Agência Bancária

André Barbosa – ei01043@fe.up.pt Filipe Montenegro – ei01104@fe.up.pt

- **Objectivos:**

- Implementar uma aplicação em Java para a gestão de uma agência bancária.

- O trabalho inclui a criação de dados para representação das várias entidades envolvidas, e métodos associados às operações básicas inerentes ao funcionamento de uma agência bancária.

- Os dados existentes devem ser armazenados em ficheiro para utilização posterior.

- O trabalho deve ainda incluir a geração de diversos relatórios.

- **Descrição:**

Antes de mais, no nosso trabalho cada utilizador tem um código e uma password únicos. Estes são usados para efectuar o *login* na agência bancária e posteriormente para ter acesso ás opções disponibilizadas. Como base da agência bancária está sempre um director (código: daf_00001 e passsword "web"). Para distinção entre os diversos utilizadores atribuímos diferentes iniciais ao código (administrativo – fa_xxxxx; gestor – fg_xxxxx; empresa – ce_xxxxx; individuo – ci_xxxxx). Estas iniciais explicam-se da seguinte maneira: fa = **f**uncionário **a**dministrativo, ce = **c**liente **e**mpresa, …

Os principais conceitos da aplicação, são representados do seguinte modo:

Estabelecemos uma hierarquia, que tem como base as classes **Funcionario** e **Cliente**. As duas subclasses de **Funcionario** são as classes **Administrativo** e **Gestor** e as subclasses de **Cliente** são as classes **Empresa** e **Individuo**. Tivemos em conta que uma hierarquia só é consistente quando um objecto de uma subclasse é, sem dúvida, um objecto mais específico da sua superclasse, ou seja, um administrativo é um funcionário.

A classe **Banco** não é mais do que uma base de dados contendo toda a informação do banco (administrativos, gestores, empresas, …). Esta informação é guardada em *LinkedLists*, ou seja, um objecto do tipo **Banco** contém uma *LinkedList* onde são guardados todos os administrativos (objectos) do banco, uma *LinkedList* onde são guardados todos os gestores (objectos) do banco, uma *LinkedList* onde são guardadas todas as empresas (objectos) do banco, …

A classe **Conta** representa as contas do banco. Tanto os clientes como os gestores usam a classe **Conta** (contêm objectos do tipo **Conta**).

A classe **Movimento** representa os movimentos das contas. Apenas as contas têm objectos deste tipo e esta só pode ser acedida pelos clientes.

A classe **Interface** estabelece uma interacção com o utilizador. Esta classe usa as classes **Funcionario**, **Administrativo**, **Gestor**, **Cliente**, **Empresa**, **Individuo**, **Conta**, **Movimentos** (já que a pedido do utilizador cria novos objectos destas classes e usa também os seus métodos *public* em diversas ocasiões e para diversos fins), **Banco** (já que é um objecto do tipo **Banco** que guarda toda a informação do banco).

É importante referir que, para ser possível guardar o banco (é guardado num ficheiro um objecto do tipo **Banco** que contém toda a informação do banco), foi necessário implementar *Serializable* nas classes **Funcionario** (as duas subclasses que descendem de **Funcionario** automaticamente implementam *Serializable*), **Cliente** (as duas subclasses que descendem de **Cliente** automaticamente implementam *Serializable*), **Conta**, **Movimento** e **Banco** (exemplo: *public class Banco implements Serializable*). A classe **Banco** implementa *Serializable* já que vai ser um objecto

deste tipo que vai ser guardado num ficheiro. As outras classes implementam *Serializable* já que um objecto do tipo **Banco** contém objectos dessas classes (excepção feita às classes **Funcionario** e **Cliente**).

Nota: Neste trabalho as classes **Funcionario** e **Cliente** são do ponto de vista do código iguais, mas tendo em conta a vida real duma agência bancária estas classes não podem ser só uma (escalões diferentes).

