Atividade – 04/08/2023

Sistema bancário em Java – Explicado.

Imagine um banco real onde você tem vários clientes. Cada cliente possui informações básicas, como nome, idade e CPF.

Nessa analogia, a classe **Cliente** no seu código Java é como uma representação eletrônica dessas informações de um cliente no sistema bancário.

Classe **Cliente**

Atributos:

- **nome**: É como o nome completo de uma pessoa na vida real. Representa quem o cliente é dentro do banco.

- **idade**: Assim como a idade de uma pessoa indica quantos anos ela tem, este atributo armazena a idade do cliente.

- **cpf**: O CPF é um número único que identifica cada cidadão brasileiro, assim como neste código ele identifica cada cliente do banco de maneira única.

Construtores:

- **Cliente()** (sem parâmetros):
Esse é como um cadastro em branco. Imagine que você acabou de entrar no banco e recebeu um formulário para preencher suas informações. No início, todas as informações estão vazias. Isso é semelhante ao construtor vazio.

- **Cliente(String nome, Integer idade, String cpf)** (com parâmetros):
Esse construtor é como se você já tivesse preenchido o formulário com todas as suas informações. Você está entregando ao banco seu nome, idade e CPF, e eles estão registrando isso em seu sistema.

Métodos:

- **Getters**: Esses métodos são como perguntar ao banco sobre as informações de um cliente. Por exemplo, **getNome**() é como perguntar: "Qual é o nome deste cliente?".

- **Setters**: Esses métodos são como atualizar as informações de um cliente no banco. Por exemplo, se você mudar de nome por algum motivo legal, você iria ao banco e diria: "Por favor, atualize meu nome no sistema", que é análogo a chamar **setNome(String nome).**

- **toString()**: Este método é como um cartão de identificação que exibe todas as informações básicas do cliente. Quando chamado, ele retorna uma representação em texto das informações do cliente, semelhante a mostrar seu cartão de identidade para alguém.

Resumo:

Então, a classe `Cliente` pode ser pensada como uma ficha eletrônica dentro do sistema do banco. Ela contém todas as informações necessárias para identificar e descrever um cliente, assim como a ficha real faria em um banco físico.

Link do código (sem comentários):

https://github.com/felipe-laudano/estudando-java/blob/main/src/main/java/aula_040823/atividade/Cliente.java

Código explicado:

package aula_040823.atividade; // Isso define o pacote em que a classe está contida.

public class Cliente { //Isso declara a classe `Cliente`.

private String nome; //Variável private (só para uso nesta classe), tipo String, representará o nome do cliente.

private Integer idade;// Integer, representará a idade do cliente.

private String cpf; // String, representará o cpf do cliente.

public Cliente() { //Esse é o construtor vazio. É como pegar um formulário em branco quando você entra no banco.

}

public Cliente(String nome, Integer idade, String cpf){

//Esse é o construtor com parâmetros. É como preencher o formulário com suas informações e entregá-lo ao banco.

this.nome = nome; //A palavra-chave this ajuda o compilador a entender que você quer referir-se aos atributos da instância atual da classe, e não aos parâmetros do construtor (String nome, Integer idade, String cpf). Ela é uma referência à instância atual da classe em que está sendo usada. Ela ajuda a resolver a ambiguidade quando os parâmetros do construtor têm o mesmo nome que os atributos da classe.

this.idade = idade; // Para usar uma analogia, pense nos atributos da classe como "caixas de armazenamento" dentro do perfil de um cliente no sistema do banco. Quando um novo cliente é criado e você diz this.idade = idade;, é como se você estivesse pegando a informação da idade fornecida e colocando-a na caixa de armazenamento de idade do perfil desse cliente específico no banco. A palavra-chave this indica que você está trabalhando com as caixas de armazenamento (ou atributos) daquele cliente específico.

this.cpf = cpf; //this.cpf = cpf; significa "pegue o valor do parâmetro cpf fornecido a este construtor e atribua-o ao atributo cpf da instância atual da classe Cliente."

