Formabase
Aprenda passo a passo

Java
Projeto Final - Parte II

Metodologia

Interprete o documento calmamente e com atenção.

Acompanhe a execução do exercício no seu computador.

Não hesite em consultar o formador para o esclarecimento de qualquer questão.

Não prossiga para o ponto seguinte sem ter compreendido totalmente o ponto anterior.

Caso seja necessário, execute várias vezes o exercício até ter compreendido totalmente o processo.

Conteúdo programático

1. Banco

Para integrarmos todo o projeto é necessária a classe `Banco`. Esta classe será a responsável por várias ações, como a procura de clientes entre outras. Vamos começar por criar as funcionalidades relacionadas com os clientes.

Adicione ao seu projeto a classe `Banco.java` existente na pasta **Objetos/Projeto**

Analise o código desta classe e **implemente** os métodos relacionados com os clientes

Baseie-se na documentação dos métodos para as suas implementações.

Para **testar** estes métodos **use** o método `testarBancoClientes` da classe `TestesProjeto`

Precisamos também de incluir métodos para manusear contas que serão úteis mais tarde.

Adicione os seguintes métodos à classe `Banco.java`:

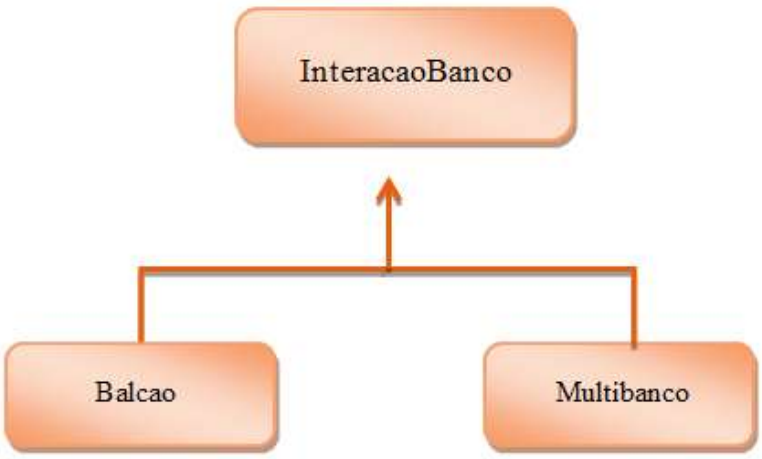
```
/**
* Procura em todos os clientes por uma conta com o nib recebido como parâmetro
* Devolve o objeto conta caso exista ou null caso não exista
* @param nib nib da conta a procurar
* @return Conta com o nib especificado
*/
public static Conta obterConta(int nib){
   //implementar o codigo deste método
}

/**
* Procura o cliente com o id recebido por parâmetro. Caso este exista adiciona
* o objeto conta recebido por parâmetro às contas desse cliente
* @param idcliente id do cliente a adicionar a Conta
* @param c Conta a adicionar
*/
private static void adicionaConta(int idcliente, Conta c) {
   //implementar o codigo deste método
}

/**
* Cria uma conta para o cliente c com o tipo tipoconta
* ESTE METODO ASSUME QUE O TIPO DE CONTA E VALIDO (1 - DEBITO / 2 - PRAZO)
* @param c O cliente sobre o qual vai ser criada a conta
* @param tipoconta o tipo da conta a criar
* @return O nib da nova conta criada
*/
public static int criarConta(Cliente c, int tipoconta){
   //implementar o codigo deste método
}
```

Agora **desenvolva** o código para os métodos que acabou de adicionar

Para **testar** os dois últimos métodos **use** o método `testarBancoContas` da classe `TestesProjeto`

