**01Composição de Objetos**

**Transcrição**

Demos pouca atenção para o atributo titular do tipo String. Podemos utilizar esse atributo dentro da classe TestaMetodo sem dificuldades.

public class TestaMetodo {

System.out.println(contaDaMarcela.saldo);

System.out.println(contaDoPaulo.saldo);

contaDoPaulo.titular = "paulo silveira";

System.out.println(contaDoPaulo.titular);

}Copiar código

Foi dito que Java zera o valor dos atributos quando acionamos a palavra-chave new. Agora compreenderemos melhor o que acontece no caso dos tipos não numéricos como String.

Suponhamos que a conta bancária do **ByteBank** além de guardar as informações de saldo, agência, número e o nome do titular, também guardará o *CPF* do titular e sua *profissão*. Uma possível solução seria incluir esses novos atributos à classe Conta.

public class Conta {

double saldo;

int agencia;

int numero;

String titular;

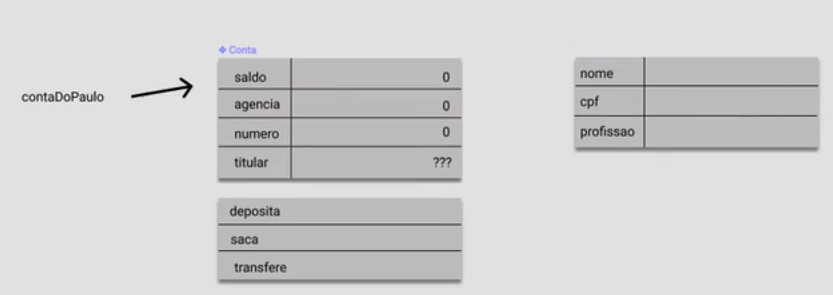
String cpf;

String profissão;

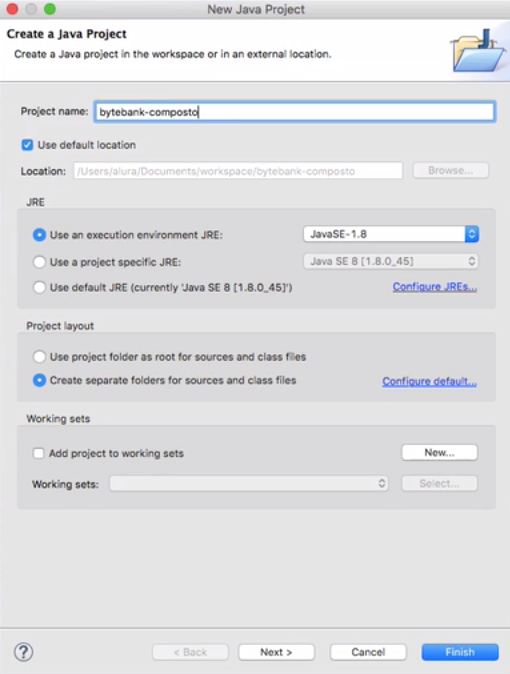
<!-- ... -->Copiar código

Percebam que a classe começa a ficar "inchada" e com muitas informações que não dizem respeito exatamente a uma conta bancária, como a profissão do titular e seu CPF.

Para resolver essa questão, podemos criar um tipo novo chamado Cliente, que terá os atributos de nome, cpf e profissão.

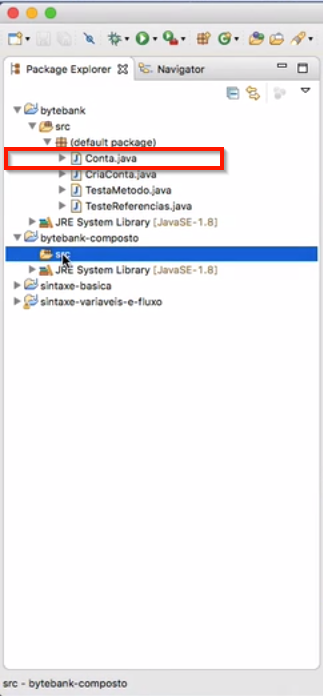


Para separar bem a organização das contas bancárias e dos titulares, criaremos um novo projeto Java intitulado bytebank-composto.



Na área do *Package Explorer*, selecionaremos a classe Conta.Java e a copiaremos utilizando o atalho "Ctrl + C", e a colaremos na pasta src via atalho "Ctrl + V".

