Gestão de uma Instituição Bancária

Implementação de uma Solução em C++

6 de Novembro de 2011

Trabalho elaborado por:

André Freitas - ei10036 Cristiano Alves - ei10153 Vasco Gonçalves - ei10054

Índice

Índice de Ilustrações

Introdução

A gestão de uma instituição de grandes dimensões, com diversos serviços associados e recursos humanos, necessita de ser feita de um modo automatizado, eficaz e seguro, dada à enorme quantidade de dados a manipular. Assim sendo, faz todo o sentido recorrer a uma aplicação de *Software* robusta que seja fácil de usar, que não cause problemas e que saiba lidar com dados que sejam sensíveis no que toca à privacidade e à importância dos mesmos, de modo a não perdê-los e a não serem facilmente acedidos por terceiros.

Ora, uma Instituição Bancária lida com informações financeiras que se enquadram na área de dados muito sensíveis, daí que a solução a ser implementada tenha de funcionar em pleno, visto que os dados não podem ser corrompidos nem perdidos. Um Banco é constituído por Funcionários e Clientes, que podem ser particulares ou coletivos, que partilham informações comuns como o Nome, Data de Nascimento e afins. Esta entidade é ainda constituída por contas bancárias que podem ser a prazo e a ordem, onde se associa o histórico de todos os movimentos feitos pelos clientes.

Implementando uma solução em C++ recorremos ao conceito de programação orientada a objetos e, para salvar toda a informação manipulada, fizemos uso de ficheiros que são facilmente manipulados com a biblioteca *fstream* desta linguagem. A interface da aplicação de gestão do Banco assenta sobre o ambiente de linha de comandos e não sobre Caixas de Diálogo. Foi tido em conta a validação de dados como o NIF e NIB a fim de detectar fraudes nas operações por parte dos clientes, usando os algoritmos estipulados pela lei portuguesa e afins.

Implementação em C++

Tal como foi referido, a solução para a gestão da instituição bancária foi implementada na linguagem C++, que é uma linguagem muito flexível e eficaz para projetos que envolvam uma grande/média complexidade de especificação do mesmo. Na decisão das classes foram usados os conceitos de heranças e polimorfismo, inerentes da programação orientada a objetos nesta linguagem. Os erros durante a execução são tratados com exceções.

Brainstorming da Solução

Numa situação de desenvolvimento de código em equipa, uma abordagem *bottom-up* é extremamente eficaz na modularização. Foi feito um Brainstorming de como seria a estrutura da implementação em termos das entidades e sub-entidades.

