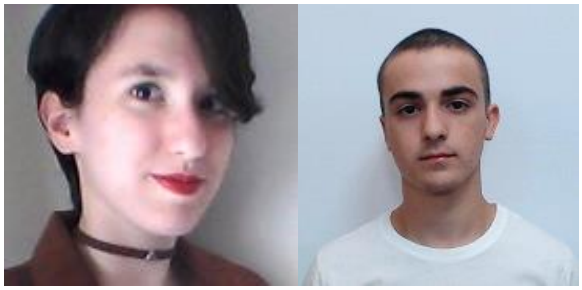
APLICAÇÃO TRAZAQUI!: DESENVOLVIMENTO

PROGRAMAÇÃO ORIENTADA AOS OBJETOS LCC/2020

GRUPO 23
BÁRBARA FARIA A85774
JOSÉ LUÍS PIRES A84552



Índice

0. Introdução

No âmbito da cadeira de Programação Orientada aos Objetos, integrante no programa do 2º ano da Licenciatura em Ciências da Computação, foi desenvolvida a aplicação TrazAqui!. Esta é um resultado de todo o conhecimento adquirido ao longo do semestre por parte dos membros da equipa, sempre baseado nos tópicos abordados durante as aulas teóricas e práticas lecionadas. O apoio dos docentes no esclarecimento de algumas dúvidas e disponibilização de algum material, mais especificamente ficheiros de log, provou-se vital à interpretação final do enunciado dado.

1. Visão Geral

TrazAqui! tem como objetivo de base o acesso a um serviço de entregas de encomendas. Todas as entidades constituintes neste processo devem conseguir comunicar entre si, para além de lhes serem apresentadas diferentes funções neste sistema de distribuição, consoante o seu cargo.

Destacamos desde já que apesar, da presença das variáveis e métodos com visa ao reconhecimento de Encomendas Médicas na nossa versão inicial da aplicação, estes nunca chegaram a mostrar-se funcionais, e, portanto, após desenvolvimento posterior do código, foi concluído que era desnecessária a sua presença no final. A sua adição levaria a alteração de grandes porções do código que, no grande esquema do trabalho, não afetaria em grande o seu funcionamento. Daí a sua referência ter sido removida nas classes delas dependentes: Encomenda, Voluntarios e Empresa.

Será de seguida efetuada uma análise mais profunda a cada uma das classes presentes no API desenvolvido e suas funcionalidades implementadas.

