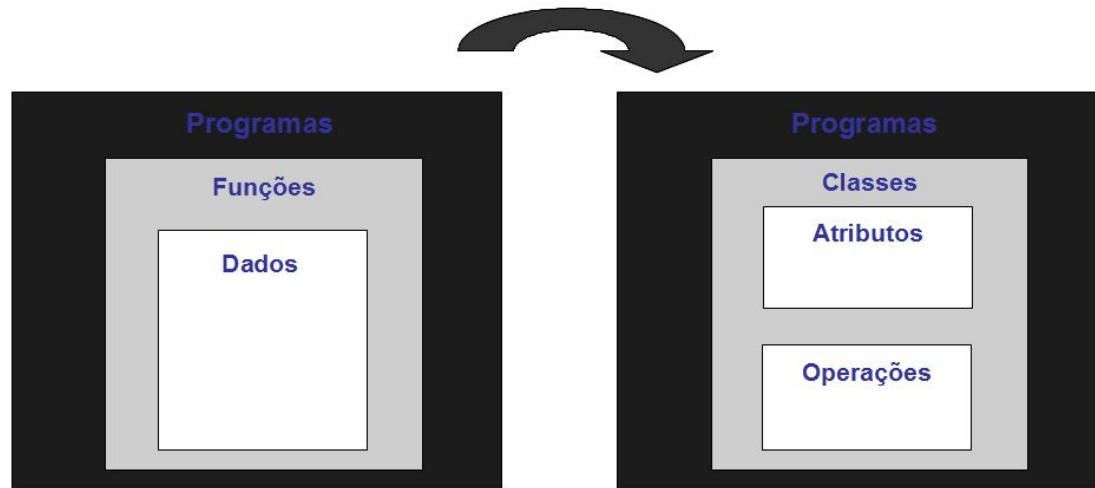


DCC205 – PROGRAMAÇÃO ORIENTADA A OBJETOS

Aula 04

Carlos Bruno Oliveira Lopes
carlosbrunocb@gmail.com

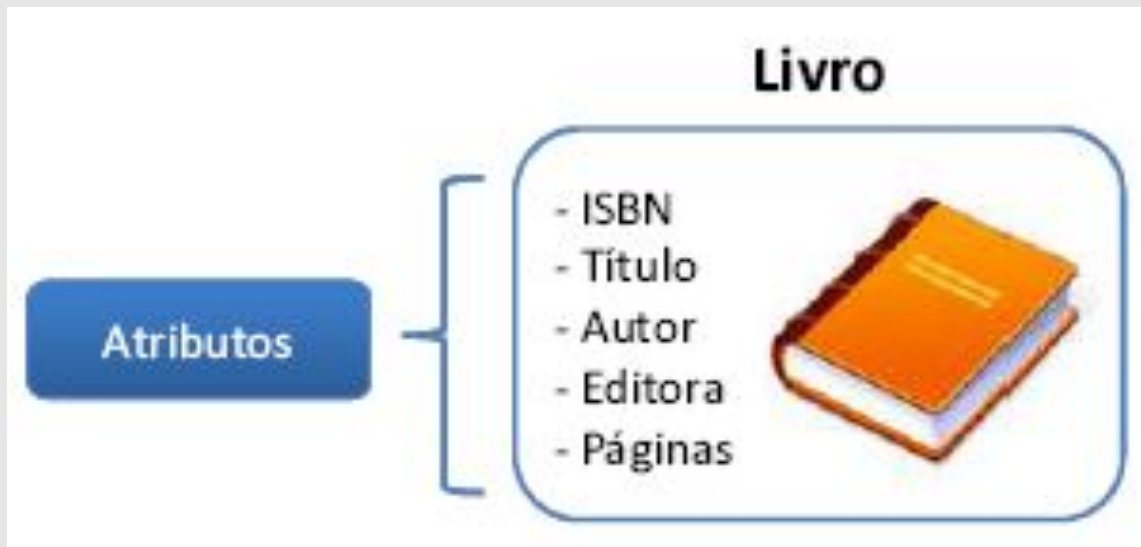
- Mudança de Enfoque

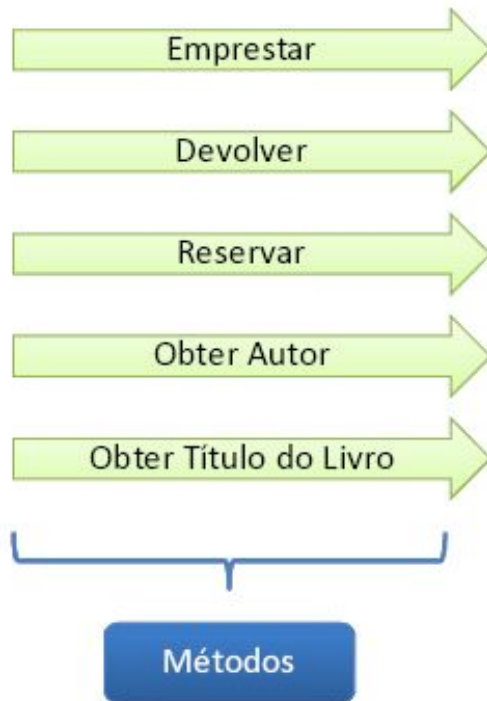


**Comparação
visual entre
paradigmas**

Classes: Estruturas de Dados e comportamentos

- Uma classe representa um tipo de dados
- É uma estrutura de dados (no caso os atributos) e o comportamento (no caso os métodos)





