Universidade do Minho

Licenciatura em Engenharia Informática

Geocaching

Hélder Filipe Fernandes Machado

João Carlos Delgado Monteiro

Índice

1. Introdução

O presente projeto tem como base a utilização de linguagem *JAVA* apoiada pelo *BlueJ*, uma ferramenta de desenvolvimento integrado.

Tal projeto consiste na concepção de uma aplicação com o objetivo de registar e simular atividades de descobertas de caches. A aplicação deverá essencialmente permitir aos utilizadores registar e consultar essas mesmas atividades, as atividades de todos os utilizadores na sua rede de amigos e ainda a criação de novas caches.

A implementação da solução será segundo os principios da Programação Orientada aos Objetos (POO). Neste projeto são explorados mecanismos muito poderosos do JAVA como *Hierarquia* e *Herança* e *Composiçao de Classes*, quanto a POO preocupamo-nos essencialmente com o encapsulamento de dados.

2. Contextualização

Geocaching é um passatempo ao ar livre no qual se utiliza um receptor de navegação por satélite (GPS) para encontrar uma "geocache" (ou simplesmente "cache") colocada em qualquer sítio do mundo. Uma cache típica é uma pequena caixa (rolo fotográfico ou até tupperware), fechada e à prova de água, que contém um livro de registo e, em alguns casos, alguns objectos como canetas, afia-lápis, moedas ou bonecos para troca.

Essas caches são criadas pelos utilizadores e publicadas na Internet com a respetiva localização (coordenadas GPS WSG84) e alguma informação sobre o esconderijo onde se encontra. Os outros utilizadores (também chamados de “geocachers”) consultam essa informação e tentam descobrir a cache e quando o conseguem registam a descoberta no respetivo livro de registo que se encontra na cache. Os geocachers podem também colocar e retirar os objetos das caches para que haja sempre alguma recordação para guardar.

Existem vários tipos de caches como micro-caches onde apenas se encontra um pequeno livro para os geocachers registarem a descoberta, virtual-cache onde não existe nenhuma cache física escondida mas que supostamente deverá levar a uma localização com algo interessante ou apenas bonito (neste caso o registo necessita de uma prova da visita como por exemplo uma foto ou uma descrição do que fez e do que viu), multi-caches que necessitam que os utilizadores visitem pontos intermédios de modo a descobrir as coordenadas da cache final, entre várias outras.

3. Motivação e objetivos

A maior motivação para este trabalho foi a aprendizagem obtida sobre uma das linguagens de programação mais populares em todo o mundo.

Tendo em conta que o objetivo principal da nossa aplicação é o registo e consulta de atividades e estatísticas, o principal objetivo para o desenvolvimento deste projeto passa por tentar implementar funções simples e eficientes e expor de maneira organizada e intuitiva para o utilizador todas as funcionalidades disponíveis e consultas pretendidas. Para além disto, e como este projeto simula uma interação com utilizadores do mundo real, um dos principais objetivos a cumprir é conseguir pôr em prática as especificações requeridas para esta aplicação.