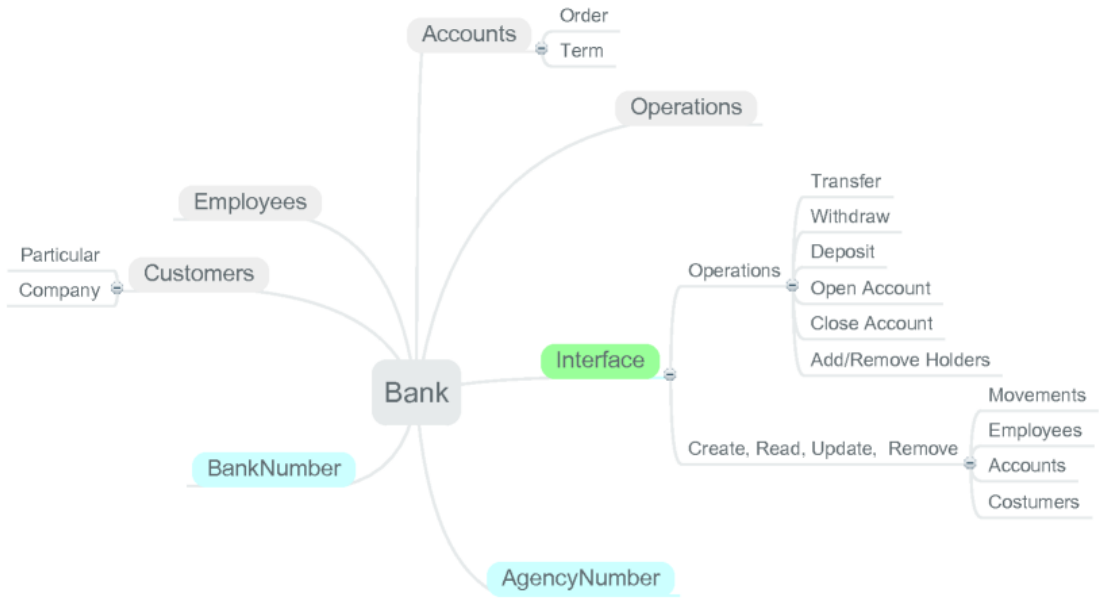


Ilustração 1 - Mapa de ideias do sistema de gestão do Banco

A partir deste mapa podemos identificar as principais entidades, como as contas bancárias, funcionários e clientes. Um Banco é sempre identificado pelo número de quatro dígitos relativo à instituição Bancária que representa mais outros quatro dígitos do número da Agência em questão. A interface necessita de ter as ações básicas de manipulação de dados de Criar, Ler, Actualizar e Eliminar, mais conhecida por *CRUD*. Precisa de também ter as operações habituais que temos num Banco que são: abrir e fechar uma conta; fazer levantamentos, depósitos e transferência; adicionar e remover titulares de uma conta.

Colocadas assim as ideias sobre a "mesa", é uma questão de as colocar em acção em C++, implementando tudo em classes, com a referida abordagem *bottom-up.*

Diagrama UML das Classes

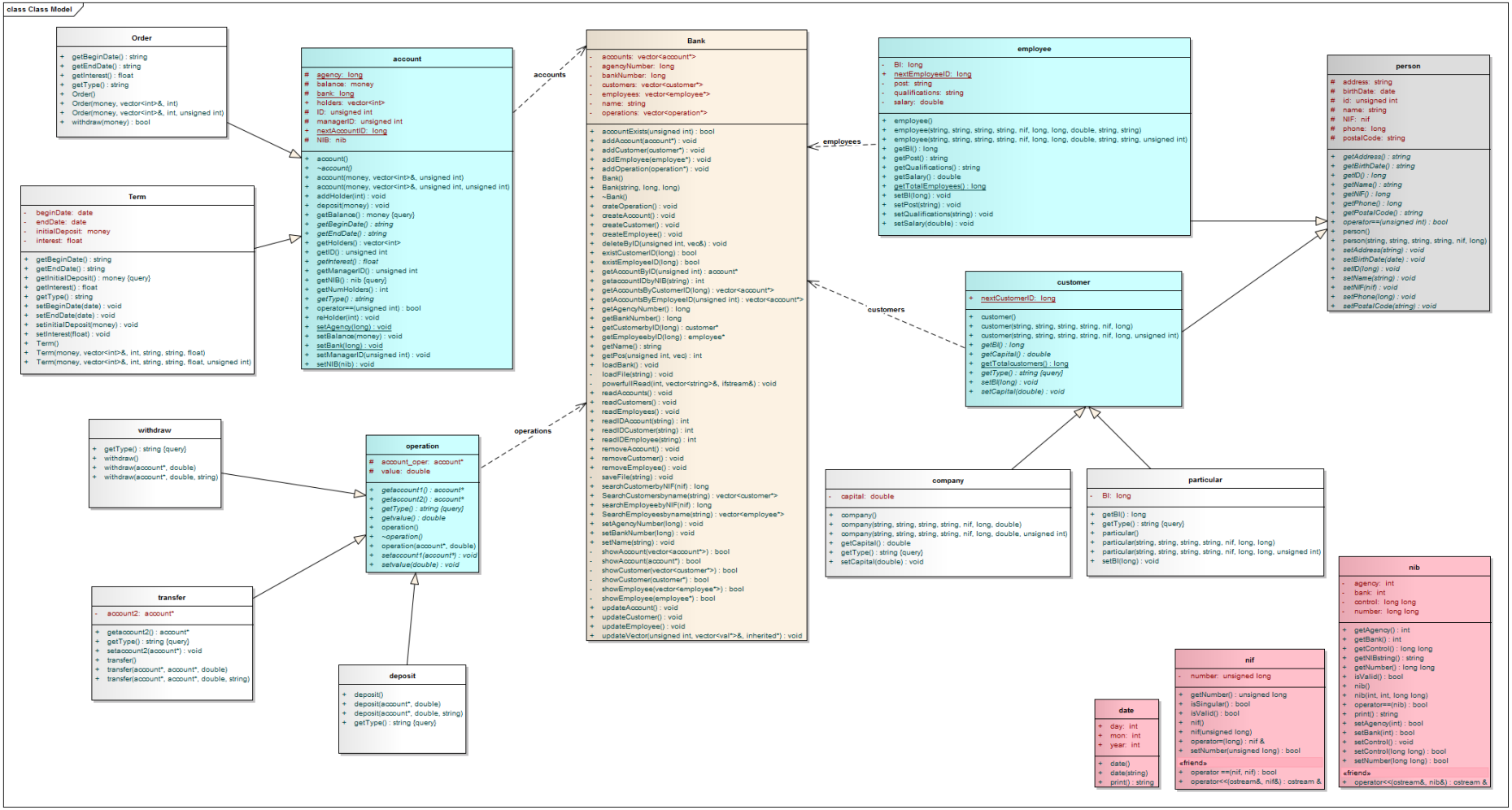


Ilustração 2 - Diagrama UML das classes em C++

Como podemos verificar, a complexidade da solução em termos de codificação é considerável, pelo que problemas de lógica e organização poderão surgir em certas alturas da implementação.

