Projeto Prático de Programação Orientada aos Objetos

LEI

Relatório do Projeto GeocachingPOO

2014/15

Grupo 38

Trabalho realizado por:

- Alexandre Lopes Mandim da Silva (A73674)
- Francisco José Torres Ribeiro (A70712)
- Rafael Mota Oliveira (A72307)

Índice:

Introdução:

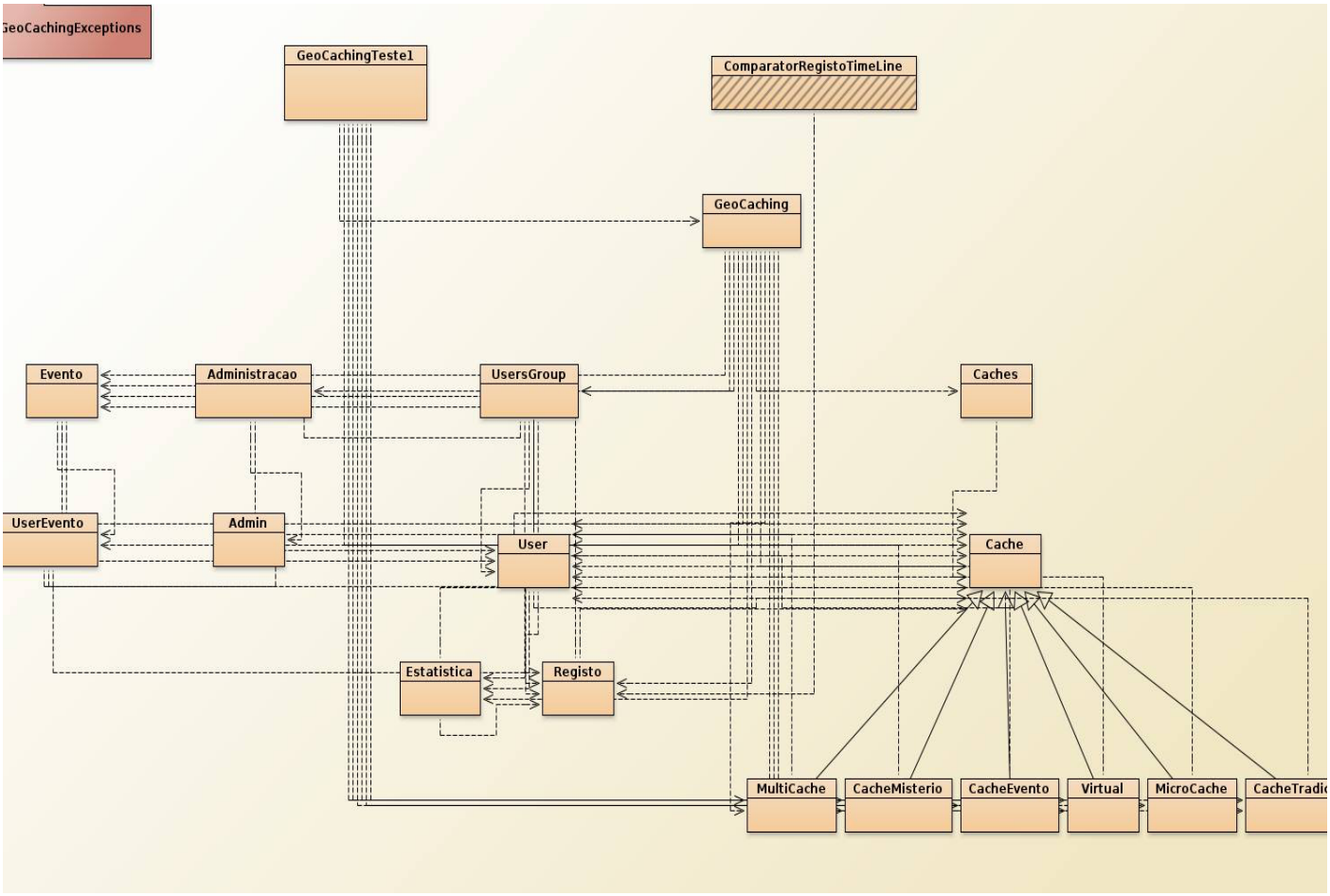
Este trabalho foi realizado no âmbito da unidade curricular de Programação Orientada aos Objetos. O principal objetivo do trabalho foi o de colocar em prática a matéria e os conhecimentos leccionados ao longo deste semestre.

