FACULDADE
DE CIÊNCIAS
UNIVERSIDADE DE LISBOA

Relatório

Análise e Desenho de Software

Grupo ads004

Alessandro Melo - 42530
Daniel Marques - 42161
Márcio Domingues - 42555

2013 - 2014

Índice

Introdução

Foi decidido desenvolver o caso de uso “abrir ordem” pois as ordens são a parte fulcral de todo o sistema.

Decidiu-se implementar todas as classes auxiliares, mas necessárias, à funcionalidade abrir ordem com os serviços mínimos por esta requiridos.
Optou-se por dar um maior ênfase à camada de persistência, através da implementação de dois modos de persistência em ficheiro e em base de dados.
A decisão de usar bases de dados para a persistência prende-se sobretudo de uma visão para este projecto como um sistema distribuído, como foi explanado no relatório inicial. Se este sistema implementar dois serviços um no terminal de carga e outro na área de secretariado seria mais difícil manter a consistência de dados entre os dois serviços através de ficheiros. Problema que seria minortado através de um servidor de base de dados partilhado por ambos os serviços.

Para programar este cenário, a nossa interface mostra o formulário de novas ordens e a lista de ordens no sistema, tudo na mesma janela, desviando-se do ambiente distribuido apenas para simplificar o cenário e para mostrar feedback ao utilizador sobre a persistencia dos dados. Num cenário realista, numa fase mais avançada da implementação, a lista de ordens seria actualizada em todos os terminais ligados ao sistema.
Como já foi referido, nem todas as classes usadas presentes no modelo de desenho estão completas pois não era necessário incluir toda a sua funcionalidade para este particular cenário

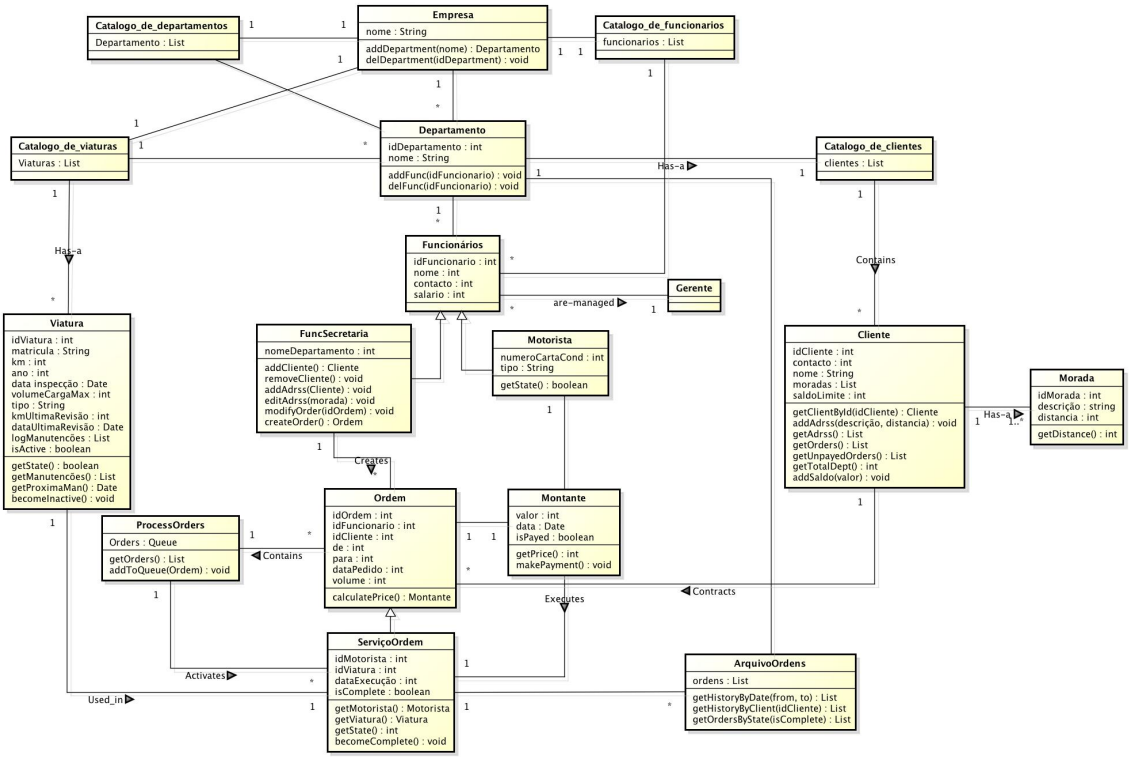
Decisões Tomadas

Talvez a principal decisão digna de nota seja o modo de simulação do calculo das distancias entre as moradas e a firma. Foi utilizado o tamanho da *string* usada para descrever a morada para simular o valor da distancia, deste modo dois clientes com a mesma morada teráo sempre o mesmo valor para a distancia.