Descrição das Classes

CLASSE BANK

A classe *Bank* diz respeito à instituição Bancária que tem o seu número de identificação da entidade que representa e o número da agência. Esta classe é constituída por vectores de clientes, funcionários, contas e operações bancárias.

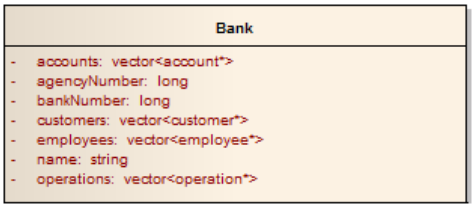


Ilustração 3 - Atributos da Classe Bank

CLASSE ACCOUNT

A classe *account* diz respeito a uma conta bancária que é constituída por um saldo, titulares, gestor de conta, o seu número de identificação e o seu NIB. São ainda definidas as variáveis estáticas para a agência e o número do banco, devido a serem necessárias para a instanciar a variável do NIB. Desta classe mãe, derivam as classes *Order*, que diz respeito às Contas à Ordem e a classe *Term*, que é uma conta a prazo que possui uma data de início e fim de contracto, o depósito inicial e o valor percentual dos juros. De notar que a class *Order* não acrescente mais atributos.

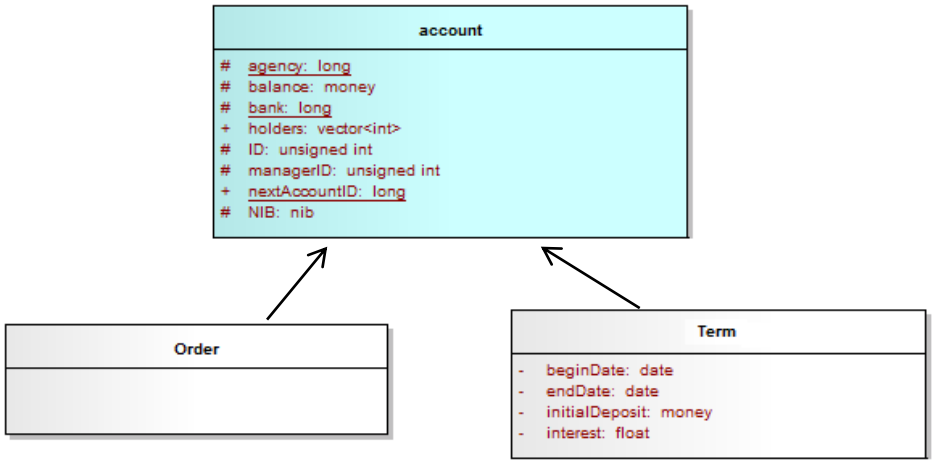


Ilustração 4 - Atributos da Classe *account* e das suas subclasses

CLASSE OPERATION

A classe *operation* descreve as diversas operações bancárias que um Banco está sujeito. Uma operação é constituída simplesmente pelo seu valor e a referência da conta sobre a qual a operação está a ser feita. Desta classe mãe, derivam as classes *transfer*, *deposit* e *withdraw*, que dizem respeito a uma transferência, depósito e levantamento. A classe *transfer* acrescenta a referência para a conta de destino do montante a ser transferido.