Com este trabalho esperamos consolidar todos estes conteúdos, obter uma boa preparação para o teste de avaliação e fomentar o trabalho em equipa, aptidão fundamental no mundo do trabalho.

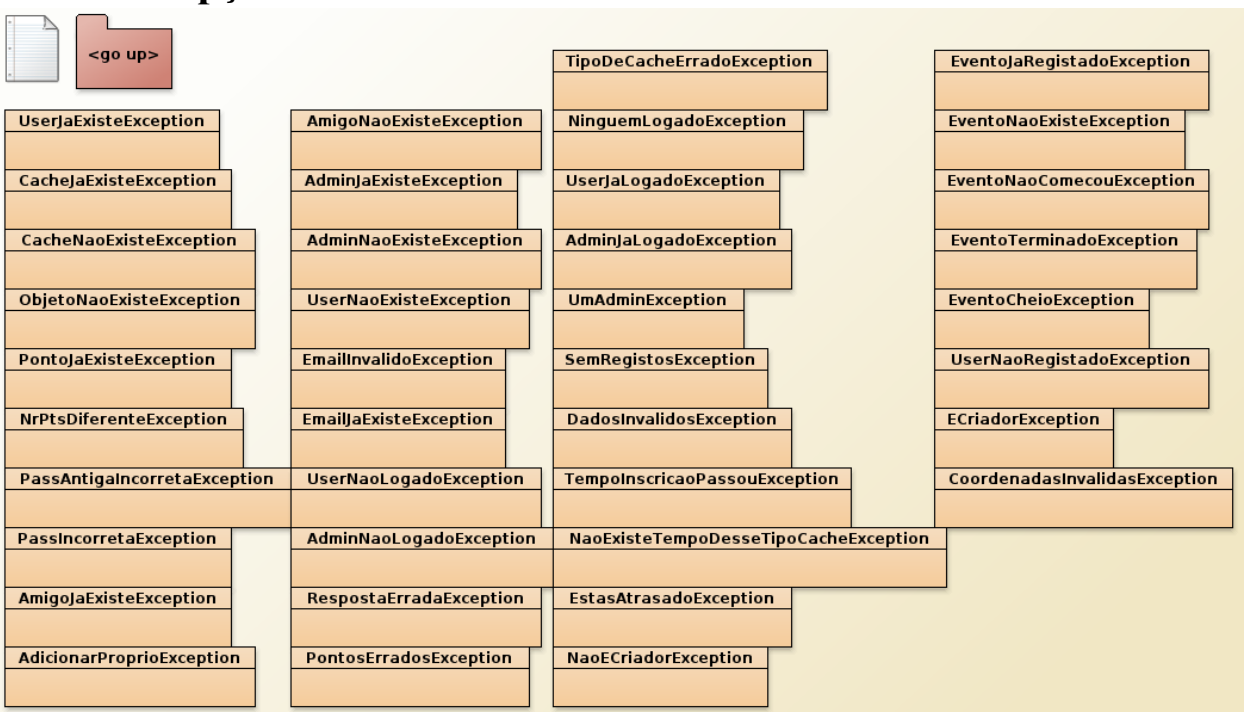
De modo a facilitar a implementação inicial do projeto começamos por realizar um esboço generalista e simplista da aplicação, implementando as diversas classes por alto. Ao longo do projeto fomos aumentando a complexidade das mesmas. Porém a estrutura do projeto não variou muito do traçado inicialmente.

Nos capítulos seguintes procede-se uma explicação mais detalhada do trabalho.

Arquitectura de classes



- **Excepções**



Estruturação de classes

A classe GeoCachingTeste é responsável por apresentar ao utilizador os menus necessários à utilização da aplicação bem como por recolher os inputs do utilizador necessários para a interação deste com o programa.

A classe GeoCaching serve de intermediário entre os dados enviados pela classe GeoCachingTeste e as restantes classes que permitem o funcionamento da aplicação.

Primeiramente, vamo-nos referir às classes relacionadas com as caches. Foi criada uma classe chamada Cache que é super classe de várias classes que representam os diferentes tipos de cache. De modo a representar um conjunto de caches criamos outra classe a que demos o nome de Caches.

De forma a representar um utilizador possuímos uma classe User e uma outra designada de UsersGroup para, tal como em Cache e Caches, poder ter todos os *users* existentes agrupados.