Dois aspectos do projecto que deveriam ter sido implementados de outra forma e que teriam que ser modificados numa fase seguinte, seriam um *script* de auto-incrementação para o valor de identificação da ordem. Que no caso da persistência em base de dados seria fácil pois o MySQL pode ser defido para o gerar automaticamente.
O mosso projecto foi implementado de modo a que não seja possível criar ordens com o mesmo identificador mas não obriga, nesta fase, a uma sequencialização.
Outro decisão de melhoria no projecto seria uma maior separação entre as camadas, principalmente entre as camadas de interface com utilizador e de domínio, Numa próxima fase deveriam ser criadas classes de controlo que geriam pedidos entre camadas, implementando um padrão MVC.
Para o GUI faltou igualmente redefinir o método RowSorter de modo a que a tabela consegui-se lidar com a ordenação de números inteiros.

Modelo de Desenho Inicial

(para mais detalhe ver anexo)



Para o caso de uso abrir ordem foi decidido usar apenas as seguintes classes do domínio:

- FuncSecretaria
- Ordem
- ProcessOrders
- Cliente
- Morada

Camada de Persistencia

Para a camada de persistência tomou-se a decisão de implementar dois tipos de persistência uma usando ficheiros de formato CVS e outra usando base de dados MySQL.

Para isso foi criado um interface Persistence onde são definidos os métodos a utilizar pela classes que o implementam. E foram criadas duas classes:

- FilePersistence
- sqlPersistence

Estas implementam os métodos:

- getOrdens, utilizado para retirar a informação guardada no inicio do programa.
- addOrdem, que trata de salvar uma nova ordem criada pelo utilizador.

Um utilizador pode seleccionar que método de persistência a utilizar.

Notas sobre Persistencia

FilePersistence:

A persistência em ficheiro é feita num ficheiro que acompanha o projecto de *eclipse* com o nome Ordens.cvs

sqlPersistence:

Setup do JDBC no Eclipse

Ir buscar o JDBC Connector
http://dev.mysql.com/downloads/connector/j/

Num Mac
configure the build path in Eclipse:

- In Eclipse Right click on your Project folder
- Click Build Path and Configure Build Path
- Click the Library tab
- Click Add External jars and locate the file named mysql-connector-java-5.1.19-bin.jar

Outro modo
copiar o ficheiro .jar para a directoria /System/Library/Java/Extensions.

Num Pc
Adicionar ao ClassPath

- Open Control Panel and double click on System
- Click the Advanced Tab
- Click Environment Variables
- At the end of the classpath add ;c:\the location of mysql-connector-java-5.1.19-bin.jar

Setup do MySQL

Instalação
seguir instruções em http://dev.mysql.com/usingmysql/get_started.html

Criação e Manipulacao da Base de Dados

setup root password pela primeira vez
$ mysqladmin -u root password NEWPASSWORD
modificar password
$mysqladmin -u root -p'oldpassword' password NEWPASSWORD

(NEWPASSWORD tera que ser igual a marcio1981 porque esta hardcoded no programa java)

ou então

Correr o Mysql com as credenciais ja existentes
Criar a base de dados "samp_db"
mysql> CREATE DATABASE samp_db;
dar premicoes ao utilizador "root" com a password "marcio1981"
mysql> GRANT ALL ON samp_db.* TO root@localhost IDENTIFIED BY 'marcio1981';
testar com
$ mysql -u root -p marcio1981;

(ver http://www.cyberciti.biz/faq/mysql-user-creation/)

Criar table na base de dados

```
CREATE TABLE ordens
(
        idOrdem int NOT NULL UNIQUE,
        idFuncionario int NOT NULL,
        idCliente    int NOT NULL,
        origem      VARCHAR(40) NOT NULL,
        destino     VARCHAR(40) NOT NULL,
        dataPedido     DATE NOT NULL,
        volume      int NOT NULL,
        PRIMARY KEY (idOrdem)
);
```

Se preferir ter alguns dados iniciais na tabela criada.