}

public String getNome() { //Esse método permite que alguém pergunte ao banco pelo nome do cliente. E ele retorna o nome.

return nome; //A instrução return nome; está retornando o valor do atributo nome da instância atual da classe Cliente. Em outras palavras, ela está recuperando o nome do cliente que está armazenado no objeto e o enviando de volta a quem chamou o método. Neste caso, o retorno precisa ser do mesmo tipo especificado na declaração do método (public String getNome()) que é String, então, em return nome; "nome" precisa ser do tipo String.

}

Se você imaginar o método como um funcionário do banco, quando alguém pergunta: "Qual é o nome deste cliente?", o funcionário verifica o registro e diz: "O nome deste cliente é 'João'." O nome 'João' é o que está sendo retornado.

<u>Comparação: return x System.out.println():</u>

return: Utilizado para enviar um valor de volta ao chamador. É como pedir um café em uma cafeteria e receber o café para usar como quiser (beber, derramar no copo, adicionar açúcar, etc.).
System.out.println(): Utilizado para mostrar informações no console. É como o barista na cafeteria anunciando em voz alta que o café está pronto. Todos no local podem ouvir a informação, mas ela não é "utilizável" da mesma maneira que o café real entregue nas suas mãos.

public void setNome(String nome) { //Este método permite que o cliente atualize seu nome no banco. "void" indica que o método não retorna nada.

this.nome = nome;

}

public Integer getIdade() { // Similar ao método `getNome()`, este permite perguntar a idade do cliente.

```
        return idade; //retorna a idade
    }
    public void setIdade(Integer idade) { // Este método permite que o cliente atualize sua idade no banco.
        this.idade = idade; }
    public String getCpf() { // Este método permite perguntar pelo CPF do cliente.
        return cpf; //retorna o cpf
    }


        public void setCpf(String cpf) { // Este método permite que o cliente atualize seu CPF no banco.
        this.cpf = cpf;
    }


    @Override // Esta anotação informa ao compilador que o método subsequente deve sobrescrever um método na classe pai (neste caso, a classe Object). Isso é útil como uma verificação em tempo de compilação. Se, por algum motivo, o método não estiver sobrescrevendo um método da classe pai (por exemplo, devido a um erro de digitação no nome), um erro será gerado.
    public String toString() { // Este método é como um cartão de identidade. Ele exibe todas as informações do cliente de maneira organizada. Esta linha declara o método como público, significando que ele pode ser chamado por qualquer código que tenha acesso a uma instância da classe.
        return "Cliente - nome: " + nome + ", idade: " + idade + ", cpf: " + cpf; // Essa linha está construindo uma String que representa o objeto. Ela concatena o texto "nome: " com o valor do atributo nome, e faz o mesmo com os atributos idade e cpf.
```

O resultado é uma String que descreve o objeto, incluindo seus valores atuais para nome, idade e cpf. Ele retorna uma String, que é a representação em texto do objeto.

```
    }
}
```

Ainda sobre o toString():

Analogia

Imagine que o objeto Cliente é uma pessoa em uma festa, e alguém pergunta: "Quem é você?". A pessoa então responde: "Meu nome é João, tenho 30 anos e meu CPF é 123.456.789-00." Esta resposta é o que o método toString() está fazendo. É como uma apresentação textual do objeto, contendo as informações mais relevantes que o descrevem.

Uso

O método toString() é muitas vezes chamado automaticamente quando você tenta imprimir um objeto usando, por exemplo, System.out.println(cliente). Se o método toString() foi sobrescrito na classe Cliente, ele imprimirá a String retornada por esse método, fornecendo uma descrição clara e legível do objeto. Isso facilita a depuração e a compreensão do estado do objeto durante o desenvolvimento.

Em resumo, cada linha do código contribui para a criação, manutenção e recuperação de informações de um cliente, assim como seria feito em um banco real. É uma representação virtual do processo de gerenciar informações do cliente em um ambiente bancário.

Classe **Conta**

A classe Conta representa uma conta bancária dentro de um sistema de banco. Ela mantém informações cruciais sobre a conta, incluindo o número da conta, o saldo atual e informações sobre o cliente (representado por um objeto da classe Cliente).

Aqui está uma descrição das principais partes da classe:

Atributos: A classe possui três atributos privados, que são o número da conta (um número inteiro), o saldo (um número de ponto flutuante) e o cliente associado à conta (um objeto da classe Cliente).