4. Estrutura da Aplicação

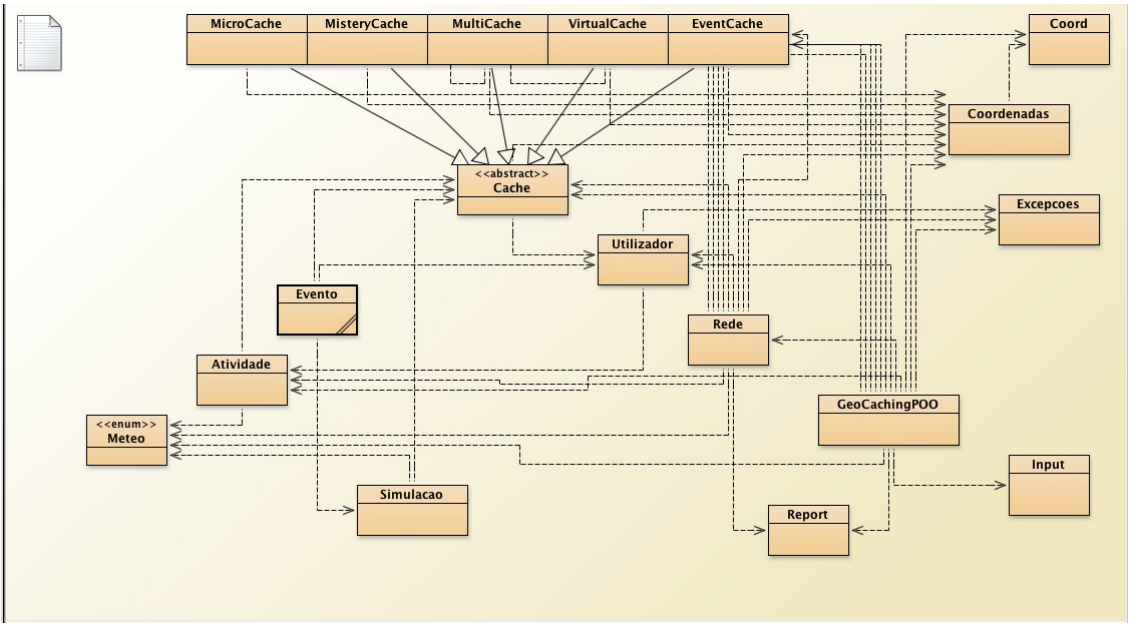

Figura 1: Arquitetura da Aplicação

4.1. Classe Abstrata Cache

Optou-se por uma classe abstrata pois, para além de permitir criar um "esqueleto" para suas subclasses reutilizando código, obriga também que as subclasses implementam vários métodos necessários ao funcionamento da aplicação. Isso faz com que a aplicação seja desde logo extensível no que toca a novas classes de caches.

Para além dos construtores, *getters* e *setters,* ela apenas contém as variáveis de instância:

- *String código*, que define o nome/código identificador da cache;
- *GregorianCalendar data,* define a data da criação da cache;
- *String descrição_extra,* informaçãoo extra que o utilizador considera útil sobre a cache;
- *Coordenadas coord,* coordenadas da localização da cache;
- *String criador,* email do criador.

4.1.1 Classe MicroCache

Subclasse da classe Cache que, para além das variáveis e métodos herdados de *Cache,* implementa os métodos *toString()* e *clone()*.

4.1.2 Classe MisteryCache

Para além das variáveis e métodos herdados de *Cache,* esta subclasse teve os seu construtores, *getters* e *setters* refinados para os seus requisitos e tem implementado também os métodos *toString()* e *clone()*.

```
private String obj;              //descrição do objeto que se encontra na cache
private String tipo;               //Puzzle, Challenge....
private String charada;          // charada a resolver
```

Figura 2: Variáveis de instância da subclasse MisteryCache

4.1.3 Classe MultiCache

Para além das variáveis e métodos herdados de *Cache,* esta subclasse teve os seu construtores, *getters* e *setters* refinados para os seus requisitos e tem implementado também os métodos *toString()* e *clone()*.

```
private String obj;      //descrição do objeto que se encontra na cache
private ArrayList<Coordenadas> lista_coord;   // lista das coordenadas a visitar
```

Figura 3: Variáveis de instância da subclasse MultiCache

4.1.4 Classe VirtualCache

Esta subclasse implementa os métodos *toString()* e *clone()*, para além das variáveis e métodos herdados de *Cache*.

4.1.5 Classe EventCache

Para além das variáveis e métodos herdados de *Cache,* esta subclasse teve os seu construtores, *getters* e *setters* refinados para os seus requisitos e tem implementado também os métodos *toString()* e *clone()*.

```
private TreeSet<String> nomes; //lista dos participantes
private GregorianCalendar dataEvento;  //dia do encontro
```

Figura 4: Variáveis de instância da subclasse EventCache

4.2. Classe Utilizador

Na Classe Utilizador, para além dos campos básicos de informação como nome, morada, credenciais de acesso etc., o grupo adicionou à classe as variáveis de instância identificadas pela figura abaixo, sendo estas as mais merecedoras de atenção pela maior complexidade.

```
private TreeSet<String> amigos;
private TreeSet<String> pedido_amizade;
private HashMap<String,Atividade> atividades;
private TreeSet<String> eventos;
```