INSERT INTO ordens (idOrdem, idFuncionario,idCliente, origem, destino, dataPedido, volume) VALUES (1, '123456789','987654321','rua valor sul n2','rua valor sul n32', '1732-02-22','12');

INSERT INTO ordens (idOrdem, idFuncionario,idCliente, origem, destino, dataPedido, volume) VALUES (2, '123456789','987654321','rua valor sul n32','rua arrabida centro n16','1735-10-30','57');

INSERT INTO ordens (idOrdem, idFuncionario,idCliente, origem, destino, dataPedido, volume) VALUES (3, '987654321','123456789','rua arrabida norte n20','rua valor sul n2','1743-04-13','345');

INSERT INTO ordens (idOrdem, idFuncionario,idCliente, origem, destino, dataPedido, volume) VALUES (4, '987654321','123456789','rua venda do pinheiro n30','rua arrabida norte n20','1751-03-16','600');

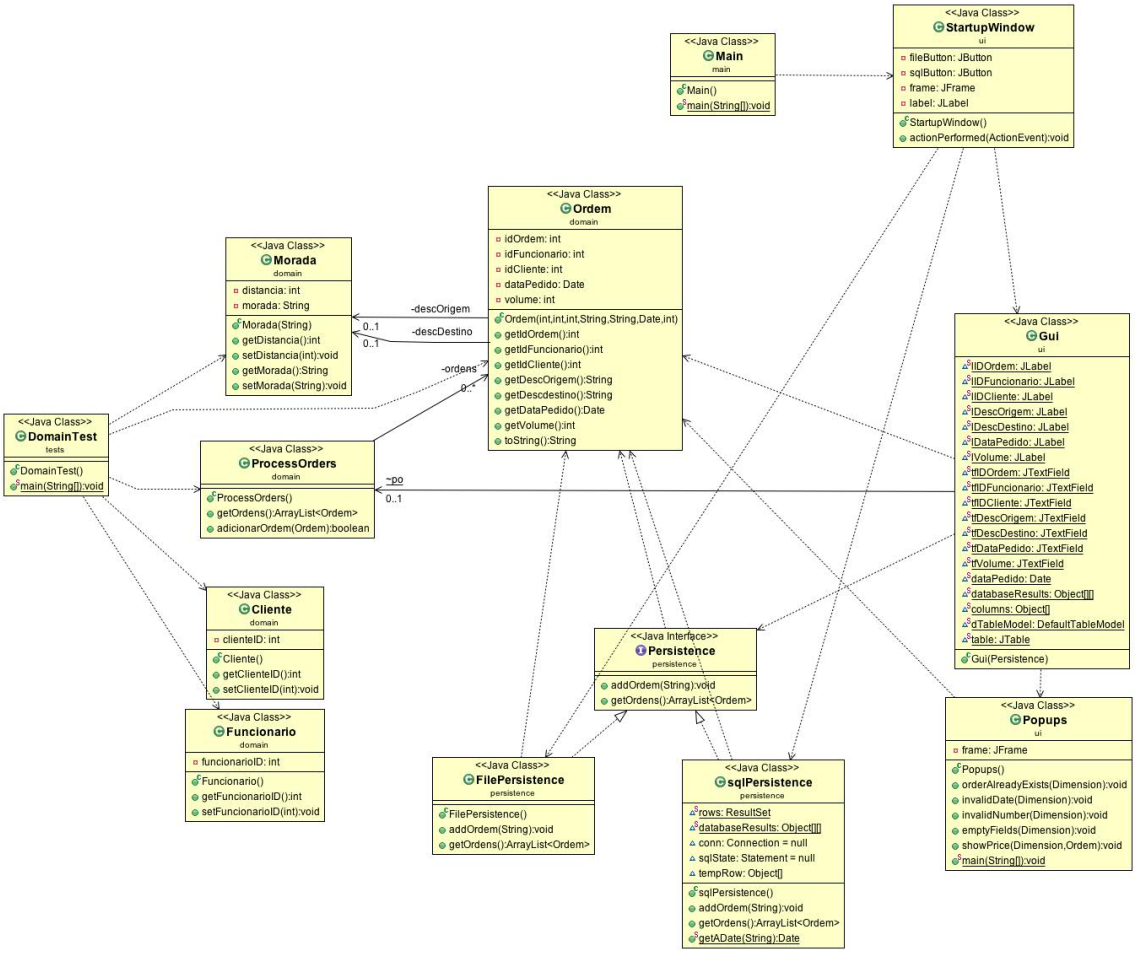
Camada de Apresentação

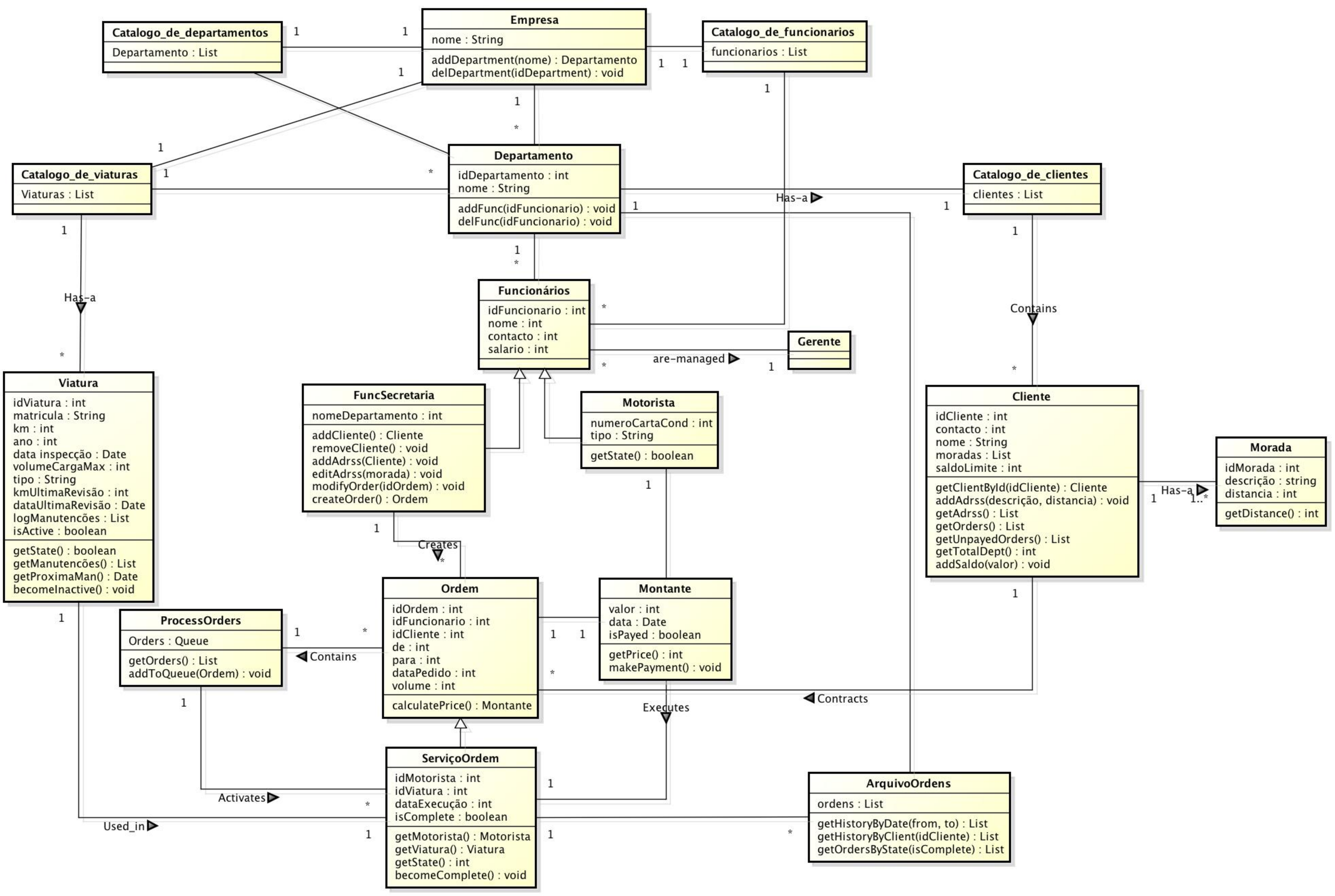
A camada de apresentação foi realizada em ambiente gráfico Java Swing