Construtores: Existem dois construtores, um sem parâmetros e outro que aceita todos os três atributos como parâmetros. Eles são usados para criar instâncias da classe Conta.

Getters e Setters: Esses métodos fornecem acesso controlado aos atributos da classe, permitindo que os valores sejam recuperados ou alterados de maneira segura.

Métodos de Depósito e Saque: A classe possui dois métodos essenciais que permitem depositar dinheiro na conta e sacar dinheiro dela. Eles incluem verificações para garantir que os depósitos sejam positivos e que os saques não excedam o saldo disponível.

Método toString(): Esse método foi sobrescrito para fornecer uma representação textual da instância de Conta, facilitando a impressão e depuração.

Você pode pensar na classe Conta como uma carteira digital dentro do sistema bancário. Ela contém tudo o que você precisa saber sobre a conta bancária, como o número de identificação (número da conta), quanto dinheiro está disponível (saldo) e a quem pertence (cliente).

Os métodos de depósito e saque são como colocar dinheiro dentro da carteira ou tirá-lo, e os getters e setters são como verificar a carteira ou alterar algo dentro dela (como adicionar um cartão de visita).

Link do código sem comentários:
https://github.com/felipe-laudano/estudando-java/blob/main/src/main/java/aula_040823/atividade/Conta.java

Código explicado:

```
public class Conta {
```

```
    private Integer numero;
```
// Representa o número da conta. É declarado como privado para que somente métodos dentro da classe possam acessar diretamente.

```
    private Double saldo;
```
// Representa o saldo atual da conta.

```
    private Cliente cliente;
```
// É um objeto da classe Cliente, representando a pessoa ou entidade proprietária da conta.

```
    public Conta() {
```
// Este é um construtor sem parâmetros que permite criar uma instância de Conta sem fornecer detalhes específicos.

```
    }
```

```
    public Conta(Integer numero, Double saldo, Cliente cliente) {
```
// Construtor com Parâmetros, permite criar uma instância de Conta, fornecendo o número da conta, saldo inicial e o cliente associado à conta.

this.numero = numero; // this é usado para referir-se aos atributos da instância atual da classe.

this.saldo = saldo;

this.cliente = cliente; // Este é o ponto mais interessante. O parâmetro cliente é um objeto da classe Cliente, e ele é atribuído ao atributo cliente da nova instância da classe Conta. Isso significa que o objeto Cliente passado ao construtor se tornará parte da nova instância da classe Conta, representando o cliente associado a essa conta específica.

}

Getters e Setters: Estes métodos permitem acessar e modificar os valores dos atributos:

Getters: getNumero(), getSaldo(), e getCliente() retornam os valores dos atributos correspondentes.

Setters: setNumero(Integer numero), setSaldo(Double saldo), e setCliente(Cliente cliente) permitem definir novos valores para os atributos correspondentes.

Seguem abaixo:

```
public Integer getNumero() {
    return numero;
}

public void setNumero(Integer numero) {
    this.numero = numero;
}
```

```
    public Double getSaldo() {
        return saldo;
    }

    public void setSaldo(Double saldo) {
        this.saldo = saldo;
    }

    public Cliente getCliente() {
        return cliente;
    }

    public void setCliente(Cliente cliente) {
        this.cliente = cliente;
    }
```

```
    public void depositar(Double valor) { // public: Indica que o método pode ser chamado de qualquer lugar fora da classe. void: Significa que o método não retorna um valor. depositar(Double valor): É o nome do método, e ele aceita um único parâmetro, que é o valor que será depositado na conta.
        if(valor > 0.0) { // Antes de depositar, o método verifica se o valor fornecido é maior que zero. Depositar um valor negativo não faz sentido no contexto de uma conta bancária, por isso a verificação é feita.
            this.saldo += valor; // Se o valor for positivo, ele é adicionado ao saldo atual da conta. Aqui, this.saldo refere-se ao atributo saldo da instância atual da classe Conta, e += é um atalho para adicionar o valor ao saldo atual.
        }
```

```
        else {
            System.out.println("Insira um valor positivo");
        }
```
// Se o valor fornecido for negativo ou zero, o método imprimirá uma mensagem de erro. Isso serve como uma forma de feedback para o chamador do método, indicando que algo deu errado.

```
    }
```

Imagine que a conta bancária seja uma carteira, e o método depositar seja como colocar dinheiro nessa carteira.