Figura 5: Variáveis de instância da Classe Utilizador

Variáveis e métodos de instância

No que toca a variáveis de instância:

- **amigos** contém todos os emails dos utilizadores que são amigos do utilizador
- **pedido_amizade** contém todos os pedidos de amizades pendentes
- **atividades** contém todas as caches descobertas pelo utilizador
- **eventos** contém todos os eventos participados pelo utilizador, sem repetidos

De seguida descrevemos os métodos mais críticos desta classe:

- **public boolean validaLogin(String p)**: método que valida se a string recebida é igual à password deste utilizador
- **public void insereAtiv(Atividade at) throws Excepcoes**: método que recebe uma atividade e a insere na hash table (com o código e a própria atividade)
- **public void removeAtiv(String codigo) throws Excepcoes**: dado o código de uma atividade, este método verifica se esse mesmo código existe na lista de atividades. Se existir, remove-a da HashMap.
- **public void inserePedido(String email) throws Excepcoes**: método que recebe um mail e adiciona esse utilizador aos pedidos de amizade
- **public void removePedido(String email) throw Excepcoes**: método que recebe um email e remove o pedido de amizade
- **public boolean pedidoExiste(String email):** verifica se o email recebido está nos pedidos de amizade
- **public boolean amigoExiste(String email):** verifica se o dado email encontra-se na lista de amigos
- **public void adicionaAmigo(String email) throws Excepcoes:** método que recebe um utilizador e adiciona-o à lista de amigos, caso ainda não exista nessa lista

- **public void removeAmigo(String email) throws Excepcoes: :** método que recebe um utilizador e remove-o da lista de amigos, caso estiver nessa lista
- **public ArrayList<Atividade> ultimasDez():** método que devolve as últimas 10 atividades de um utilizador
- **public ArrayList<Atividade> ordenaAtividades():** método para ordenar as atividades de um utilizador por ordem cronológica
- **public ArrayList<ArrayList<Atividade>> statsMensais(int year) throws Excepcoes:** Neste método o grupo optou por criar um arraylist com posições fixas. São 12 posições sendo que cada uma corresponde às estatísticas de um mês, posição 0 para janeiro até posição 11 para dezembro. Por sua vez cada cada posição do arraylist é também um outro arraylist com todas as atividades realizadas nesse mês (para o ano passado como argumento). Deste modo permite à aplicação uma procura e consulta eficiente para os detalhes de um dado mês e para os detalhes de uma atividade do mês a consultar.
- **public HashMap<Integer, ArrayList<Atividade>> statsAnuais() throws Excepcoes:** stats anuais retorna um hashmap com todos os anos em que foram registadas atividades pelo utilizador. Essa hashmap tem como chave os próprios anos e como valor associado um arraylist com as atividades desse dado ano.

4.3. Classe Atividade

Esta classe é referente ao registo de informações de atividades relativas à descoberta de uma cache. Para além dos construtores, *getters*, *setters*, tem implementado também os métodos *toString()*, *clone()* e um método *public String dataString()* que retorna a data atual da descoberta da cache.

```
private String nome;    //nome ou código desta atividade
private String cod_cache; // código da cache
private String descricao; //descricao da atividade por parte do utilizador que a descobriu
private Meteo meteo;   //meteorologia no decorrer desta atividade
private String obj_retirado;
private String obj_colocado;
private GregorianCalendar data;
```

Figura 6: Variáveis de instância da Classe Atividade

4.4. Classe Evento

```
private String info;          //Informação/Descrição do evento
private String nome;          //Nome do evento
private int maxp;             //Máximo de participantes
private ArrayList<Cache> caches;
private HashMap<String,Integer> participantes;
private GregorianCalendar prazo_insc;
private GregorianCalendar data;
private String vencedor;
```

Figura 7: Variáveis de instância da Classe Evento

Para além das variáveis de instância descritas na figura acima, a Classe Eventos contém também os construtores usuais, os *getters* e *setters*.