2. Interação com o Banco

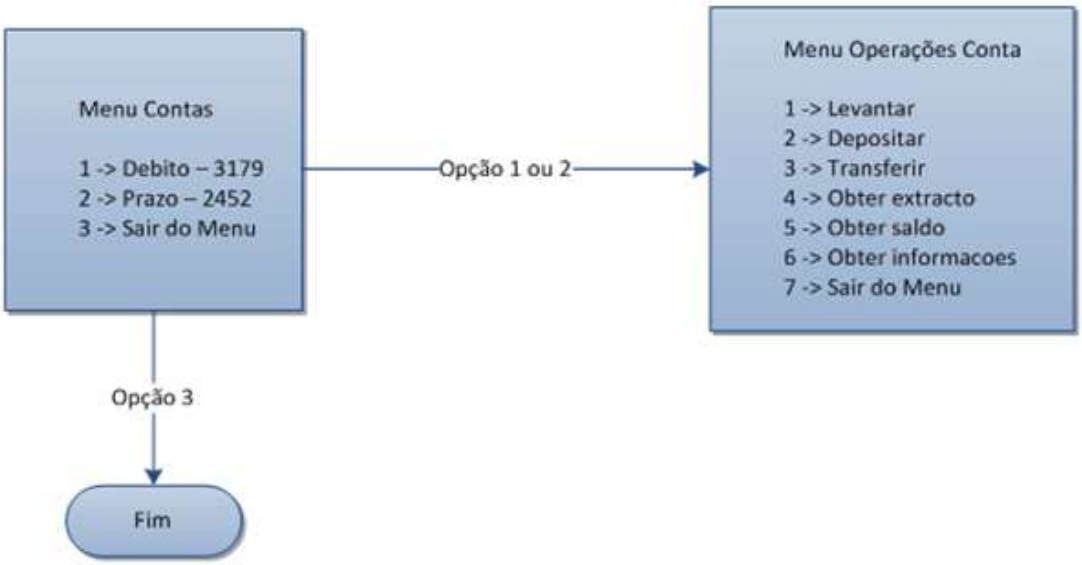
Neste momento faltam-nos apenas as classes que interagem diretamente com o banco e fornecem a navegação através de menus disponibilizando as operações ao utilizador e operador. Para este fim precisamos de três classes que se relacionam de acordo com o seguinte diagrama:



Comece por **integrar** no seu projeto a classe `InteracaoBanco.java` existente na pasta **Objetos/Projeto**

