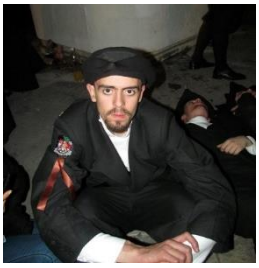
Universidade do Minho

Licenciatura em Engenharia Informática

Programação Orientada aos Objetos

GeocachingPOO

GRUPO 8

		
Célia Figueiredo a67637	Gil Gonçalves A67738	José Carlos Faria

Braga 6 de Junho de 2015

Conteúdo

Introdução

No âmbito da cadeira de Programação Orientada aos Objetos, perante o problema apresentado iremos desenvolver em Java uma aplicação que permita registar e simular atividades de registo e descoberta de caches.

Para começar é importante saber em que consiste esta atividade, portanto, ***Geocaching*** é um passatempo e um desporto ao ar livre no qual se utiliza um recetor de navegação por satélite (GPS), para encontrar uma "geocache" (ou simplesmente "cache") colocada em qualquer local do mundo. Uma cache típica é uma pequena caixa (ou tupperware), fechada e à prova de água, que contém um livro de registo e alguns objetos, como canetas, afia-lápis, moedas ou bonecos para troca, e o geocacher aponta no livro de registos as coordenadas (latitude e longitude) da cache. Estas, em conjunto com outra informação sobre o local do esconderijo, são publicadas na Internet. Os outros geocachers, os descobridores, leem essa página e, com recetores GPS, procuram-na. Quando o conseguem, registam o achado na mesma página. Os Geocachers são livres de colocar ou retirar objetos da cache, normalmente por troca de coisas de pequeno valor, de modo a haver sempre qualquer recordação para trazer.

Para a realização deste projeto vai implementar-se um conjunto de classes que definem os objetos presentes no sistema de software. Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos. Será também construído um menu para facilitar a utilização dos métodos anteriormente referidos, e tornar o programa mais funcional e interessante.

Análise e especificação do projeto

Este projeto insere-se na Unidade Curricular de Programação Orientada aos Objectos, que consiste em criar uma aplicação que funcionará de maneira semelhante à já existente "Geocaching", que será descrita como foi implementada de seguida.

Descrição Informal do Projeto

Após a leitura e discussão do enunciado decidiu-se que teríamos de ter uma classe **Utilizador** para que fosse possível criar um perfil (nome, idade, género, data de nascimento, morada), este associado a um e-mail (único) e uma palavra-passe (5 caracteres). Também será possível adicionar e remover amigos, procurar e esconder caches, participar em atividades de lazer e eventos, também poderá fazer report da cache, caso o utilizador veja que algo está errado com ela. Para além disso o utilizador também poderá aceder às estatísticas das suas atividades, como também as estatísticas das caches que encontrou, assim como aceder às condições meteorológicas do local, se houver uma atividade que esteja à muito tempo no sistema e o utilizador a quiser apagar também irá ser possível, contudo se o utilizador encontrou/escondeu alguma cache durante a execução da atividade, esta cache não irá ser apagada.

Visto que temos vários tipos de caches será necessário a criação de uma super Classe (**Cache**) para definir os métodos mais gerais para as subclasses (**Cache_misterio, Micro_cache, Mult_cache**) herdarem os métodos.

Será necessária a criação de uma classe **Geocaching** para fazer a comunicação entre o sistema e o utilizador. Assim como a classe **Gestor** que servirá para interação do utilizador com o sistema, isto é a interação input/output.

De seguida foram explicadas as classes que foram criadas para a concretização deste projeto.

Classe Admin

A classe **Admin** permite remover as cache que foram reportadas, permite também que ele próprio adicione e remova caches que ele criou.

Classe Atividade

A classe **Atividade** é uma classe abstrata que permite identificar o tipo de atividade, a data em que ela foi feita, no local e a meteorologia, foram usados os métodos gets e sets.

Classe Cache

A classe **Cache** é uma classe abstrata, que permite identificar o local onde se encontra, o tamanho da cache, o tipo da cache, a data que ela foi criada, a data em que foi encontrada pelo 'geocacher', o nome da cache e o e-mail de quem a inseriu. Os métodos utilizados são os gets e sets.

