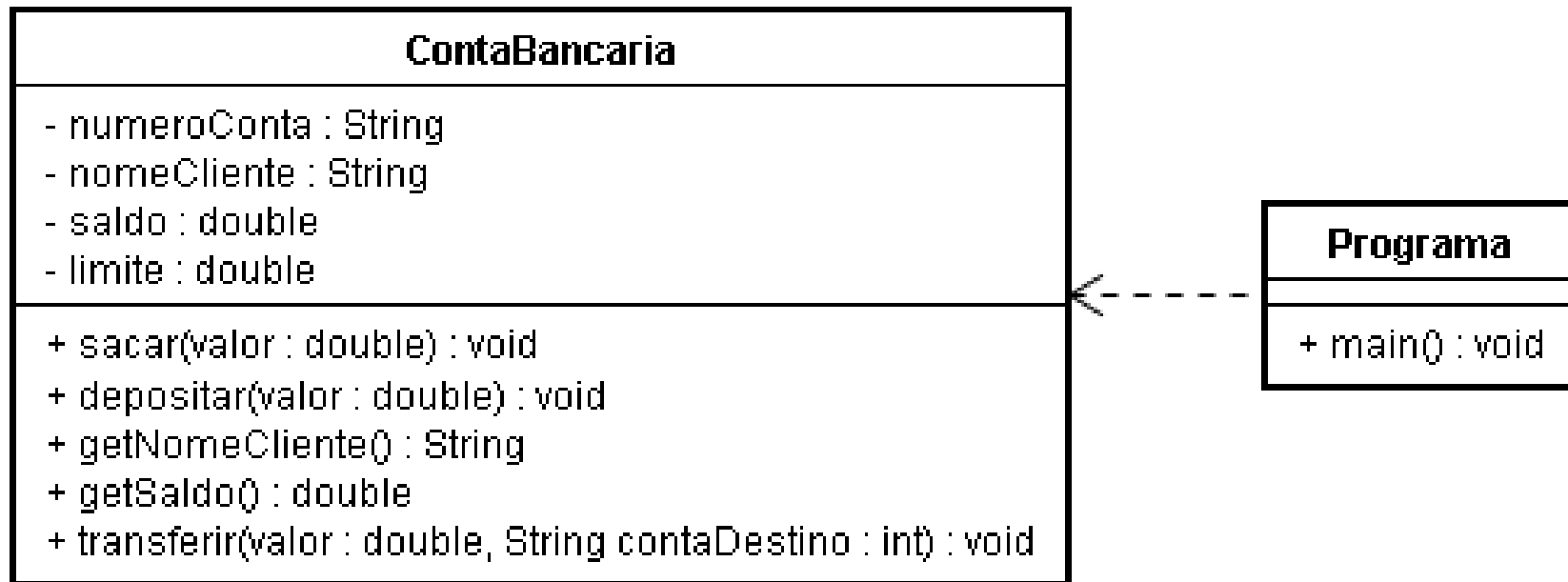
- Considerando um programa para um banco, é bem fácil perceber que uma **entidade** extremamente importante para o nosso sistema é a **conta**.
- Nossa idéia aqui, é generalizarmos alguma informação, juntamente com funcionalidades que toda conta deve ter.



Classe – exemplo

- O que toda conta tem e é importante para nós?
 - número da conta
 - nome do cliente
 - saldo
 - limite
- O que toda conta faz e é importante para nós? Isto é, o que gostaríamos de “pedir à conta”.
 - saca uma quantidade x
 - deposita uma quantidade x
 - imprime o nome do dono da conta
 - devolve o saldo atual
 - transfere uma quantidade x para uma outra conta y
 - devolve o tipo de conta

Representação Gráfica - Diagrama de classes UML





Classe – código JAVA

```
public class ContaBancaria {  
    String numeroConta;  
    String nomeCliente;  
    double saldo;  
    double limite;  
  
    public void sacar(double valor){}  
    public void depositar(double valor){}  
    public String getNomeCliente(){ return this.nomeCliente; }  
    public double getSaldo(){ return this.saldo;}  
    public void transferir(double valor, String contaDestino){}  
}
```



Criação e utilização de um objeto

- O que temos? – uma classe **ContaBancaria** que especifica o comportamento de todos os objetos dessa classe;
- Para criar (construir, instanciar) uma conta, basta usar a palavra chave **new**:
- `ContaBancaria conta = new ContaBancaria();`