Classes e seus métodos

Atributos X Métodos

Atributos

Características da classe
Normalmente representados
por substantivos



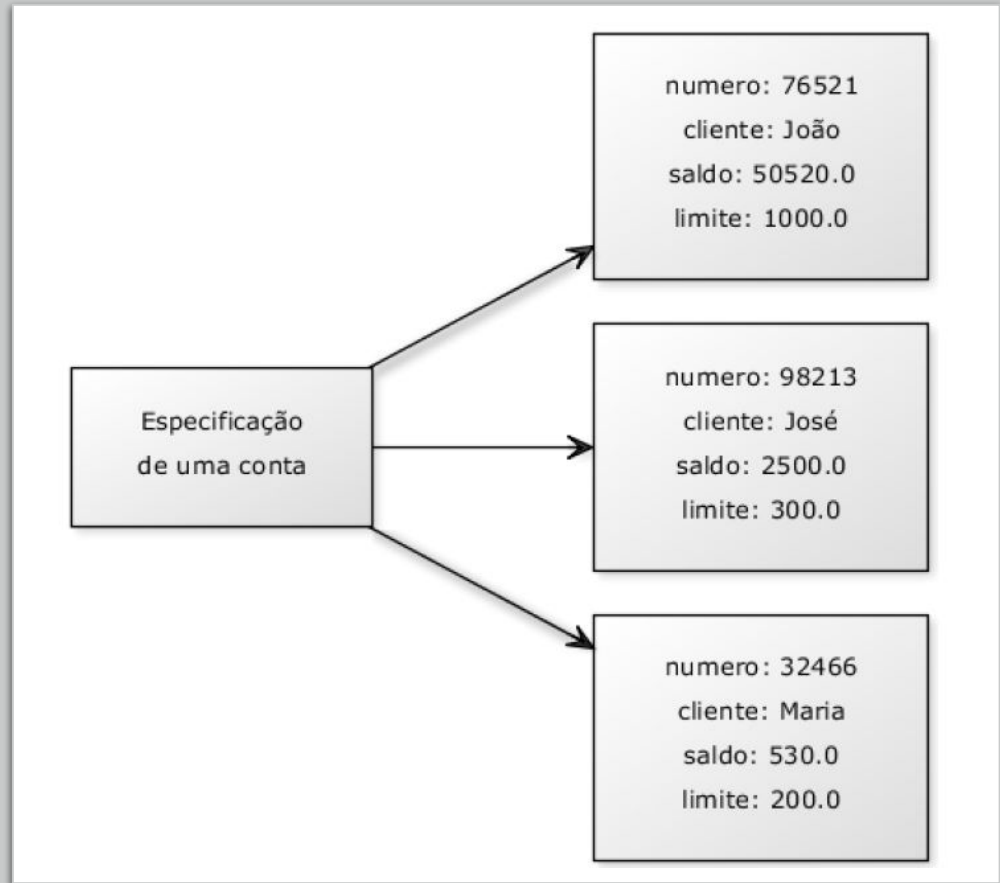
Métodos

Operações que a classe é capaz
de realizar
Normalmente representados
por verbos

Criando um Tipo

- **O que toda conta tem e é importante para nós?**
 - número da conta
 - nome do dono da conta
 - saldo
 - limite
- **O que toda conta faz e é importante para nós?**
 - 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