Se você tentar colocar uma quantidade positiva de dinheiro na carteira, ela simplesmente adicionará esse dinheiro ao que já está lá.

Se você tentar colocar uma quantidade negativa de dinheiro na carteira (o que não faria sentido na vida real), a carteira (ou o método depositar) responderá dizendo: "Insira um valor positivo".

Assim, este método simula a ação real de depositar dinheiro em uma conta bancária, com verificações para garantir que a ação seja lógica e significativa no contexto de um sistema bancário.

```
    public void sacar(Double valor) {
```
// O método sacar é outro componente essencial da classe Conta, permitindo que o dinheiro seja retirado da conta. public: Indica que o método pode ser chamado de qualquer lugar fora da classe. void: Significa que o método não retorna um valor. sacar(Double valor): É o nome do método, e ele aceita um único parâmetro, que é o valor que será retirado da conta.

Corpo do Método

```
        if (this.saldo >= valor) {
```
// Antes de fazer o saque, o método verifica se o saldo atual da conta é maior ou igual ao valor solicitado para saque. Isso garante que a conta tenha fundos suficientes para cobrir o saque.

this.saldo -= valor; // Se o saldo for suficiente, o valor solicitado é subtraído do saldo atual da conta. Aqui, this.saldo refere-se ao atributo saldo da instância atual da classe Conta, e -= é um atalho para subtrair o valor do saldo atual. } //fim do if

else {

System.out.println("Saldo insuficiente"); // Se o saldo não for suficiente para cobrir o valor solicitado, o método imprimirá uma mensagem de erro. Isso fornece feedback ao chamador do método, informando que o saque não pode ser realizado.

}

}

Imagine a conta bancária como uma jarra de água, e o método sacar é como tirar água dessa jarra.

Se você tentar tirar uma quantidade de água que é menor ou igual ao que está na jarra, você pode fazer isso facilmente, e a jarra ficará com menos água.

Se você tentar tirar mais água do que a jarra contém, a jarra (ou o método sacar) responderá dizendo: "Saldo insuficiente". Neste caso, a água na jarra permanece inalterada.

Assim, este método simula a ação real de sacar dinheiro de uma conta bancária, com verificações para garantir que a ação seja permitida no contexto do sistema bancário.

A verificação de saldo garante que você não possa sacar mais dinheiro do que realmente possui na conta, assim como não pode retirar mais água de uma jarra do que realmente existe nela.

```
    @Override
    public String toString() { // Semelhante ao método toString() na classe Cliente, este método retorna uma representação em String da instância de Conta, incluindo o número, saldo, e informações do cliente.
        return "\nConta - Numero: " + numero + ", saldo: " + saldo + "\n" + cliente;
    }
}
```

A classe Conta é como um dossiê de uma conta bancária que o banco mantém. Contém todas as informações importantes: o número da conta, o saldo e quem é o proprietário.

Os métodos depositar e sacar são como as transações que você faz em um caixa eletrônico ou balcão de um banco. Você pode adicionar dinheiro (depósito) ou retirar dinheiro (saque), e a conta reflete essas mudanças.

Os getters e setters são como perguntar ao seu gerente de banco para obter informações ou fazer mudanças na sua conta. Você pode perguntar "Qual é o meu saldo?" ou pedir "Mude o número da minha conta para X."

Classe **Banco**

A classe Banco serve como uma representação de um banco no sistema bancário que você está construindo. Essencialmente, ela age como um gerente central que cuida de várias contas bancárias. Assim como um banco real, ela tem a capacidade de gerenciar várias contas, criar novas contas para os clientes, buscar uma conta específica pelo número e listar todas as contas existentes.

Principais Componentes

Atributos: A classe tem um único atributo privado que é uma lista de contas (ArrayList de objetos Conta). Esse atributo simula o conjunto de contas que o banco gerencia.