Exemplo – Programa.java

```
public class Programa {  
    public static void main(String[] args) {  
        ContaBancaria minhaConta;  
        minhaConta = new ContaBancaria();  
    }  
}
```



Exemplo

- Através da variável `minhaConta`, podemos acessar o objeto recém criado para alterar seu nome, seu saldo etc:
- ```
public class Programa {
 public static void main(String[] args) {
 ContaBancaria minhaConta;
 minhaConta = new ContaBancaria();
 minhaConta.nome = "Duke";
 minhaConta.saldo = 1000.0;
 System.out.println("Saldo atual: " + minhaConta.saldo);
 }
}
```
- É importante fixar que o ponto foi utilizado para acessar algo em `minhaConta`. Agora, `minhaConta` pertence ao Duke, e tem saldo de mil reais.



# Métodos

---

- Representam as ações realizadas sobre os atributos de um objeto;
- Equivalem às funções e procedimentos da programação estruturada;
- A declaração e implementação dos métodos ocorre dentro das classes;



## Exercício 01

---

- Vamos implementar o método `sacar()` da classe `ContaBancaria`, onde o parâmetro é o valor a ser debitado da conta. Este método não retorna valor algum;
- Em seguida, implemente o método `depositar()`



# Exercício 01 – solução sacar()

```
public class ContaBancaria {
 String numeroConta;
 String nomeCliente;
 double saldo;
 double limite;

 public void sacar(double valor) {
 double novoSaldo = this.saldo - valor;
 this.saldo = novoSaldo;
 }
 public void depositar(double valor) {}
 public String getNomeCliente() { return this.nomeCliente; }
 public double getSaldo() { return this.saldo; }
 public void transferir(double valor, String contaDestino) {}
}
```



## Exercício 02

---

- Crie a classe **SacaEDeposita**, onde seu método `main()` utiliza a classe `ContaBancaria` para sacar, depositar e imprimir o saldo da conta, após cada transação;



# Métodos com retorno

---

- São métodos que retornam algum valor para o código que o chamou;
- Ex.: Um cliente não pode sacar um valor maior do que o saldo de sua conta bancária;
- Com isso, podemos alterar o código do método `ContaBancaria.sacar()`, a fim de que seja retornado um valor booleano;
- Operação realizada == true;
- Operação não realizada == false;



## ContaBancaria.sacar()

---

```
public boolean sacar(double valor){
 boolean resultado;
 if(valor > this.saldo){
 resultado = false;
 }else{
 double novoSaldo = this.saldo - valor;
 this.saldo = novoSaldo;
 resultado = true;
 }
 return resultado;
}
```





## Exercício 03

---

- Crie uma nova classe Operacao;
- No método main(), crie um novo objeto ContaBancaria;
- Realize um depósito de R\$ 100;
- Realize um saque de R\$ 150;
- De acordo com o resultado do saque, imprima as mensagens:
  - Saque realizado com sucesso OU
  - Saldo insuficiente;



# Operacao.main()

---

```
public class Operacao {
 public static void main(String[] args) {
 ContaBancaria conta = new ContaBancaria();
 conta.depositar(100.0);
 if(conta.sacar(150)){
 System.out.println("Saque realizado com sucesso.");
 }else{
 System.out.println("Saldo insuficiente.");
 }
 }
}
```



# Acesso por referência

---

- Quando declaramos uma variável para associar a um objeto, na verdade, essa variável não guarda o objeto, e sim uma maneira de acessá-lo, chamada de **referência**.
- É por esse motivo que, diferente dos tipos primitivos como **int** e **long**, precisamos usar o **new** depois de declarada a variável:
- Ex.: Conta c1;  
c1 = **new** Conta();