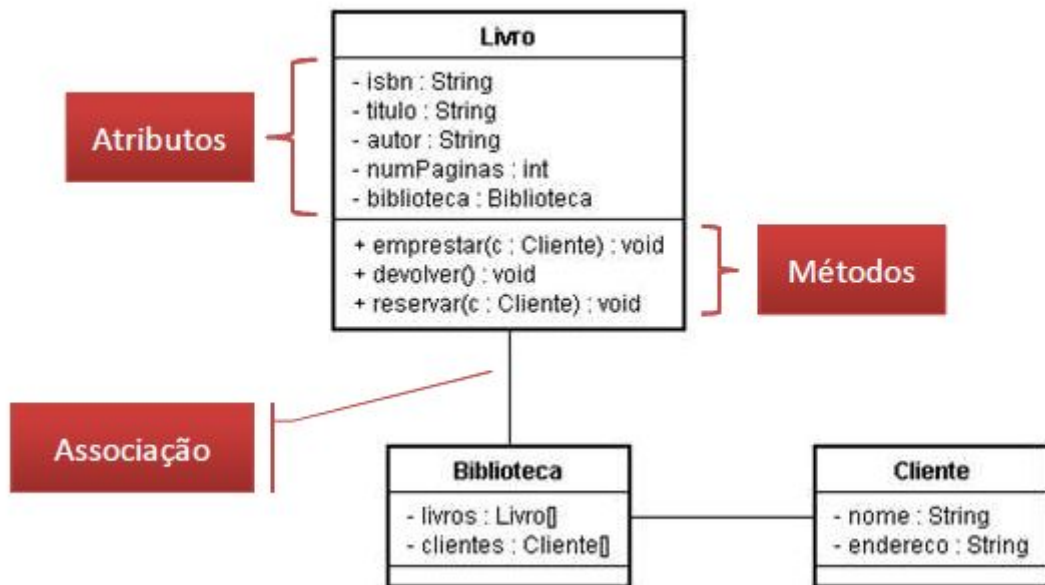
Exemplo de conta com atributos



Relembrando a notação UML

- Unified Modeling Language
- Utilizada para documentar sistemas orientados a objetos
- Composta por diversos diagramas
 - Um deles é o Diagrama de Classes, que mostra as classes do sistema, juntamente com seus respectivos métodos e atributos

A Notação UML: Diagrama de Classes



Classes X Objetos

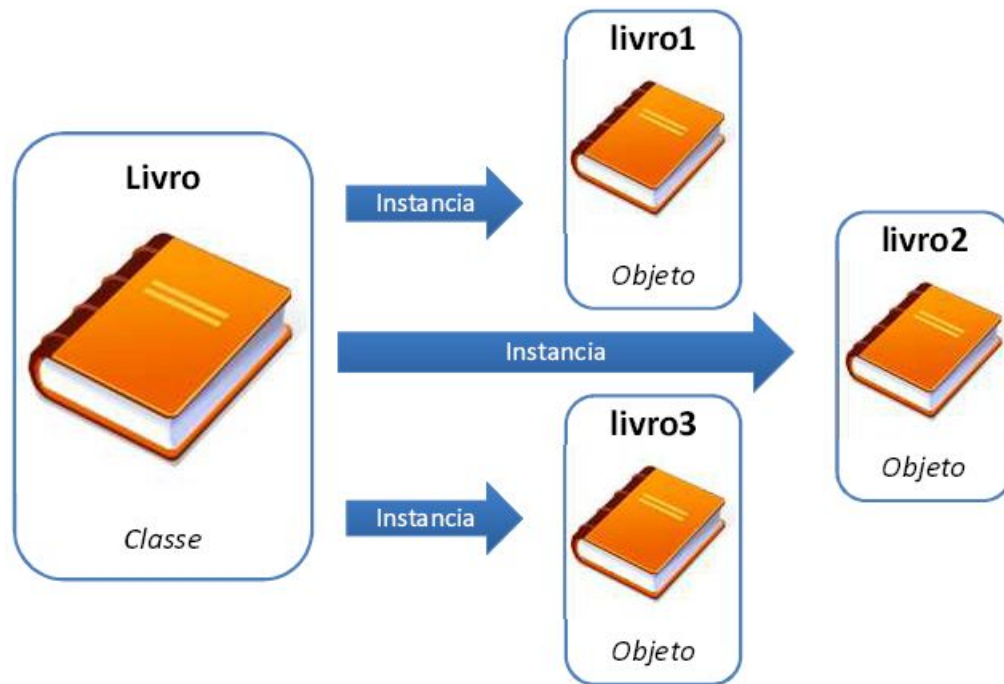
A estrutura do Livro ou Conta a qual nós nos referimos não representa um livro ou Conta propriamente...

Ela é apenas uma estrutura (**classe**) usada como modelo para construir os livros propriamente ditos (**objetos**)

Classe e Objeto são conceitos diferentes!

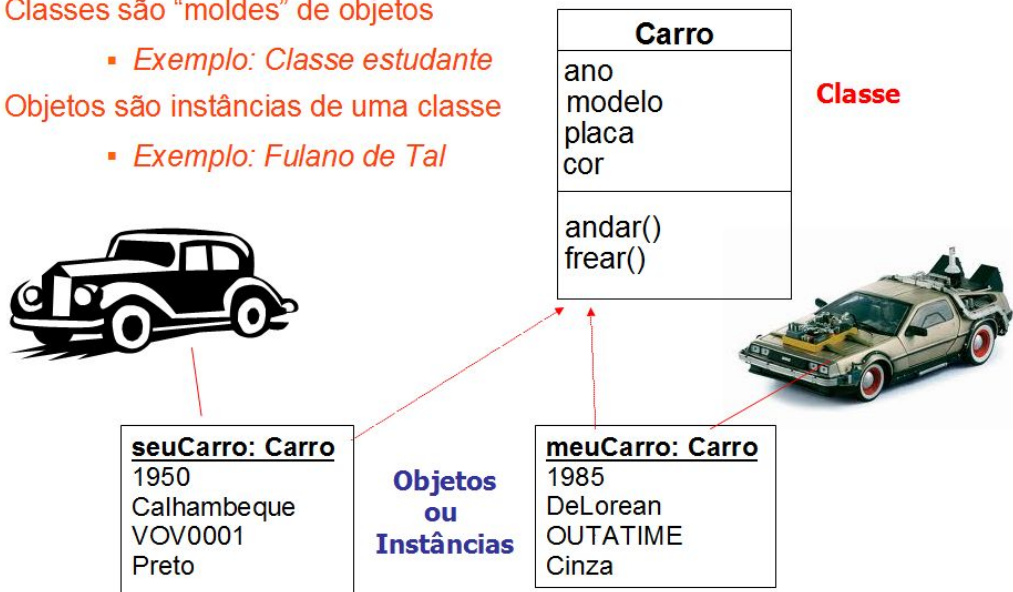
Classes são usadas para **instanciar** objetos