**Atenção!** Quando temos duas classes com o mesmo nome, ainda que em projetos diferentes, podemos nos confundir e fazer edições na classe errada. O Eclipse disponibiliza um recurso para que se feche o projeto que não está sendo trabalhado. Na área *Package Navigator*, basta clicar com o botão direito sobre o nome do projeto e selecionar a opção "Close Project".



No novo projeto, criaremos uma nova classe intitulada Cliente. Tal classe conterá os atributos de nome, cpf e profissao como já foi dito.

public class Cliente {

String nome;

String cpf;

String profissao;

}Copiar código

Iremos estabelecer uma relação entre Contae Cliente, ou seja, toda Conta faz uma referência a um Cliente.

Não é mais interessante para o nosso projeto que o atributo titular seja uma String, e sim que faça referência a um cliente específico.

Criaremos uma nova classe chamada TestaBanco. Faremos um main e criaremos uma referência para um cliente que chamaremos de paulo.

public class TestaBanco {

public static void main(String[] args) {

Cliente paulo = new Cliente();

}

}Copiar código

Vamos popular este objeto, criando seus atributos.

public class TestaBanco {

public static void main(String[] args) {

Cliente paulo = new Cliente();

paulo.nome = "Paulo Silveira";

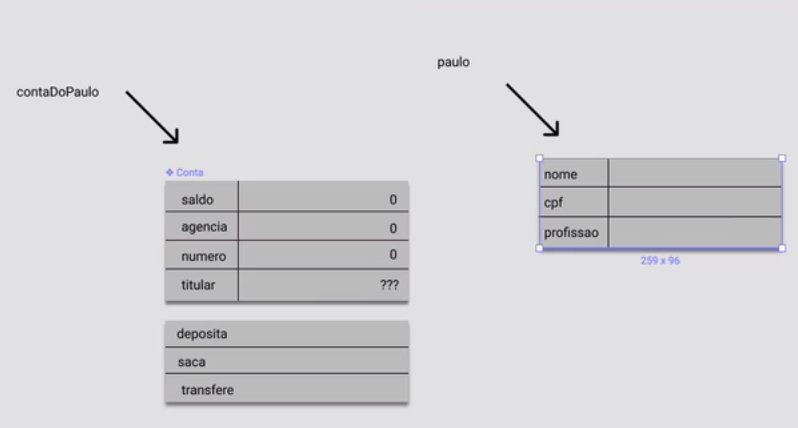
paulo.cpf = "222.222.222-22";

paulo.profissao = "programador";

}

}Copiar código

A referência para este cliente está populada com os dados estipulados.



Criaremos a conta do cliente referido, e depositaremos um valor de 100 reais.

public class TestaBanco {

public static void main(String[] args) {

Cliente paulo = new Cliente();

paulo.nome = "Paulo Silveira";

paulo.cpf = "222.222.222-22";

paulo.profissao = "programador";

Conta contaDoPaulo = new Conta();

contaDoPaulo.deposita(100);

}

}Copiar código

Agora temos uma classe Conta e outra Cliente. Queremos que o atributo titular não seja uma String, mas sim, uma referência para um objeto do tipo Cliente. Em nossa classe Conta, alteraremos o tipo do atributo titular para ser do tipo Cliente.

public class Conta {

double saldo;

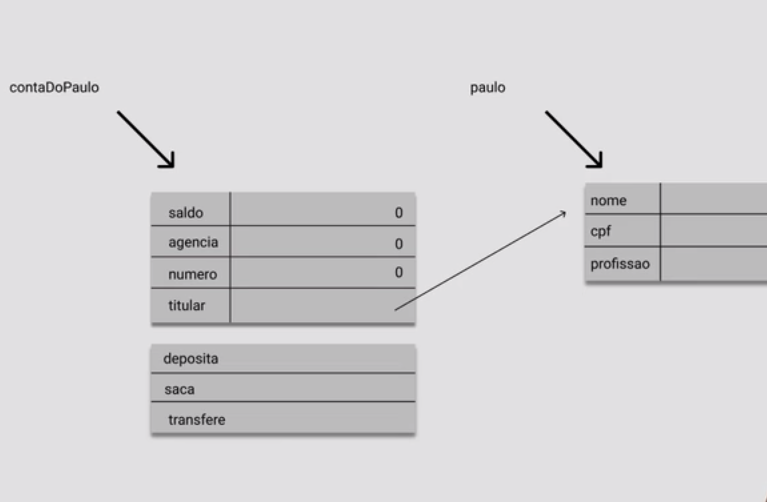
int agencia;

int numero;

Cliente titular;

<!-- ... -->Copiar código

A nossa ideia pode ser ilustrada pelo diagrama. Queremos que o atributo titular faça uma referência a um cliente específico, ou seja, iremos fazer uma associação entre objetos.