4.5. Classes Coordenadas e Coord

Para o tratamento das coordenadas de uma cache, utilizou-se essas duas classes Coord e Coordenadas:

- **Coord,** que implementa o básico das coordenadas (grau, minuto, segundo e a direção). Contém também os construtores, *getters*, *setters*, e os métodos *toString()* e *clone()*

  ```
  private int grau;
  private int min;
  private int seg;
  private char direcao;
  ```

  Figura 8: Variáveis de instância da Classe Coord

- **Coordenadas,** classe criada para tratar da latitude e longitude. Para além das variáveis de instância(***Coord latitude*** e ***Coord longitude***), os construtores, *getters*, *setters*, *toString()*, *clone()*, também implementa 2 métodos essenciais:
  - **Public boolean validaLat(int g, int m, int s, char d):** método para validar a latitude. Recebe os 4 argumentos da latitude, retornando *true* ou *false* para valores válidos ou inválidos, respectivamente. Os argumentos são o grau (entre 0 e 90), os minutos e segundos (ambos entre 0 e 60) e a direção (norte, 'N' e sul, 'S')
  - **Public boolean validaLon(int g, int m, int s, char d):** método para validar a longitude. Recebe os 4 argumentos da longitude, retornando *true* ou *false* para valores válidos ou inválidos, respectivamente. Os argumentos são o grau (entre 0 e 180), os minutos e segundos (ambos entre 0 e 60) e a direção (este, 'E' e oeste, 'W')

4.6. Classe GeocachingPOO (Main)

Classe de tratamento dos vários menus da nossa aplicação. Ela caracteriza-se pela invocação dos vários métodos afetos às outras classes para serem utilizados nos menus a que são específicos, implementando também métodos próprios necessário para o complementar as várias funções requisitadas nos menus de cada utilizador e afins. Contém também as funções de gravação e carregamento de ficheiros.

```
static Input input;
static Rede rede;
static Utilizador user;     //user é a variável global utilizada para guardar o utilizador atual
static String nomeFicheiro;
```

Figura 9: Variáveis de instância da Classe GeocachingPOO

4.7. Classe Rede

Classe implementada para tratamento das várias ações referentes a um utilizador e à sua rede de amigos, que passamos a citar:

- Listagem dos amigos de um utilizador específico
- A validação de um login (administrador ou outro utilizador)
- Criação de novos utilizadores
- Verificar se um email é válido ou se já existe na lista de amigos
- Remover/adicionar um amigo
- Aceitar/remover/enviar um pedido de amizade
- Criação dos vários tipos de caches
- Inserir/remover atividades do próprio utilizador
- Listar todas ou as dez últimas atividades de um utilizador ou de um amigo
- Fazer *report abuse* de uma cache
- Listar as estatísticas mensais e anuais do próprio utilizador ou de um amigo

As estruturas de dados mais utilizadas nesta classe foram a **HashMap** e o **Arraylist.**

```
private HashMap<String,Utilizador> users; //<email,utilizador>
private HashMap<String,Cache> caches;       //<codigo da cache, cache>
private ArrayList<Report> reports;     //lista de reports feitos a caches
private Utilizador admin;
```

Figura 10: Variáveis de instância da Classe Rede

Para além das variáveis de instância descritas na figura acima, e também os construtores usuais, *getters* e *setters,* a classe Rede tem também os seguintes métodos