Funcionalidades disponibilizadas ao utilizador: (definidas na classe Interface)

[Start]

- Criar novo banco
- Abrir banco existente
- Sair

[Inicial]

- Login
- Sair e gravar

[Menu Director]

- Adicionar administrativo
- Remover administrativo
- Voltar

[Menu Administrativos]

- Listar gerentes de conta e informação relativa ás contas associadas
- Alterar dados pessoais
- Adicionar gestor
- Remover gestor
- Adicionar cliente
- Voltar

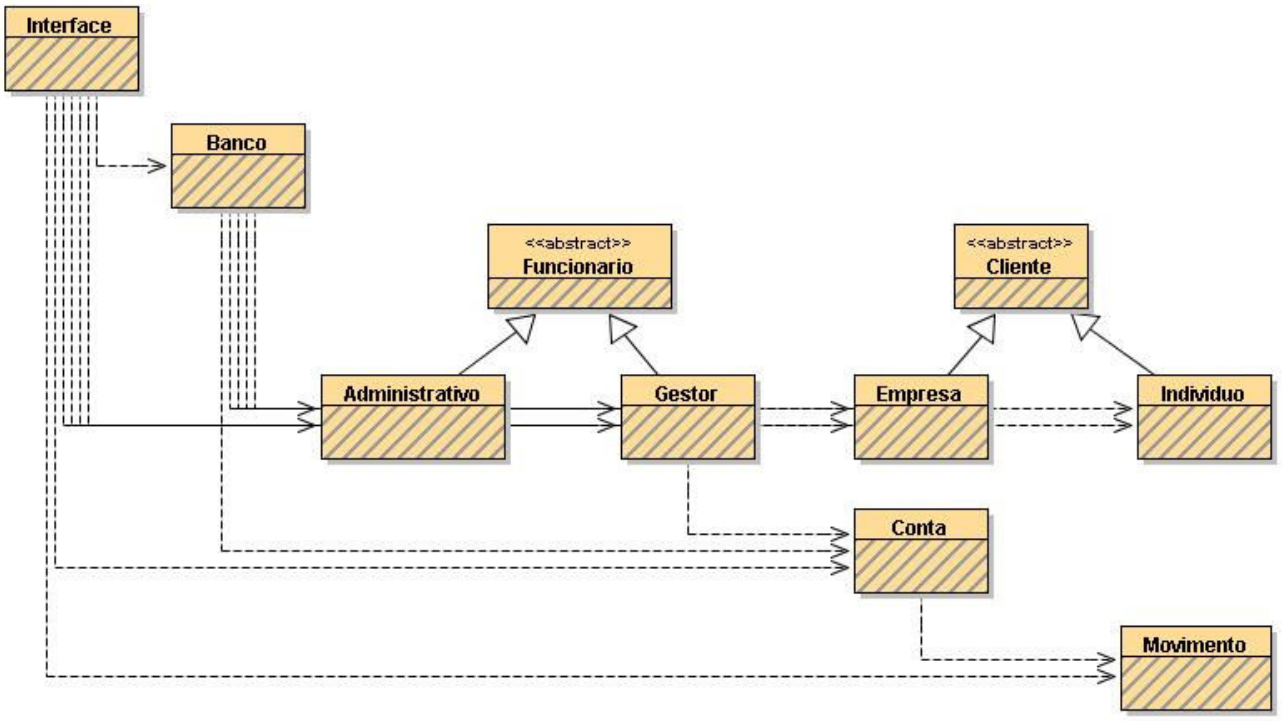
[Menu Gestores]

- Listar os clientes e saldos de conta respectivos
- Alterar dados pessoais
- Voltar

[Menu Empresas] = [Menu Individuos]

- Levantar dinheiro
- Depositar dinheiro
- Requisitar cheques
- Cancelar conta
- Alterar gerente de conta
- Consultar saldo de conta
- Consultar movimentos a partir de uma determinada data

- Alterar dados pessoais
- Voltar



As linhas a cheio representam uma relação de herança (exemplo: a classe **Administrativo** herda da classe **Funcionario**) e as linhas a tracejado representam uma relação de uso (exemplo: a classe **Banco** usa a classe **Conta** já que contém objectos deste tipo).