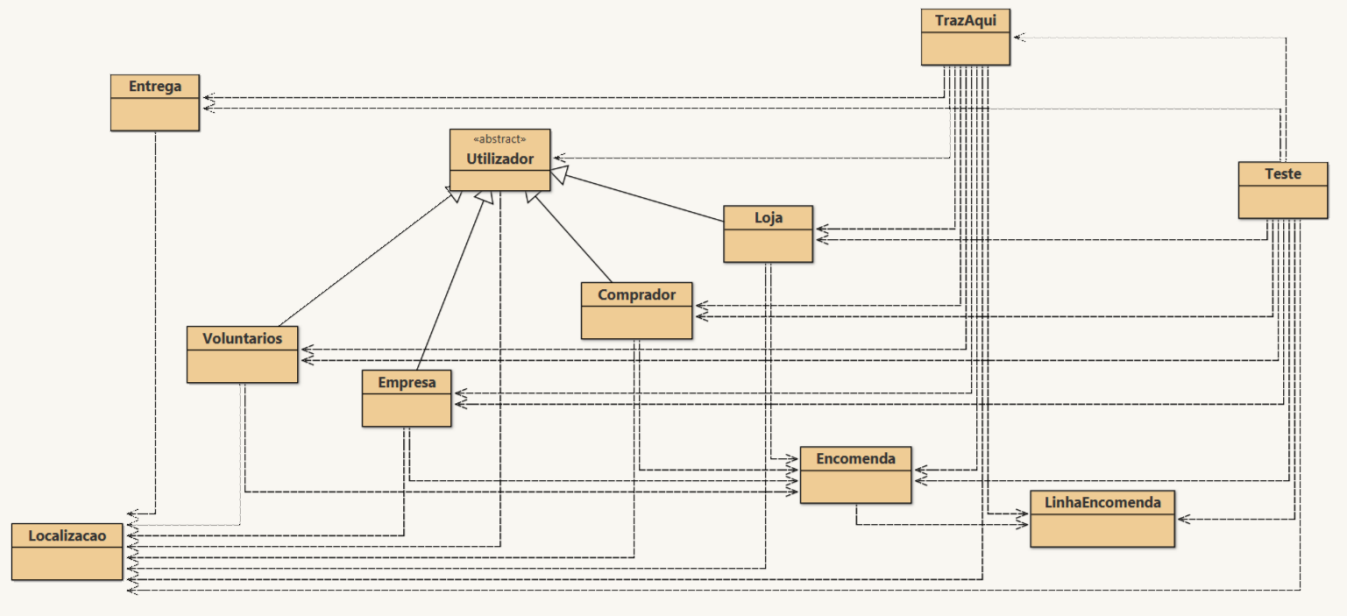


Figura 1 Visão geral de TrazAqui! através do programa de desenvolvimento Java, BlueJ.

2. As Classes

2.1. Utilizador

Após leitura atenta do enunciado, foi concluído que todo o Comprador, Voluntário, Loja e Empresa ligados ao app não são mais do que utilizadores "especiais" da aplicação, apresentando um papel essencial mas distinto no funcionamento de todo o processo de entrega. Assim, definimos a superclasse Utilizador, na qual todas as entidades anteriormente referidas terão acesso para informações de mesma natureza, como email, palavra passe, nome e coordenadas atuais. A sua individualidade será destacada na sua subclasse respetiva.

```
public class Comprador extends Utilizador
public class Voluntarios extends Utilizador
public class Loja extends Utilizador
public class Empresa extends Utilizador
```

Figura 2 Implementação de Utilizador como superclasse de Comprador, Voluntarios, Loja e Empresa.

Temos, portanto, um estabelecimento de uma relação hierárquica de herança em que, consoante o tipo de utilizador presente, lhe será atribuída uma subclasse correspondente de especialização.

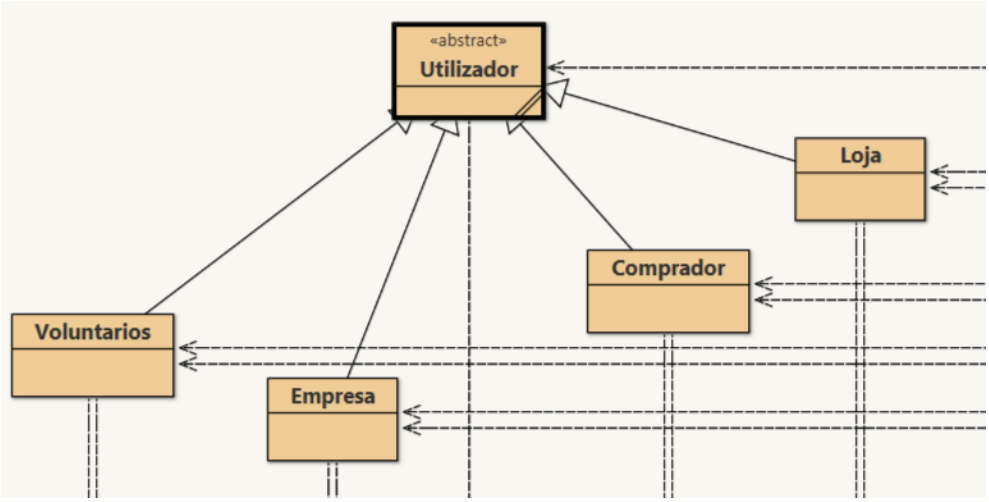


Figura 3 Classe abstrata Utilizador no contexto da hierarquia.

É uma das classes mais simples num ponto de vista estrutural, uma vez que apresenta apenas as variáveis de instância e métodos de alteração e acesso das mesmas. Métodos Equals e toString também se encontram presentes na integra, tendo Clone e stringtoFile sido chamados em modo abstrato, uma vez que apresentarão variações de acordo com as subclasses que os implementam.

```
/**
 * Clone por abstração
 */
public abstract Utilizador clone();
```

```
/**
 * toFile por abstração
 */
public abstract String stringtoFile();
```

Figura 4 Métodos Clone e stringtoFile abstratos.