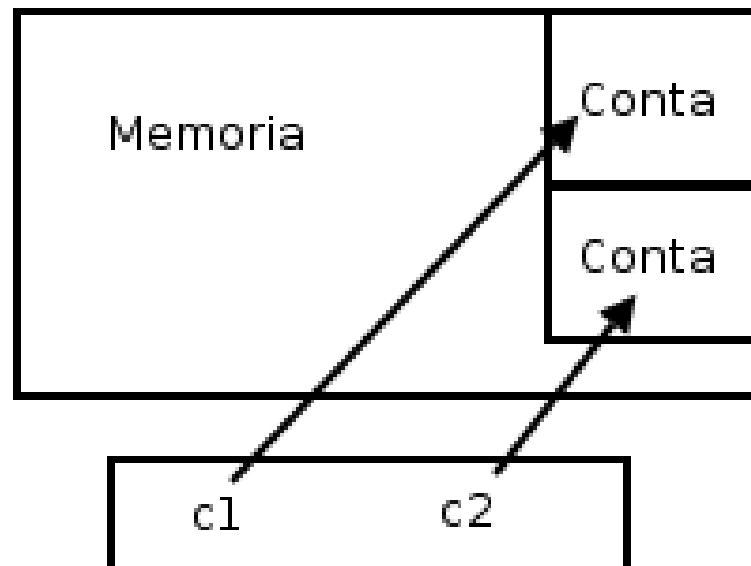
# Acesso por referência

---

- Em JAVA, uma variável nunca é um objeto, mas uma referência a objeto;
- Uma variável referência guarda o endereço de memória onde está o objeto;
- O que acontece quando criamos e iniciamos uma variável de referência?
  - `Conta c1 = new Conta();`
  - `Conta c2 = new Conta();`

# Acesso por referência

- Internamente, c1 e c2 vão guardar um número que identifica em que **posição da memória** cada Conta se encontra.





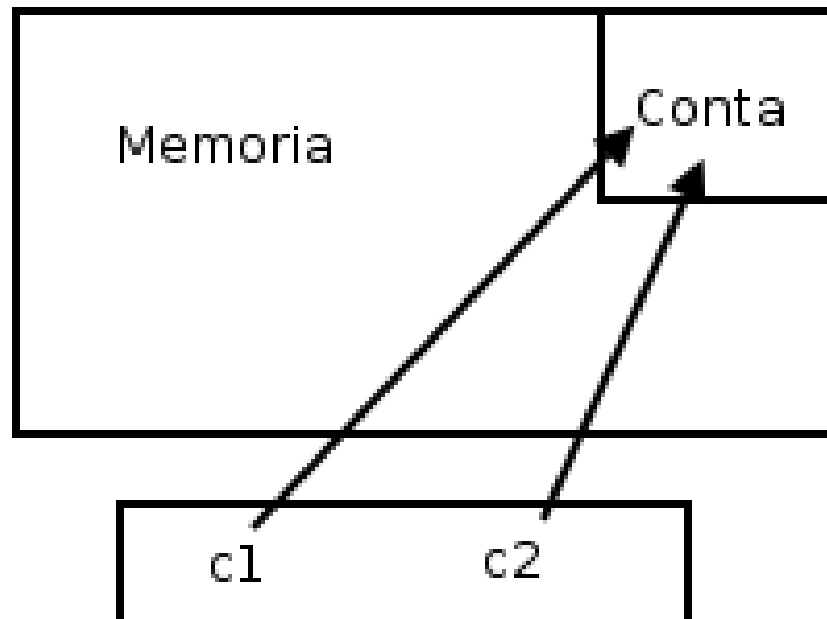
## Outro exemplo de referência

---

- O que será impresso após a execução do código abaixo?
- ```
public static void main(String args[]) {  
    ■ Conta c1 = new Conta();  
    ■ c1.deposita(100);  
    ■ Conta c2 = c1; // linha importante!  
    ■ c2.deposita(200);  
    ■ System.out.println(c1.saldo);  
    ■ System.out.println(c2.saldo);  
}
```

Referência ao mesmo objeto

- C2 copia o endereço de memória armazenado em C1 e passa a apontar para o mesmo objeto ($C2 == C1$):





Referência a objetos distintos

- O que será impresso com o código abaixo?
- ```
public static void main(String args[]) {
 ■ Conta c1 = new Conta();
 ■ c1.nome = "Duke";
 ■ c1.saldo = 227;
 ■ Conta c2 = new Conta();
 ■ c2.nome = "Duke";
 ■ c2.saldo = 227;
 ■ if (c1 == c2) {
 ■ System.out.println("Contas iguais");
 ■ }else{
 ■ System.out.println("Contas diferentes")
 ■ }
 ■ }
```



# O método

## ContaBancaria.transferir()

- E quando precisamos transferir dinheiro entre contas duas contas?
- Na chamada do método, já temos a conta de origem(**this**). Só precisamos da conta destino e do valor a transferir;
- Como fica o código alterado da classe ContaBancaria.transferir()?

