




Java SE e Orientação a Objetos



Conteúdo



1. Introdução ao Java
2. Variáveis em Java: primitivas e de referência
3. Estruturas de Controle
4. Trabalhando com estruturas de arrays em Java
5. **Orientação a Objetos I: Objetos e Classes** 
6. Orientação a Objetos II: Herança e Polimorfismo
7. E as IDE's ??? NetBeans, Eclipse, ...





5. OO I - Objetos e Classes

- Enfim, vamos ao que interessa: programação orientada a objetos em Java...
- O Java foi, desde seu projeto, concebido para desenvolver sistemas dentro do paradigma de orientação a objetos
- A programação orientada a objetos (POO) tem os objetos como elementos básicos de seus programas, sendo eles, uma representação computacional o mais natural possível de alguma entidade existente no mundo real dentro do contexto do problema, representando assim sua solução no mundo computacional



5. OO I - Objetos e Classes



- Os objetos possuem propriedades (também chamadas de características) e comportamentos
- Por exemplo, um objeto pessoa possui as propriedades idade, peso, cor dos olhos, pressão sanguínea, nome, cpf, titulação, etc...
- Além disso, pessoa também possui comportamentos como comer, dormir, chutar, roubar, virar, deitar, etc...
- Essas nomenclaturas são referentes à abstração do objeto no mundo real



5. OO I - Objetos e Classes



- No mundo computacional (no código), essas abstrações são modeladas como atributos e métodos
- Portanto, objeto é uma coleção de atributos e métodos relacionados
- A estrutura fundamental da POO é a classe, cuja finalidade é modelar um objeto do mundo real
- A classe seria o metadado do objeto, o que ele contém e o que ele faz



5. OO I - Objetos e Classes

- A classe é a receita do bolo, o objeto é o bolo que saiu do forno, você pode fazer quantos quiser, com recheios diferentes (atributos diferentes), mas mesmo aqueles com atributos iguais nunca são exatamente os mesmos (OID)
- Outra metáfora é considerarmos a classe como a planta de uma casa num condomínio, e a partir desta, todas as casas são feitas de forma semelhante, mas cada uma tem sua peculiaridade
- O objeto é uma instância de uma classe



5. OO I - Objetos e Classes



- Um dos principais benefícios da utilização do conceito de classes é a reutilização de código, pois a partir de uma mesma receita e uma mesma forma você fabrica muitos bolos
- Além disso, técnicas como encapsulamento e polimorfismo (que veremos mais tarde) melhoram a qualidade do projeto, inserindo características desejáveis como alta coesão, baixo acoplamento, manutenibilidade, entre muitas outras



5. OO I - Objetos e Classes



- Uma classe é composta por:
 - Atributos de objeto
 - Métodos de objeto
 - Atributos de classe (estático, p.ex. um contador)
 - Métodos de classe (estático)



5. OO I - Objetos e Classes



- Todo sistema começa modelando seus objetos de negócio, do mundo real, projetando-o; essa é a nossa classe
- Depois temos um programa que utiliza, constrói esse objeto a partir de seu projeto, esse é o objeto
- Então, um programa instancia um ou mais objetos a partir de uma classe existente
- Você não pode comer a receita de um bolo, mas pode fazê-lo a partir dela



5. OO I - Objetos e Classes

- Vamos começar a programar e para melhor entendimento, vamos modelar um sistema bancário nos nossos exemplos de OO
- Qual a principal entidade de um sistema bancário?
- Uma conta...
- O que toda conta tem?
- O que toda conta faz?



5. OO I - Objetos e Classes

- Quando definimos o que toda conta tem, estamos projetando seus atributos
- Vamos considerar que uma conta tem:
 - O número da conta
 - O nome do cliente que associado à conta
 - Um saldo
 - Um limite

```
1 class Conta {  
2     int numero;  
3     String nome;  
4     double saldo;  
5     double limite;  
6 }  
7  
8
```



5. OO I - Objetos e Classes



- Quando definimos o que toda conta faz, estamos projetando seus métodos
- Vamos considerar que toda conta faz:
 - Saca uma determinada quantidade x
 - Deposita uma quantidade x
 - Imprime o nome do dono da conta
 - Retorna o saldo atual
 - Transfere uma quantidade x para uma conta y
 - Retorna o tipo de conta