A classe Registo permite-nos ter informação acerca de atividades realizadas, como uma descoberta ou criação de uma cache por parte de um *user*. A classe Estatística agrupa todos os registos de um mês de um *user*.

Uma aplicação Geocaching tal como qualquer outra tem de possuir uma administração que controle e garanta o seu correto funcionamento. Foi então criada uma classe Admin que contém informação que permite representar um administrador e uma classe Administração que contém todos os administradores existentes.

De modo a simular e representar eventos criamos uma classe chamada UserEvento que contém informação necessária para representar um utilizador que esteja inserido num evento. A classe Evento possui todos os utilizadores que participam no evento e outras informações que serão abordadas mais detalhadamente mais adiante.

Estruturas de dados e estratégias adoptadas

Tendo em conta os aspetos das diferentes *collections* e tirando partido das diferentes vantagens que cada uma delas apresenta, adaptámos as nossas necessidades às várias características das mesmas.

Claro que, em muitos casos, muitos dos tipos nativos de dados (int, double, …) serviam as nossas necessidades, bem como algumas classes disponibilizadas pelo Java (como por exemplo, Point2D.Double).

- Cache

Variáveis de instância:

Cada Cache, independentemente do seu tipo, tem umas coordenadas que corresponde à sua posição geográfica.

Uma *String* que da informação referente ao local de esconderijo, um nome da cache. Um *int* que representa a dificuldade desta cache, ou seja, cada cache tem um valor associado entre 1 e 10 que representa a sua dificuldade, quanto maior o valor maior essa dificuldade.

Possui também como variável de instância um *ArrayList<String>* designado de

livroRegisto que representa o nome de cada utilizador que a descobriu.

Por fim, temos como última variável de instância uma *String* que é o nome do criador da cache.

As seguintes seis classes representam as sub-classes da classe Cache, portanto todas elas têm as variáveis de instância da classe Cache:

- MultiCache

Variáveis de instância:

```
private ArrayList<Point2D.Double> todasCaches;
```

Uma Multi Cache é uma cache ao qual o utilizador tem de percorrer vários pontos para alcançar o ponto final, ponto esse que corresponde às coordenadas da cache final. Com isto, uma MultiCache tem como variável de instância todasCaches que é um *ArrayList<Point2D.Double>* com todos os pontos intermédios.

- CacheMisterio

Variáveis de instância:

```
private String puzzle;
private String resposta;
```

Uma Cache Mistério necessita que um utilizador resolva um puzzle para encontrá-la. Sendo então que criamos uma *String* puzzle e uma *String* resposta.

- CacheEvento

Variáveis de instância:

```
private GregorianCalendar data;
```

Uma Cache Evento é um encontro de geoCachers, sendo que criamos uma variável data que corresponde à data do evento.

- Virtual

Variáveis de instância:

```
private String pergunta;
private String prova;
```

Uma Cache Vitual é um local a visitar sem caixas escondidas. É necessário uma prova para validar este tipo de Cache, então criamos duas *String's* que são uma pergunta e uma prova.

- MicroCache

Uma Micro Cache é uma cache normal apenas difere no tamanho, sendo então que esta MicroCache apenas tem as variáveis de instância herdadas de Cache.

- CacheTradicional

Variáveis de instância:

```
private ArrayList<String> objetos;
```

Uma Cache Tradicional permite inserir e retirar objetos do seu interior, sendo que apenas foi necessário criar uma variável objetos que é um *ArrayList<String>* que tem o vários nomes dos objetos existentes na Cache.

- Caches

```
private TreeMap<String,Cache> todasCaches, cachesEliminadas;
```

A classe Cache tem dois *TreeMap<String,Cache>,* uma representa todas as Caches possíveis de se descobrir denominada de todasCaches. Outro deles designa-se de cachesEliminadas que tal como o nome sugere representa todas as classes que foram eliminadas. Estes *TreeMap's* associam o nome da cache(*Key*) à sua cache correspondente(*Cache*).

Optamos por um *TreeMap* para podermos ter a associação de nome de cache e a Cache, o que facilita o acesso.

- User

Variáveis de instância:

```
private String email;
private String password;
private String nome;
private char genero;
private String morada;
private int pontos;
private GregorianCalendar data_de_nascimento;
```

Cada um dos *users* registados na aplicação têm um email, uma password, um nome e uma morada associados que são representados, respetivamente, pelas Strings email, password e nome e morada.