## Exercício 04 – Implemente o método **transferir()**;

```
public boolean transferir(ContaBancaria contaDestino, double valor){
 //Variavel que informa se a transferencia foi realizada
 boolean transferiu = false;
 //Verifica se a conta tem saldo suficiente
 boolean retirou = this.sacar(valor);
 if(!retirou){
 JOptionPane.showMessageDialog(null,
 "Não foi possível realizar a transferência." +
 "\n SALDO INSUFICIENTE.");
 }else{
 //realiza o deposito na conta destino
 contaDestino.depositar(valor);
 transferiu = true;
 }
 return transferiu;
}
```



# Questionamentos

---

- O que acontece com a **conta** que foi passada como parâmetro?
- O objeto é “clonado”?
- A passagem de parâmetros é uma atribuição simples do valor da variável;
- Com isso, quando passamos um objeto como parâmetro, estamos, na verdade, passando o seu endereço de memória;

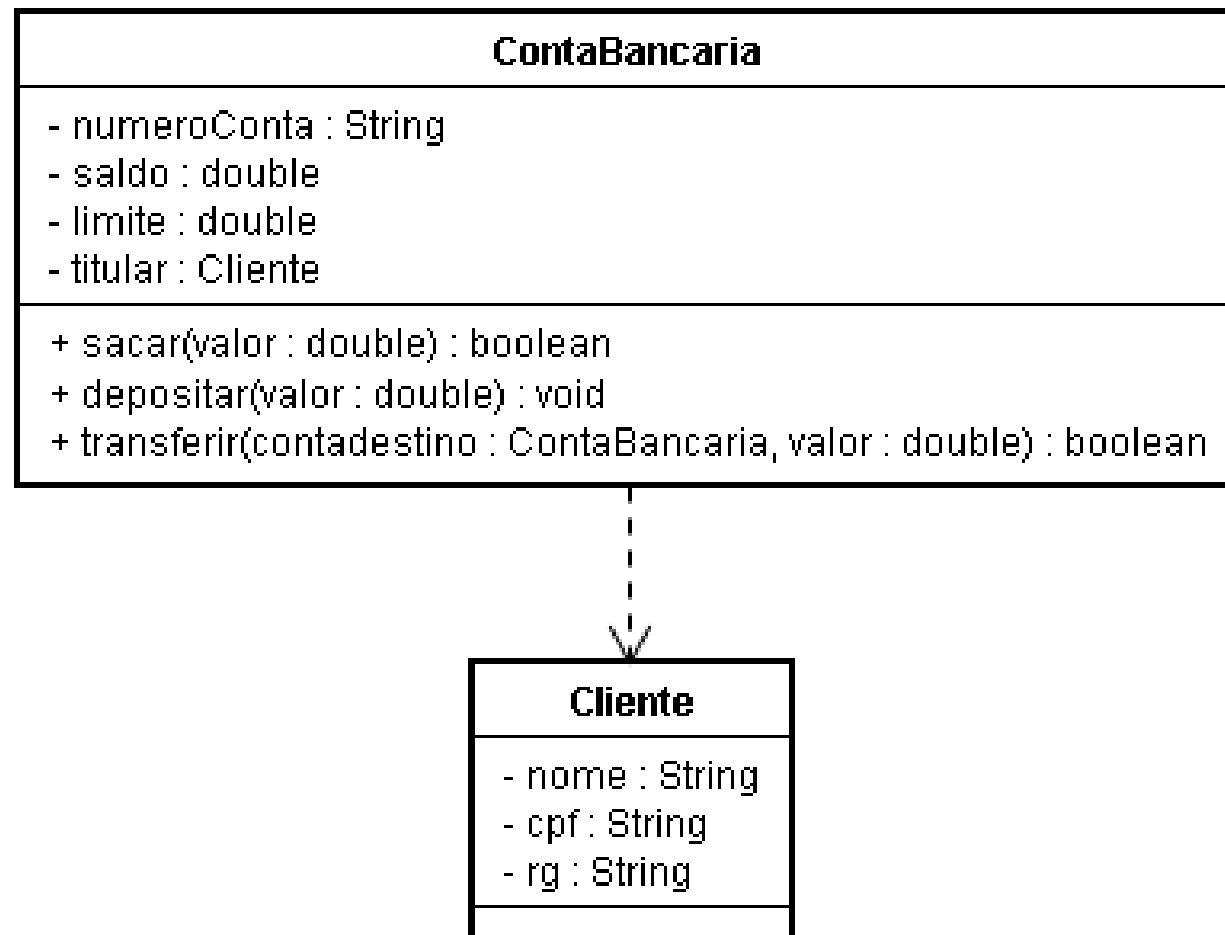


# Melhorando os atributos

---

- Digamos, agora, que precisamos aumentar a classe ContaBancaria e incluir os atributos nome, cpf e rg do cliente dono da conta;
- **Ops**, mas nome, cpf e rg são informações do cliente, não da conta;
- Então, precisamos criar uma classe Cliente e atualizar os atributos de ContaBancaria;

# Representação gráfica – Diagrama de classes UML





# Classes atualizadas

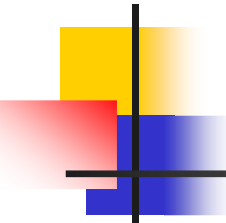
```
public class ContaBancaria {
 String numeroConta;
 double saldo;
 double limite;

 Cliente titular;

 public boolean sacar(double valor) {
 public void depositar(double valor) {
 public boolean transferir(ContaBancaria
 public double getSaldo() {
 }
}
```

```
public class Cliente {
 String nome;
 String cpf;
 String rg;
}
```

# Como vincular um cliente a uma conta bancária?



```
public class TesteClienteConta {
 public static void main(String[] args) {
 //Declaracao e criacao da conta bancaria
 ContaBancaria conta = new ContaBancaria();

 //Declaracao e criacao, criacao e carga do cliente
 Cliente meuCliente = new Cliente();
 meuCliente.cpf = "12345678901";
 meuCliente.rg = "12345678";
 meuCliente.nome = "Eduardo Braga";

 //Atribuicao do cliente como titular da conta bancaria
 conta.titular = meuCliente;
 }
}
```