5. OO I - Objetos e Classes



- Vamos começar implementando um método que saca e outro que deposita...
- Para o cliente sacar eu preciso definir um valor, bem como para depositar, uma vez feito isso a operação é executada e não precisa mais nada por enquanto



5. OO I - Objetos e Classes

- Assinatura de métodos
- Argumento ou parâmetros do método
- Retorno void
- Escopo de variáveis
- Variável auxiliar e this

```
1 class Conta {  
2     int numero;  
3     String nome;  
4     double saldo;  
5     double limite;  
6  
7     void saca(double valor) {  
8         double saldoAlterado = saldo - valor;  
9         saldo = saldoAlterado;  
10    }  
11  
12    void deposita(double valor) {  
13        this.saldo += valor;  
14    }  
15 }  
16
```



5. OO I - Objetos e Classes

- Não esqueçam de um detalhe, este é apenas o projeto de uma conta, precisamos de um programa que vai utilizá-la, ou instanciá-la
- Para instanciar uma conta, precisamos de um espaço em memória que caiba conta, então esse programa precisa ter uma variável do tipo conta (declaração), e a partir dessa variável precisamos construir conta (instanciar), que é feito com o operador *new*
- Criada uma conta, podemos acessar seus atributos e métodos através do operador "."



5. OO I - Objetos e Classes

- Vamos acessar e imprimir seus atributos

```
1  class TestaConta {
2      public static void main(String[] args) {
3          Conta c1;
4          c1 = new Conta();
5          //A declaração e instanciação pode ser na mesma linha
6          // Conta c1 = new Conta;
7
8          c1.numero = 001;
9          c1.nome = "Gama";
10         c1.saldo = 100.0;
11         c1.limite = 10.0;
12
13         System.out.println("Numero da conta: "+c1.numero);
14         System.out.println("Titular: "+c1.nome);
15         System.out.println("Saldo atual: "+c1.saldo);
16         System.out.println("Limite: "+c1.limite);
17     }
18 }
19
20
```



5. OO I - Objetos e Classes

- Vamos agora testar para ver se conseguimos sacar e depositar...

```
1 class TestaConta {
2     public static void main(String[] args) {
3         Conta c1;
4         c1 = new Conta();
5         //A declaração e instanciação pode ser na mesma linha
6         // Conta c1 = new Conta;
7
8         c1.numero = 001;
9         c1.nome = "Gama";
10        c1.saldo = 100.0;
11        c1.limite = 10.0;
12
13        System.out.println("Numero da conta: "+c1.numero);
14        System.out.println("Titular: "+c1.nome);
15        System.out.println("Saldo atual: "+c1.saldo);
16        System.out.println("Limite: "+c1.limite);
17
18        c1.saca(10);
19        System.out.println("Saldo atual: "+c1.saldo);
20
21        c1.deposita(1000);
22        System.out.println("Saldo atual: "+c1.saldo);
23    }
24 }
25
26
27
```



5. OO I - Objetos e Classes



- Temos um problema de regra de negócio...o que acontece se eu sacar um valor maior que o saldo mais o limite?
- Do jeito que está, nada! Devemos consertar isso
- Vamos utilizar métodos com valor de retorno e a declaração return



5. OO I - Objetos e Classes

- Nossa nova classe conta será assim...

```
1 class Conta2 {  
2     int numero;  
3     String nome;  
4     double saldo;  
5     double limite;  
6  
7     boolean saca(double valor) {  
8         if ((this.saldo+this.limite) < valor) {  
9             return false;  
10        } else {  
11            this.saldo -= valor;  
12            return true;  
13        }  
14    }  
15  
16    void deposita(double valor) {  
17        this.saldo += valor;  
18    }  
19 }  
20
```



5. OO I - Objetos e Classes

- E para executarmos os testes, podemos fazer dessa maneira...

```
1 class TestaConta2 {
2     public static void main(String[] args) {
3         Conta2 c2;
4         c2 = new Conta2();
5         //A declaração e instanciação pode ser na mesma linha
6         // Conta c2 = new Conta;
7
8         c2.numero = 001;
9         c2.nome = "Gama";
10        c2.saldo = 100.0;
11        c2.limite = 10.0;
12
13        System.out.println("Numero da conta: "+c2.numero);
14        System.out.println("Titular: "+c2.nome);
15        System.out.println("Saldo atual: "+c2.saldo);
16        System.out.println("Limite: "+c2.limite);
17
18        int saque = 110;
19        boolean resultado = c2.saca(saque);
20        if (resultado) {
21            System.out.println("Voce sacou R$"+saque+" e seu saldo atual eh R$"+c2.saldo);
22        } else {
23            System.out.println("Saque nao efetuado, valor acima do possivel");
24        }
25    }
26 }
27
28
```