Faremos essa associação na classe TestaBanco, montando, assim, a nossa composição de objetos.

public class TestaBanco {

public static void main(String[] args) {

Cliente paulo = new Cliente();

paulo.nome = "Paulo Silveira";

paulo.cpf = "222.222.222-22";

paulo.profissao = "programador";

Conta contaDoPaulo = new Conta();

contaDoPaulo.deposita(100);

contaDoPaulo.titular = paulo;

System.out.println(contaDoPaulo.titular.nome);

}

}Copiar código

Ao executarmos a aplicação, veremos que será impresso o resultado Paulo Silveira.

Para testarmos o comportamento no programa, tentaremos imprimir apenas o titular.

public class TestaBanco {

//...

contaDoPaulo.titular = paulo;

System.out.println(contaDoPaulo.titular.nome);

System.out.println(contaDoPaulo.titular);

}

}Copiar código

Ao executarmos a aplicação veremos que o resultado será uma espécie de Id ( Cliente@15db9742), que possui o mesmo valor que a variável paulo, afinal, trata-se da referência para um mesmo objeto.

**02Extraindo o que é comum**

Juarez criou as seguintes classes:

public class Pessoa {

String nome;

String cpf;

int idade;

String logradouro;

String complemento;

String numero;

String bairro;

String cidade;

String cep;

}

public class Empresa {

String razaoSocial;

String cnpj;

String logradouro;

String complemento;

String numero;

String bairro;

String cidade;

String cep;

}Copiar código

Podemos perceber que os atributos logradouro, complemento, numero, bairro, cidade e cep são os mesmos nas duas classes. Essas informações são fortes candidatas para serem externalizadas na classe Endereco e associadas às classes Pessoa e Empresa através de composição.

public class Endereco {

String logradouro;

String complemento;

String numero;

String bairro;

String cidade;

String cep;

}Copiar código

Marque a única alternativa verdadeira que modifica corretamente as classes Pessoa e Empresa para utilizarem a classe Endereco.

Parte superior do formulário

* public class Pessoa {
* String nome;
* String cpf;
* int idade;
* Endereco endereco;
* }
* public class Empresa {
* String razaoSocial;
* String cnpj;
* Endereco;
* }
* Alternativa incorreta
* public class Pessoa {
* String nome;
* String cpf;
* int idade;
* }
* public class Empresa {
* String razaoSocial;
* String cnpj;
* Endereco endereco;

}

* Alternativa incorreta
* public class Pessoa {
* String nome;
* String cpf;
* int idade;
* Endereco endereco;
* }
* public class Empresa {
* String razaoSocial;
* String cnpj;
* Endereco endereco;

}

Correta!

Parte inferior do formulário

Parabéns, você acertou!

Este conteúdo foi útil para o seu aprendizado?

Sim

Não muito

**03Referência Null**

**Transcrição**

Voltaremos para a questão do valor 0 dos atributos e como isso se dá nos casos do tipo String. Qual seria o valor zerado do atributo titular da classe Conta, uma vez que ela faz referência à outra classe Cliente?

public class Conta {

double saldo;

int agencia;

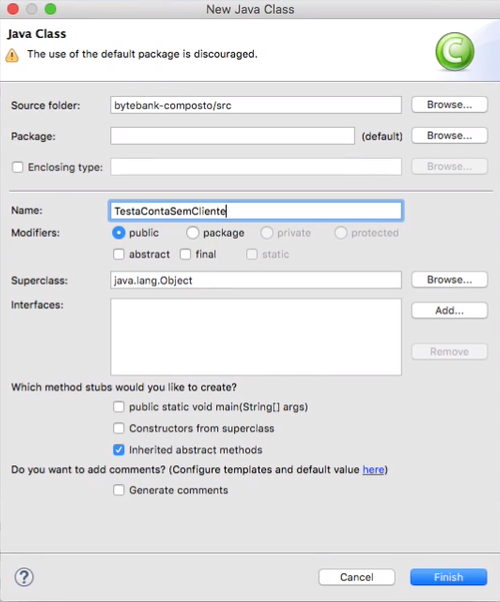
int numero;

Cliente titular;

// ...