É importante referir que as classes **Funcionario** e **Cliente** são abstractas e por isso não se podem criar objectos dessas classes. No entanto, isto não significa que estas classes não possam ser construtores (como acontece nas classes referidas) já que uma coisa não invalida a outra (os construtores são utilizados pelas suas subclasses).

No desenvolvimento do projecto utilizaram-se as seguintes bibliotecas Java:

java.lang.* - fundamental na implementação do projecto já que contém métodos (exemplo: *System.out.println*) e objectos (exemplo: *String*) essenciais em qualquer aplicação em Java. É importante referir que não é necessário fazer o *import* a esta biblioteca pois esta é implicitamente usada (exemplo: escrever *String* é o mesmo que escrever *java.lang.String*)

java.util.* - necessário para a utilização de *LinkedLists*

java.io.* - necessário para a leitura de dados do teclado, para gravar um objecto num ficheiro, para abrir um objecto de um ficheiro e para implementar *Serializable.*

- ## Implementação:

Existem 8 tipos distintos de objectos dentro da agência bancária. Os objectos são os seguintes, e caracterizam-se da seguinte forma:

Interface – Objecto que interage com o utilizador. O seu construtor não tem parâmetros. Objecto da classe **Interface**

Banco – Base de dados que contém toda a informação do banco. Todos os objectos que, a pedido do utilizador são criados, são armazenados num objecto do tipo **Banco**. Objecto da classe **Banco**;

Administrativo – Pessoa responsável pela administração, caracterizada através de todas as características de funcionario* (já que um administrativo é um funcionário). Objecto da classe **Administrativo**;

Gestor – Pessoa responsável pela gestão das contas, caracterizada através de todas as características de funcionario* (já que um gestor é um funcionário) e também pelas contas que gere. Objecto da classe **Gestor**;

Empresa – Pessoa que possui uma conta no banco, caracterizada através de todas as características de cliente** (já que uma empresa é um cliente) e também pelo seu ramo. Objecto da classe **Empresa**;

Individuo – Pessoa que possui uma conta no banco, caracterizada através de todas as características de cliente** (já que um individuo é um cliente) e também pelo seu ramo. Objecto da classe **Individuo**;

Conta – Objecto no qual os clientes operam a fim de efectuarem transferências. Tem como características o código do cliente, a data de criação e o saldo. Objecto da classe **Conta**;

Movimento – Objecto que guarda as modificações feitas nas contas. Tem como características o montante, a data da operação e o tipo de movimento. Objecto da classe **Movimento**.

***Nota**: características do funcionário – código, nome, contacto e password.

****Nota**: características do cliente – código, nome, contacto e password.

Em todas as classes todas as variáveis foram declaradas como *private*. Optámos por fazer já que queremos restringir o seu acesso mas no entanto disponibilizamos métodos *public* (métodos *set* e *get*) para aceder ás mesmas. Desta forma protegemos as características do objecto sem deixar no entanto de promover opções para a sua alteração e/ou visualização. Resumindo, é uma boa regra de programação declarar as variáveis de uma classe como *private* e, quanto muito, disponibilizar métodos *public* para aceder a essas mesmas variáveis, já que desta forma controlamos o seu acesso e temos a opção de as manter com dados consistentes (exemplo: na classe **Funcionario** a variável nome é declarada como *private* e disponibilizamos o método *public setNome*).