Construtores: Dois construtores estão disponíveis, um para iniciar um banco sem contas e outro para iniciar com uma lista preexistente de contas.

Métodos: A classe possui métodos para criar uma conta para um cliente específico, buscar uma conta pelo número e listar todas as contas do banco. Método toString: Usado para fornecer uma representação em string do objeto Banco.

Se você pensar na classe Banco como um edifício de banco real:

A lista de contas seria como os cofres ou contas de poupança/armazenamento dentro do banco.

Os métodos como criarConta, buscarConta e listarContas são como os serviços que um banco real oferece aos seus clientes e funcionários.

A instância de Banco representa um banco específico que pode ter várias filiais ou ser parte de uma rede maior de bancos.

Em resumo, a classe Banco é o coração do sistema bancário, e os métodos e atributos dentro dela permitem que ela execute funções vitais que esperaríamos de um banco real.

Link do código sem comentários:

https://github.com/felipe-laudano/estudando-java/blob/main/src/main/java/aula_040823/atividade/Banco.java

Código explicado:

// A classe Banco representa um banco que contém várias contas

// Essas linhas são instruções de importação que permitem que a classe Banco use as classes ArrayList e List da biblioteca padrão do Java.

import java.util.List; // Esta linha importa a interface List do pacote java.util. A interface List é uma parte do Java Collections Framework e define o contrato para classes que representam uma coleção ordenada (também conhecida como sequência). A classe que implementa essa interface precisa fornecer métodos para acessar elementos por índice, procurar elementos na lista, iterar sobre os elementos da lista, etc.

import java.util.ArrayList; // Esta linha importa a classe ArrayList do pacote java.util. ArrayList é uma das várias classes que implementam a interface List. Ela usa uma matriz interna para armazenar os elementos e fornece métodos para acessar, adicionar, remover e modificar elementos.

Para entender isso de uma forma mais tangível, pense na interface List como a planta genérica de uma casa, descrevendo o que uma casa deve ter (por exemplo, quartos, banheiros, uma cozinha, etc.), sem especificar exatamente como essas partes são construídas ou organizadas.

A classe ArrayList, então, é como uma versão específica dessa casa, seguindo a planta genérica mas com detalhes

particulares (por exemplo, paredes de tijolos, um telhado de ardósia, janelas de vidro duplo, etc.).

Importar essas duas classes permite que a classe Banco crie e gerencie uma lista de contas, usando a interface genérica List e a implementação específica ArrayList. É como ter os planos e os materiais prontos para construir um bairro de casas (contas) dentro da cidade (banco).

```
public class Banco {
```

```
    private List<Conta> contas;
```
// Aqui, temos um atributo privado chamado contas, que é uma lista de objetos Conta. Isso representa todas as contas que o banco gerencia. Utiliza a classe implementada anteriormente "Conta".

```
    public Banco() {
```
// Esta linha define o construtor. A palavra-chave public significa que o construtor pode ser chamado de fora da classe (ou seja, é acessível a qualquer um que queira criar uma instância da classe Banco). O nome Banco indica que este é o construtor da classe Banco, e os parênteses () vazios indicam que o construtor não aceita nenhum parâmetro. Em outras palavras, é um construtor sem argumentos.

```
        this.contas = new ArrayList<Conta>();
```
// A palavra-chave this é uma referência ao objeto atual, ou seja, a instância da classe Banco que está sendo criada. this.contas se refere ao atributo contas da classe Banco (private List<Conta> contas), e esta linha de código está atribuindo a nova ArrayList vazia a esse atributo. ArrayList<Conta>() cria uma nova instância de ArrayList que é projetada para conter objetos da classe Conta. Essa nova lista está vazia no momento da criação.

```
    }
```

```
    public Banco(List<Conta> contas) {
```
//Este construtor, por outro lado, aceita uma lista de contas (List<Conta>) como argumento e inicializa o

atributo contas com essa lista fornecida. Ele é útil quando você quer criar um banco que já tem algumas contas existentes.

```
        this.contas = contas;
    }
```

Diferenças Principais entre os dois construtores:

Parâmetros: O construtor sem argumentos não requer informações adicionais para criar uma instância de Banco, enquanto o construtor com argumento requer uma lista preexistente de contas.