- **public Utilizador getUtilizador(String email) throws Excepcoes** : método que recebe um email e retorna o utilizador que tem esse mesmo email como chave no HashMap.
- **public String getListaUsers()**: método que imprime a uma lista com o nome e respetivo email dos utilizadores existentes.
- **public boolean validaLoginAdmin(String email, String pass) throws Excepcoes** : método que valida as credenciais de admin recebidas como argumento retornando um booleano com o resultado.
- **public boolean validaLogin(String email, String pass) throws Excepcoes**: este método recebe um mail e uma password e indica se o par é válido retornando um booleano;
- **public void insereNovo(String e, Utilizador ut) throws Excepcoes**: recebendo como parâmetros um email e um utilizador, este método adiciona o novo utilizador na hashmap com o email como chave caso este ainda não exista.
- **public boolean validaMail(String email) throws Excepcoes:** verifica se o email recebido como parâmetro já se encontra registado na hashmap
- **public boolean emailExiste(String mail):** método que retorna um booleano que indica se o email já existe na hashmap
- **public boolean cachesIsEmpty():** este método indica se a lista de caches está vazia ou se já há caches criadas
- **pupublic Utilizador getUser(String email) throws Excepcoes :** método que recebe um email e retorna o utilzador correspondente caso ele exista
- **public void removeAmigo(String email_amigo, String email) throws Excepcoes :** Este método elimina um email da lista de amigos do utilizador com email igual ao do segundo parâmetro. De seguida faz a mesma operação no sentido inverso.
- **public void aceitaPedido(String email_amigo, String email) throws Excepcoes:** dado um email, caso este se encontre na lista de pedidos de amizade do utilizador do segundo parâmetro, este é removido na lista e o email é adicionado na sua lista de amizades (e vice-versa).
- **public void removePedido(String email_amigo, String email) throws Excepcoes:** Este método o é similar ao anterior mas apenas remove o pedido de amizade da lista do utilizador.
- **public int enviarPedido(String email_amigo, String email) throws Excepcoes:** Método que adiciona o email do utilizador na lista de pedidos de amizade do utilizador pretendido (primeiro parâmetro).

- **public boolean existeAmigo(String email_amigo, String email):** Este método verifica de dois utilizadores são amigos.

- .**public boolean cacheExiste(String c):** Este método recebe um código e retorna um booelano indicando se existe uma cache registada com esse código.
- **public Cache getCache(String cod) :** Método que retorna a cache com o código recebido como parâmetro.
- **public ArrayList<Atividade> getDezAtividades(String email):** Método que retorna as ultimas dez atividades, por ordem cronológica, de um dado utilizador.
- **public void insereAtividade(String email, String nome, String cod, String desc, Meteo met, String obr, String obc, GregorianCalendar data) throws Excepcoes:** Método que cria uma nova atividades com os parâmetros recebidos e adiciona essa atividade la lista de atividade de um dado utilizador (passado como parâmetro também)
- **public void removeAtividade(String email, String cod) throws Excepcoes:** Método que recebe o código de uma atividade e um email e remove essa atividade da lista de atividades do utilizador indicado.
- **public void insereReport(String email, String motivo, String cod):** Insere um novo report na lista de reports às caches.
- **Public void setReports(ArrayList<Report> rep):** recebe uma lista atualizada de reports e substitui pela anterior.
- **public void removeCache(String email, String cod) throws Excepcoes:** Método que remove a cache indicada caso a indicação seja feita pelo seu criador ou pelo admin.
- **public ArrayList<Atividade> getAtividades(String email):** Este método retorna a lista de todas as atividades de um dado utilizador.
- **public ArrayList<ArrayList<Atividade>> getStatsMensais(String email, int year) throws Excepcoes:** retorna as estatísticas de cada mês de um dado utilizador.
- **public HashMap<Integer,ArrayList<Atividade>> getStatsAnuais(String email) throws Excepcoes:** retorna as estatísticas de um dado utilizador organizadas pelo respectivo ano de registo
- **public ArrayList<ArrayList<String>> cachesMensais(int year):** retorna a lista de caches criadas num dado ano organizadas pelo mês da sua criação.
- **public ArrayList<ArrayList<String>> registosMensais(int year):** função semelhante à anterior mas desta vez referente ao registo de utilizadores.
- **public HashMap<Integer,ArrayList<String>> cachesAnuais():** retorna um hashmap que tem como chave um ano e como valor a lista de caches criadas nesse ano.
- **public HashMap<Integer,ArrayList<String>> registosAnuais():** método semelhante ao anterior mas referente ao registo de utilizadores.