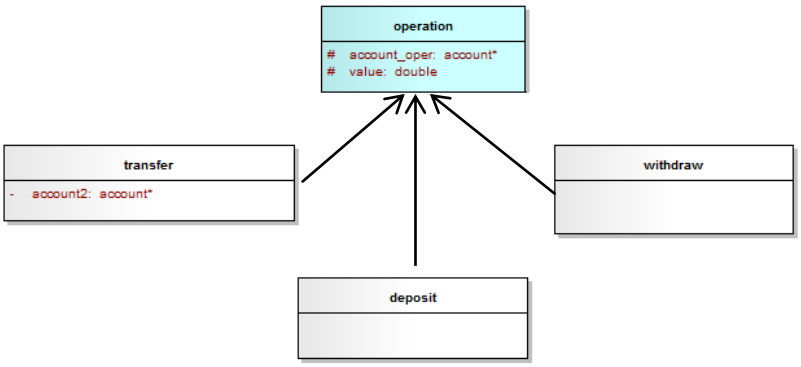


Ilustração 5 - Atributos da Classe *operation* e das suas subclasses

CLASSE PERSON

A classe *person* descreve os dados inerentes a uma pessoa/indivíduo, ou seja, morada, data de nascimento, a sua identificação, nome, contribuinte, telefone e código postal. Desta classe mãe derivam as classes *customer* e *employee*. Um cliente pode ser do tipo empresa ou particular, por isso, da classe *customer* ainda derivam as classes *company* e *particular*, que adicionam os atributos da capital e BI, respectivamente. Um funcionário tem ainda o seu cargo, qualificações, salário e o seu número de BI.

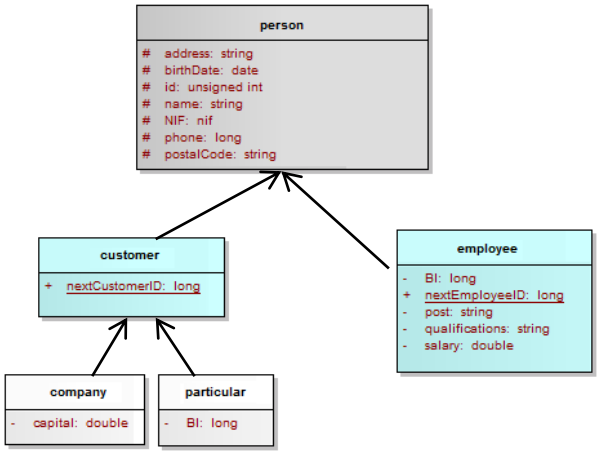


Ilustração 6 - Atributos da Classe *person* e das suas subclasses

CLASSES AUXILIARES

Como constatado anteriormente, foram criadas classes auxiliares, tendo em conta o rigor da especificação do projeto. Um objecto do tipo data pode ser estanciado com o argumento do tipo *string* da data: *data d1("21/10/1992")*, no formato *dd/mm/(aa)aa.* Já a implementação da classe *nif* permite detectar se o nif é válido ou não. No que toca ao *nib*, este é constituído por 4 campos, o banco, a agência, o número de conta e o dígito de controlo que é gerado a partir de um algoritmo conhecido. É possível também validar um número NIB com esta implementação da classe.

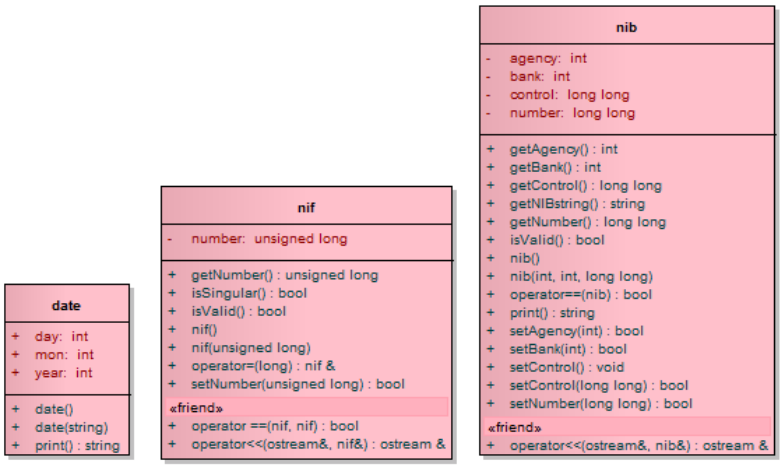


Ilustração 7 - Definição das Classes date, nif e nib

Tratamento de erros e exceções

O tratamento de situações que incorrem em erro quer em termos de lógica da implementação quer em termos da própria execução é importante. Assim sendo, implementamos as seguintes regras:

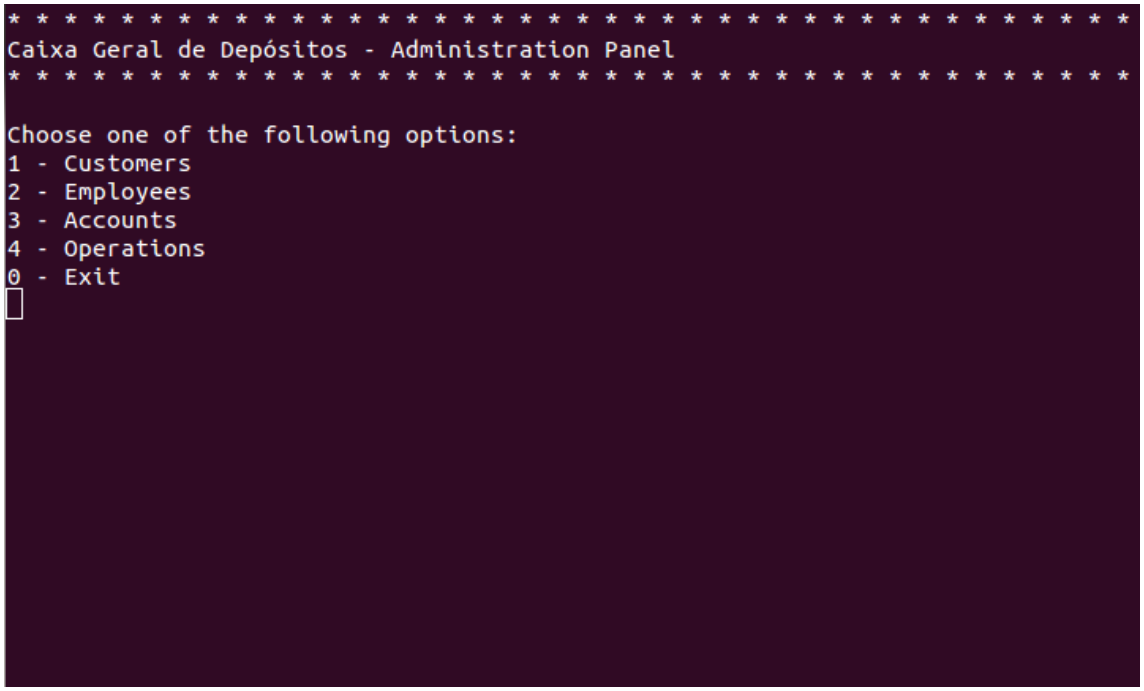
1. Não é possível criar contas enquanto não existir cliente e/ou empregados;
2. Quando se apaga um ou mais clientes que sejam os únicos titulares de uma conta esta também é apagada;
3. Ao apagar um empregado, se ele é responsável por alguma conta bancária, pede ao utilizador para escolher outro caso sobre mais do que um empregado, visto que só sobrar um este é automaticamente associado à conta. Porém, se ficar sem empregados, as contas ficam "sem empregado" e não é possível adicionar contas até ser adicionado um novo empregado;
4. Não é possível ter mais titulares nas contas bancárias do que o total de clientes existentes no banco;
5. Não é possível ter titulares repetidos.

Casos de Utilização

A Aplicação desenvolvida é compilada em Ambiente Windows e Linux, cumprindo assim os *standards* da linguagem C++.

Criação de um Banco

Menu Principal



Menu de Contas

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Account
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

Choose one of the following options:
1) List all Accounts
2) Add new Account
3) Edit Account
4) List operations of an account
0) Return to Menu
```

Menu de Clientes

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Customer
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

Choose one of the following options:
1) List all Customer
2) Add new customer
3) Edit customer
0) Return to Menu
```

Menu de Empregados

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Employee
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

Choose one of the following options:
1) List all Employee
2) Add new Employee
3) Edit Employee
0) Return to Menu
```

Menu de Operações

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Operations
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

Choose one of the following options:
1) Choose Operation
2) Historical List of operations
0) Return to Menu
```