Classes X Objetos



▪ Diferença entre Classes e Objetos

- Classes são “moldes” de objetos
 - Exemplo: Classe estudante
- Objetos são instâncias de uma classe
 - Exemplo: Fulano de Tal

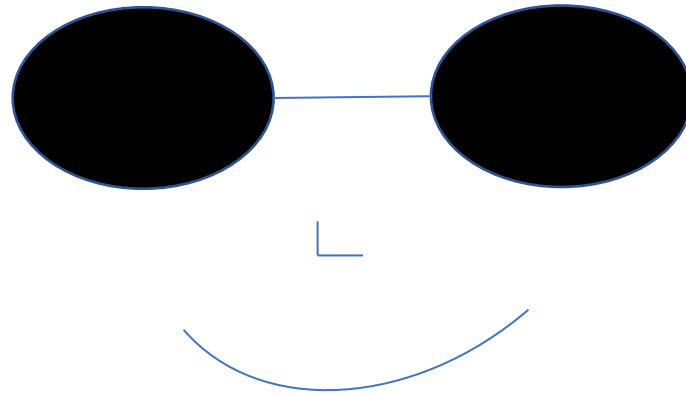


Classes x Objetos

Definições básicas: Objetos

- **No mundo real:** elementos que interagem entre si, onde cada um deles desempenha um papel específico
- **Na modelagem de sistemas:** *Um objeto é uma entidade (instância) que incorpora uma abstração relevante no contexto de uma aplicação*
- Um objeto possui **um estado, características e propriedades (atributos)**, **exibe um comportamento bem definido, expresso por operações (métodos)** e tem **identidade única**

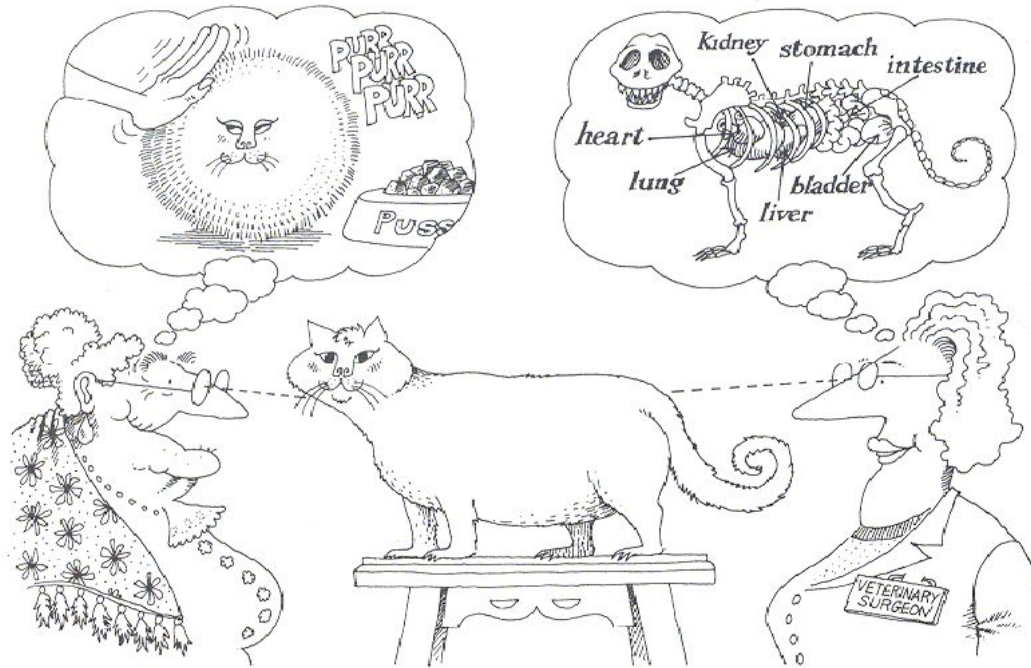
O que você vê?





Abstração em POO

- Podemos ver a figura do slide anterior como a simplificação de algo mais complexo, como um homem usando óculos escuros.
- Em POO, abstração são modelos que representam simplificações de algo, onde só os pontos considerados relevantes são representados
- Enfoque em características essenciais
- Uma abstração é qualquer modelo que **inclui os aspectos relevantes** de alguma coisa, **ao mesmo tempo em que ignora os menos importantes.**



Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

Abstração em POO


```
class Conta {  
    int numero;  
    String dono;  
    double saldo;  
    double limite;  
  
    // ..  
}
```

Uma classe em Java

Quando uma variável é declarada dentro do escopo da classe, é chamada de variável de objeto, ou ATRIBUTO

Um arquivo .java pode ter apenas uma classe declarada como pública dentro dele. O **nome da classe deve ser o nome do arquivo.**

Criando e usando um objeto

Para criar (construir, instanciar) uma Conta, basta usar a palavra chave new.