Funções dos métodos em todas as classes

Classe Banco – classe que serve de base de dados armazenando toda a informação da agência bancária

public Banco () – Cria um banco com dados pré-estabelecidos

public void addAdministrativo (Administrativo administrativo) – Adiciona um novo administrativo ao banco

public void removeAdministrativo (int index) – Remove um administrativo do banco

public Administrativo getAdministrativo (int index) – Retorna um administrativo do banco

public String geraCodigoAdministrativo () – Retorna o código para um novo administrativo

public int getSizeAdministrativos *()* – Retorna o número de administrativos do banco

public void actualizaAdministrativos (int numRemovido) – Actualiza os administradores do banco

public void addGestor (Gestor gestor) – Adiciona um novo gestor ao banco

public void removeGestor (int índex) – Remove um gestor do banco

public Gestor getGestor (int índex) – Retorna um gestor do banco

public String geraCodigoGestor () – Retorna o código para um novo gestor

public Gestor getRandomGestor () – Retorna um gestor do banco escolhido aleatoriamente

public int getSizeGestores () – Retorna o número de gestores do banco

public void actualizarGestor (int numRemovido) – Actualiza os gestores do banco

public void addEmpresa (Empresa empresa) – Adiciona nova empresa ao banco

public void removeEmpresa (int index) – Remove uma empresa do banco

public Empresa getEmpresa (int índex) – Retorna uma empresa do banco

public String geraCodigoEmpresa () – Retorna código para uma nova empresa

public int getSizeEmpresas () – Retorna o número de empresas do banco

public void actualizaEmpresa (int numRemovido) - Actualiza as empresas do banco

public void addIndividuo (Individuo individuo) – Adiciona um individuo ao banco

public void removeIndividuo (int index) – Remove um individuo do banco

public Individuo getIndividuo (int índex) – Retorna um individuo do banco

public String geraCodigoIndividuo () – Retorna código para um novo individuo

public int getSizeIndividuos () – Retorna o número de individuos do banco

public void actualizaIndividuo (int numRemovido) - Actualiza os individuos do banco

public void addConta (Conta conta) – Adiciona uma conta ao banco

public void removeConta (int index) – Remove uma conta do banco

public Conta getConta (int índex) – Retorna uma conta do banco

public int getSizeContas () – Retorna o número de contas do banco

Classe Administrativo – Classe representante dos administrativos

public Administrativo (String codigo , String password , String nome , String contacto) – Cria um administrativo com dados recebidos

Classe Gestor – classe representante dos gestores

public Gestor (String codigo , String password , String nome , String contacto) – Cria um gestor com dados recebidos

public void addConta (Conta conta) – Adiciona uma nova conta ao gestor

public void removeConta (int índex) – Remove uma conta do gestor

public Conta getConta (int índex) – Retorna uma conta do gestor

public int getSizeContas () – Retorna o número de contas do gestor

Classe Empresa – Classe representante das empresas

public Empresa (String codigo , String password , String nome , String contacto , String ramo) - Cria uma empresa com dados recebidos

public void setRamo (String ramo) – Modifica o ramo da empresa

… existem métodos set semelhantes para todas as variáveis da classe **Empresa**

public String getRamo () – Retorna o ramo da empresa

… existem métodos get semelhantes para todas as variáveis da classe **Empresa**

Classe Individuo – Classe representante dos indivíduos

public Individuo (String codigo , String password , String nome , String contacto , String profissão) – Cria um individuo com dados recebidos

public void setProfissao (String profissao) – Modifica a profissão do individuo

… existem métodos set semelhantes para todas as variáveis da classe **Individuo**

public String getProfissao () – Retorna a profissão do individuo

… existem métodos get semelhantes para todas as variáveis da classe **Individuo**

Classe Conta – Classe representante das contas

public Conta (String codCliente , String data , double saldo) – Cria uma conta com dados recebidos

public void setCodCliente (String newCodCliente) – modifica o código do cliente da conta