Assim, Utilizador, para além de ser o nível mais elevado da relação de hierarquia em que se encontra inserido, é também uma classe abstrata.

```
public abstract class Utilizador
```

Figura 5 Chamada de Utilizador como Classe Abstrata.

2.2. Comprador

Comprador pode ser tomado como um "Utilizador Standard" da aplicação, ou seja, o utilizador para a qual a aplicação tem a intenção principal de servir utilidades.

A única e importante variável presente nesta classe é a posse de uma lista de Encomenda, que fará o registo de todas as encomendas efetuadas pelo Comprador na aplicação, quer as já recebidas como as por receber. Um género de "histórico", por assim dizer.

Apresenta, portanto, uma lista de variável de instância, bem como todos os métodos de acesso e alteração da mesma. Métodos Equals e toString característicos, Clone e stringtoFile adaptados à sua estrutura. Uma vez pertencente à hierarquia, dados associados à sua superclasse são chamados sempre que necessários nestas implementações através da referência super.m() e this.m(), consoante ocasião.

```
public Comprador(Comprador c){
    super(c);
    this.encomendas = c.getEncomenda();
}
```

Figura 6 Acesso a dados de Comprador enquanto subclasse de Utilizador.

```
/**
 * toString
 */
public String toString(){
    return "Comprador / " + super.toString();
}
```

```
/**
 * toFile
 */
public String stringtoFile(){
    StringBuilder sb = new StringBuilder();
    sb.append("Utilizador:"+this.getEmail());
    sb.append(","+this.getPass());
    sb.append(","+this.getName());
    sb.append(","+this.getCoord().getLat());
    sb.append(","+this.getCoord().getLon());
    return(sb.toString());
}
```

Figura 7 Exemplos de chamadas super.m() e this.m() referentes a superclasse.

2.3. Voluntarios

Os voluntários serão um dos dois tipos disponíveis de entregadores de encomendas presentes na aplicação. Estes trabalham dentro de um raio de ação que estabelecem, sinalizam a sua disponibilidade, cobram uma taxa pelo transporte e apresentam a sua avaliação, a qual lhes é posteriormente atribuída quando um comprador o avalia no final de uma dada entrega.

Apresentam por isso valores estáticos de taxa cobrada pelo peso de uma encomenda e o preço por Km percorrido que efetuam, de forma a compensar o seu trabalho. Para além disso, apresentam também variáveis para o raio, em específico um booleano para manifestar a sua disponibilidade para distribuição ou ocupação, respetiva avaliação corrente e número de avaliações totais para realização do cálculo da dada média.

```
private static double precoKM = 0.5;
private static double taxaPeso = 0.1;
```

Figura 8 Preço por nº de Kms efetuados por um voluntário numa entrega, bem como taxa adicional pelo peso de dada encomenda. Valores estáticos e em cêntimos.

Para além das variáveis e métodos de alteração e acesso usuais, Clone e toString, Equals e stringtoFile, com recurso a super.m() e this.m(), esta subclasse apresenta em especial um método para o custo do valor total de transporte efetuado por um voluntário, dada a relação entre uma certa encomenda e distância da loja de levantamento ao comprador em questão, com recurso às taxas previamente estabelecidas.

```
public double custoTransV(Encomenda e, Localizacao loja, Localizacao comprador){
    double r = 0;
    double edistl = this.getCoord().distancia(loja);
    double ldistc = loja.distancia(comprador);
    r = e.getPeso()*this.taxaPeso + (edistl + ldistc)*this.precoKM;

    return r*1.10; //(1.10 compensação pelo tempo de espera na loja)
}
```

Figura 9 Custo de Transporte cobrado por um voluntário, em cêntimos.

2.4. Loja

As lojas presentes na aplicação servem como pontos de recolha para os voluntários dos artigos de dada encomenda, quando estes não são escolhidos para serem entregues por empresas transportadoras. São por isso vistas como as fontes dos produtos a ser encomendados.

Uma vez que vários produtos podem ser encomendados e tem de se notar uma ordem de entrega para os mesmos, esta classe apresenta duas variáveis de instância com vista a esse reconhecimento: um booleano que permite saber se existe correntemente uma fila de espera em dada loja e, se este se provar verdadeiro, a correspondente lista de espera de encomendas. A lógica seguida será sempre que uma nova encomenda é realizada à loja por parte de um comprador, esta será adicionada ao fim da lista. Assim que a encomenda do topo tiver sido entregue, é removida.