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

Criando e usando um objeto

- Código anterior cria um objeto do tipo Conta, mas como acessar esse objeto que foi criado?
- Precisamos ter alguma forma de nos referenciarmos a esse objeto. Precisamos de uma variável:

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

Criando e usando um objeto

- Através da variável `minhaConta`, podemos acessar o objeto recém criado para alterar seu dono, seu saldo, etc:

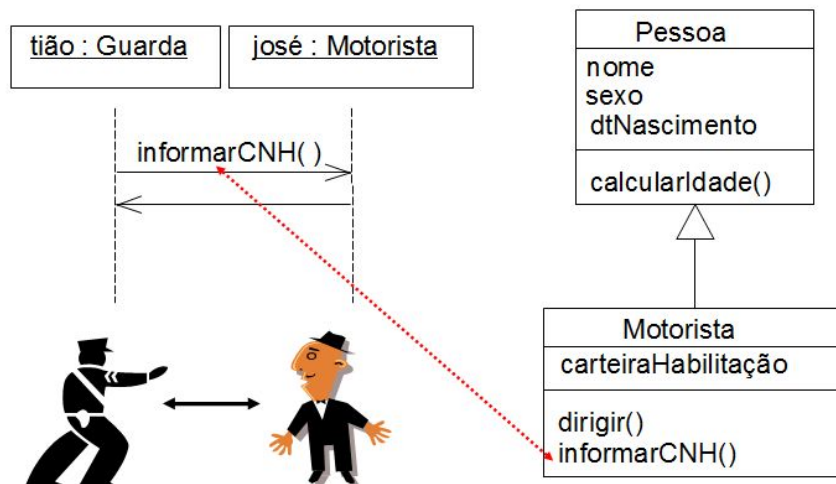
```
class Programa {  
    public static void main(String[] args) {  
        Conta minhaConta;  
        minhaConta = new Conta();  
  
        minhaConta.dono = "Duke";  
        minhaConta.saldo = 1000.0;  
  
        System.out.println("Saldo atual: " + minhaConta.saldo);  
    }  
}
```

Definições Básicas: Mensagens

- Meio de comunicação entre objetos.
- As mensagens são responsáveis pela ativação de todo e qualquer processamento.
- Para que um objeto realize alguma tarefa, é necessário enviar a ele uma mensagem, solicitando a execução de um método específico.

Definições Básicas: Mensagens

- Mensagem (Falbo, 2003)



Métodos

- Dentro da classe, também declararemos **o que cada conta faz** e como isto é feito - os comportamentos que cada classe tem.
- Especificaremos isso dentro da própria classe Conta, e não em um local desatrelado das informações da própria Conta.

Declarando Métodos e Atributos

24

Atributos

Métodos

```
public class Livro {  
    String isbn;  
    int numPaginas;  
    ...  
  
    void emprestar(Cliente c) {  
        ...  
    }  
  
    void devolver() {  
        ...  
    }  
}
```

O Java inicializa os atributos automaticamente

Métodos

- Queremos criar um método que saca uma determinada quantidade e não devolve nenhuma informação para quem acionar esse método:

```
class Conta {  
    double saldo;  
    // ... outros atributos ...  
  
    void saca(double quantidade) {  
        double novoSaldo = this.saldo - quantidade;  
        this.saldo = novoSaldo;  
    }  
}
```

Repare que a conta pode estourar o limite fixado pelo banco.

Métodos

Da mesma forma, temos o método para depositar alguma quantia:

```
class Conta {  
    // ... outros atributos e métodos ...  
  
    void deposita(double quantidade) {  
        this.saldo += quantidade;  
    }  
}
```

Para mandar uma mensagem ao objeto e pedir que ele execute um método, também usamos o ponto. O termo usado para isso é **invocação de método**.

Métodos

```
class TestaAlgunsMetodos {  
    public static void main(String[] args) {  
        // criando a conta  
        Conta minhaConta;  
        minhaConta = new Conta();  
  
        // alterando os valores de minhaConta  
        minhaConta.dono = "Duke";  
        minhaConta.saldo = 1000;  
  
        // saca 200 reais  
        minhaConta.saca(200);  
  
        // deposita 500 reais  
        minhaConta.deposita(500);  
        System.out.println(minhaConta.saldo);  
    }  
}
```

Método com retorno

- Um método sempre tem que definir o que retorna, nem que defina que não há retorno, como nos exemplos anteriores onde estávamos usando o void.

```
class Conta {  
    // ... outros métodos e atributos ...  
  
    boolean saca(double valor) {  
        if (this.saldo < valor) {  
            return false;  
        }  
        else {  
            this.saldo = this.saldo - valor;  
            return true;  
        }  
    }  
}
```

Especificação da Classe Conta

Conta
+numero: int +saldo: double +limite: double +nome: String
+saca(valor: double): boolean +deposita(valor: double)

Exemplo de uso

```
minhaConta.saldo = 1000;
boolean consegui = minhaConta.saca(2000);
if (consegui) {
    System.out.println("Consegui sacar");
} else {
    System.out.println("Não consegui sacar");
}
```