… existe um método semelhante para a variável saldo da classe **Conta**

public String getCodCliente () – Retorna o código do cliente

… existem métodos semelhantes para todas as variáveis da classe **Conta**

public void addMovimento (Movimento movimento) – adiciona um novo movimento à conta

public Movimento getMovimento (int índex) – retorna um movimento da conta

public int getSizeMovimentos () – retorna o número de movimentos da conta

Classe Funcionário – Classe que é usada como superclasse abstracta das classes **Administrativo** e **Gestor**

public Funcionario (String codigo, String password, String nome, String contacto) – Cria um funcionário com dados recebidos

public void setNome (String newNome) – Modifica o nome do funcionário

… existem métodos set semelhantes para todas as variáveis da classe **Funcionário**

public String getNome () – Retorna o nome do funcionário

… existem métodos get semelhantes para todas as variáveis da classe **Funcionário**

Classe Cliente – Classe que é usada como superclasse abstracta das classes **Empresa** e **Individuo**

public Cliente (String codigo, String password, String nome, String contacto) – Cria um cliente com dados recebidos

public void setNome (String newNome) – Modifica o nome do cliente

… existem métodos set semelhantes para todas as variáveis da classe **Cliente**

public String getNome () – Retorna o nome do cliente

… existem métodos get semelhantes para todas as variáveis da classe **Cliente**

Classe Movimento – Classe representante dos movimentos

public Movimento (double montante, String data, String tipo) – cria um movimento com dados recebidos

public double getMontante () – retorna o montante do movimento

… existem métodos get semelhantes para todas as variáveis da classe **Movimento**

Classe Interface – classe que gere a agência bancária possibilitando acções ao utilizador

public Interface () – cria uma interface vazia

public void clearScreen () – limpa o output

public void menuStart () – visualiza o menuStart (abrir novo banco, utilizar banco existente, sair) e processa a opção do utilizador

public void leDados () – abre um banco já existente dum ficheiro definido pelo utilizador

public void gravaDados () – grava um banco num ficheiro definido pelo utilizador

public void menuInicial () – visualiza o menuInicial (login, sair e gravar) e processa a opção do utilizador

public void menuLogin () – visualiza o menuLogin (introduzir código e password) e processa as opções do utilizador

public void menuDirector () – visualiza o menuDirector (adicionar e remover administrativo, voltar) e processa a opção do utilizador

public void menuAdministrativos (int numCodigo) – visualiza o nome e o código do administrativo e o menuAdministrativos (listar gerentes de conta e informação relativa ás contas associadas, alterar dados pessoais, adicionar e remover gestor, adicionar cliente, voltar) e processa a opção do utilizador

public void menuGestores (int numCodigo) – visualiza o nome e o código do gestor e o menuGestores (listar clientes e saldos de conta respectivos, alterar dados pessoais, voltar) e processa a opção do utilizador

public void menuClientesEmpresas (int numCodigo) – visualiza o nome e o código da empresa e o menuClientesEmpresas (levantar dinheiro, depositar dinheiro, requisitar cheques, cancelar conta, alterar gerente de conta, consultar saldo da conta, consultar movimentos a partir de uma determinada data, alterar dados pessoais, voltar) e processa a opção do utilizador

public void menuClientesIndividuos (int numCodigo) – visualiza o nome e o código do individuo e o menuClientesIndiviudos (levantar dinheiro, depositar dinheiro, requisitar cheques, cancelar conta, alterar gerente de conta, consultar saldo da conta, consultar movimentos a partir de uma determinada data, alterar dados pessoais, voltar) e processa a opção do utilizador

public void menuAdicionarAdministrativo () – visualiza o menuAdicionarAdministrativo (nome, contacto, password) e adiciona um administrativo ao banco com as opções do utilizador

public void menuAdicionarGestor () – visualiza o menuAdicionarGestor (nome, contacto, password) e adiciona um gestor ao banco com as opções do utilizador

public void menuAdicionarCliente () – visualiza o menuAdicionarCliente (empresa, individuo, voltar) e processa a opção do utilizador