Apesar de ter sido implementada a fila para a noção de tempo percorrido, este é também um dos pontos que não chegou a ser explorado no código final da aplicação, pelos mesmos motivos de Encomendas Médicas. Para completar a classe, no entanto, decidimos que a sua declaração deveria permanecer presente.

```
private boolean existefila;
private List<Encomenda> filaespera;
```

Figura 10 Chamada de instâncias para reconhecimento de existência de fila de espera numa dada loja.

Métodos de acesso e alteração de instâncias encontram-se presentes, Clone, toString, Equals e stringtoFile também, como seria expectável.

2.5. Empresa

De forma semelhante aos voluntários, as empresas transportadoras são o 2º tipo de entregadores disponível na aplicação TrazAqui!.

Trabalham também dentro de um raio de ação que estabelecem, cobram uma taxa pelo transporte e exibem a sua avaliação posteriormente atribuída quando um comprador as avalia no final de uma entrega.

Apresentam um valor estático para a taxa cobrada pelo peso de uma encomenda, de forma a compensar o seu deslocamento. O preço por Km percorrido que exigem também se encontra presente, porém varia de empresa para empresa, ao contrário de Voluntarios, motivo de não se apresentar estático nesta classe.

Para além disso estão ainda presentes as variáveis para o NIF da empresa, avaliação corrente e número de avaliações totais para realização do cálculo da média.

Ainda diferente de Voluntarios, demos como possível uma empresa conseguir realizar a distribuição de mais de uma encomenda ao mesmo tempo. Esta interpretação será essencial para o estabelecimento de um número limite de entregas simultâneas.

Métodos de acesso e alteração de variáveis, Clone, toString, Equals e stringtoFile encontram-se presentes. Método para cálculo do custo do transporte igualmente adaptado para novos valores de taxa e peso cobrados.

```
public double custoTransE(Encomenda e, Localizacao loja, Localizacao comprador){
    double r = 0;
    double edistl = this.getCoord().distancia(loja);
    double ldistc = loja.distancia(comprador);
    r = e.getPeso()*this.txpeso + (edistl + ldistc)*this.precoKM;

    return r*1.10; //(1.10 compensação pelo tempo de espera na loja)
}
```

Figura 11 Custo de Transporte cobrado por uma empresa transportadora, em cêntimos.

2.6. Encomenda

A classe Encomenda serve o propósito de registar informações de uma encomenda efetuada.

Possui, por isso variáveis para código da encomenda, Comprador, Loja e entregador intervenientes. O peso total dos artigos encomendados, um estado com visa ao seguimento da distribuição e uma lista LinhaEncomenda, que apresenta todos os produtos constituintes da encomenda em questão.

Todas estas variáveis privadas apresentam respetivos métodos de acesso e alteração. Clone, toString, Equals e stringtoFile encontram-se, mais uma vez, presentes na classe.

Foram estabelecidos logo à partida 6 estados possíveis que uma encomenda pode manifestar ao longo do seu processamento:

• DISPONIVEL/NAO DISPONIVEL, atribuídos pela loja, notam se a mesma já disponibilizou os artigos para entrega.

• POR CONFIRMAR/CONFIRMADO, dado pelo utilizador, consoante sua aceitação da empresa transportadora.

• ACEITE se um entregador se encontra no processo de entrega.

• ENTREGUE assim que entregador completa a sua missão e produtos chegam ao destino.

```
private static String naodisp = "NAO DISPONIVEL";
private static String disp = "DISPONIVEL";
private static String nconfirm = "POR CONFIRMAR";
private static String confirm = "CONFIRMADO";
private static String aceite = "ACEITE";
private static String entregue = "ENTREGUE";
```

Figura 12 Estados estáticos presentes para seguimento da entrega de uma encomenda.

Alguns destes estados apresentam métodos de alteração remotos, consoante a sua necessidade de utilização noutros códigos.