Os seguintes métodos recebem os parâmetros das caches corresnpondentes, criam uma nova cache e adicionam essa mesma cache na hashmap de caches da rede:

- **public void criaMicroCache(String cod, GregorianCalendar data, String descricao, Coordenadas cor, String criador) throws Excepcoes;**

- **public void criaMisteryCache(String cod, GregorianCalendar data, String descricao, Coordenadas cor, String criador, String obj, String tipo, String charada) throws Excepcoes;**

- **public void criaMultiCache(String cod, GregorianCalendar data, String descricao, Coordenadas cor, String criador, String obj, ArrayList<Coordenadas> lista) throws Excepcoes;**

- **public void criaVirtualCache(String cod, GregorianCalendar data, String descricao, Coordenadas cor, String criador) throws Excepcoes;**

- **public void criaEventCache(String cod, GregorianCalendar data, String descricao, Coordenadas cor, String criador, GregorianCalendar dataEncontro) throws Excepcoes ;**

Os seguintes métodos recebem como parâmetro o código de uma cache e confirmam se a classe dessa cache é a indicada ou não:

- **public boolean isMultiCache(String c);**
- **public boolean isMisteryCache(String c);**
- **public boolean isVirtualCache(String c);**
- **public boolean isEventCache(String c);**

5. Decisões tomadas

- Optamos por definir a **Classe Meteo** como **enum** para o tratamento das constantes referentes à meteorologia utilizadas pela nossa aplicação, conforme ilustrada na figura abaixo

```
public enum Meteo
{
    CHFR{
        public String toString(){ return "Chuva fraca"; }
    },
    CH{
        public String toString(){ return "Chuva"; }
    },
    CHFT{
        public String toString(){ return "Chuva forte"; }
    },
    SOL{
        public String toString(){ return "Sol"; }
    },
    NEV{
        public String toString(){ return "Nevoeiro"; }
    },
    TEMP{
        public String toString(){ return "Tempestade"; }
    },
    NUB{
        public String toString(){ return "Nublado"; }
    };
}
```

Figura 11: Descrição da Classe Meteo

- A **Classe Input**, nada mais é do que a nossa implementação do *Scanner*. Tomámos essa decisão para assegurar, por exemplo, que quando é necessário um tipo de input (por exemplo inteiro) a aplicação só avança quando for introduzido o tipo esperado(se esperar por um inteiro e o user introduzir um char ele invalida e pede de novo o input).

  Os métodos da classe utilizados foram:

  - **public static String lerString()**
  - **public static int lerInt()**
  - **public static double lerDouble()**
  - **public static float lerFloat()**
  - **public static boolean lerBoolean()**
  - **public static short lerShort()**
  - **public static char lerChar()**
- A **Classe Excepcoes** foi implementada para o tratamento dos vários casos de excepcoes da aplicação. É uma extensão da classe *Exception* do JAVA
- O grupo decidiu utilizar os *HashMap's* para uma pesquisa mais eficiente e melhor organização (por exemplo, para os utilizadores da rede)
- Os *Set's* foram utilizados para evitar a repetição de valores (por exemplo, nos pedidos de amizade, não se pode ter mais que um pedido de amizade de cada user. Ou seja, se um utilizador 1 manda 3 pedidos a um outro utilizador 2, o email do utilizador 1 só fica registado nos pedidos uma só vez)

6. Conclusão

Após a conclusão deste projeto, podemos afirmar que adquirimos bastantes conhecimentos no que toca à Programação Orientada a Objetos. Fizemos uso das diversas capacidades da linguagem JAVA, incluindo o polimorfismo que se alcança com a abstração de classes.

Um dos objectivos foi a hierarquia das classes para assim haver reutilização do código e facilitar inserções de novas classes e subclasses.

Com o código feito desta maneira evita-se repetição de código, já que o que é comum a alguns tipos encontra-se na classe principal, por exemplo, a informação comum a todas as caches encontra-se na classe Cache e o que é característicos de cada tipo de cache é especificado em cada subclasse. Também assim se torna mais fácil a leitura.

Para guardar informação utilizou-se coleções do tipo *Map's, Set's e ArrayList.* Esta decisão foi bastante importante pra o grupo, pois uma boa escolha destas coleções permite um dimensionamento bastante superior do programa e tempos inferiores de execução.

Uma etapa realizada com sucesso foi a criação dos métodos que tornaram possível a utilização do programa. Com isso realizado, o programa possuí uma grande variedade de funcionalidades que o utilizador pode tirar proveito.