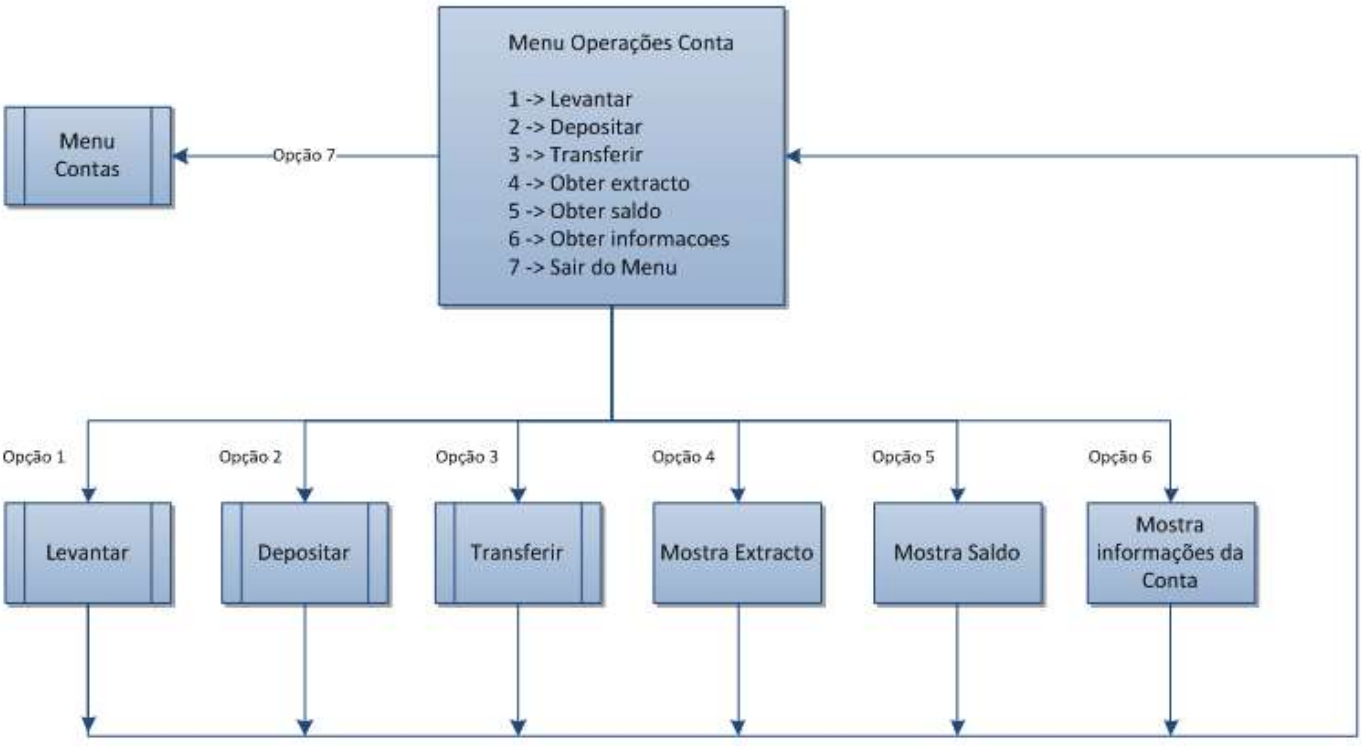
2.1. Menu Contas

A classe `InteracaoBanco.java` tem dois métodos que apresentam diferentes menus e as respetivas operações. O fluxograma seguinte exemplifica o funcionamento do primeiro menu desta classe. Este menu mostra as contas que o cliente escolhido possui, e ainda uma última opção para sair do menu. O menu das contas relaciona-se diretamente com o menu de operações de cada conta.



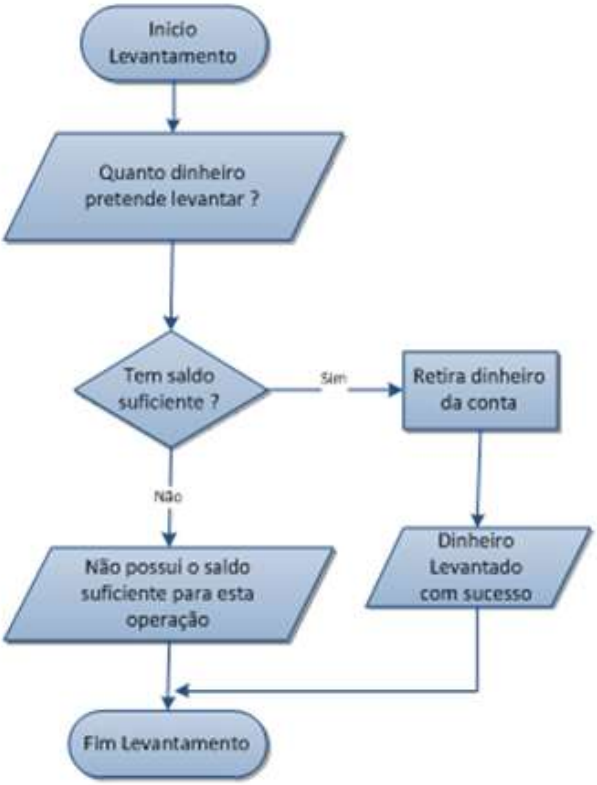
2.2. Menu Operações Conta

O menu de operações da conta possui as várias operações disponíveis para a gerir. Vamos agora observar o respetivo diagrama em maior detalhe:



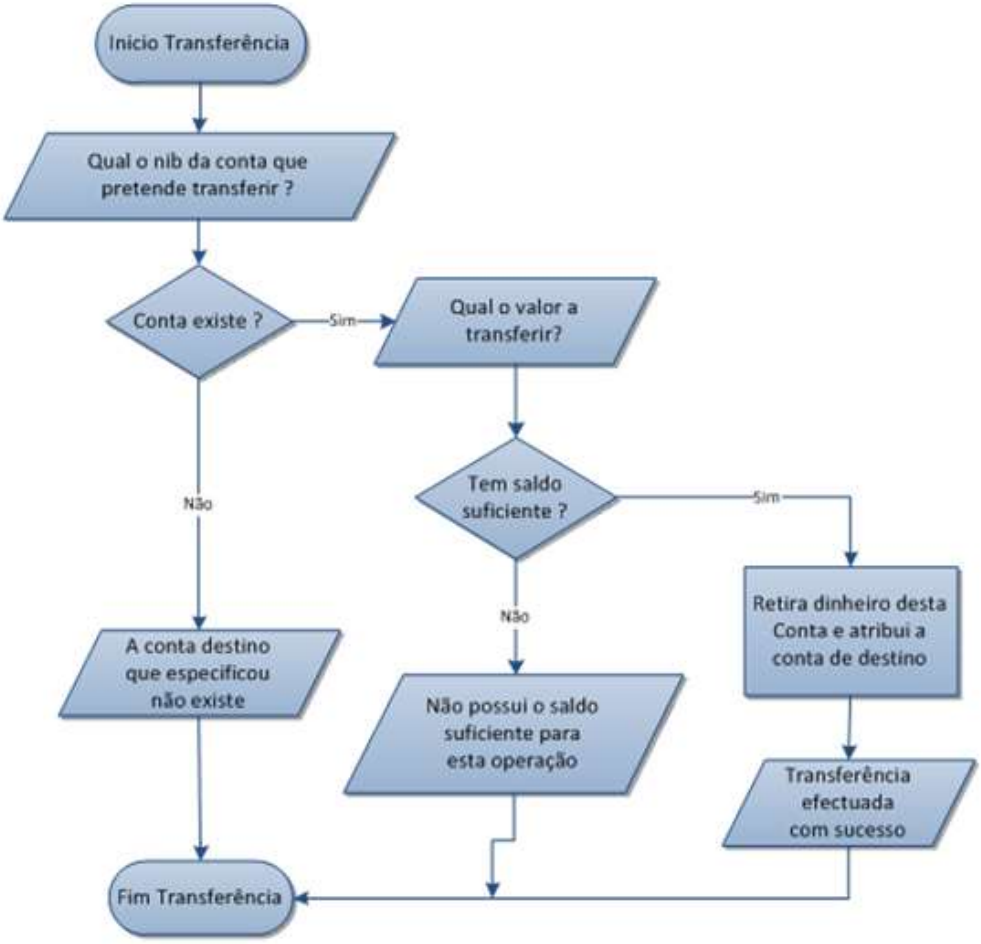
As opções com o retângulo recortado são mostradas em maior detalhe a seguir.

2.3. Levantar



2.4. Depositar

2.5. Transferir



3. Implementação

Implemente na classe `InteracaoBanco.Java` estes dois menus com o respetivo código para cada uma das operações

Para as ações de levantamento, depósito e transferência baseie-se nos diagramas apresentados nos pontos anteriores.

3.1. Testes

É necessário testarmos o código que acabámos de desenvolver mas, para este caso, existe uma particularidade. É essencial existir no banco pelo menos um cliente com as respetivas contas.

Para a classe `Banco.java` **adicione** o método `simularDados` e **substitua** o método `iniciar`:

```
public static void simularDados(){
    Cliente joao = new Cliente("joao",111,222);
    joao.adicionarConta(new Prazo());
    joao.adicionarConta(new Debito());
```

```
    Cliente pedro = new Cliente("pedro",123,456);
    pedro.adicionarConta(new Prazo());
    pedro.adicionarConta(new Debito());

    clientes.add(joao);
    clientes.add(pedro);
}

public static void iniciar(){
    clientes = new ArrayList<Cliente>();
    simularDados();
}
```

Ficámos assim com alguns dados de teste.

