Formabase
Aprenda passo a passo

Java
Projeto Final

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. Objetivo

Pretende-se desenvolver uma aplicação que funcione como um banco. Esta aplicação assume que o mundo bancário se resume a um único banco, onde existem vários clientes, cada um com as respetivas contas que podem ser de débito ou a prazo.

Um cliente pode utilizar o multibanco para efetuar as seguintes operações:

- Levantar dinheiro

- Depositar dinheiro
- Transferir dinheiro
- Obter extrato da conta
- Obter saldo da conta
- Obter informações da conta

Estas operações são executadas de forma diferente consoante o tipo de conta: a prazo ou débito. Este programa abrange ainda uma vertente balcão, onde se assume que é um funcionário do banco que executa as respetivas operações. Desta forma, o funcionário difere do cliente no sentido em que consegue executar as mesmas operações, assim como as seguintes:

- Criar um cliente
- Desativar um cliente
- Criar uma conta
- Desativar uma conta
- Listar clientes

Vamos começar por ver este projeto a funcionar para percebermos melhor o objetivo:

- **Aceda** à pasta **Objetos** do **Java** e, de seguida, à pasta **Projeto**

- **Copie** para a sua **pasta de aluno** os ficheiros `balcao.jar`, `multibanco.jar` e a pasta **dados**

- **Abra** a **Linha de comandos**

- **Execute** o comando `P:`

- Para executar o programa na vertente de cliente, **execute** `java –jar multibanco.jar`

- **Faça** várias experiências de maneira a perceber o funcionamento da aplicação

  Como dados de login pode usar como utilizador **111** e como password **222**, referentes à conta joao. Caso pretenda, pode sempre analisar os ficheiros .csv da pasta dados para ver quais os outros utilizadores disponíveis.

- Para a vertente balcão **execute** `java –jar balcao.jar`

Agora que já temos a perceção do que se pretende, vamos passar ao desenvolvimento. Todos os programas devem ser desenvolvidos de forma faseada, testando os vários blocos de código que são construídos de maneira a minimizar a existência de erros. O programa contempla várias entidades que se traduzem em classes.

As classes são:

- Conta
  - Debito
  - Prazo
- InteraccaoBanco
  - Balcao
  - Mulibanco
- Cliente
- Transaccoes
  - Levantamento
  - Transferencia
  - Deposito
  - CapitalizacaoJuros

- Banco

> Para todas as classes que desenvolvermos, devemos ter em mente o conceito de encapsulamento abordado nos módulos anteriores.

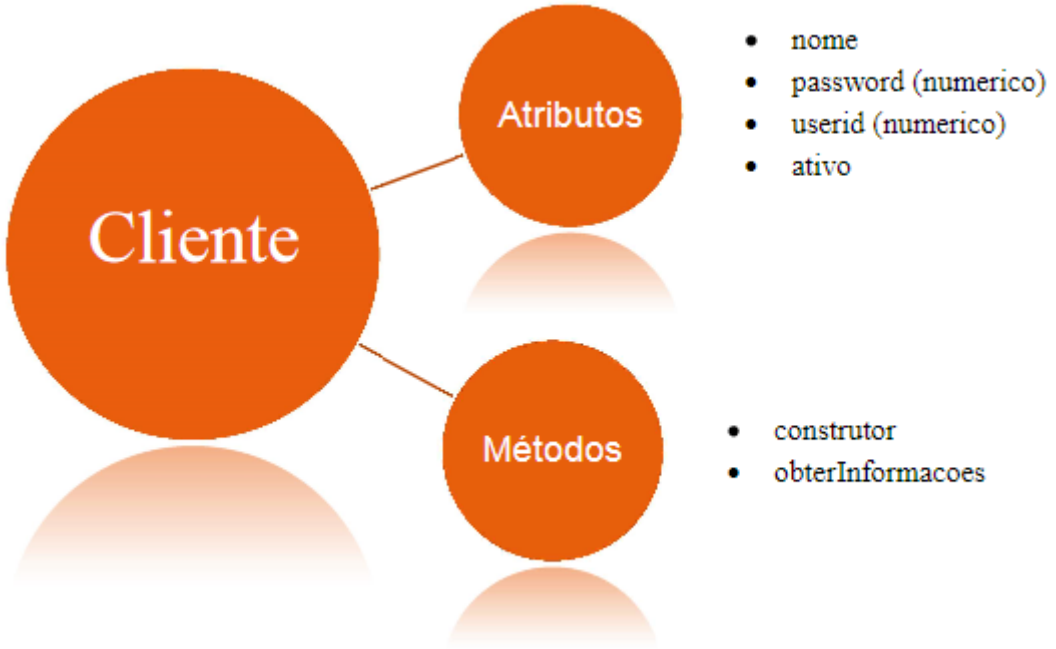
- **Crie** um novo projeto com o nome **ProjetoFinal**

2. Clientes

O banco possui uma lista de clientes, da qual todo o programa depende. Vamos começar por desenvolver a classe `Cliente` que irá representar cada um dos elementos dessa lista.