7. ANEXOS

Anexo 1: Primeiro menu

Anexo 2: Menu de *Login*

```
 ____   ____   ____   ____   ____   ____   _  _   ___   _  _   ____
| ___| | ___| |  _ | | ___| |  _ | | ___| | || | |_ _| | \ | | | ___|
||  __  ||___  | | | | ||    | |_| | ||    | |_| |  ||  |  \| | ||  __   __   __   __
|| |_ | | ___| | | | | ||    |  _  | ||    |  _  |  ||  |     | || |_ | | _ || ||  ||  |
||___|| ||____ | |_| | ||___ | | | | ||___ | | | | _||_ | |\  | ||___|| | _|| | || | |
|_____| |_____| |____| |_____| |_| |_| |____| |_| |_| |___| |_| \_| |_____| |_|  |__||__|


  ***          MENU PRINCIPAL             ***


1 - Lista de utilizadores
2 - Amigos
3 - Caches & Atividades
4 - Consultar dados pessoais
5 - Editar dados pessoais
6 - Estatísticas Mensais GeoCachingPOO
7 - Estatísticas Anuais GeoCachingPOO
8 - Gravar
9 - Logout
```

Anexo 3: Menu principal (utilizador já fez *login*)

```
*** Lista de utilizadores ***

        NOME        |       EMAIL
-----------------------------------------
Maria Joao - mjoao@gmail.com
Helder Machado - hmachado@poo.pt
Luisa - luisa@hotmail.com
Maria - maria@gmail.com
Edison - edison@gmail.com
Rafael - rafael123@gmail.com
Pedro - pedro@yahoo.com
admin - admin@geocachingpoo.pt
Joao - john@gmail.com
Joao Monteiro - jmonteiro@uminho.pt
Helder - helder@gmail.com
John - user1989@gmail.com
-----------------------------------------

Prima ENTER para continuar...
```

Anexo 4: Listagem dos utilizadores registados

```
  ***            AMIZADES           ***

1 - Lista de amigos
2 - Lista de pedidos
3 - Aceitar pedido
4 - Remover pedido
5 - Enviar pedido
6 - Remover amigo
7 - Consultar dados pessoais de uma amigo
0 - Sair
```

Anexo 5: Menu referente a operações sobre amizades

```
  ***          CACHES & ATIVIDADES          ***

1  - Criar cache
2  - Remover Cache
3  - Consultar caches
4  - Registar nova atividade
5  - Últimas 10 atividades
6  - Consultar todas as atividades
7  - Remover atividade
8  - Estatísticas Mensais
9  - Estatísticas Anuais
10 - Report Abuse
11 - Consultar Reports(Admin)
0  - Sair
```

Anexo 6: Menu Caches e Atividades

```
** Estatísticas do ano 2015 **

Mês 1 -> Utilizadores registados: 0 | Caches criadas: 0
Mês 2 -> Utilizadores registados: 0 | Caches criadas: 1
Mês 3 -> Utilizadores registados: 0 | Caches criadas: 0
Mês 4 -> Utilizadores registados: 0 | Caches criadas: 0
Mês 5 -> Utilizadores registados: 0 | Caches criadas: 0
Mês 6 -> Utilizadores registados: 12 | Caches criadas: 0
Mês 7 -> Utilizadores registados: 0 | Caches criadas: 0
Mês 8 -> Utilizadores registados: 0 | Caches criadas: 0
Mês 9 -> Utilizadores registados: 0 | Caches criadas: 0
Mês 10 -> Utilizadores registados: 0 | Caches criadas: 0
Mês 11 -> Utilizadores registados: 0 | Caches criadas: 0
Mês 12 -> Utilizadores registados: 0 | Caches criadas: 0



1 - Consultar emails registados num dado mês
2 - Consultar códigos de caches criadas num dado mês
0 - Sair
```

Anexo 7: Estatísticas mensais da aplicação de um dado ano

```
*** Registo de utilizadores ***

Ano 2015 -> Utilizadores registados: 12

_____________________________________

*** Criação de caches ***

Ano 2015 -> Caches criadas: 12



1 - Consultar emails registados num dado ano
2 - Consultar códigos de caches criadas num dado ano
3 - Consultar Estaísticas mensais de um ano
0 - Sair
```

Anexo 8: Estatísticas anuais da aplicação