Copiar código

Realizaremos um teste para tentar descobrir o que acontece caso não criemos um Cliente. Para isso, criaremos uma nova classe chamada TestaContaSemCliente.



Criaremos na nova classe o main, bem como uma referência para Conta chamada contaDaMarcela utilizando a palavra-chave new.

public class TestaContaSemCliente {

public static void main(String[] args) {

Conta contaDaMarcela = new Conta();

System.out.println(contaDaMarcela.saldo);

}

}Copiar código

Acionando o sysout para saldo, o código é compilado e o atributo zerado sem nenhum problema. Mas se tentarmos fazer um procedimento parecido com titular, sem definirmos um Cliente para este atributo?

public class TestaContaSemCliente {

public static void main(String[] args) {

Conta contaDaMarcela = new Conta();

System.out.println(contaDaMarcela.saldo);

contaDaMarcela.titular.nome = "Marcela";

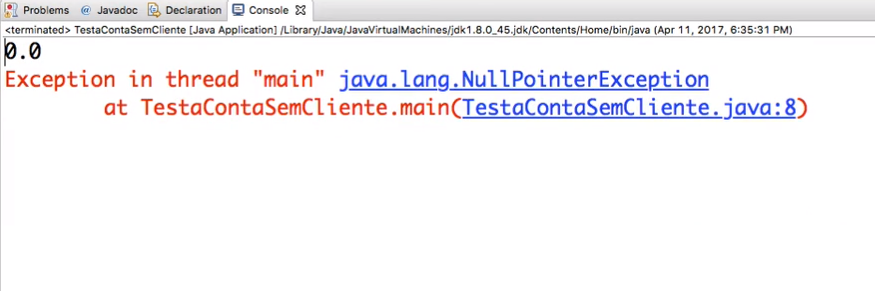
System.out.println(contaDaMarcela.titular.nome);

}

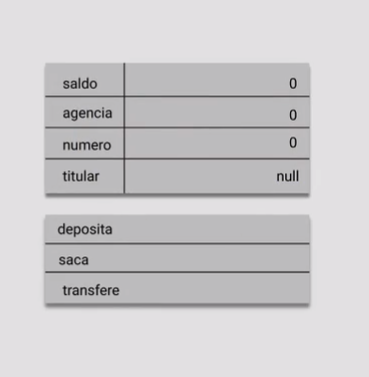
}Copiar código

Ao executarmos a aplicação, veremos que houve uma mensagem de erro. Por enquanto, o conteúdo dessa mensagem ficará nebuloso, mas analisaremos os termos da mensagem posteriormente.

Percebam que está em destaque a linha que ocasionou o erro de aplicação, que é TestaContaSemClinete.java:8.



O "zero" de um atributo ou variável do tipo referência, chamamos de null, que significa algo como "referência para lugar nenhum".



Podemos ter uma referência para nada no nosso código. Na linha 7 do nosso código, podemos realizar um sysout em contaDaMarcela e titular.

public class TestaContaSemCliente {

public static void main(String[] args) {

Conta contaDaMarcela = new Conta();

System.out.println(contaDaMarcela.saldo);

System.out.println(contaDaMarcela.titular);

contaDaMarcela.titular.nome = "Marcela";

System.out.println(contaDaMarcela.titular.nome);

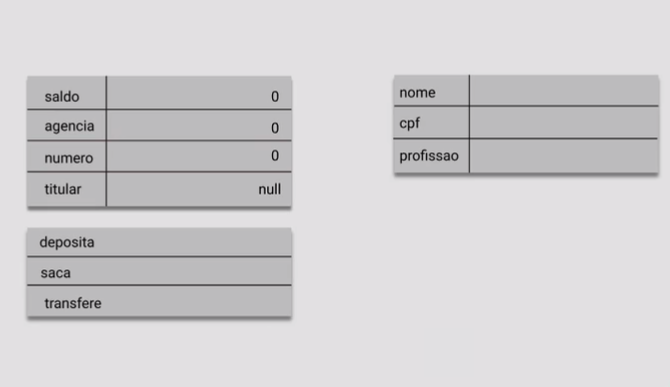
}

}Copiar código

Ao executarmos novamente o código, veremos que o resultado da impressão será a mensagem null antes do surgimento do erro.