5. OO I - Objetos e Classes



- Podemos instanciar mais de uma conta no mesmo programa??? Faça o teste...
- Como funciona na memória???
- Já vimos que num tipo primitivo, a variável tem acesso por valor, enquanto nos objetos a variável tem acesso por referência
- Ou seja, enquanto no primeiro caso, a variável guarda o próprio valor, nos objetos guardamos uma referência para o objeto
- Complicado??? Mais tarde veremos como isso é importante e que valor agrega à POO



5. OO I - Objetos e Classes



- Por enquanto, vamos apenas exemplificar como funciona

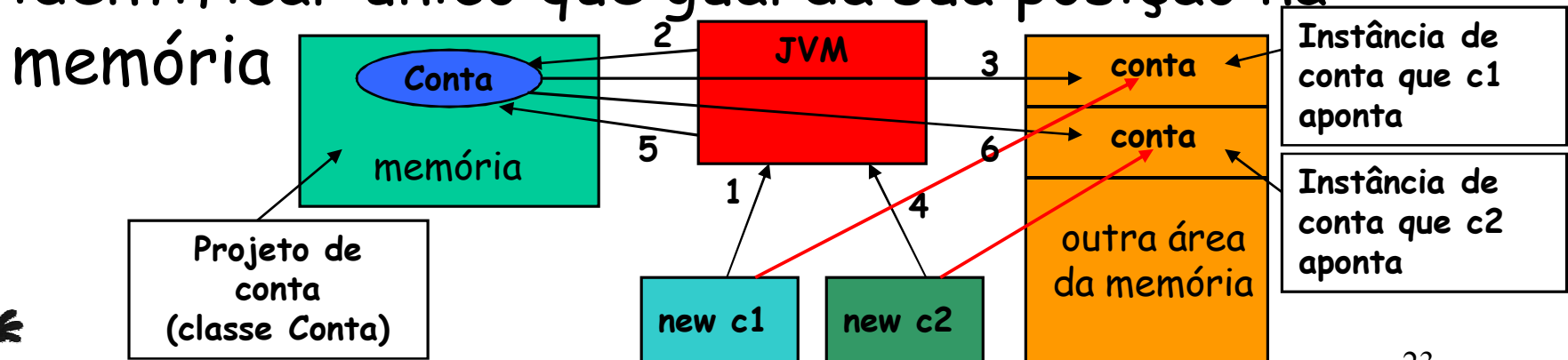
```
1 class TestaConta3 {  
2     public static void main(String[] args) {  
3         Conta c1;  
4         c1 = new Conta();  
5  
6         Conta c2;  
7         c2 = new Conta();  
8  
9         ...  
}
```

- No código acima, não criamos um objeto c1 e outro objeto c2, ambos do tipo conta, mas duas referências para dois endereços de memória diferentes que suportam objetos conta



5. OO I - Objetos e Classes

- Por força do hábito e para simplificar, com o tempo acostumamos a dizer "instanciar um objeto c1 do tipo conta", por exemplo
- Tudo bem, o importante é você saber o conceito correto e o que acontece realmente na memória, onde cada objeto instanciado (*new*) tem um identificador único que guarda sua posição na memória