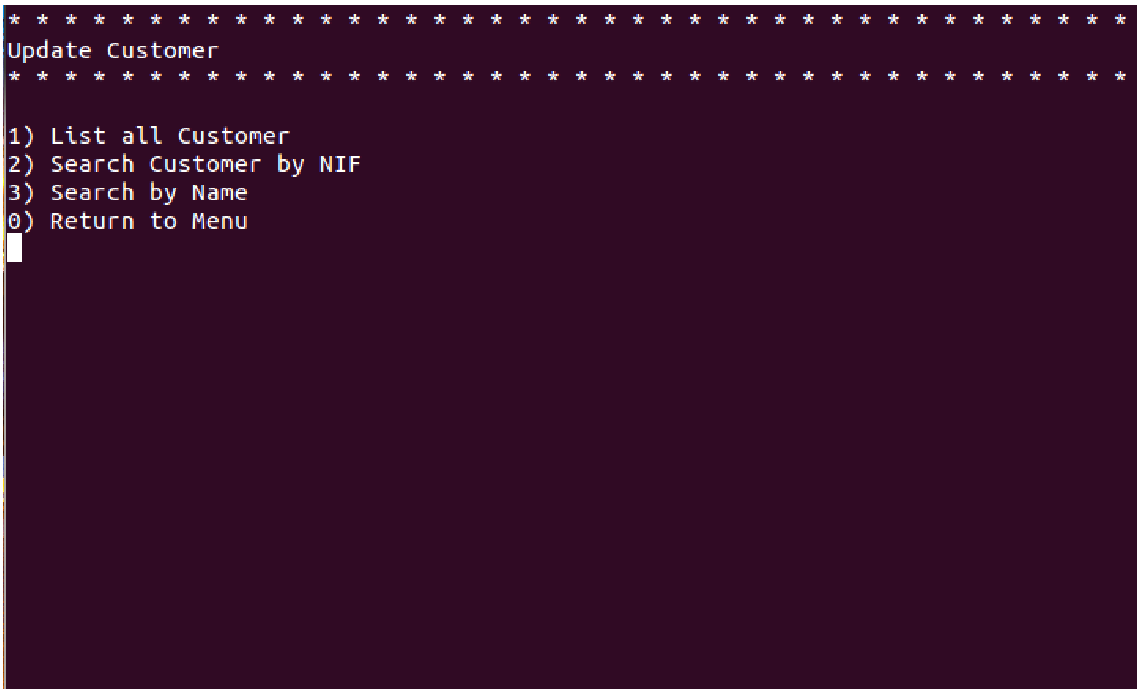
Criar Cliente

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Create Customer
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

1) Particular
2) Company
```

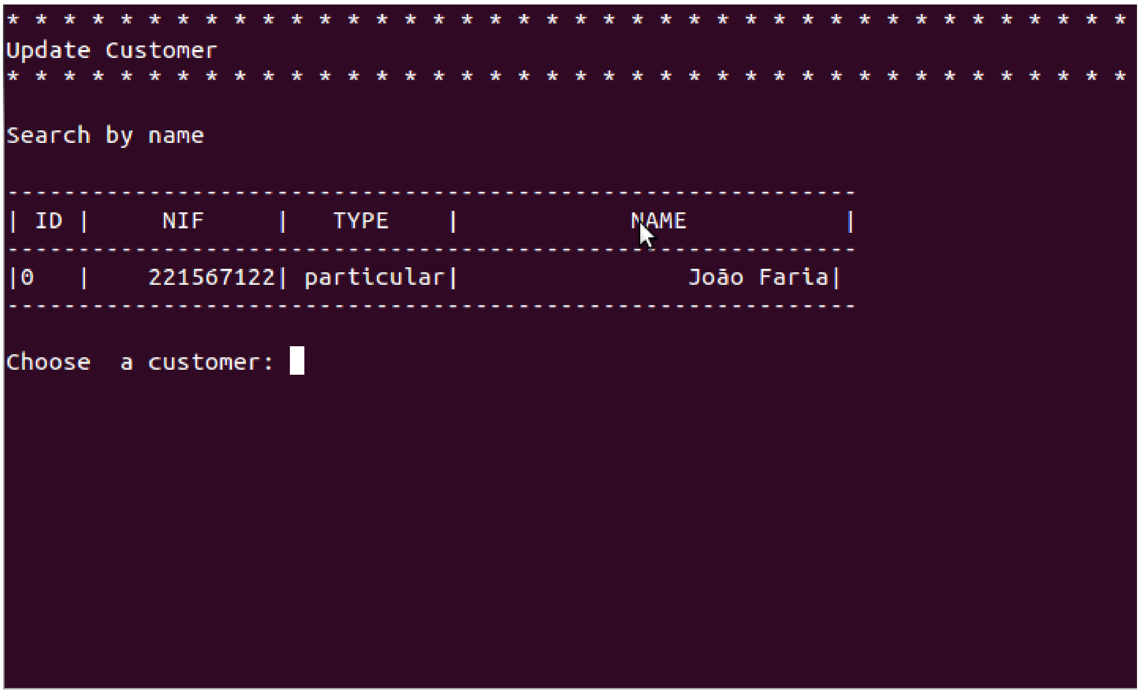
Listar Clientes

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
List Customer

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


-------------------------------------------------------------
| ID |     NIF     |   TYPE    |            NAME           |
-------------------------------------------------------------
|0   |    221567122| particular|                João Faria|
-------------------------------------------------------------
|1   |    123678123| particular|                Maria Alves|
-------------------------------------------------------------
|2   |    567890121|    company|         Vinhos e Uvas, LDA|
-------------------------------------------------------------
|3   |    241441451| particular|            Alberto Ferrais|
-------------------------------------------------------------

Press ENTER to continue
```

Atualizar Clientes



Pesquisa de Clientes



Atualizar Clientes (cont.)