public void menuAdiconarClienteEmpresa () – visualiza o menuAdicionarClienteEmpresa (nome, ramo, contacto, password) e adiciona uma empresa banco e uma conta ao respectivo gestor

public void menuAdiconarClienteIndividuo () - visualiza o menuAdicionarClienteIndividuo (nome, profissão, contacto, password) e adiciona um individuo ao banco e uma conta ao respectivo gestor

public Conta menuAdicionarConta (String codCliente) – visualiza o menuAdicionarConta (data de criação, saldo incial) e adiciona uma conta ao banco com as opções do utilizador

public void menuRemoverAdministrativo () – visualiza o menuRemoverAdministrativo (escolha do administrativo a remover) e processa a opção do utilizador

public void menuRemoverGestor (int numCodigoAdministrativo) – visualiza o menuRemoverGestor (escolha do gestor a remover) e processa a opção do utilizador

public void menuAlterarDadosAdministrativo (Administrativo administrativo) – visualiza o menuAlterarDadosAdministrativo (nome, contacto, password) e modifica o administrativo com as opções do utilizador

public void menuAlterarDadosGestor (Gestor gestor) – visualiza o menuAlterarDadosGestor (nome, contacto, password) e modifica o gestor com as opções do utilizador

public void menuAlterarDadosEmpresa (Empresa empresa) – visualiza o menuAlterarDadosEmpresa (nome, ramo, contacto, password) e modifica a empresa com as opções do utilizador

public void menuAlterarDadosIndividuo (Individuo individuo) – visualiza o menuAlterarDadosIndividuo (nome, profissão, contacto, password) e modifica o individuo com as opções do utilizador

public void removerConta (String codCliente) – remove uma conta do banco e do gestor correspondente

public void listarClientesSaldos (int numCodigoGestor) – lista os clientes e os saldos de conta respectivos

public void listarGestoresContas () – lista os gerentes de conta e informação relativa ás contas associadas

public void levantamento (Conta conta) – efectua um levantamento na conta

public void deposito (Conta conta) – efectua um depósito na conta

public void requisitarCheques (Conta conta) – efectua um levantamento de cheques na conta

public void alterarGestor (Conta conta) – altera o gerente da conta

public void consultarSaldo (Conta conta) – visualiza o saldo da conta

public void menuListarMovimentos (Conta conta) – visualiza o menuListarMovimentos (levantamentos, depósitos, todos) e processa a opção do utilizador

public void menuListarMovimentosLevantamentos (Conta conta) – visualiza a listagem dos levantamentos

public void menuListarMovimentosDepositos (Conta conta) – visualiza a listagem dos depósitos

public void menuListarMovimentosTodos (Conta conta) – visualiza a listagem dos levantamentos e depósitos

public static void main (String[] args) – método executável da classe **Interface**

- ## Comentários:

Neste trabalho o mais difícil foi começar… Antes de começarmos a escrever o código investimos algum tempo na estruturação do projecto do trabalho, o que mais tarde se revelou um factor importante no desenvolvimento do mesmo.

Como opções mais relevantes que tomamos na elaboração da estrutura destacam-se as seguintes:

- A identificação de cada utilizador através do seu código e da sua password
- O facto de associarmos a cada gestor uma lista de contas e não uma referência ao gestor nas contas;
- O facto de associarmos a cada conta uma lista de movimentos e não uma referência à conta nos movimentos;

- A distinção entre funcionários e clientes que permite uma maior aproximação da realidade do dia-a-dia;

• Conclusões e melhoramentos:

No final da implementação podemos verificar que a nossa estruturação foi bem projectada e que tudo funcionava conforme o previsto.

Uma das coisas que podia ser melhorada no nosso trabalho é a apresentação, usando para isso interface gráfica. Deste modo seria mais perceptível para o utilizador as apresentações dos menus e as opções a tomar.

• Referências:

Publicações:

C, How to Program, Third Edition
Deitel & Deitel
Prentice Hall

Recursos Web:

java.sun.com