No método `main` da classe `InteracaoBanco.java`, **defina** as primeiras instruções da seguinte forma:

```
Banco.iniciar();
Cliente c = Banco.procurarCliente(111);
cli = c;
processaMenuContas(c.obterContas());
```

Estas instruções fazem com que os clientes de teste sejam adicionados à classe `Banco`. Para além disso especifica o cliente corrente da `InteracaoBanco`, com o identificador "111" (João).

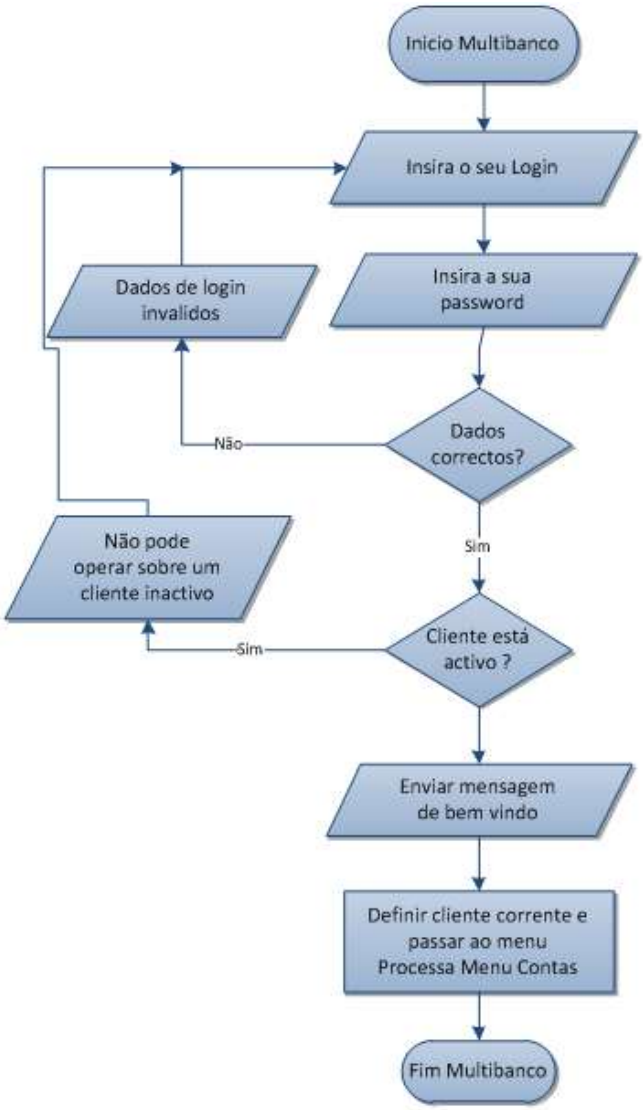
Teste esta classe

4. Multibanco

A classe `Multibanco` deriva da classe `InteracaoBanco`. Esta classe apenas acrescenta a parte da validação do utilizador no seu início, com o respetivo identificador e password.

Crie a classe `Multibanco.java`

Implemente esta classe tendo em consideração a lógica do fluxograma seguinte:



Deve ser usado o método `validarLogin` da classe `Banco` para a validação dos dados.

Teste esta classe

Para o teste desta classe deve usar no método `main` apenas o método `iniciar` do banco, pois a seleção do cliente corrente deve ser feita pela classe `Multibanco`. Esta seleção varia consoante o login e password inseridos.