5. OO I - Objetos e Classes

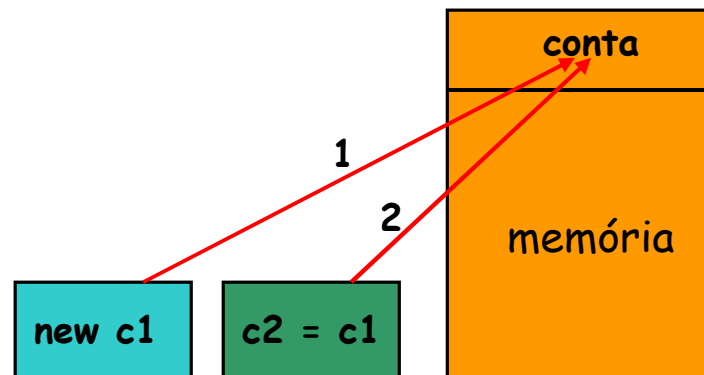


- Temos uma estrutura idêntica à de um ponteiro, mas todo o gerenciamento do ciclo de vida destes objetos em relação à memória (alocação, desalocação, etc) é feita por uma máquina que está atenta a muito mais detalhes do que estaria mesmo um bom programador, a JVM
- Um exemplo interessante e importante para entendermos alguns conceitos no futuro, é o que acontece quando atribuímos um objeto a um outro objeto já instanciado



5. OO I - Objetos e Classes

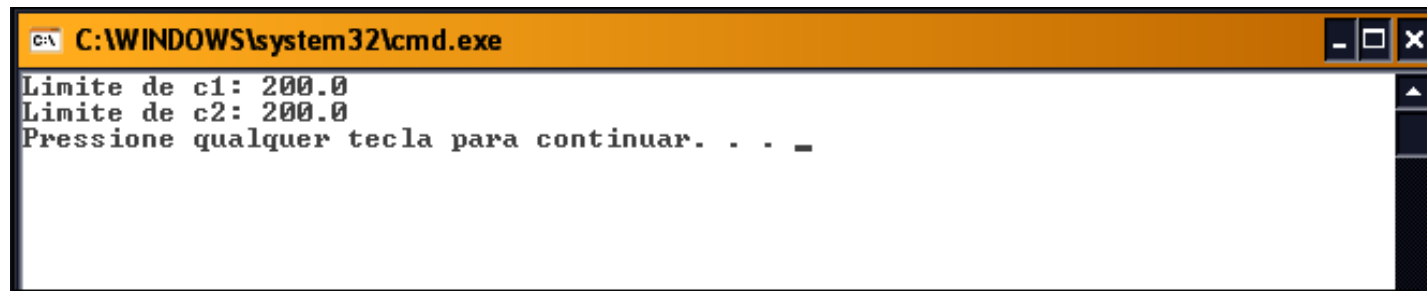
- Como sempre, o operador de atribuição "=" copia o valor de uma variável, mas a variável no caso de objeto não guarda o objeto, mas sim uma referência que aponta para ele, ou seja, a referência é copiada
- Conclusão, os dois objetos possuem a mesma referência, ou seja, eles apontam para o mesmo espaço de memória
- É o mesmo objeto!



5. OO I - Objetos e Classes

- O exemplo anterior pode ser exemplificado pelo código abaixo...

```
1 class TestaConta4 {  
2     public static void main(String[] args) {  
3         Conta c1;  
4         c1 = new Conta();  
5         c1.limite = 100;  
6  
7         Conta c2;  
8         c2 = c1;  
9         c2.limite = 200;  
10  
11         System.out.println("Limite de c1: "+c1.limite);  
12  
13         System.out.println("Limite de c2: "+c2.limite);  
14     }  
15 }  
16
```



```
C:\WINDOWS\system32\cmd.exe  
Limite de c1: 200.0  
Limite de c2: 200.0  
Pressione qualquer tecla para continuar. . . _
```



5. OO I - Objetos e Classes

- O que acontecerá no exemplo abaixo?

```
1  class TestaConta5 {  
2      public static void main(String[] args) {  
3          Conta c1;  
4          c1 = new Conta();  
5          c1.nome = "Gama";  
6  
7          Conta c2;  
8          c2 = new Conta();  
9          c2.nome = "Gama";  
10  
11         if (c1 == c2) {  
12             System.out.println("As contas sao iguais");  
13         } else {  
14             System.out.println("As contas sao diferentes");  
15         }  
16     }  
17 }  
18
```



5. OO I - Objetos e Classes



- Passando objetos como argumento de um método
- Como você faria uma operação de transferência?
O que seria necessário? Como seria o código?
- Existe alguma pré-condição?
- Implemente esse código
- Vamos ver um pouco de UML???