É ainda destacada a presença de métodos para adicionar encomenda a lista de previamente realizadas , de atualização do peso total no contexto da adição de um novo produto e de determinação do custo global.

```
public void addLinhaEncomenda(LinhaEncomenda linha){
    this.le.add(linha.clone());
}
        public void acrescentaPeso(double p){
            this.peso = this.getPeso()+p;
        }
    public double custoEnc(){
        double r = 0;

        for(LinhaEncomenda les : this.le){
            r = r + les.calculaValorLinhaEnc();
        }
        return r;
    }
```

Figura 13 Métodos de adição de encomenda, atualização de peso e custo total, respetivamente.

2.7. LinhaEncomenda

LinhaEncomenda, intrinsecamente criada para apoio de Encomenda, armazena dados de um determinado artigo pedido.

Foi desenvolvida com base na relação entre EncEficiente e LinhaEncomenda criadas no contexto das avaliações práticas e adaptada para as novas necessidades do enunciado fornecido.

Possui variáveis de código do produto, e respetivas descrição, quantidade e preço associados, os seus métodos de acesso e alteração, Clone, toString, Equals e stringtoFile.

É, em conjunto com Utilizador, uma das mais simples presentes no API, em termos de estrutura.

2.8. Entrega

Como o nome indica, esta classe fornece-nos todos os dados de uma entrega realizada ou a decorrer. Coordenadas e código de identificação do comprador, loja e entregador são fornecidas. O custo total da entrega e identificação da encomenda a ela relacionada também se encontram presentes como variáveis, bem como a avaliação e data de receção.

Apresenta ainda dois estados estáticos, CLASSIFICADA/NÃO CLASSIFICADA para reconhecimento por parte do comprador, entregador e loja se a sua avaliação já foi atribuída após finalização da operação.

```
private static String nclass = "NAO CLASSIFICADA";
private static String classificada = "CLASSIFICADA";
```

Figura 14 Variáveis estáticas para notar corrente estado de avaliação de dada entrega.

Métodos de acesso e alteração, Clone, toString, Equals e stringtoFile mais uma vez presentes, sem falha.

2.9. Localização

Tendo por base a classe Ponto e derivadas, estudadas ao longo do semestre em diversas ocasiões, servimo-nos de inspiração para a conceção desta classe.

Localização dá-nos acesso através de duas coordenadas, longitudinais e latitudinais, da posição de todo o utilizador da aplicação TrazAqui!. Consequentemente, apresenta vários métodos, para além dos de acesso e alteração das variáveis de instância.

Para além de Clone, toString, e stringtoFile, estão, portanto, presentes ainda métodos para deslocamento, soma e movimento de coordenadas, cálculo de distâncias, reconhecimento de valores positivos introduzidos e ainda notação de igualdade, tanto entre latitude e longitude introduzidas como duas localizações gerais.

2.10. TrazAqui

TrazAqui serve de arquivo a todos os métodos essenciais para o decorrer da aplicação, quando desenvolvida a classe de teste. Estes dizem respeito a ações que cada utilizador pode tomar durante a sua experiência.

Sendo uma classe bastante extensiva, e com diversos métodos semelhantes, para os diferentes tipos de utilizadores, faremos de seguida um destaque dos mais relevantes.

Acrescenta-se ainda que é recorrido o uso de Maps, como forma de acesso mais organizada aos dados para cada método desenvolvido.

```
private Map<String,Utilizador> users;
private Map<String,Encomenda> encomendas;
private Map<String,Entrega> entregas;
```

Figura 15 Maps em questão para acesso a dados de utilizadores, encomendas e entregas.

```
public int addUser(Utilizador u){
    if(this.users.containsValue(u)) return -1;
    else{
        this.users.put(u.getEmail(),u.clone());
        return 1;
    }
}
```

Figura 166 Adiciona um novo Utilizador a aplicação.

```
public List<Entrega> entregasPeriodo(String emailE, LocalDate inicio, LocalDate fim){
    ArrayList<Entrega> res = new ArrayList<>();

    if(this.users.containsKey(emailE)){

        for(Entrega e : this.entregas.values()){
            if(e.getEntregador().equals(emailE) && e.getData().isAfter(inicio) && e.g
                res.add(e.clone());
            }
        }
    }
    return res;
}
```