- **Crie** a classe `Cliente.java` dentro do novo projeto

- **Implemente** os **métodos** e **atributos** da classe `Cliente` de acordo com a figura que se segue:



- O construtor desta classe deve receber o **nome**, o **userid** e a **password**.

Os testes de todas as classes a desenvolver serão feitos através da classe `TestesProjeto.java`.

- **Inclua** no seu projeto a classe `TestesProjeto.java` existente na pasta **Objetos/Projeto**

- **Remova** os **comentários** do método `testarClientes` na classe `TestesProjeto` e **adicione** no `main` a chamada a esta função

⚠ Os comentários das instruções de `import` existentes no início do ficheiro devem ser removidos à medida que vão sendo necessárias nos métodos de teste.

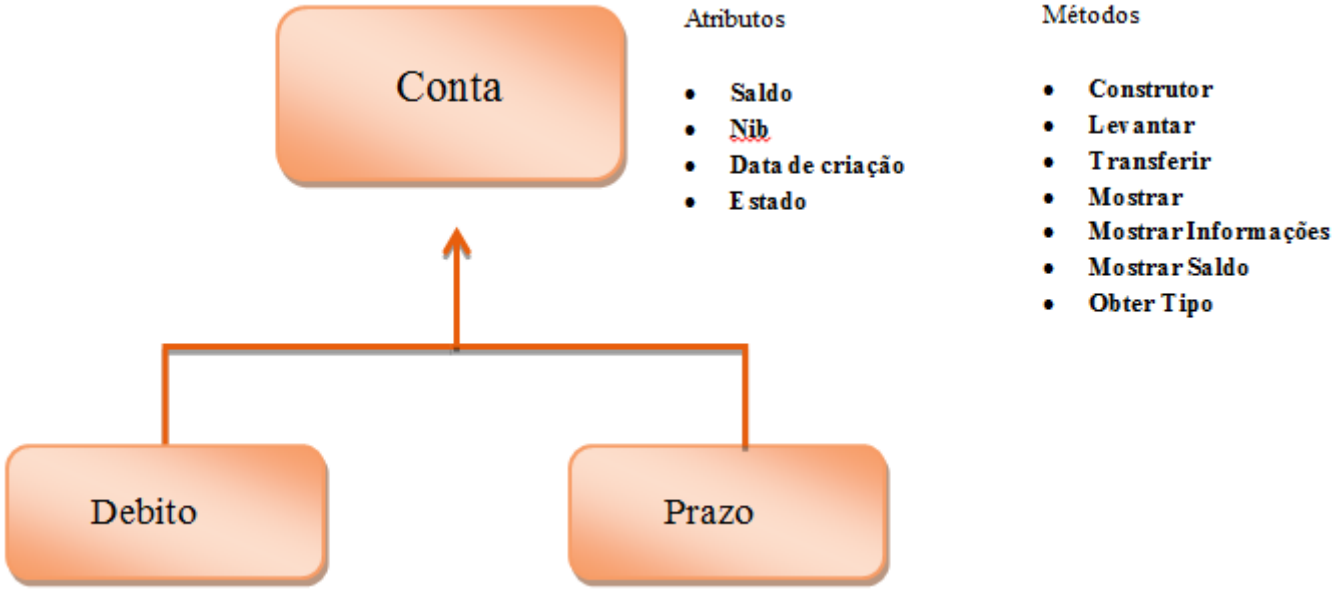
- Observe o método `testarClientes` de forma a compreender o seu funcionamento. Caso tenha dúvidas não hesite em solicitar ajuda ao formador.

- **Execute** a classe `TestesProjeto` e **observe** o resultado

  - Se criou corretamente a classe `Cliente`, não deverá ver qualquer erro na consola.