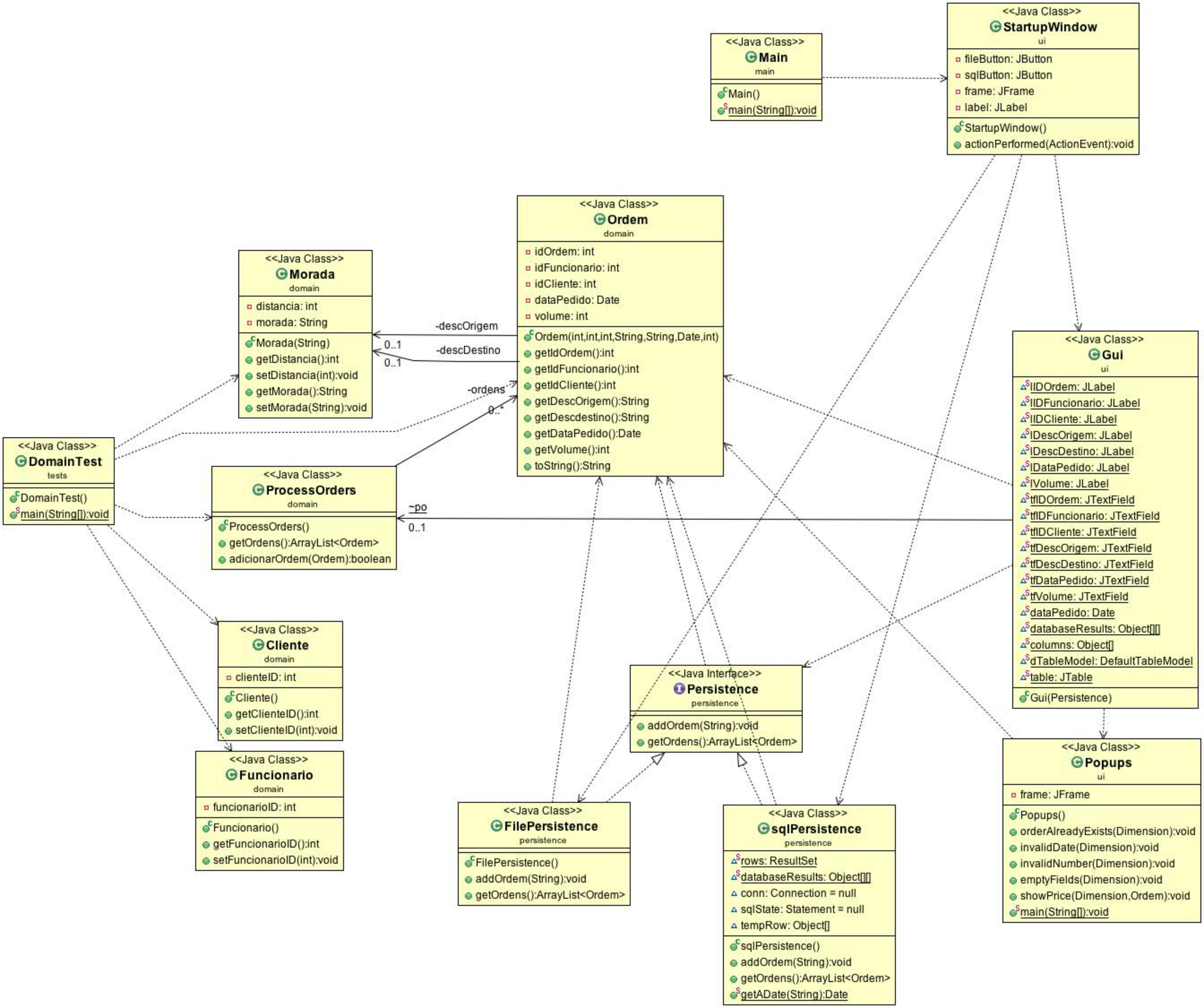
Foram criadas 3 classes:
Gui, responsável por toda a interacção com o utilizador quando este realiza operações dentro do âmbito do caso de uso escolhido.
Popups, responsável por todas as janelas de aviso em caso de confirmações ou erros de introdução de valores.
StartupWindow, responsável pela janela de arranque do ambiente gráfico onde é dado a escolher ao utilizador o modelo de persistência a utilizar.

Diagrama de Classes Implementado

(para mais detalhe ver anexo)