5. OO I - Objetos e Classes

- Vamos continuar nosso sistema bancário...
- O sistema precisa também de informações cadastrais sobre a conta, ou melhor, sobre o dono da conta
- Não faz sentido essa informação ficar na própria conta, até porque os dados do cliente podem estar dissociados da conta
- Vamos então abstrair essa decomposição e transformar esses dados em uma classe Cliente



5. OO I - Objetos e Classes

```
1 class Cliente {  
2     String nome;  
3     String sobreNome;  
4     String cpf;  
5 }
```

```
1 class ContaTransf2 {  
2     int numero;  
3     double saldo;  
4     double limite;  
5     //String nome;  
6     Cliente cliente;  
7 }
```

- Agora a classe ContaTransf2 tem um cliente
- Será que funciona? Vamos testar...

```
1 class TestaContaCliente{  
2     public static void main(String args[]){  
3         ContaTransf2 c1 = new ContaTransf2();  
4         c1.cliente.nome = "Mike";  
5         System.out.println(c1.cliente.nome);  
6     }  
7 }  
8 }
```



5. OO I - Objetos e Classes

- E agora dessa maneira abaixo

```
1  class TestaContaCliente2{
2      public static void main(String args[]){
3          ContaTransf2 c1 = new ContaTransf2();
4          Cliente cliente1 = new Cliente();
5          c1.cliente = cliente1;
6          c1.cliente.nome = "Mike";
7          System.out.println(c1.cliente.nome);
8      }
9  }
10 }
```

- Sempre que os atributos são declarados e não são inicializados através do operador de atribuição eles assumem valores padrão: false para boolean, 0 (zero) para números e *null* para objetos, por isso tivemos problema no exemplo anterior



5. OO I - Objetos e Classes

- Uma solução que pode ser utilizada dependendo da regra de negócio e que pode reduzir a quantidade de problemas como a aplicação esquecer de instanciar o objeto de negócio cliente é a agregação (relação "toda instância de a tem um b")

```
1 class ContaTransf3 {  
2     int numero;  
3     double saldo;  
4     double limite;  
5     //String nome;  
6     Cliente cliente = new Cliente();
```

Instanciamos
Cliente dentro de
ContaTransf3



```
1 class TestaContaCliente3{  
2     public static void main(String args[]){  
3         ContaTransf3 c1 = new ContaTransf3();  
4         //Cliente cliente1 = new Cliente();  
5         //c1.cliente = cliente1;  
6         c1.cliente.nome = "Mike";  
7         System.out.println(c1.cliente.nome);  
8     }  
9 }  
10 }
```

E o mesmo código
que dava erro no
primeiro exemplo
agora funciona

5. OO I - Objetos e Classes

- Vamos ver o UML deles?
- Continuando nosso sistema bancário...
- Além de conta e cliente, nosso banco também possui funcionários, vamos implementá-lo e testá-lo

```
1 class Funcionario{
2     String nome;
3     String cpf;
4     String departamento;
5     String dataAdmissao;
6     double salario;
7     String status;
8
9     void bonificacao(double bonus){
10         salario +=bonus;
11     }
12 }
```

```
1 class TestaFuncionario {
2     public static void main(String args[]){
3         Funcionario f1 = new Funcionario();
4         f1.nome = "Mark";
5         f1.salario = 500.0;
6         f1.status = "ativo";
7         System.out.println(f1.salario);
8         f1.bonificacao(50);
9         System.out.println(f1.salario);
10     }
11 }
```



5. OO I - Objetos e Classes

- Até agora, o que conseguimos enxergar nessa forma de programar, através de classes, é o poder de reusabilidade e manutenibilidade, veremos muito mais...
- Da mesma forma que fazemos com variáveis primitivas, com variáveis do tipo classe, ou seja, objetos, declaramos com um identificador mas na sua inicialização somos obrigados a adicionar o operador *new* que faz a criação (ou instanciação) e a inicialização com valores default ou não através do construtor



5. OO I - Objetos e Classes



- Assim, você primeiro declara o tipo e identificador do objeto e depois ele é criado ou instanciado através do operador *new*
- O operador *new* aloca memória para um objeto do mesmo tipo da classe e retorna uma referência
- Ao ser criado o objeto, o método construtor é invocado, ele possui o mesmo nome da classe e pode ser inicializado com valores default ou não