3. Contas

Assim como o banco possui clientes, também os clientes possuem contas. No entanto, este caso já é um pouco diferente do cliente, uma vez que a conta possui vários tipos. Por este motivo iremos ter uma classe base `Conta` e as respetivas classes derivadas. A seguinte figura exemplifica este conceito:



A classe `Conta` é uma classe abstrata, o que significa que não podem ser criados objetos deste tipo diretamente.

- **Crie** um **package** com o nome **contas**

- **Adicione** o ficheiro `Conta.java` existente na pasta **Objetos/Projeto** a este package

- **Analise** o ficheiro e **interprete os comentários** dos métodos

> Na classe `Conta`, foi utilizado um gerador de números aleatórios para a atribuição do número da conta. Esta geração de números aleatórios baseia-se numa classe já existente no Java. Mais à frente iremos substituir este pedaço de código por um método do Banco para gerar os números de conta.

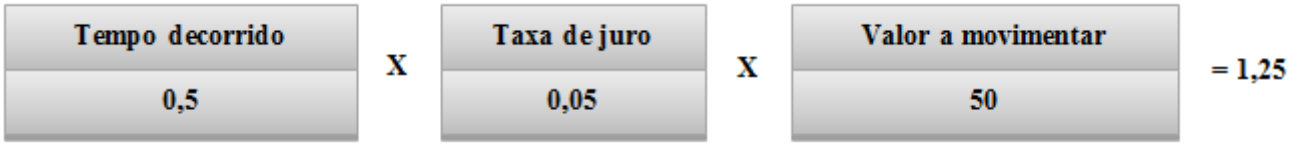
Para a classe `Debito` o funcionamento dos métodos corresponde exatamente à descrição na classe `Conta`. Para a classe `Prazo` tenha em consideração o seguinte:

- **Atributos** – Deve especificar atributos para guardar a taxa de juro, validade da conta e valor de juros acumulado.
- **Construtor** – Define a data de validade, como um ano após a data de criação da conta.
- **Aplicar taxa de Juro** – Calcula o valor de juros a ser acumulado na conta no final do prazo. Este valor não é adicionado diretamente ao saldo, mas sim guardado à parte.

O cálculo do valor dos juros contempla os seguintes fatores:

1. Tempo decorrido entre a data de criação da conta e a data corrente. Este valor corresponde a uma percentagem do valor total da validade da conta.
2. Taxa de juro.
3. Valor a movimentar.

Exemplo: Imaginemos que desejamos levantar **50€** de uma conta a prazo ao fim de 6 meses da sua criação. Antes de retirarmos o dinheiro devemos calcular o juro que foi acumulado por essa quantia, durante o tempo em que esteve na conta. Uma vez que o prazo é de 1 ano, e 6 meses correspondem a ½ do prazo, ficamos com o valor **0,5** para o tempo decorrido. A taxa de juro é de 5%, que corresponde ao valor **0,05**, ficando assim o cálculo:



Este valor será então somado ao acumulado de juros da conta.

- **Levantamento** – Caso exista saldo suficiente, levanta o dinheiro. É usado o cálculo anterior para aplicação da taxa de juro.
- **Transferência** – Caso exista saldo suficiente transfere o dinheiro. É usado o cálculo anterior para aplicação da taxa de juro.
- **Mostrar Saldo** – Mostra o saldo da conta e o valor de juros acumulado até ao momento.

- **Implemente** as classes `Debito` e `Prazo` com os respetivos atributos e métodos

- Após ter estas duas classes implementadas, **remova** os **comentários** da classe `TestesProjeto` no método `testarContas`

- **Adicione** ao `main` desta classe uma chamada ao método `testarContas`

- **Teste** o funcionamento do programa

Para podermos terminar este ponto falta-nos a integração dos clientes com as contas. Isto significa que vamos voltar à classe `Cliente` e adicionar métodos para manipular as contas.

A interação entre as duas entidades compreende três métodos na classe `Cliente`:

- **adicionarConta** – Recebe um objeto do tipo conta e adiciona-o à lista de contas do cliente.
- **obterContas** – Devolve um `ArrayList<Conta>` que representa todas as contas desse cliente.
- **obterConta** – Recebe um **nib** de uma conta e devolve o objeto "Conta" com esse **nib**, ou `null` caso não exista.

> Lembre-se que deve também adicionar um atributo à classe `Cliente` para guardar a lista de contas.