O char genero pode ser 'M' ou 'F', que corresponde ao género masculino ou feminino, respetivamente.

O inteiro pontos corresponde à pontuação acumulada pelo *user* por todas as suas atividades de descoberta de caches ou participação em eventos.

A data_de_nascimento corresponde a um objeto da classe GregorianCalendar que representa, tal como o nome indica, a data de nascimento do *user*.

- UsersGroup

Variáveis de instância:

```
private TreeMap <String, User> treeUsers;
```

Um *UsersGroup* representa um conjunto de utilizadores e, para tal, utilizamos um TreeMap, visto querermos associar o email (Key) de cada *user* a um objeto da classe *User* (Value).

Optamos por um *TreeMap* para podermos ter a associação de email de um utilizador e o *User*, o que facilita o acesso.

- Estatística

Variáveis de instância:

```
private GregorianCalendar data;
private ArrayList<Registo> registoMes;
```

Uma *Estatistica* possui uma data associada, em que o respetivo mês corresponde à altura a que o *ArrayList* de *Registo* registoMes se refere.

Optamos por um *ArrayList* porque queríamos ter ordem de Registos em time line e esta *Collection* possui essa característica.

- Registo

Variáveis de instância e de classe:

```
private static int contador = 0;

private int idRegisto;
private GregorianCalendar dataAtividade;
private Cache cache;
private Evento evento;
private boolean encontrado, criado;
private int condicoesMeteorologicas;
```

Registo possui uma variável de classe contador. Esta variável tem como função manter o número mais alto de idRegisto acessível a todas as instâncias de *Registo*. Desta forma, ao atribuírmos um idRegisto a cada *Registo* temos a garantia de que este é único, visto que esta variável de classe contador é incrementada a cada atribuição de um idRegisto.

A dataAtividade corresponde à data a que o *Registo* diz respeito.

A cache corresponde à *Cache* a que diz respeito o *Registo*.

O evento diz respeito ao *Evento* a que o *Registo* corresponde, se for o caso. Caso contrário, atribui *null* a evento.

Como um *Registo* pode dizer respeito a uma criação ou a uma descoberta de uma *Cache*, os *boolean* encontrado e criado especificam esta circunstância. Não podem, portanto, ser os dois *true* ou os dois *false*.

As condições meteorológicas associadas a um *Registo* (geradas aleatoriamente) são representadas por condicoesMeteorologicas.

- Admin

Variáveis de instância:

```
private String email;
private String password;
```

Um *Admin* tem um email e uma password associados, representados respetivamente pelas *Strings* email e password.

- Administração

Variáveis de instância:

```
private TreeMap<String, Admin> grupoAdmins;
private TreeSet<Cache> cachesReportadas;
```

A *Administração* corresponde ao conjunto de todos os *Admins* e de todas as caches reportadas como abusivas.

O *TreeMap* grupoAdmins associa o email (Key) de cada *Admin* ao *Admin* em questão (Value).

O *TreeSet* cachesReportadas contém todas as caches que foram denunciadas

pelos *Users* para estarem disponíveis para os *Admins* tomarem as devidas acções relativamente a elas (remoção ou permissão, visto que um *Admin* pode concordar ou discordar do *report abuse* feito por um *User*).

Obtamos por um *TreeMap* para representar o grupoAdmins visto ser rápido acesso e possuir uma associação *email's* com os administradores. Usamos um *TreeSet* de Caches para representar as caches reportadas visto não termos caches reportadas repetidas.

- UserEvento

Variáveis de instância:

```
private String email;
private TreeMap<String,Double> tempoMinimo;
private ArrayList<Registo> registos;
private int pontos;
private boolean terminado;
```

A classe UserEvento tem uma *String* email que é o email do utilizador que participará no evento. Um *TreeMap<String,Double>* que corresponde ao historial médio de cada utilizador tem em descobrir cada tipo de Cache, esta variável designa-se de tempoMinimo.

Tem também uma variável que corresponde aos registos de atividade do utilizador no evento. Estes registos estão armazenados num *ArrayList<Registo>* .

Temos também um *int* pontos para representar o total de pontos que esse utilizador tem e, por fim, esta classe possui também um *boolean* terminado que indica se está num evento ou não.

Decidimos implementar um *TreeMap* para representar o tempoMinimo porque assim associamos de forma rápida um email de um utilizador ao seu historial de tempos para descobrir cada tipo de cache.