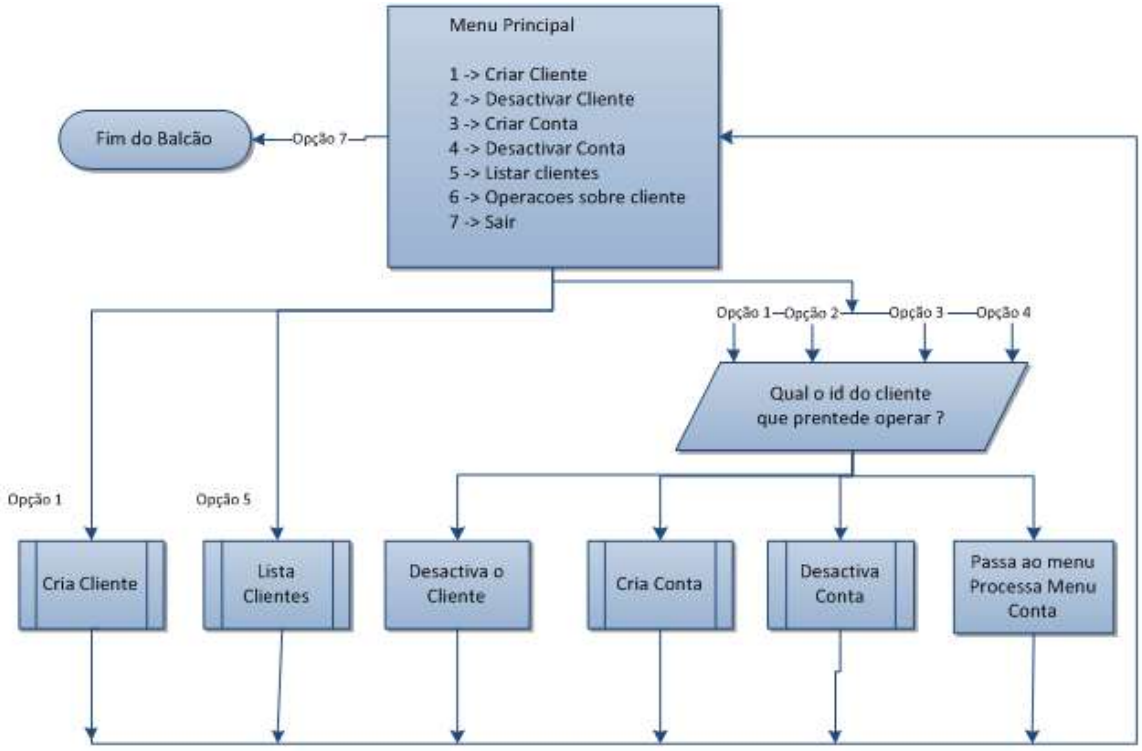
5. Balcão

A classe `Balcao` deriva também da classe `InteracaoBanco`. Para além dos métodos desta última, define mais um menu onde constam as operações apenas disponíveis ao operador do banco, como criar um cliente, listar clientes, etc.

Por questões de simplificação assumimos que a vertente de balcão do banco funciona internamente, não existindo quaisquer validações de acesso para os operadores.

5.1. Menu Principal

O menu principal da classe `Balcao` e as respetivas opções são explicadas no seguinte fluxograma:

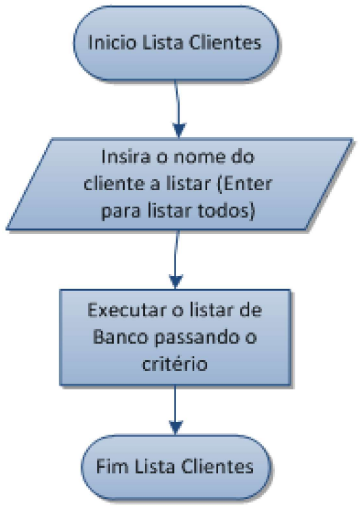


5.2. Ação Cria Cliente

Esta ação recebe os dados para o novo cliente e cria-o através da classe `Banco`. O identificador do utilizador para o cliente deve ser gerado aleatoriamente à semelhança do número da conta.