Figura 177 Permite-nos aceder a lista de entregas realizadas por um dado entregador, num período a definir.

```
public double lucroPeriodo(String emailE, LocalDate inicio, LocalDate fim){
    double res=0;

    if(this.users.containsKey(emailE)){

        for(Entrega e : this.entregas.values()){
            if(e.getEntregador().equals(emailE) && e.getData().isAfter(inicio) && e.getData().isBefore(fim)){
                res=res+e.getCusto();
            }
        }
    }
    return res;
}
```

Figura 18 Dá-nos o lucro obtido por um entregador num período a definir.

```
public List<Encomenda> encomendasNaoDisp(String emailLoja){
    List<Encomenda> r = new ArrayList<>();

    for(Encomenda e : this.encomendas.values()){
        if(e.getCLoja().equals(emailLoja) && e.getState().equals("NAO DISPONIVEL")){
            r.add(e.clone());
        }
    }
    return r;
}
```

Figura 19 Lista das encomendas ainda por disponibilizar de uma determinada loja.

```
public List<Loja> dentrodoRaioV(String emailE){
    List<Loja> l = new ArrayList<>();
    Voluntarios v = (Voluntarios) this.users.get(emailE);

    for(Utilizador u : this.users.values()){
        if(u.getClass().getName().equals("Loja")){
            Loja ls = (Loja) u;
            if(v.getCoord().distancia(u.getCoord())<=v.getRaio()){
                l.add(ls.clone());
            }
        }
    }
    return l;
}
```

Figura 20 Lojas que se encontram dentro do raio de ação de um entregador: voluntário. (Semelhante para empresa)

```
public boolean estadoV(String emailV){
    Voluntarios v = (Voluntarios) this.users.get(emailV);
    return v.getAvailable();
}
```

Figura 21 Altera estado de um voluntário.

```
public double kmpercorridos(String email){
    double r = 0;

    for(Entrega e : this.entregas.values()){
        if(e.getEntregador().equals(email)){
            r = r + e.getCentregador().distancia(e.getCloja()) + e.getCloja().distancia(e.getCbuyer());
        }
    }
    return r;
}
```

Figura 22 Kms percorridos por uma dada empresa transportadora.

```
public void top10Users(){
public void top10Entregadores(){
```

Figura 23 Top 10 utilizadores e entregadores da aplicação.

```
public void classificar1entrega(String emailC){
```

Figura 24 Classifica uma entrega após a sua receção.

```
public void classificarEmpresa(String email, double classificacao){
    Empresa e = (Empresa) this.users.get(email);
    if(e!=null ){
        double notaT = (e.getNota()*e.getReviews()+classificacao)/(e.getReviews()+1);
        e.setNota(notaT);
        e.setReviews(e.getReviews()+1);
    }
}
```

Figura 25 Classifica uma empresa. (Semelhante para voluntário)

```
private static Comprador lerComprador(String input){
    String[] buffer = input.split(",");
    String email = buffer[0];
    String pass = buffer[1];
    String nome = buffer[2];
    double gpsx = Double.parseDouble(buffer[3]);
    double gpsy = Double.parseDouble(buffer[4]);
    Localizacao l = new Localizacao(gpsx,gpsy);
    Comprador c = new Comprador(email,pass,nome,l,new ArrayList<>());
    return c;
}
```

Figura 26 Método de leitura de ficheiro: Cria um comprador através dos dados dos logs. (Semelhante para Loja, Voluntario, Empresa, Encomenda e Entrega)

2.11. Teste

É em Teste que a aplicação TrazAqui! irá correr.

Podemos encontrar aqui métodos para terminar e iniciar a aplicação, inscrever um novo utilizador, aceder ao menu respetivo de cada um, entre outros.

Uma vez que esta classe se apresenta ainda mais complexa que TrazAqui, decidimos mostrar as suas funcionalidades visualmente, prosseguindo, assim, para o manual de utilização da aplicação desenvolvida.

3. Manual de Utilizador

• Para a inicialização de TrazAqui! é necessário o correr da classe Teste em BlueJ.