Inicialização de Contas: O construtor sem argumentos cria uma nova lista vazia, enquanto o construtor com argumento utiliza uma lista fornecida. O primeiro é como abrir um banco totalmente novo, enquanto o segundo é como assumir o controle de um banco existente com contas já ativas.

Pense no construtor sem argumentos como construir um novo edifício bancário do zero, sem clientes ou contas. Tudo é novo e vazio.

O construtor com argumento é como assumir o controle de um edifício bancário existente, com clientes e contas já ativos. Você está começando com uma operação que já está em andamento.

Essas duas abordagens oferecem flexibilidade na criação de um objeto Banco, dependendo das necessidades específicas do código que está utilizando essa classe. Você pode começar com um banco vazio ou com um conjunto preexistente de contas, conforme necessário.

//Métodos Getters e Setters. Esses métodos permitem obter e definir a lista de contas. São padrão em muitas classes Java para manipular atributos privados.

```
public List<Conta> getContas() {
    return contas;
}

public void setContas(List<Conta> contas) {
    this.contas = contas;
}
```

```
public void criarConta(Cliente cliente) { // O método criarConta é responsável por criar uma nova conta bancária e adicioná-la à lista de contas do banco. public: Indica que o método pode ser chamado de fora da classe. void: Significa que o método não retorna um valor. criarConta: É o nome do método. (Cliente cliente): É o parâmetro que o método aceita. Ele exige um objeto Cliente para associar à nova conta.
    Conta conta = new Conta(this.contas.size()+1, 0.0, cliente); //Cria uma nova instância da classe Conta. this.contas.size()+1: Calcula o próximo número de conta com base no tamanho atual da lista de contas (this.contas.size()) e adiciona 1 a ele. 0.0: Define o saldo inicial da conta como 0.0. cliente: Usa o objeto Cliente passado como argumento para associar o cliente à conta.
    contas.add(conta); //Adiciona a nova conta criada à lista de contas do banco (atributo contas).
}
```

Pense no método criarConta como o processo de abrir uma nova conta bancária em um banco físico.

Recepção do Cliente: O banco recebe um cliente (representado pelo parâmetro Cliente cliente).

Criação de uma Nova Conta: O banco então prepara todos os documentos necessários (representado pela criação da instância Conta), atribui um número de conta único (baseado no número total de contas já existentes), e define o saldo inicial como zero. Adição à Lista de Contas: Finalmente, o banco registra a conta no sistema, adicionando-a à lista geral de contas.

O resultado é que o banco agora tem uma nova conta associada ao cliente fornecido, e essa conta foi adicionada à lista interna de contas que o banco gerencia. O método encapsula todo esse processo, permitindo que ele seja executado com uma única chamada de método, passando apenas o cliente como parâmetro.

```
//O método buscarConta na classe Banco é responsável por encontrar e retornar uma conta bancária com base no número da conta fornecido.

  public Conta buscarConta(Integer numeroConta) { //public: Indica que o método pode ser chamado de fora da classe. Conta: É o tipo de retorno do método, indicando que ele retornará uma instância da classe Conta ou null. buscarConta: É o nome do método. (Integer numeroConta): É o parâmetro que o método aceita. Ele exige um número de conta (como um Integer) que servirá como critério de busca.

    for(Conta conta : contas) { //sse é um loop for-each que percorre cada conta na lista de contas do banco (atributo contas).

     If (conta.getNumero().equals(numeroConta)) { //Dentro do loop, ele compara o número da conta atual (conta.getNumero()) com o número da conta fornecido (numeroConta) através do equals. Se forem iguais, ele encontrou a conta desejada.

         return conta; //Se a condição acima for verdadeira, ele retorna a conta encontrada e o método termina aqui.

      }

    }
```

```
        return null; //Se o loop termina sem encontrar uma conta com o número fornecido, o método retorna null, indicando que a conta solicitada não foi encontrada. Este return fica fora do loop for.
    } //fim do método buscarConta
```

Pense no método buscarConta como o processo de um funcionário do banco procurando os detalhes da conta de um cliente no sistema do banco.