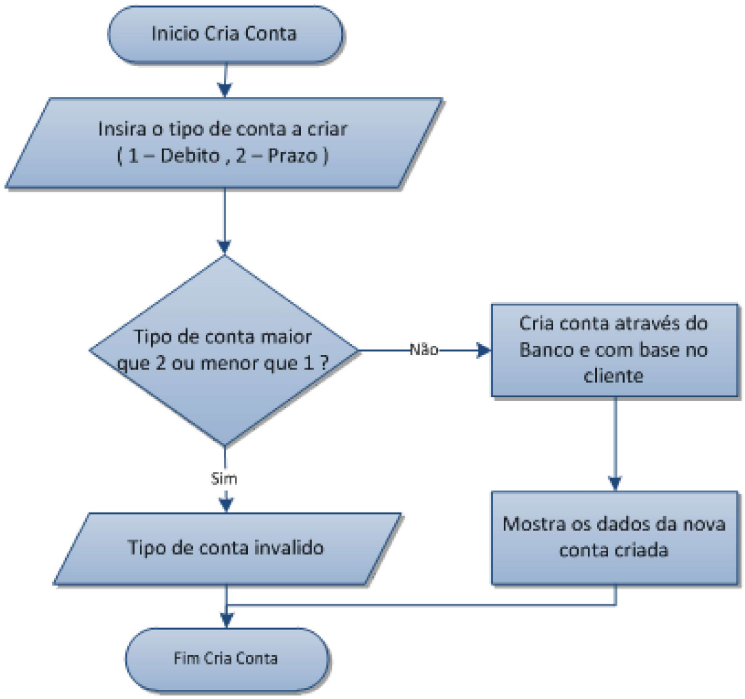
5.3. Ação Lista Cliente

Para a listagem de clientes deve ser pedido o critério, que pode ser o nome do cliente a pesquisar ou texto vazio para listar todos. Esta pesquisa é feita através do método `listarClientes` da classe Banco.



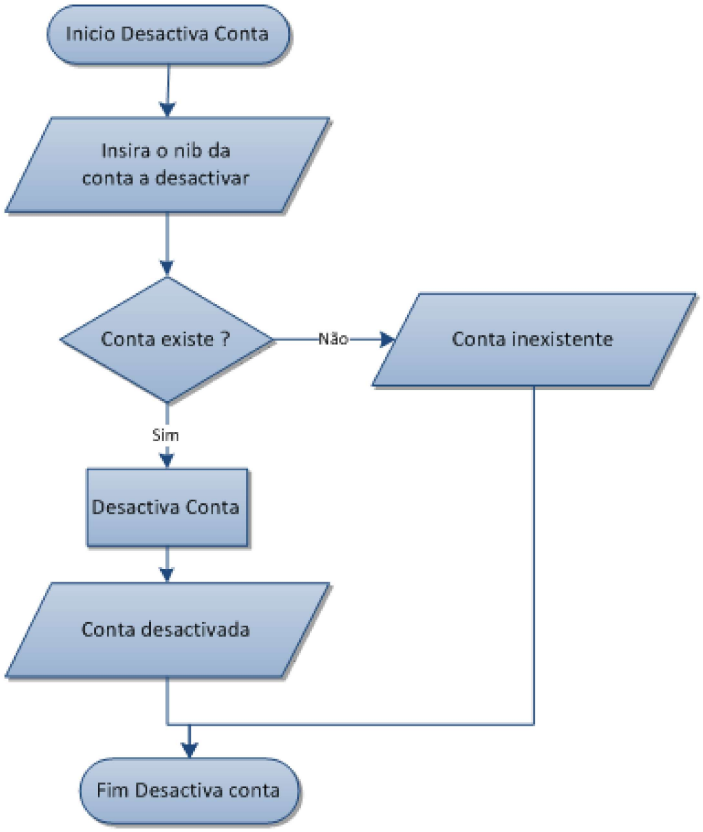
5.4. Ação Cria Conta

Esta ação necessita de conhecer o tipo de conta que se pretende criar. A criação é feita através do identificador de cliente já solicitado no diagrama do menu. Após a criação os dados da conta são mostrados no ecrã.