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Update Customer
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* * * * * * * *
Customer Data *
* * * * * * * *

===============================================================================

Name: João Faria
Brirth Date: 21/7/1990
Adress: Rua da Alegria nº 12
Postal Code: 4567-123
NIF: 221567122
BI/CC: 14071124
===============================================================================

You really want to change this Customer?
1) Yes
2) No
```

Listar Empregados

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
List Employees

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


------------------------------------------------------------
| ID |      NIF      |    POST     |            NAME           |
------------------------------------------------------------
|0   |     234877123|Executive Chief|              João Navarro|
------------------------------------------------------------
|1   |     456890123|         CEO|               Carlos Manso|
------------------------------------------------------------

Press ENTER to continue
```

Listar Contas

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
List Accounts

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


-------------------------------------------------------------------------------------
| ID |           NIB            | BALANCE |     HOLDERS      |     MANAGER     |
-------------------------------------------------------------------------------------
|1   |0036 0010 0000 0000 0017 6|  2500.23|     Maria Alberta|     Carlos Manso|
-------------------------------------------------------------------------------------
|2   |0036 0010 0000 0000 0027 5|   200.23|     Maria Alberta|     Carlos Manso|
-------------------------------------------------------------------------------------
|4   |0036 0010 0000 0000 0047 3|  4500.12|Vinhos e Uvas, LDA|    João Navarro|-
-------------------------------------------------------------------------------------
Press ENTER to continue
```

Depósito

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Deposit
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Amount to transfer: 2400.23
```

Levantamento

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Withdraw
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Amount to transfer: 1121211
The withdraw can not be executed because there are insufficient funds
Press ENTER to continue
```

Listar Operações

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
List Operations

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


----------------------------------------------------------------------------------
|        DESTINATION         |           SOURCE           |   VALUE    |    TYPE    |
----------------------------------------------------------------------------------
|0036 0010 0000 0000 0017 6|--------------------------|          12|     deposit|
----------------------------------------------------------------------------------
|0036 0010 0000 0000 0027 5|--------------------------|         121|     deposit|
----------------------------------------------------------------------------------
|0036 0010 0000 0000 0017 6|0036 0010 0000 0000 0027 5|         200|    transfer|
----------------------------------------------------------------------------------
Press ENTER to continue
```

Progressão do Projeto

Consideramos que a elaboração deste projeto foi fundamental para a correta aplicação dos conhecimentos adquiridos até agora numa componente mais prática. Como em todos os projetos de grupo foi difícil repartir as tarefas entre os membros do grupo, devido à complexidade do mesmo. Contudo, esta barreira foi superada e pensamos ter conseguido que todos os membros trabalhassem com afinco e dedicação.

Sentimos mais dificuldade no desenvolvimento da aplicação na escrita e leitura de clientes, visto que existem clientes com tipos de dados variáveis. Esta dificuldade foi igualmente sentida de modo análogo na manipulação das contas nos ficheiros. É uma fase do projeto que envolveu muito depuramento para assegurar que não iria comprometer o futuro do trabalho.

O trabalho foi distribuído da seguinte forma:

Cristiano Alves:

- Interface do utilizador;
- Desenvolvimento das classes;
- Colaboração no Relatório;
- Colaboração no Diagrama.

André Freitas:

- Criação dos esqueletos das classes;
- Documentação;
- Desenvolvimento do diagrama de classes;
- Desenvolvimento do relatório

Vasco Gonçalves:

- Implementação de funções nas classes;
- Implementação da gravação e leitura de ficheiros;
- Documentação;
- Desenvolvimento do diagrama de classes;
- Desenvolvimento do relatório.

Em suma, a progressão do projeto foi faseada e sempre ponderada, eliminando sempre problemas de partes que iria suportar o desenvolvimento final da aplicação.

Conclusão

Com este trabalho sentimos a dificuldade de desenvolver uma aplicação que lide com dados importantes e privados. Foi difícil para nós implementar as soluções tendo em conta tanto a facilidade de utilização como a segurança nos dados. Ao utilizarmos ficheiros temporários pensamos ter criado uma nova camada de segurança na utilização deste programa, mas admitimos que este não é perfeito pois poderíamos ter implementado mais medidas de segurança como encriptação dos dados.

Consideramos que o projeto cumpre com todos os pontos solicitados e supera ainda muitos outros problemas que não eram referidos mas que foram encontrados ao longo do desenvolvimento do mesmo.