Recepção do Número da Conta: O funcionário recebe o número da conta que deseja buscar (representado pelo parâmetro Integer numeroConta).

Procura no Sistema: O funcionário então começa a procurar essa conta nos registros do banco, olhando cada conta uma por uma (representado pelo loop for-each).

Encontrando a Conta: Se o funcionário encontrar a conta com o número correspondente, ele a entrega ao cliente (representado pelo return conta;).

Conta Não Encontrada: Se o funcionário não conseguir encontrar a conta, ele informa ao cliente que a conta não existe no banco (representado pelo return null;).

Esse método encapsula a lógica necessária para encontrar uma conta específica em uma lista de contas, fornecendo uma interface simples para buscar contas por número.

//O método listarContas na classe Banco é responsável por exibir informações sobre todas as contas armazenadas no banco.

public void listarContas() { //public: Indica que o método pode ser chamado de fora da classe. void: Significa que o método não retorna um valor. listarContas: É o nome do método.

for(Conta conta : contas) { //Este é um loop for-each que percorre cada conta na lista de contas do banco (atributo contas).

System.out.println(conta); //entro do loop, ele chama o método System.out.println, passando a conta atual como argumento. Isso irá imprimir a representação em String da conta atual, que é definida pelo método toString da classe Conta.

}

}

Pense no método listarContas como um funcionário do banco abrindo o sistema do banco e lendo em voz alta os detalhes de cada conta registrada.

Procura no Sistema: O funcionário começa do início da lista de contas e vai para cada registro um por um (representado pelo loop for-each). Leitura dos Detalhes: Para cada conta, o funcionário lê os detalhes em voz alta (representado pelo System.out.println(conta);). O resultado é uma listagem completa de todas as contas no banco, impressa na saída padrão (geralmente o console). Isso fornece uma maneira rápida e simples de visualizar todas as contas armazenadas no banco, útil para fins de depuração ou para fornecer uma visão geral das contas para o usuário.

//Este método retorna uma representação em String da classe Banco, que inclui a lista de contas.

@Override

public String toString() {

return "{ " + contas + " }";

```
    }
}
```

Imagine a classe Banco como um prédio de banco real, onde:

O atributo contas são os cofres onde o dinheiro de cada cliente é guardado.

O método criarConta é como um cliente abrindo uma nova conta no banco.

O método buscarConta é como um atendente do banco procurando os detalhes da conta de um cliente.

O método listarContas é como um relatório de todas as contas existentes no banco.

Esta classe encapsula todas as operações básicas que você esperaria de um banco, como gerenciar contas, procurar contas, e listar todas as contas.

Vamos analisar detalhadamente cada parte do código dentro do método main da classe SystemBank. Este método é o ponto de entrada para executar o programa e simula um cenário bancário utilizando as classes Banco, Cliente e Conta previamente definidas.

Código sem comentários:

https://github.com/felipe-laudano/estudando-java/blob/main/src/main/java/aula_040823/atividade/SystemBank.java

Explicação do código:

```
public class SystemBank { //inicia a classe SystemBank
```

```
    public static void main(String[] args) { //declaração do método main
```
(principal) é dentro dele que chamaremos todos os métodos das outras classes que criamos

```
        Banco itau = new Banco(); //Aqui é criada uma instância do objeto
```
Banco chamada itau, usando o construtor padrão. Pode-se pensar nisso como a criação de uma nova agência bancária chamada Itaú.

//Neste ponto, três clientes são criados com seus respectivos nomes, idades e números de CPF.

```
        Cliente Jose = new Cliente("Jose", 21, "405.342.598-65");

        Cliente Renato = new Cliente("Renato", 14, "401.342.598-61");

        Cliente Maria = new Cliente("Maria", 40, "422.342.598-62");


        System.out.println(Jose); //Utilizando o método toString da classe
```
Cliente, os detalhes do cliente José são impressos no console.

```
Cliente - nome: Jose, idade: 21, cpf: 405.342.598-65

Process finished with exit code 0
```

//Utilizando o método criarConta da classe Banco, são criadas contas para cada cliente no banco Itaú.

```
itau.criarConta(Jose);

itau.criarConta(Renato);

itau.criarConta(Maria);
```