5.5. Ação Desativa Conta

Assim como na ação anterior também esta já contempla um cliente selecionado no diagrama do menu. A conta é obtida sobre este cliente com base no **nib**. Caso exista uma conta com o respetivo **nib** esta é desativada.



6. Implementação

Implemente a classe `Balcao` considerando os diagramas vistos anteriormente

Para testar esta mesma classe use também o método `iniciar` do `Banco`.

Confirme que todas as suas classes funcionam corretamente.

Para o nosso modelo de dados ficar conceptualmente correto, não devemos permitir a criação de objetos da classe `InteracaoBanco`. Isto porque as duas únicas formas que nos permitem interagir com o banco são o balcão e multibanco.

Torne a classe `InteracaoBanco` **abstrata**

Esta redefinição faz com que não seja possível instanciar objetos desta classe.

7. Ordenação Clientes

Após todas estas funcionalidades vamos adicionar uma que é bastante comum, a ordenação de arrays. Para isso é necessário voltar a rever a utilização de interfaces. O Java dispõe já de várias interfaces com diversos objetivos sendo que uma delas possibilita a ordenação - a interface **Comparable**. Uma classe que implemente esta interface possibilita a comparação dos seus objetos. É com base nessa comparação que os métodos de ordenação funcionam.

> Qualquer lista ou array de tipos que implementem a **interface Comparable** podem ser ordenados pelos métodos do Java.

Para ordenar os nossos clientes no banco necessitamos então de implementar esta interface na classe `Cliente`.

Modifique a linha de declaração da classe `Cliente` de acordo com o seguinte:

```
public class Cliente implements Comparable{
```

Agora é necessário implementar o único método que a interface especifica, o método `CompareTo`. Este método devolve o resultado da comparação, para este caso, entre dois clientes. Somos livres de definir o critério de comparação entre dois clientes, alterando assim a sua ordenação.

Adicione o método `compareTo` através do código seguinte:

```
@Override
public int compareTo(Object arg0) {

}
```

O `compareTo` devolve um inteiro com as seguintes definições:

- **Menor que zero** – Este cliente é menor que o recebido por parâmetro.
- **Igual a zero** – Ambos os clientes são iguais.
- **Maior que zero** – Este cliente é maior que o recebido por parâmetro.

Para a nossa ordenação iremos utilizar o nome como critério. Para este fim precisamos então de fazer uso do nome neste método `compareTo`.

A classe `String`, assim como várias outras no Java, já implementa a interface `Comparable`.

Adicione ao método `compareTo` o código:

```
Cliente clienteAComparar = (Cliente) arg0;
return nome.compareTo(clienteAComparar.nome);
```

A primeira linha permite-nos converter num `Cliente` o objeto recebido como parâmetro.

Com este código, se tentarmos comparar um cliente com outro tipo de objetos que não um cliente, obtemos um erro de conversão.

A segunda linha, que é a linha de retorno, indica que a comparação de dois clientes é a comparação entre os nomes desses dois clientes.

Agora basta invocar o método de ordenação já disponível no Java.

Adicione a seguinte instrução no início do método `listarClientes` da classe `Banco`:

```
Collections.sort(clientes);
```

O método de ordenação faz parte de uma classe que abrange todas as coleções do Java.

Execute a classe `Balcao` e **teste** a **ordenação**

Para se utilizar outro tipo de ordenação basta alterar o método `compareTo` da classe `Cliente`. Para ordenar pelo

identificador do cliente bastaria fazer o seguinte:

```
public int compareTo(Object arg0) {
    Cliente clienteAComparar = (Cliente)arg0;
    return userid-clienteAComparar.userid;
}
```

Repare agora como a comparação se baseia na diferença entre os dois identificadores. Esta subtração irá resultar numa ordenação ascendente pelos identificadores dos Clientes.

> Caso possua dúvidas neste ponto não hesite em solicitar ajuda ao formador.