• Após inicialização, somos deparados com o menu de Login da aplicação. Vamos criar um novo utilizador do tipo Comprador.

```
Seja bem vindo!!
1: Login
2: Registo
0: Cancelar
Selecione uma das opções:
```

Figura 27 Menu inicial para Login/Registo.

```
Seja bem vindo!!
1: Login
2: Registo
0: Cancelar
Selecione uma das opções: 2
1: Comprador
2: Loja
3: Voluntário
4: Empresa Transportadora
1
Indique o seu email:
exemplo@gmail.com
Escreva o seu nome:
BL
Crie uma password:
234
Indique as coordenadas da localização para deixar as suas encomendas
Gpsx:
26.789
Gpsy:
34.456
```

Figura 28 Exemplo de inscrição enquanto Comprador.

• Após inscrição é-nos dado imediato acesso ao menu relativo a um comprador.

```
Selecione a operação que pretende efetuar:
1: Fazer uma encomenda
2: Autorizar o transporte de encomendas por empresas transportadoras
3: Listagem das minhas encomendas
4: Entregas feitas por um voluntário ou empresa num determinado periodo
5: Top 10 utilizadores que mais usam a aplicação para fazer compras
6: Top 10 empresas transportadoras por Kms percorridos
7: Classificar um voluntário ou empresa transportadora
0: Sair
```

Figura 29 Menu TrazAqui! para Comprador: BL.

• Realizando o Login através de uma Loja, Voluntário ou Empresa da aplicação, obteremos menus semelhantes.

```
É bom vê-lo/a novamente Punt Roma!

Selecione a operação que pretende realizar:
1: Disponiblizar um encomenda para ser transportada
2: Entregas feitas por um voluntário ou empresa num determinado período
3: Top 10 utilizadores que mais usam a aplicação para fazer compras
4: Top 10 empresas transportadoras por Kms percorridos
0: Sair
```

Figura 30 Menu TrazAqui! para Loja 8: Punt Roma.

```
É bom vê-lo/a novamente Rui Pedro Gomes Coelho!

Selecione a operação que pretende efetuar:
1: Sinalizar que está pronto para recolher encomendas
2: Sinalizar que não está pronto para recolher encomendas
3: Ver encomendas disponíveis para levantamento
4: Registar entrega
5: Ver as encomendas que entreguei num dado período
6: Ver total faturado entre datas
7: Top 10 utilizadores que mais usam a aplicação para fazer compras
8: Top 10 empresas transportadoras por Kms percorridos
9: Classificação atual
0: Sair
```

Figura 31 Menu TrazAqui! para Voluntário 73: Rui Pedro Gomes Coelho.

```
É bom vê-lo/a novamente ZENITHINDEX!

Selecione a operação que pretende realizar:
1: Ver encomendas disponíveis para levantamento
2: Levantar encomendas autorizadas
3: Registar entrega
4: Ver as encomendas que a empresa entregou num dado período
5: Ver total faturado entre datas
6: Top 10 utilizadores que mais usam a aplicação para fazer compras
7: Top 10 empresas transportadoras por Kms percorridos
8: Classificação atual
0: Log out
```

Figura 32 Menu TrazAqui! para Empresa Transportadora 31: ZENITHINDEX.

• Cada opção presente no menu dos utilizadores encontra-se perfeitamente funcional, verificando-se a sua respetiva atualização em logs, quando esta é necessária.