//O método buscarConta da classe Banco é chamado com o número de conta 1, e os detalhes da conta são impressos utilizando o método toString da classe Conta. ele fica oculto

```
System.out.println(itau.buscarConta(1));
```

```
Conta - Numero: 1, saldo: 0.0
Cliente - nome: Jose, idade: 21, cpf: 405.342.598-65

Process finished with exit code 0
```

//Utilizando o método buscarConta para encontrar uma conta e depositar 15.000 nela e em seguida, são sacados R$ 250 dessa conta.

```
itau.buscarConta(1).depositar(15000.0);

itau.buscarConta(1).sacar(250.0);
```

//Os detalhes da conta 1 são impressos novamente, refletindo as alterações feitas nos passos anteriores.

```
System.out.println(itau.buscarConta(1));
```

```
Conta - Numero: 1, saldo: 0.0
Cliente - nome: Jose, idade: 21, cpf: 405.342.598-65

Conta - Numero: 1, saldo: 14750.0
Cliente - nome: Jose, idade: 21, cpf: 405.342.598-65

Process finished with exit code 0
```

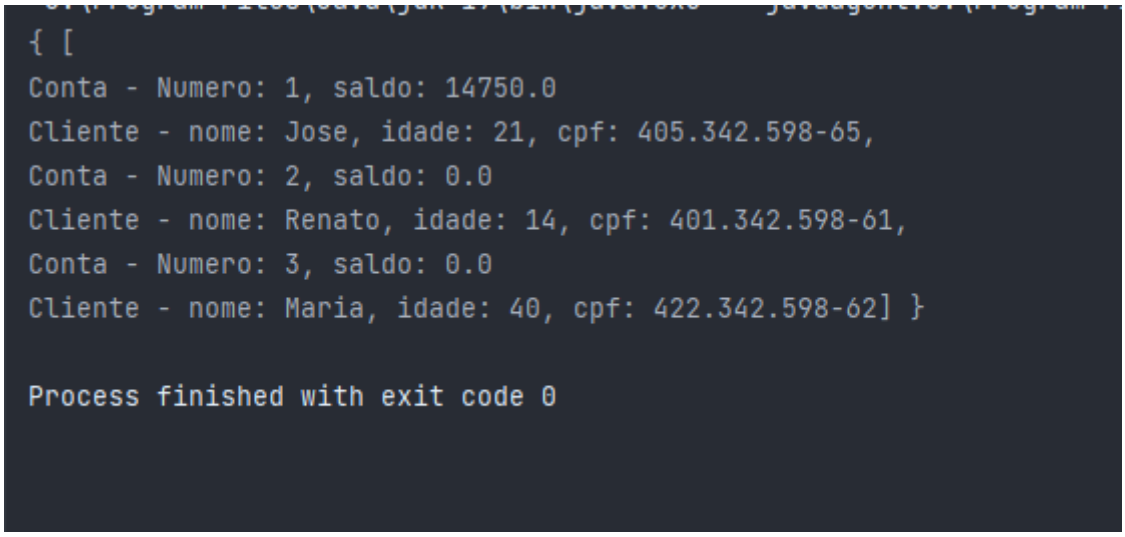
//Utilizando o método listarContas da classe Banco, são impressos os detalhes de todas as contas do banco Itaú.

```
itau.listarContas();
```

```
Conta - Numero: 1, saldo: 14750.0
Cliente - nome: Jose, idade: 21, cpf: 405.342.598-65

Conta - Numero: 2, saldo: 0.0
Cliente - nome: Renato, idade: 14, cpf: 401.342.598-61

Conta - Numero: 3, saldo: 0.0
Cliente - nome: Maria, idade: 40, cpf: 422.342.598-62

Process finished with exit code 0
```

// Utilizando o método toString da classe Banco, os detalhes de todas as contas do banco Itaú são impressos no console. Muito parecido com o método listarContas.

```
System.out.println(itau);
```



} //fim do main

}//fim da classe SystemBank

Link do código:

https://github.com/felipe-laudano/estudando-java/tree/main/src/main/java/aula_040823/atividade

Espero que tenha te ajudado! (;