O nome do titular("Marcela") não faz referência a nenhum tipo Cliente. Para que a aplicação seja executada corretamente, precisamos criar um novo cliente e fazer a associação entre Conta e Cliente.



Já temos no código a associação ao objeto Conta através da variável contaDaMarcela. Nosso próximo passo é fazer a associação entre titulare nome. Nas atividades anteriores, declaramos que a variável paulo era responsável por essa associação entre objetos. Neste caso, faremos de outro modo.



Faremos com que titular deixe de ser null, fazendo-o receber um novo cliente: new Cliente.

public class TestaContaSemCliente {

public static void main(String[] args) {

Conta contaDaMarcela = new Conta();

System.out.println(contaDaMarcela.saldo);

contaDaMarcela.titular = new Cliente();

System.out.println(contaDaMarcela.titular);

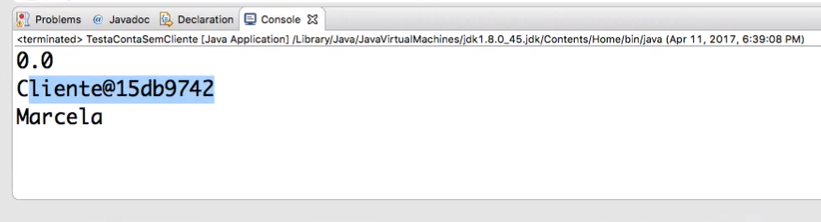
contaDaMarcela.titular.nome = "Marcela";

System.out.println(contaDaMarcela.titular.nome);

}

} Copiar código

Há casos em que não há necessidade de criar uma variável temporária, podemos criar a associação em uma linha, como é o caso. Ao executarmos a aplicação, será impresso um Id referente ao Cliente, revelando a associação feita entre os objetos.



Normalmente no Java, são criadas grandes redes de objetos interconectados que se referenciam, e através da invocação de métodos conseguimos fazer com que eles trabalhem entre si. O resultado é que temos códigos curtos, mas que atuam em grandes conjuntos. Com isso, temos uma maior organização no projeto, pois são códigos mais fáceis de ler e de realizar manutenções.

O null é uma referência que você encontrará com muita frequência, e não há necessidade de se preocupar com ela.

Uma referência é tida como null porque ainda não foi populada.

Para popular uma referência basta inserirmos um valor dentro dela, normalmente através de um new ou apontando para uma referência já existente, como fizemos no código anterior com paulo.

Um último desafio: lembre-se que podemos setar valores *default*. Na classe Conta, podemos dizer que toda vez que uma conta é aberta no **ByteBank** o saldo se inicia com 100.

public class Conta {

double saldo = 100;

int agencia;

int numero;

Cliente titular;

// ...Copiar código

Do mesmo modo, podemos fazer com o que toda a vez que o new é acionado para uma Conta, temos um novo Cliente. Ou seja, toda Conta já se associa a um Cliente, com isso, não nos preocuparíamos com o null neste caso em particular.

public class Conta {

double saldo = 100;

int agencia;

int numero;

Cliente titular = new Cliente();

// ...Copiar código

No nosso projeto não é uma opção muito interessante, pois toda a conta tem de ser associada à um cliente novo, banindo a possibilidade de um cliente ter duas contas, por exemplo. Porém, em muitos casos, essa é uma alternativa interessante.

**04Problema não esperado**

Fernanda decidiu aplicar seu conhecimento sobre composição utilizando as classes Pessoa e Endereço criadas por Juarez. Porém, seu código não esta funcionando em tempo de execução (*runtime*). Vejamos seu código:

public class Pessoa {

String nome;

String cpf;

int idade;

Endereco endereco;

}

public class Endereco {

String logradouro;

String complemento;

String numero;

String bairro;

String cidade;

String cep;

}

public class Programa {

public static void main(String[] args) {

Pessoa p = new Pessoa();

p.nome = "Paulo";

p.endereco.logradouro = "Avenida XYZ";

}

}Copiar código

Marque a opção que explica corretamente o erro no código de Fernanda.