• Verificamos, por exemplo, que após criação de um novo Utilizador, logs atualizou com as suas respetivas informações inseridas.

```
........ TrazAqui ........
Transportadora:t9,251351,TORRESTIR TRANSITARIOS LDA,-9.798416,29.869064,923953011,128.0,1.5,0.0,0.0
Utilizador:pyro@hotmail.com,fire,Luisa,666.66,666.66
Loja:l8,25412351,Punt Roma,57.339508,-86.066315
Loja:l58,35131,Primark,78.73474,66.239685
Loja:l13,2421,LA Kids & Junior,-27.339592,-54.781532
Transportadora:t51,21214,SABIA ATITUDE CONSTRUCOES,-39.519154,8.964966,455159118,167.0,3.5,0.0,0.0
Utilizador:u97,54835,Pedro Filipe Carvalho Barbosa,-68.78327,-50.26914
Voluntario:v73,231351,Rui Pedro Gomes Coelho,-45.424522,-79.517136,15.0,true,0.0,0.0
Transportadora:t31,516513,ZENITHINDEX,31.47522,76.58536,599618020,192.0,1.5,0.0,0.0
Utilizador:u33,2145,Tomas Salazar Valente,98.079666,-36.973225
Utilizador:u78,65461,Luis Pedro da Silva Vila,64.87256,43.373962
Voluntario:v33,1254,Simao Pedro Santa Cruz Oliveira,11.959328,95.35214,81.0,false,6.5,1.0
Voluntario:v59,2121,Joao Carlos Delgado Monteiro,87.60193,-67.35995,56.0,false,0.0,0.0
Utilizador:u7,54154,Rodrigo Tiago Oliveira Ferreira,-14.681885,72.73595
Loja:l83,3651,ColchaoNet.com,95.71132,-57.792557

........ TrazAqui ........
Transportadora:t9,251351,TORRESTIR TRANSITARIOS LDA,-9.798416,29.869064,923953011,128.0,1.5,0.0,0.0
Utilizador:pyro@hotmail.com,fire,Luisa,666.66,666.66
Loja:l8,25412351,Punt Roma,57.339508,-86.066315
Loja:l58,35131,Primark,78.73474,66.239685
Loja:l13,2421,LA Kids & Junior,-27.339592,-54.781532
Transportadora:t51,21214,SABIA ATITUDE CONSTRUCOES,-39.519154,8.964966,455159118,167.0,3.5,0.0,0.0
Utilizador:u97,54835,Pedro Filipe Carvalho Barbosa,-68.78327,-50.26914
Voluntario:v73,231351,Rui Pedro Gomes Coelho,-45.424522,-79.517136,15.0,true,0.0,0.0
Transportadora:t31,516513,ZENITHINDEX,31.47522,76.58536,599618020,192.0,1.5,0.0,0.0
Utilizador:u33,2145,Tomas Salazar Valente,98.079666,-36.973225
Utilizador:u78,65461,Luis Pedro da Silva Vila,64.87256,43.373962
Voluntario:v33,1254,Simao Pedro Santa Cruz Oliveira,11.959328,95.35214,81.0,false,6.5,1.0
Voluntario:v59,2121,Joao Carlos Delgado Monteiro,87.60193,-67.35995,56.0,false,0.0,0.0
Utilizador:exemplo@gmail.com,234,BL,26.789,34.456
Utilizador:u7,54154,Rodrigo Tiago Oliveira Ferreira,-14.681885,72.73595
Loja:l83,3651,ColchaoNet.com,95.71132,-57.792557
```

Figura 33 Ficheiro logs antes e após nova incrição de um Utilizador do tipo Comprador.

• Outras operações serão mais bem exploradas quando apresentada a aplicação em avaliação individual. Apesar disso, podemos ainda mostrar em ação os tops.

```
É bom vê-lo/a novamente BL!

Selecione a operação que pretende efetuar:
1: Fazer uma encomenda
2: Autorizar o transporte de encomendas por empresas transportadoras
3: Listagem das minhas encomendas
4: Entregas feitas por um voluntário ou empresa num determinado periodo
5: Top 10 utilizadores que mais usam a aplicação para fazer compras
6: Top 10 empresas transportadoras por Kms percorridos
7: Classificar um voluntário ou empresa transportadora
0: Sair
5

Top 10 utilizadores que mais usam a aplicação:
1º Francisco Coutinho Martins Correia : 2 encomenda(s) recebida(s)
2º Luis Pedro da Silva Vila : 1 encomenda(s) recebida(s)
3º Barbara : 1 encomenda(s) recebida(s)
```

```
6

Top 10 empresas transportadoras por Kms percorridos:
1º ZENITHINDEX : 352.6704778005792 Kms
```

Figura 34 Top 10 utilizadores (compradores) da aplicação, bem como Top 10 Empresas Transportadoras.

4. Conclusão

Através deste projeto foi-nos possível aprofundar o nosso conhecimento e praticar as técnicas de desenvolvimento adquiridas ao longo do semestre na cadeira de Programação Orientada aos Objetos.

Todos os módulos estudados, desde os mais básicos, criação de classes, definição de variáveis e métodos de acesso e alteração, ao trabalho com arrays, lists, maps, hierarquias, classes abstract, entre outros; foram utilizados com sucesso no contexto da aplicação TrazAqui!.

O guião apresentado foi encarado como um desafio pelos membros da equipa, onde viram a oportunidade de expandir as suas capacidades e pôr à prova todo o trabalho realizado ao longo destes últimos meses.