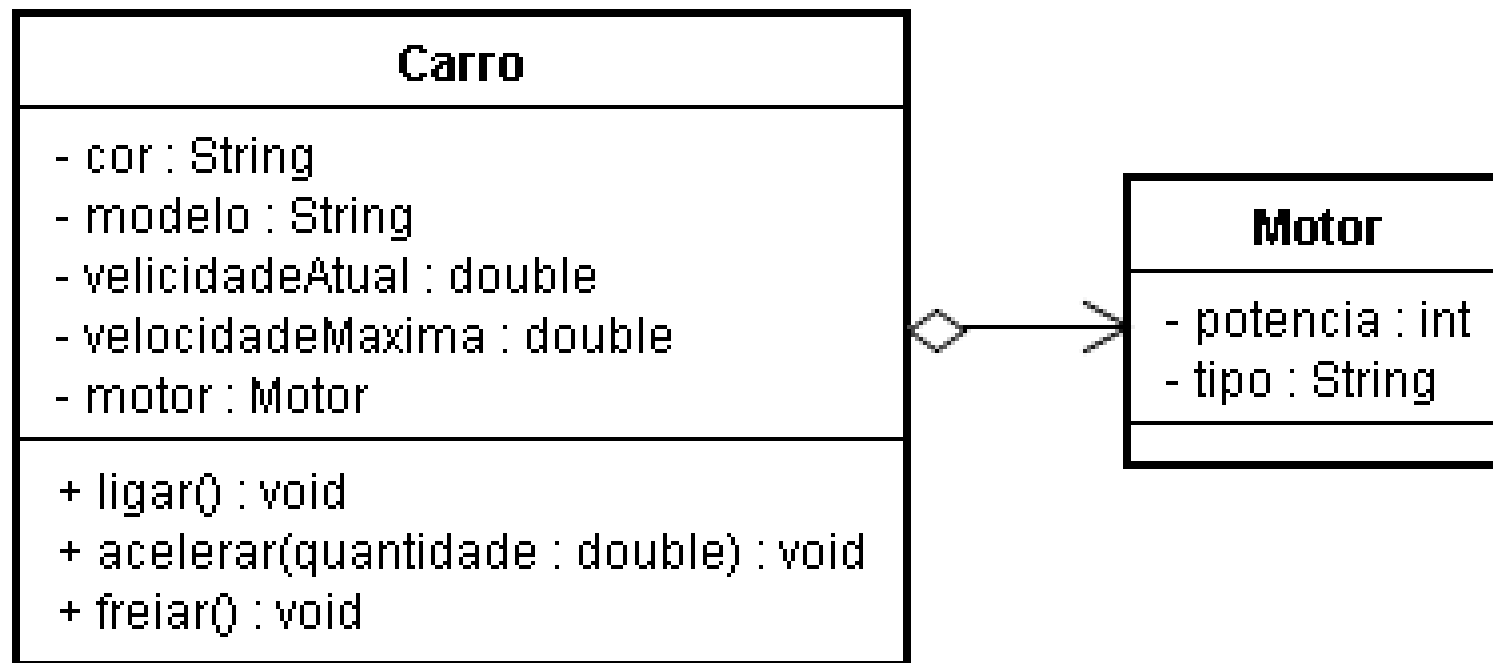
# Acesso aos dados de cliente

---

- Após a atribuição, podemos acessar os dados de Cliente a partir da Conta:
  - `conta.meuCliente.nome = "Luiz Inácio";`
  - `System.out.print(conta.meuCliente.nome);`
- Um sistema OO é um conjunto de classe que se comunicam através de mensagens (métodos);
- Cada classe tem um papel bem definido
- Mas, e se eu não usa-se o new em cliente e tentasse acessá-lo? **NullPointerException**



# Exercício 05: Implemente o Diagrama de Classe a seguir





## Exercício 05 – Classe Carro

```
public class Carro {
 String cor;
 String modelo;
 double velocidadeAtual;
 double velocidadeMaxima;
 Motor motor;

 public void ligar() {
 System.out.println("O carro está ligado");
 }
 public void acelerar(double quantidade) {
 this.velocidadeAtual += quantidade;
 }
 public void freiar() {
 System.out.println("O carro está freiando");
 }
}
```



## Exercício 06

---

- Escreva uma classe `FabricaCarro` que simule uma fábrica de carros. Todos os dias o gerente informa:
  - A quantidade de carros a ser produzida;
  - A cor, modelo e a velocidade máxima;
  - A potência e o tipo de motor;
- No método `main()`, implemente o armazenamento dos carros produzidos em um Array;



# Criação de arrays dinâmicos

---

- API Collections Framework: Capítulo 10;
- Facilita busca, remoção e tamanho “infinito”;
- As classes Object e ArrayList;
  - Criação: `ArrayList lista = new ArrayList();`