Catalogo_de_departamentos
Departamento : List
Empresa
nome : String
addDepartment(nome) : Departamento
delDepartment(idDepartment) : void
Catalogo_de_funcionarios
funcionarios : List
Catalogo_de_viaturas
Viaturas : List
Departamento
idDepartamento : int
nome : String
addFunc(idFuncionario) : void
delFunc(idFuncionario) : void
Has-a
Catalogo_de_clientes
clientes : List
Funcionários
idFuncionario : int
nome : int
contacto : int
salario : int
Gerente
are-managed
Has-a
Contains
Viatura
idViatura : int
matricula : String
km : int
ano : int
data inspecção : Date
volumeCargaMax : int
tipo : String
kmUltimaRevisão : int
dataUltimaRevisão : Date
logManutencões : List
isActive : boolean
getState() : boolean
getManutencões() : List
getProximaMan() : Date
becomeInactive() : void
FuncSecretaria
nomeDepartamento : int
addCliente() : Cliente
removeCliente() : void
addAdrss(Cliente) : void
editAdrss(morada) : void
modifyOrder(idOrdem) : void
createOrder() : Ordem
Motorista
numeroCartaCond : int
tipo : String
getState() : boolean
Creates
Cliente
idCliente : int
contacto : int
nome : String
moradas : List
saldoLimite : int
getClientById(idCliente) : Cliente
addAdrss(descrição, distancia) : void
getAdrss() : List
getOrders() : List
getUnpayedOrders() : List
getTotalDept() : int
addSaldo(valor) : void
Has-a
Morada
idMorada : int
descrição : string
distancia : int
getDistance() : int
Ordem
idOrdem : int
idFuncionario : int
idCliente : int
de : int
para : int
dataPedido : int
volume : int
calculatePrice() : Montante
Montante
valor : int
data : Date
isPayed : boolean
getPrice() : int
makePayment() : void
ProcessOrders
Orders : Queue
getOrders() : List
addToQueue(Ordem) : void
Contains
Executes
Contracts
Activates
Used_in
ServiçoOrdem
idMotorista : int
idViatura : int
dataExecução : int
isComplete : boolean
getMotorista() : Motorista
getViatura() : Viatura
getState() : int
becomeComplete() : void
ArquivoOrdens
ordens : List
getHistoryByDate(from, to) : List
getHistoryByClient(idCliente) : List
getOrdersByState(isComplete) : List


<<Java Class>>
Main
main
Main()
main(String[]):void
<<Java Class>>
StartupWindow
ui
fileButton: JButton
sqlButton: JButton
frame: JFrame
label: JLabel
StartupWindow()
actionPerformed(ActionEvent):void
<<Java Class>>
Ordem
domain
idOrdem: int
idFuncionario: int
idCliente: int
dataPedido: Date
volume: int
Ordem(int,int,int,String,String,Date,int)
getIdOrdem():int
getIdFuncionario():int
getIdCliente():int
getDescOrigem():String
getDescdestino():String
getDataPedido():Date
getVolume():int
toString():String
<<Java Class>>
Morada
domain
distancia: int
morada: String
Morada(String)
getDistancia():int
setDistancia(int):void
getMorada():String
setMorada(String):void
-descOrigem
0..1
-descDestino
0..1
-ordens
0..*
<<Java Class>>
Gui
ui
lIDOrdem: JLabel
lIDFuncionario: JLabel
lIDCliente: JLabel
lDescOrigem: JLabel
lDescDestino: JLabel
lDataPedido: JLabel
lVolume: JLabel
tfIDOrdem: JTextField
tfIDFuncionario: JTextField
tfIDCliente: JTextField
tfDescOrigem: JTextField
tfDescDestino: JTextField
tfDataPedido: JTextField
tfVolume: JTextField
dataPedido: Date
databaseResults: Object[][]
columns: Object[]
dTableModel: DefaultTableModel
table: JTable
Gui(Persistence)
<<Java Class>>
DomainTest
tests
DomainTest()
main(String[]):void
<<Java Class>>
ProcessOrders
domain
ProcessOrders()
getOrdens():ArrayList<Ordem>
adicionarOrdem(Ordem):boolean
~po
0..1
<<Java Class>>
Cliente
domain
clienteID: int
Cliente()
getClienteID():int
setClienteID(int):void
<<Java Class>>
Funcionario
domain
funcionarioID: int
Funcionario()
getFuncionarioID():int
setFuncionarioID(int):void
<<Java Interface>>
Persistence
persistence
addOrdem(String):void
getOrdens():ArrayList<Ordem>
<<Java Class>>
FilePersistence
persistence
FilePersistence()
addOrdem(String):void
getOrdens():ArrayList<Ordem>
<<Java Class>>
sqlPersistence
persistence
rows: ResultSet
databaseResults: Object[][]
conn: Connection = null
sqlState: Statement = null
tempRow: Object[]
sqlPersistence()
addOrdem(String):void
getOrdens():ArrayList<Ordem>
getADate(String):Date
<<Java Class>>
Popups
ui
frame: JFrame
Popups()
orderAlreadyExists(Dimension):void
invalidDate(Dimension):void
invalidNumber(Dimension):void
emptyFields(Dimension):void
showPrice(Dimension,Ordem):void
main(String[]):void