- Evento

Variáveis de instância:

```
private String nome;
private int nrUsersMax, condicoesMeteo; /* 0 <= condicoesMeteo <= 5 */
private double tempoMedioEncontrarCache;
private GregorianCalendar datalimiteInscricao, dataInicioEvento;
private boolean terminado;
private Point2D.Double localizacaoEvento;
TreeMap <String, Cache> cachesEvento;
TreeMap <String, UserEvento> utilizadoresRegistados;
```

A classe Evento tem uma *String* nome que representa o nome do Evento, um *int* nrUsersMax que é o número máximo de utilizadores suportado no evento e outro *int* condicoesMeteo que representa as condições meteorológicas do evento.

Existe também duas datas, uma datalimiteInscriçao que delimita a data de inscrição e outra data dataInicioEvento que representa a data do início do evento.

Um *boolean* terminado que indica se o evento está a decorrer ou não. Existe um *Point2D.Double* localizacaoEvento que indica as coordenadas do evento.

Por fim, esta classe possui dois *TreeMap,* um com a chave *String* que corresponde ao nome de uma cache e o *value Cache* que é a cache correspondente. Esta variável chama-se de cachesEvento. O outro *TreeMap* tem como *key* uma *String* e *value UserEvento* que associa um email de utilizador de um evento ao seu *UserEvento.*

- GeoCaching

Variáveis de instância:

```
private String loggedUser, loggedAdmin;

private UsersGroup utilizadores;
private Caches caches;
private Administracao administracao;
private TreeMap<String,Evento> eventos;
```

Para se ter conhecimento do *User* ou do *Admin* que tem logIn efetuado na aplicação, as *Strings* loggedUser e loggedAdmin guardam o email correspondente de quem tem sessão iniciada.

GeoCaching tem também um *UsersGroup* utilizadores também associado, que tem todos os *Users* registado na aplicação.

Tem também uma *Caches* caches que tem todas as caches disponíveis para serem descobertas pelos *Users* (não contando as caches dos *Eventos*).

Possui também uma *Administração* administracao que corresponde ao conjunto dos *Admins* e das caches reportadas.

Por último, possui um TreeMap eventos que relaciona o nome de um Evento (Key) com o Evento correspondente (Value).

Estratégias adotadas para a simulação dos Eventos:

Para simular os Eventos atribuímos um tempo médio de descoberta para cada tipo de cache a um utilizador com base nos seus registos anteriores.

Primeiramente, calculamos a proporção de um tipo de cache em questão em relação ao número total de caches descobertas por esse *User*.

De seguida, com base numa fórmula que estabelecemos e achamos adequada para o efeito, subtraímos ao número 15 o valor calculado anteriormente e dividimos este resultado por 10.

Sabendo que as condições meteorológicas do momento afetam o tempo necessário para se descobrir uma cache (num ponto de vista negativo, ou seja, o tempo tem tendência para subir quanto mais adversas forem estas condições) também as tivemos em conta aquando da determinação do tempo médio necessário para a descoberta de uma cache num *Evento*. Como tal, o *int* que representa as condições meteorológicas do *Evento* em questão é dividido por 10 e a esse resultado soma-se 1.

De seguida, o valor calculado anteriormente da proporção de caches descobertas pelo *User* é multiplicado pelo valor mencionado anteriormente das condições meteorológicas. Desta forma, sabemos que o valor inicialmente estabelecido para a média de tempo para se descobrir uma cache aquando da criação do *Evento* aumenta consoante as condições meteorológicas são mais adversas (*int* que as representa toma valores maiores).

Fórmula sintetizada:

1º Passo

$$tempoMédio \times (15 - proporção / 10) = x$$

2º Passo (utilizando valor calculado anteriormente)

$$x = x * (1 + condMeteo / 10)$$

Nossa aplicação

- Login e Menu do Utilizador:

```
Erro abrir ficheiro!
Entrar -> 1
Registar -> 2
Sair -> 0
1

Insira o seu e-email:
rafa

Introduza a sua palavra-passe:
pass

Entrou com sucesso

1. Criar uma cache;
2. Remover uma cache;
3. Registar descoberta de uma cache;
4. Remover uma atividade de descoberta;
5. Ver as suas últimas 10 atividades (ver detalhadamente uma atividade);
6. Ver lista de amigos;
7. Ver 10 últimas atividades dos amigos (ver detalhadamente uma atividade);
8. Ver o perfil;
9. Editar o perfil;
10. Ver estatística mensalmente ou anualmente;
11. Adicionar um utilizador aos amigos
12. Remover um utilizador dos amigos
13. Ver todos os users
14. Ver todas as caches
15. Reportar abuso de cache
16. Mudar palavra pass
17. Ver eventos
18. Entrar num evento
19. Registar num evento
20. Guardar estado
0. LogOut
```