Anexos

Código-Fonte dos Testes em Cute

O cute é um *plugin* do eclipse que permite efetuar testes unitários do código desenvolvido. Assim sendo, em anexo segue os testes de algumas classes:

```
#include "cute.h"
#include "ide_listener.h"
#include "cute_runner.h"
// Libraries
#include "bank.h"
#include <iostream>
using namespace std;
void testCustomer() {
	vector <customer*> clientes;
	// -> Add a particular object
	particular
p1("André","21/10/1992","Funchal","9300",240446941,33643,291944026);
	clientes.push_back(&p1);
	ASSERT_EQUAL("André",clientes[0]->getName());
	ASSERT_EQUAL("Funchal",clientes[0]->getAddress());
	ASSERT_EQUAL("9300",clientes[0]->getPostalCode());
	ASSERT_EQUAL(291944026,clientes[0]->getPhone());
	ASSERT_EQUAL("21/10/1992",clientes[0]->getBirthDate());
	ASSERT_EQUAL(33643,clientes[0]->getBI());
	ASSERT_EQUAL(0,clientes[0]->getID());
	// -> Add a company object
	company c1("ACM,
LDA","1/1/1942","Funchal","9300",140426941,222888233,5000.0);
	clientes.push_back(&c1);
	ASSERT_EQUAL("1/1/1942",clientes[1]->getBirthDate());
	ASSERT_EQUAL(5000.0,clientes[1]->getCapital());
	ASSERT_EQUAL(1,clientes[1]->getID());
}

void testEmployee(){
	vector<employee*> emplo;
	// -> First employee
	employee e1("Carlos","26/3/1945","Porto","4435-056",162895513,
1872794123, 222323123, 800.00, "Worker","Secundario");
	emplo.push_back(&e1);
	ASSERT_EQUAL(emplo[0]->getName(),"Carlos");
	ASSERT_EQUAL(emplo[0]->getBirthDate(),"26/3/1945");
	ASSERT_EQUAL(emplo[0]->getAddress(),"Porto");
	ASSERT_EQUAL(emplo[0]->getPostalCode(),"4435-056");
	ASSERT_EQUAL(emplo[0]->getNIF(),162895513);
	ASSERT_EQUAL(emplo[0]->getBI(),1872794123);
	ASSERT_EQUAL(emplo[0]->getPhone(),222323123);
	ASSERT_EQUAL(emplo[0]->getPost(),"Worker");
	ASSERT_EQUAL(emplo[0]->getQualifications(),"Secundario");
	ASSERT_EQUAL(emplo[0]->getSalary(),800.0);
	employee e2("Juan","26/1/1965","Braga","4535-056",162552423, 1424123,
2222423, 1100.00, "Officer","Mestrado");
	emplo.push_back(&e2);
	ASSERT_EQUAL(emplo[1]->getID(),1);
```

```
}
void testAccounts(){
	// To show how it works
	money m1=30.0;
	ASSERT_EQUAL(m1,30.0);
	vector <account*> acc;
	// Set the static bank and agency variables
	account::setBank(2121);
	account::setAgency(1461);
	vector<int> holders1;
	holders1.push_back(1);
	holders1.push_back(6);
	// ************* Create an order account
	Order o1(200.0,holders1,10);
	acc.push_back(&o1);
	// Test the nib generation
	ASSERT_EQUAL(acc[0]->getNIB().getAgency(),1461);
	ASSERT_EQUAL(acc[0]->getNIB().getBank(),2121);
	ASSERT_EQUAL(acc[0]->getNIB().print(),"2121 1461 0000 0000 0017 0");
	// Test members
	ASSERT_EQUAL(acc[0]->getNumHolders(),2);
	ASSERT_EQUAL(acc[0]->getManagerID(),10);
	ASSERT_EQUAL(acc[0]->getManagerID(),10);
	ASSERT_EQUAL(acc[0]->getBalance(),200.0);
	acc[0]->deposit(23.0);
	ASSERT_EQUAL(acc[0]->getBalance(),223.0);
	// ************* Create an term account
	Term t1(100.0,holders1,21,"2/10/2000","2/10/2012",0.05);
	acc.push_back(&t1);
	ASSERT_EQUAL(acc[1]->getBeginDate(),"2/10/2000");
	ASSERT_EQUAL(acc[1]->getEndDate(),"2/10/2012");
	ASSERT_EQUAL(acc[1]->getInterest(),0.05);
}

void runSuite(){
	cute::suite s;
	s.push_back(CUTE(testCustomer));
	s.push_back(CUTE(testEmployee));
	s.push_back(CUTE(testAccounts));
	cute::ide_listener lis;
	cute::makeRunner(lis)(s, "The Suite");
}

int main(){
    runSuite();
    return 0;
}
```

Bibliografia

Não foram utilizadas quaisquer referências bibliográficas para a elaboração deste relatório.