5. OO I - Objetos e Classes

- Outro conceito poderoso e que mostra outras vantagens da orientação a objetos é o **encapsulamento** e para explicá-lo temos que falar de **diretivas de visibilidade** ou **modificadores de acesso**
- Um problema de segurança ou de inconsistência do sistema pode ocorrer com muita frequência caso suas variáveis estejam livres para serem sobrescritas diretamente



5. OO I - Objetos e Classes



- No nosso sistema bancário, por exemplo, isso poderia ser um problema se uma aplicação pudesse escrever diretamente na variável saldo sem passar por um método que valide a regra de negócio referente ao limite da conta do cliente
- Assim temos que protegê-la, ou encapsulá-la, e isso é feito limitando seu acesso mudando sua diretiva de visibilidade ou seu modificador



5. OO I - Objetos e Classes



- Uma forma de proteger nosso sistema, seria garantir que a variável saldo fosse modificada somente através de métodos seguros, esses sim públicos, ficando ela protegida ou privada, assim seu modificador passa a ser *private*
- Então temos a diretiva *public* e *private* por enquanto, que podem ser aplicadas tanto a atributos da classe quanto a métodos, sendo os recursos *public* acessíveis somente a própria classe e os *private* acessíveis a qualquer classe externa



5. OO I - Objetos e Classes



- Esse é o espírito do encapsulamento, principalmente já que falamos desde o começo que o que importa nas nossas classes são as interfaces e não a implementação (*Eric Gamma: Design Patterns*), pois isso reduz o acoplamento, então nada melhor do que disponibilizarmos as interfaces (métodos públicos) e protegermos os objetos da implementação (atributos privados)
- Por isso os métodos públicos são chamados interfaces de uma classe



5. OO I - Objetos e Classes



- Numa classe, o importante é o que ela faz e não como faz
- Muito mais que isso, em Java isso se torna um padrão de codificação chamado Java Beans, onde um objeto do negócio tem seus atributos privados e os acessos de leitura e escrita a esses atributos são feitos através de métodos públicos *get* e *set* (use-os somente quando necessário de acordo com as regras do negócio)



5. OO I - Objetos e Classes



- Sempre que instanciamos um objeto a partir de uma classe, utilizamos o operador *new*
- Esse operador constrói um objeto, por isso ele executa o construtor da classe
- O construtor é um conjunto de instruções com o mesmo nome da classe e pode ser implícito (construtor default) ou explícito dependendo da sua necessidade
- Até agora só utilizamos o default, que é vazio, por isso não o colocamos na classe conta



5. OO I - Objetos e Classes



- Vamos testar um primeiro construtor em conta

```
1 class ContaTransfConstrutor {
2     int numero;
3     double saldo;
4     double limite;
5     //String nome;
6     Cliente cliente = new Cliente();
7
8     ContaTransfConstrutor(){
9         System.out.println("Teste da classe conta com construtor");
10    }
11
12    boolean saca(double valor) {
```

```
1 class TestaContaClienteConstrutor{
2     public static void main(String args[]){
3         ContaTransfConstrutor cl = new ContaTransfConstrutor();
4         //Cliente clientel = new Cliente();
5         //cl.cliente = clientel;
6         cl.cliente.nome = "Mike";
7         System.out.println(cl.cliente.nome);
8
9     }
10 }
```



5. OO I - Objetos e Classes



- Esse foi só um exemplo de um construtor, mas para que serve um construtor?
- A partir do momento que você cria um construtor, o default (que é vazio) deixa de existir, sendo assim, eu posso obrigar a aplicação que utiliza a classe a me passar algum dado ou argumento obrigatoriamente, inicializando o meu objeto com algum valor explícito



5. OO I - Objetos e Classes

- Poderia, por exemplo, obrigar o sistema a somente criar uma conta se fosse passado o nome do cliente e cpf

```
1 class ContaTransfConstrutor2 {
2     int numero;
3     double saldo;
4     double limite;
5     //String nome;
6     Cliente cliente = new Cliente();
7
8     ContaTransfConstrutor2(String nome, String cpf){
9         this.cliente.nome = nome;
10        this.cliente.cpf = cpf;
11    }
12
13
14    | boolean saca(double valor) {
```

```
1 class TestaContaClienteConstrutor2{
2     public static void main(String args[]){
3         ContaTransfConstrutor2 c1 = new ContaTransfConstrutor2("Mike", "111.111.111-11");
4         System.out.println(c1.cliente.nome);
5         System.out.println(c1.cliente.cpf);
6     }
7 }
```