Ou então, posso eliminar a variável temporária, se desejado:

```
minhaConta.saldo = 1000;
if (minhaConta.saca(2000)) {
    System.out.println("Consegui sacar");
} else {
    System.out.println("Não consegui sacar");
}
```

Meu programa pode manter na memória não apenas uma conta, como mais de uma:

```
class TestaDuasContas {  
    public static void main(String[] args) {  
  
        Conta minhaConta;  
        minhaConta = new Conta();  
        minhaConta.saldo = 1000;  
  
        Conta meuSonho;  
        meuSonho = new Conta();  
        meuSonho.saldo = 1500000  
    }  
}
```

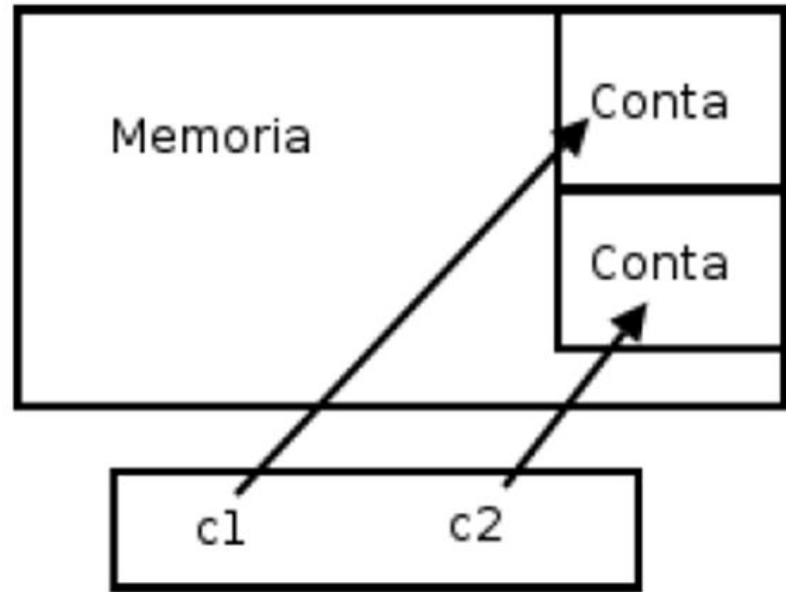
Objetos são acessados por referência

- 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.

```
public static void main(String args[]) {  
    Conta c1;  
    c1 = new Conta();  
  
    Conta c2;  
    c2 = new Conta();  
}
```



```
Conta c1;  
c1 = new Conta();  
  
Conta c2;  
c2 = new Conta();
```



Esse código nos deixa na seguinte situação:

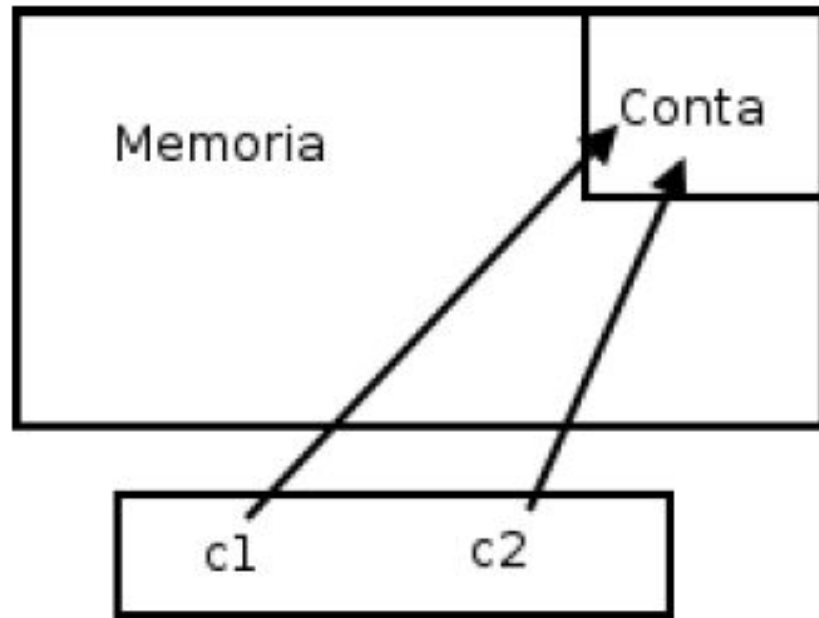
Um outro exemplo:

- Qual é o resultado do código abaixo?
- O que aparece ao rodar?

```
class TestaReferencias {  
    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);  
    }  
}
```

Na memória,
o que
acontece
nesse caso:

```
Conta c1 = new Conta();  
Conta c2 = c1;
```