  - Inclusão: `lista.add(Object objeto);`
  - Acesso: `lista.get(int indice);`



# Exemplo ArrayList - Criação

---

- `Conta c1 = new Conta ();`
- `c1.depositar(100);`
- `Conta c2 = new Conta ();`
- `c2.depositar(200);`
- `ArrayList contas = new ArrayList();`
- `contas.add(c1);`
- `contas.add(c2);`
- Para sabermos o tamanho da lista, podemos usar o método `size()`:
  - `System.out.println(contas.size());`



## Exemplo ArrayList - Acesso

---

- `for(int i = 0; i < contas.size(); i++) {`
  - `Conta cc = (Conta) contas.get(i);`
  - `System.out.println(cc.getSaldo());`
- `}`
- O que aconteceu?
- Por que precisamos do **cast** para conta?
- Podemos incluir objetos de tipos diferentes em um ArrayList?



# Bibliografia

---

- Java - Como programar, de Harvey M. Deitel
- Use a cabeça! - Java, de Bert Bates e Kathy Sierra
- (Avançado) Effective Java Programming Language Guide, de Josh Bloch



# Referências WEB

---

- SUN: [www.java.sun.com](http://www.java.sun.com)

## Fóruns e listas:

- Javaranch: [www.javaranch.com](http://www.javaranch.com)
- GUJ: [www.guj.com.br](http://www.guj.com.br)

## Apostilas:

- Argonavis: [www.argonavis.com.br](http://www.argonavis.com.br)
- Caelum: [www.caelum.com.br](http://www.caelum.com.br)





# Java Standard Edition (JSE)

---

## Capítulo 04. Orientação a Objetos: Classes, Objetos e métodos



Esp. Márcio Palheta

MSN: [marcio.palheta@hotmail.com](mailto:marcio.palheta@hotmail.com)