5. OO I - Objetos e Classes



- Podemos criar quantos construtores quisermos se isso for interessante para nossa aplicação, apenas lembrando que se o criarmos pelo menos um deles é obrigatório vir na aplicação que utiliza a classe, mesmo que seja um vazio, pois o default deixa de existir quando criamos um
- Além disso, o construtor evita que a aplicação utilize excesso de métodos set, facilitando a programação



5. OO I - Objetos e Classes



- Um outro artifício existente na programação Java que podemos precisar são os atributos de classe
- Funcionam como um contador global, pertencente a classe e não ao objeto
- Imagine por exemplo que queremos saber a quantidade total de contas, funcionários, clientes ou qualquer outro objeto do sistema
- Se criamos um contador dentro do construtor, que sempre incrementa ao instanciar um objeto vai funcionar?



5. OO I - Objetos e Classes

- Essa variável precisa ser única no sistema e isso em Java é feito quando declaramos um atributo como estático e seu acesso é feito através do nome da classe ao invés do nome do objeto

```
1 class ContaTransfConstrutor2 {  
2     private static int contadorContas;  
3     int numero;  
4     double saldo;  
5     double limite;  
6     //String nome;  
7     Cliente cliente = new Cliente();  
8  
9     ContaTransfConstrutor2(String nome, String cpf){  
10         this.cliente.nome = nome;  
11         this.cliente.cpf = cpf;  
12         ContaTransfConstrutor2.contadorContas++;  
13     }  
14  
15  
16     public int getContadorContas(){  
17         return ContaTransfConstrutor2.contadorContas;  
18     }  
19  
20     boolean saca(double valor) {
```



5. OO I - Objetos e Classes

- Vamos ver se funcionou...

```
1 class TestaContaClienteConstrutor2{
2     public static void main(String args[]){
3         ContaTransfConstrutor2 c1 = new ContaTransfConstrutor2("McCain", "111.111.111-11");
4         System.out.println(c1.cliente.nome);
5         System.out.println(c1.cliente.cpf);
6         int contador = c1.getContadorContas();
7         System.out.println(contador);
8         ContaTransfConstrutor2 c2 = new ContaTransfConstrutor2("Obama", "222.111.111-12");
9         System.out.println(c2.cliente.nome);
10        System.out.println(c2.cliente.cpf);
11        contador = c2.getContadorContas();
12        System.out.println(contador);
13    }
14 }
```

- Tudo bem, mas todas as vezes que quisermos saber o contador temos que instanciar um objeto? E se não quisermos um objeto?



5. OO I - Objetos e Classes

- Para resolvermos esse problema temos o método estático, um dos conceitos mais importantes para utilização nos padrões *factory*

```
1 class ContaTransfConstrutor3 {
2     private static int contadorContas;
3     int numero;
4     double saldo;
5     double limite;
6     //String nome;
7     Cliente cliente = new Cliente();
8
9     ContaTransfConstrutor3(String nome, String cpf){
10         this.cliente.nome = nome;
11         this.cliente.cpf = cpf;
12         ContaTransfConstrutor2.contadorContas++;
13
14     }
15
16     public static int getContadorContas(){
17         return ContaTransfConstrutor3.contadorContas;
18     }
19
20     boolean saca(double valor) {
```



5. OO I - Objetos e Classes



- E para acessarmos o contador, o fazemos direto através da classe, sem precisarmos de um objeto

```
1 class TestaContaClienteConstrutor3{
2     public static void main(String args[]){
3         ContaTransfConstrutor3 c1 = new ContaTransfConstrutor3("McCain", "111.111.111-11")
4         System.out.println(c1.cliente.nome);
5         System.out.println(c1.cliente.cpf);
6         //int contador = c1.getContadorContas();
7         //System.out.println(contador);
8         ContaTransfConstrutor3 c2 = new ContaTransfConstrutor3("Obama", "222.111.111-12");
9         System.out.println(c2.cliente.nome);
10        System.out.println(c2.cliente.cpf);
11        //contador = c2.getContadorContas();
12        int contador = ContaTransfConstrutor3.getContadorContas();
13        System.out.println(contador);
14    }
15 }
```