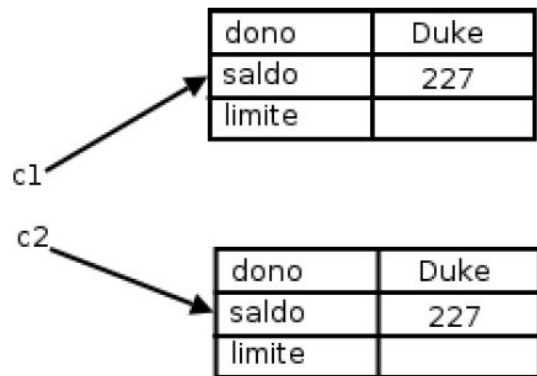


```
public static void main(String args[]) {  
    Conta c1 = new Conta();  
    c1.dono = "Duke";  
    c1.saldo = 227;  
  
    Conta c2 = new Conta();  
    c2.dono = "Duke";  
    c2.saldo = 227;  
  
    if (c1 == c2) {  
        System.out.println("Contas iguais");  
    }  
}
```

**Podemos
ver outra
situação:**

Logo, teríamos :

Para saber se dois objetos têm o mesmo conteúdo, você precisa comparar atributo por atributo. Veremos uma solução mais elegante para isso também.



O método transfere()

E se quisermos ter um método que transfere dinheiro entre duas contas? Podemos ficar tentados a criar um método que recebe dois parâmetros: conta1 e conta2 do tipo Conta. Mas cuidado: assim estamos pensando de maneira procedural. Melhor seria, assim:

```
class Conta {  
  
    // atributos e métodos...  
  
    void transfere(Conta destino, double valor) {  
        this.saldo = this.saldo - valor;  
        destino.saldo = destino.saldo + valor;  
    }  
}
```

```
boolean transfere(Conta destino, double valor) {  
    boolean retirou = this.saca(valor);  
    if (retirou == false) {  
        // não deu pra sacar!  
        return false;  
    }  
    else {  
        destino.deposita(valor);  
        return true;  
    }  
}
```

Para deixar o código mais robusto, poderíamos verificar se a conta possui a quantidade a ser transferida disponível e invocar os métodos **deposita** e **saca** já existentes:

Especificação da Classe Conta ficaria assim:

Conta
+numero: int +saldo: double +limite: double +nome: String
+saca(valor: double): boolean +deposita(valor: double) +transfere(destino: Conta, valor: double): boolean

Esse último código poderia ser escrito com uma sintaxe muito mais sucinta. Como?

- Perceba que o nome deste método poderia ser **transferePara** ao invés de só transfere. A chamada do método fica muito mais natural, é possível ler a frase em português que ela tem um sentido:
 - `conta1.transferePara(conta2, 50);`
- A leitura deste código seria “Conta1 transfere para conta2 50 reais”.



Quando passamos uma Conta como argumento, o que será que acontece na memória? Será que o objeto é clonado?

- **Resposta:** No Java, a passagem de parâmetro funciona como uma simples atribuição como no uso do “=”. Então, esse parâmetro vai copiar o valor da variável do tipo Conta que for passado como argumento. E qual é o valor de uma variável dessas? Seu valor é um endereço, uma referência, nunca um objeto. Por isso não há cópia de objetos aqui.

Continuando com Atributos

- As variáveis do tipo atributo recebem um valor padrão. No caso numérico, valem 0, no caso de boolean, valem false.
- Você também pode dar valores default, como segue:

```
class Conta {  
    int numero = 1234;  
    String dono = "Duke";  
    String cpf = "123.456.789-10";  
    double saldo = 1000;  
    double limite = 1000;  
}
```

Queremos adicionar nome, sobrenome e cpf do cliente dono da conta. Então podemos criar uma nova classe e fazer uma **composição** para evitar o excesso de atributos

```
class Conta {  
    int numero;  
    double saldo;  
    double limite;  
    Cliente titular;  
    // ..  
}
```

```
class Cliente {  
    String nome;  
    String sobrenome;  
    String cpf;  
}
```

E dentro do main da classe:

```
class Teste {  
    public static void main(String[] args) {  
        Conta minhaConta = new Conta();  
        Cliente c = new Cliente();  
        minhaConta.titular = c;  
        // ...  
    }  
}
```

O acesso a informação é dado por meio do “ponto”:

```
minhaConta.titular.nome = "Duke";
```

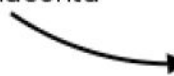
Ex.: A classe Banco usa a classe Conta que tem uma classe Cliente, que tem uma classe Endereco.

Dizemos que esses objetos colaboram ou se relacionam, trocando mensagens entre si.

Mas, e se dentro do meu código eu não desse new em Cliente e tentasse acessá-lo diretamente?

```
class Teste {  
    public static void main(String[] args) {  
        Conta minhaConta = new Conta();  
  
        minhaConta.titular.nome = "Manoel";  
        // ...  
    }  
}
```

minhaConta



numero	
saldo	
limite	
Cliente	null

caso você tente acessar um atributo ou método de alguém que está se referenciando para null, você receberá um erro durante a execução

Com esse código, toda nova Conta criada já terá um novo Cliente associado, sem necessidade de instanciá-lo logo em seguida da instanciação de uma Conta.

```
class Conta {  
    int numero;  
    double saldo;  
    double limite;  
    Cliente titular = new Cliente();    // quando chamarem new Conta,  
                                         //havera um new Cliente para ele.  
}
```

Qual alternativa você deve usar?