- Criação de uma cache (neste exemplo foi criada uma micro cache):

```
1
Indique latitude:
123
Indique longitude:
-12
Indique informações do esconderijo:
Atenção ao caminho ingreme!
Indique qual o nome:
cache aventura
Indique a dificuldade(1-10):
7
Diga qual o tipo de Cache:
1. MultiCache
2. Cache Mistério
3. Cache Evento
4. Virtual
5. MicroCache
6. Cache tradicional
0. Sair

5
Micro Cache criada com sucesso.
```

- Visualização das caches:

```
14
----------------------------------------
MICRO-CACHE
Nome: cache aventura
Coordenadas: (123.0 , -12.0).
Informação Adicional : Atenção ao caminho ingreme!
Dificuldade: 7
Criada por: rafa.
----------------------------------------
----------------------------------------
MULTI-CACHE:
Nome: cache bragaparque
Pontos: 1
Informações: Nada a referir
Criador: rafa
----------------------------------------
----------------------------------------
CACHE TRADICIONAL:
Existem: 2 objetos:
  -caneta;
  -lapis;

Nome: cache montanhosa
Coordenadas: (324.0 , 123.0).
Informação Adicional : Elevada altitude
Dificuldade: 9
Criada por: rafa.
----------------------------------------
```

- Ver perfil, alterar perfil e mostrar novamente:

```
8

Nome: Rafael
Email: rafa
Password: pass
Genero: M
Pontos: 0
Morada: Braga
Data de Nascimento: 1995/1/23
9
Introduza o seu nome:
Rafael Mota Oliveira
Introduza antiga palavra-passe(para validação):
pass
Introduza a sua morada:
Barca
Genero: '1' -> Masculino '2' -> Feminino
1
Introduza o seu ano de nascimento:
1995
Introduza o seu mes de nascimento:
1
Introduza o seu dia de nascimento:
23
Editado com sucesso!
```

```
8

Nome: Rafael Mota Oliveira
Email: rafa
Password: pass
Genero: M
Pontos: 0
Morada: Barca
Data de Nascimento: 1995/1/23
```

- Login e Menu administrador:

```
Entrar -> 1
Registar -> 2
Sair -> 0
1

Insira o seu e-email:
admin

Introduza a sua palavra-passe:
pass

Entrou com sucesso

Existe(m) 0 cache(s) reportada(s) como abusadora(s)!


1. Criar uma cache;
2. Remover uma cache;
3. Registar administrador
4. Eliminar administrador
5. Ver caches reportadas
6. Tratar cache abusadora
7. Criar Evento
8. Finalizar evento
9. Ver eventos
0. LogOut;
```

- Ver eventos:

```
9
Eventos:
-> O evento evento verao total vai decorrer em 2015/7/20 e conta com um nr máx de users de  50.
Neste evento já se encontram registados 0 jogadores.
Evento : evento verao total
Número máximo de users: 50
```

Conclusão

Para concluir, este trabalho correspondeu às expectativas mencionadas na introdução. Sentimos que melhoramos os nossos conhecimentos relativos à matéria lecionada durante o semestre, tais como, por exemplo, o encapsulamento de dados, a hierarquia de classes, o uso das *Collections*, *Streams*, etc.

Finalmente, é de salientar que o aprofundamento da matéria no projeto é uma mais valia para a preparação do teste final e consequente sucesso nesta unidade curricular.

Análise crítica:

Estamos satisfeitos com o projeto que realizamos. Sentimos que é um trabalho completo e bem estruturado.

Somos da opinião que o projeto por nós elaborado evidencia os nossos conhecimentos da matéria e do paradigma da Programação Orientada aos Objetos.

O nosso grupo tem consciência de que todos os elementos se esforçaram para a realização do trabalho e saímos, deste modo, com as nossas capacidades de organização e colaboração em equipa fortalecidas, o que é uma mais valia no mundo do trabalho.