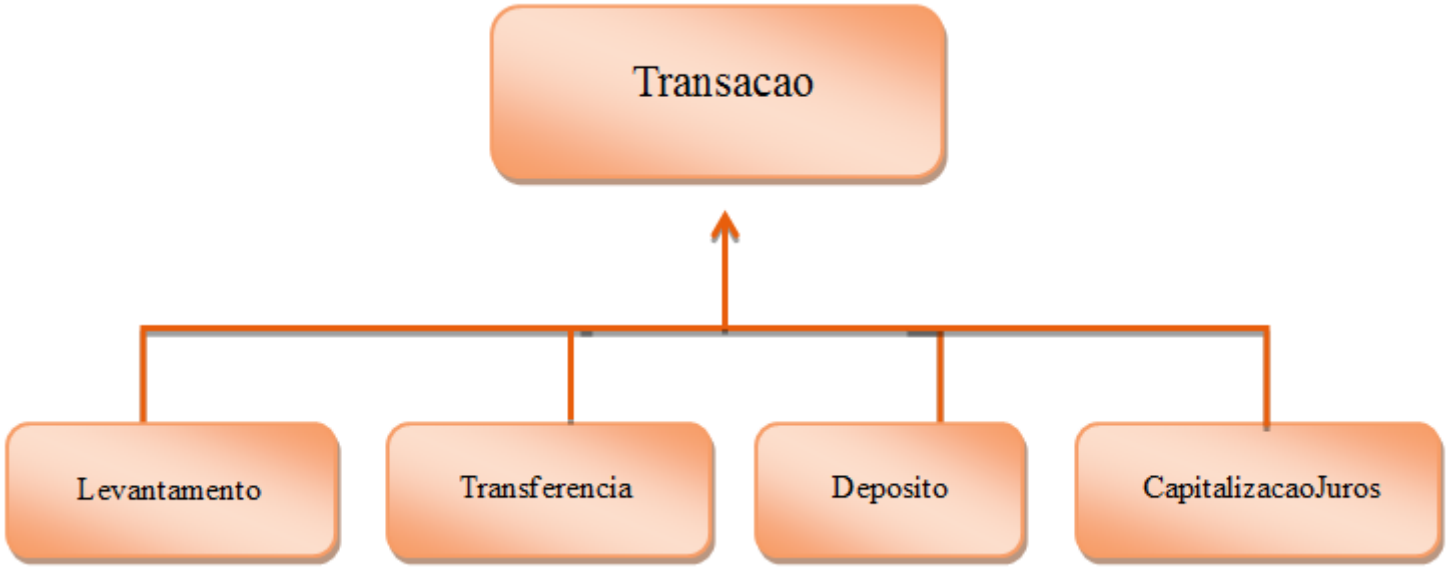
- **Implemente** estes **métodos** na classe `Cliente`

- Para **testar** as alterações **use** o método `testarContasComClientes` da classe `TestesProjeto`

4. Transações

É conveniente que o utilizador possa manter registo das operações que faz. Para este fim vamos criar uma classe `Transacao` que representa os vários movimentos das contas.

O diagrama de classes para este bloco referente às transações é o seguinte:



Cada transação deverá ter a informação da conta que refere, o valor associado ao movimento e a respetiva hora. É necessário ainda um método `mostrar` que devolva uma **String** representativa da transação, para que esta possa ser escrita no extrato com um formato semelhante:

```
Tipo – Data e Hora – Nib – Valor
```

Exemplo:

```
Deposito - 23-04-2012 17:04:00 - 2452 - 300.0€
```

- **Implemente** estas **classes** no package `transacoes`

  - A classe `Transacao` e alguns dos seus métodos devem ser abstratos.

- Para **testar** estas classes **utilize** o método `testarTransacoes` da classe `TestesProjeto`

> ⚠ Não avance sem confirmar que todos os métodos de testes mencionados até este ponto funcionam de forma correta.

Tal como no caso dos cliente e contas, falta agora integrar as transações com a respetiva conta. Tal como anteriormente é necessário alterar a classe `Conta`.

- **Implemente** os seguintes métodos na classe `Conta`:

- **mostrarExtrato** – Percorre a lista de transações da conta e mostra na consola o resultado do método `mostrar` de cada uma das transações. No fim deve imprimir o saldo da conta.
- **obterTransacoes** – Devolve a lista de transações da conta (esta deve ser do tipo `ArrayList<Transacao>`).
- **adicionarTransacao** – Adiciona um objeto "Transacao" à lista de transações.

- Para **testar** estes métodos que acabou de adicionar, **utilize** o método `testarTransacoesComContas` da classe `TestesProjeto`