Parte superior do formulário

* Fernanda não atribui valores à todas as propriedades da instância da classe Pessoa e por isso seu código não funcionará em tempo de execução.
* Alternativa incorreta

Ela esta acessando uma propriedade de um objeto que não foi inicializado.

Como a classe Pessoa não instanciou internamente a propriedade endereco ela é null. É por isso que a instrução p.endereco.logradouro resulta na exceção NullPointerException, pois p.endereco é null.

* Alternativa incorreta

Ela deveria ter feito p.logradouro = "Aveninda XYZ", pois há a composição de Endereco com Pessoa.

Parte inferior do formulário

Parabéns, você acertou!

Este conteúdo foi útil para o seu aprendizado?

Sim

Não muito

**05Solucionando o problema no código**

A classe Pessoa e Endereco de Juarez fez sucesso:

public class Endereco {

String logradouro;

String complemento;

String numero;

String bairro;

String cidade;

String cep;

}

public class Pessoa {

String nome;

String cpf;

int idade;

Endereco endereco;

}

public class Programa {

public static void main(String args[]) {

Pessoa p = new Pessoa();

p.nome = "Paulo";

p.endereco.logradouro = "Avenida XYZ";

}

}Copiar código

Marque as opções que contém a alteração que fará o código funcionar em tempo de execução:

Parte superior do formulário

* public class Pessoa {
* String nome;
* String cpf;
* int idade;
* Endereco endereco = new Endereco();
* }

A inicialização da propriedade endereco esta sendo realizada na própria classe. Dessa forma, o programador não precisará se lembrar de iniciá-la.

* Alternativa correta
* public class Programa {
* public static void main(String args[]) {
* Pessoa p = new Pessoa();
* p.nome = "Paulo";
* p.endereco = new Endereco();
* p.endereco.logradouro = "Aveninda XYZ";
* }

}

O cliente da classe Pessoa, ou seja, quem a esta utilizando, teve que atribuir uma instância de Endereco à propriedade p.endereco para que não houvesse NullPointerException.

* Alternativa correta
* public class Programa {
* public static void main(String args[]) {
* Pessoa p = new Pessoa();
* p.nome = "Paulo";
* p.endereco.logradouro = new Endereco();
* p.endereco.logradouro = "Aveninda XYZ";
* }

}

Parte inferior do formulário

Parabéns, você acertou!

**06O que aprendemos?**

Aprendemos neste capítulo:

* Realizar o relacionamento entre classes através de composição;
* Vantagens de se isolar informações repetidas em outra classe;
* Detalhes da composição;
* NullPointerException para atributos não inicializados e seus cuidados.

Este conteúdo foi útil para o seu aprendizado?

Sim

Não muito

**07Mão na massa: Referências**

Continuando nosso código...

1) Caso queira, crie um novo projeto e copie a classe **Conta**, ou apenas crie uma nova classe **Cliente** para compor nosso **bytebank**, nossa classe **Cliente** deve ter a seguinte cara:

public class Cliente {

String nome;

String cpf;

String profissao;

}Copiar código

2) Defina os atributos da classe **Cliente** como preferir, no curso, usamos **nome**, **cpf** e **profissão**.

3) Modifique a classe **Conta** adicionando um atributo do tipo **Cliente**.

public class Conta{

double saldo;

int agencia;

int numero;

Cliente titular;

}Copiar código

4) Crie um classe de teste e instancie um cliente e uma conta! Lembre-se de que o atributo **Titular** de uma conta guarda um **Cliente**

5) Repare que podemos fazer de uma forma mais direta!

Conta contaDaMarcela = new Conta();

contaDaMarcela.titular = new Cliente();Copiar código

Podemos definir o nome de um cliente fazendo:

contaDaMarcela.titular.nome = "Marcela";Copiar código

**Opinião do instrutor**

Alguma dúvida!? Pergunte no fórum!

**08Arquivos do projeto atual**

No link abaixo, você encontra o projeto até o momento atual do curso.

<https://github.com/alura-cursos/Curso-Java-parte-2-Introdu-o-Orienta-o-a-Objetos/archive/capitulo4.zip>

**Opinião do instrutor**

Lembre-se de usar o fórum em caso de dúvidas.