Resposta: Depende do caso: para toda nova Conta você precisa de um novo Cliente? É essa pergunta que deve ser respondida. Nesse nosso caso a resposta é não, mas depende do nosso problema.

Exemplo da Fábrica de Carros no Netbeans

Lembre-se de seguir a convenção java, isso é importantíssimo. Isto é, preste atenção nas maiúsculas e minúsculas, seguindo o seguinte exemplo: nomeDeAtributo, nomeDeMetodo, nomeDeVariavel, NomeDeClasse, etc...

Leitura da semana

- Leia o capítulo 4 da apostila fj11 – Orientação a objetos básica:
<http://www.caelum.com.br/apostila-java-orientacao-objetos/>
- Lição Classes e Objetos – Oracle:
<http://docs.oracle.com/javase/tutorial/java/javaOO/index.html>
- Leia o capítulo 3 – Introdução a classes e objetos do livro:
DEITEL, Harvey M. e DEITEL, Paul J. Java - Como Programar, 8ª edição. Pearson. 2010.

Exercício de Fixação



Crie classes que representam as figuras geométricas: Triângulo, Quadrado, Circunferência e Trapézio.



Cada uma destas classes deve ter um método para calcular a sua área, com a seguinte assinatura:

```
double calcularArea()
```



Note que o método **calcularArea()** não recebe parâmetros. Portanto todos os dados necessários devem ser armazenados no objeto da classe em forma de atributos para depois serem utilizados pelo método.



As fórmulas para o cálculo da área são as seguintes (próximo slide):

Exercício de Fixação

Figura	Fórmula	Elementos da Fórmula
Triângulo	$A = \frac{b \times h}{2}$	b = base h = altura
Quadrado	$A = l^2$	l = lado
Circunferência	$A = \pi \times r^2$	r = raio
Trapézio	$A = \frac{(B + b)}{2} \times h$	B = base maior b = base menor h = altura



Atividade para entregar pelo google sala de aula

O objetivo aqui é criar um sistema para gerenciar os funcionários do Banco.

1. Modele um funcionário. Ele deve ter o nome do funcionário, o departamento onde trabalha, seu salário (double), a data de entrada no banco (String) e seu RG (String). Você deve criar alguns métodos de acordo com sua necessidade. Além deles, crie um método recebeAumento que aumenta o salario do funcionário de acordo com o parâmetro passado como argumento. Crie também um método calculaGanhoAnual, que não recebe parâmetro algum, devolvendo o valor do salário multiplicado por 12 .

Atividade para entregar pelo google sala de aula

2. Teste a classe anterior instanciando um objeto da classe Funcionario. Invoque os métodos e inicialize os atributos desta classe.

Dica

Um esboço da classe que possui o main:

```
class TestaFuncionario {  
  
    public static void main(String[] args) {  
        Funcionario f1 = new Funcionario();  
  
        f1.nome = "Hugo";  
        f1.salario = 100;  
        f1.recebeAumento(50);  
  
        System.out.println("salario atual:" + f1.salario);  
        System.out.println("ganho anual:" + f1.calculaGanhoAnual());  
    }  
}
```

Funcionario
+nome: String
+departamento: String
+salario: double
+dataEntrada: String
+rg: String
+estaNaEmpresa: boolean
+bonifica(aumento:double)
+demite()
+calculaGanhoAnual(): double

Atividade para entregar pelo google sala de aula

3. Crie um método mostra(), que não recebe nem devolve parâmetro algum e simplesmente imprime todos os atributos do nosso funcionário. Dessa maneira, você não precisa ficar copiando e colando um monte de System.out.println() para cada mudança e teste que fazer com cada um de seus funcionários, você simplesmente vai fazer:

```
Funcionario f1 = new Funcionario();  
// brincadeiras com f1....  
f1.mostra();
```


Atividade para entregar pelo google sala de aula

4. Construa dois funcionários com o new e compare-os com o ==. E se eles tiverem os mesmos atributos? Para isso você vai precisar criar outra referência:

```
Funcionario f1 = new Funcionario();  
f1.nome = "Danilo";  
f1.salario = 100;  
  
Funcionario f2 = new Funcionario();  
f2.nome = "Danilo";  
f2.salario = 100;  
  
if (f1 == f2) {  
    System.out.println("iguais");  
} else {  
    System.out.println("diferentes");  
}
```

Atividade para entregar pelo google sala de aula

5. Crie duas referências para o **mesmo** funcionário, compare-os com o `==`. Tire suas conclusões. O que acontece com o `if` do exercício anterior?
6. Em vez de utilizar uma `String` para representar a data, crie uma outra classe, chamada `Data`. Ela possui 3 campos `int`, para dia, mês e ano. Faça com que seu `Funcionário` passe a usá-la. (Ex: `Data dataDeEntrada`).
7. Modifique sua Classe `TestaFuncionario` para que você crie uma `Data` e atribua ela ao `Funcionário`. Obs: Não esqueça de instanciar a classe `Data`.
8. Modifique seu método `mostra()` para que ele imprima o valor da `dataDeEntrada` daquele `Funcionário`.
9. Crie um método na Classe `Data` que devolva o valor formatado da data (“dia/mês/ano”).