Classe Geocaching

A classe Geocaching tem HasMap com os utilizadores todos, todas as caches do sistema que foram criadas por qualquer utilizador ou admin, com os reports (quando uma cache é reportada, fica escondida dos utilizadores, mais tarde o admin decide se elimina acache ou não). Os métodos mais importantes desta classe são:

- **encontrarCache**: que permite adicionar uma cache que foi encontrada pelo utilizador;
- **criarCacheMicro**: permite criar micro-cache no sistema e adiciona a respectiva cache ao HasMap das Caches que foram criadas pelo Utilizador.
- **reportAbuse** : O utilizador pode reportar algo que não esteja bem ao admin.

Classe Gestor

A classe **Gestor** faz a ponte entre o utilizador e o sistema. Juntamente com a classe "Menus", o Gestor guia o utilizador nos diversos passos de interação com a plataforma e é também aqui que é gravado o estado atual da aplicação.

Classe Utilizador

A classe **Utilizador** tem um email associado (tem de ser um email único) e uma password (com 5 caracteres), tem uma morada, um género (M/F), tem um HashMap de amigos, pedidos de amizade, caches encontradas, cahes criadas, TreeSet que está ordenado pela data.

Foram criados os métodos **<estatisticaActividades>** este que calcula as estatisticas mensais das atividades dos utilizadores; **<estatisticaAnual>** cálcula as estatísticas anuais das atividades do utilizador; **<estatisticaMensalCache>** cálcula as estatisticas mensais das caches que cada utilizador encontrou; **<estatisticaAnualCache>** cálcula as estatísticas anuais das caches encontradas pelos utilizadores; **<calculaIdade()>** calcula a idade do utilizador.

Conceção / desenho da resolução

Estruturas de dados

Na classe **Utilizador** estão inseridas várias atividades e as atividades estão ordenadas pela data, usou-se um **TreeSet<Actividade>**, que é necessário fornecer um método de comparação dos objetos **compare()** ou **compareTo()**, neste caso criou-se um método **DataComparator**, este que organiza as atividades de forma descendente, ou seja, das mais recentes para as mais antigas, sem este método de

comparação não é possível utilizar o TreeSet, a não ser para tipos de dados simples (String, Integer, etc).

Como já foi dito anteriormente a classe atividade é uma classe abstrata que foi criada dessa maneira de forma a nunca poder ser instanciada, servindo apenas de “base” para um tipo de dados que vai ser partilhado por todas as classes que a extendem.

Sub-classes da Atividade

Nesta parte escolheram-se apenas as quatro atividades que se seguem, porque já servem para ilustrar o funcionamento do programa. O facto da classe **Atividade** ser abstrata permite futuramente acrescentar o número que desejar de diferentes atividades, todas elas como as propriedades “básicas” de Atividade acrescentadas às características individuais de cada uma. Sendo as subclasses da Atividade: **Virtual, Esconder, Procura ,Cache_evento .**

A classe **Virtual**, é uma classe para as atividades de lazer dos vários Utilizadores. Nesta classe estarão implementados os métodos herdados da classe atividades, acrescentando o local onde o utilizador quer que os restantes utilizadores visitem.

A classe **Esconder** é referente ao tipo de atividades de esconder caches para que os restantes utilizadores as possam encontrar, esta classe irá ter os métodos herdados pela classe atividade acrescentando ao local os ele escondeu a cache, se o utilizador escondeu uma Multi Cache ou uma cache Mistério, esta coordenada é sempre referente à Coordenada final, contém também o tipo de cache que escondeu como o seu tamanho.

A classe **Procura** irá ter os métodos herdados pela classe Atividade, acrescentando o caminho que o utilizador fez durante a sua procura e o tipo de procura que ele fez, ou seja se foi procurar uma micro Cache, se foi procurar uma Multi Cache ou se foi procurar uma cache Mistério.

Classe GeoCaching

Na classe **GeoCaching** tem-se então três estruturas de dados que serão importantíssimas para o bom funcionamento da plataforma. A primeira usada para armazenar todos os utilizadores do sistema (**HashMap<String,Utilizador>**), associa a cada chave (o email) o respetivo utilizador. A segunda que vai guardar o nome de todas as atividades existentes no sistema **(TreeSet<String>)**. A terceira é uma HashMap com todas as caches criadas pelo utilizador, todas as caches que foram reportadas por último também tem o ‘admin’ do sistema que não faz parte da lista de utilizadores.

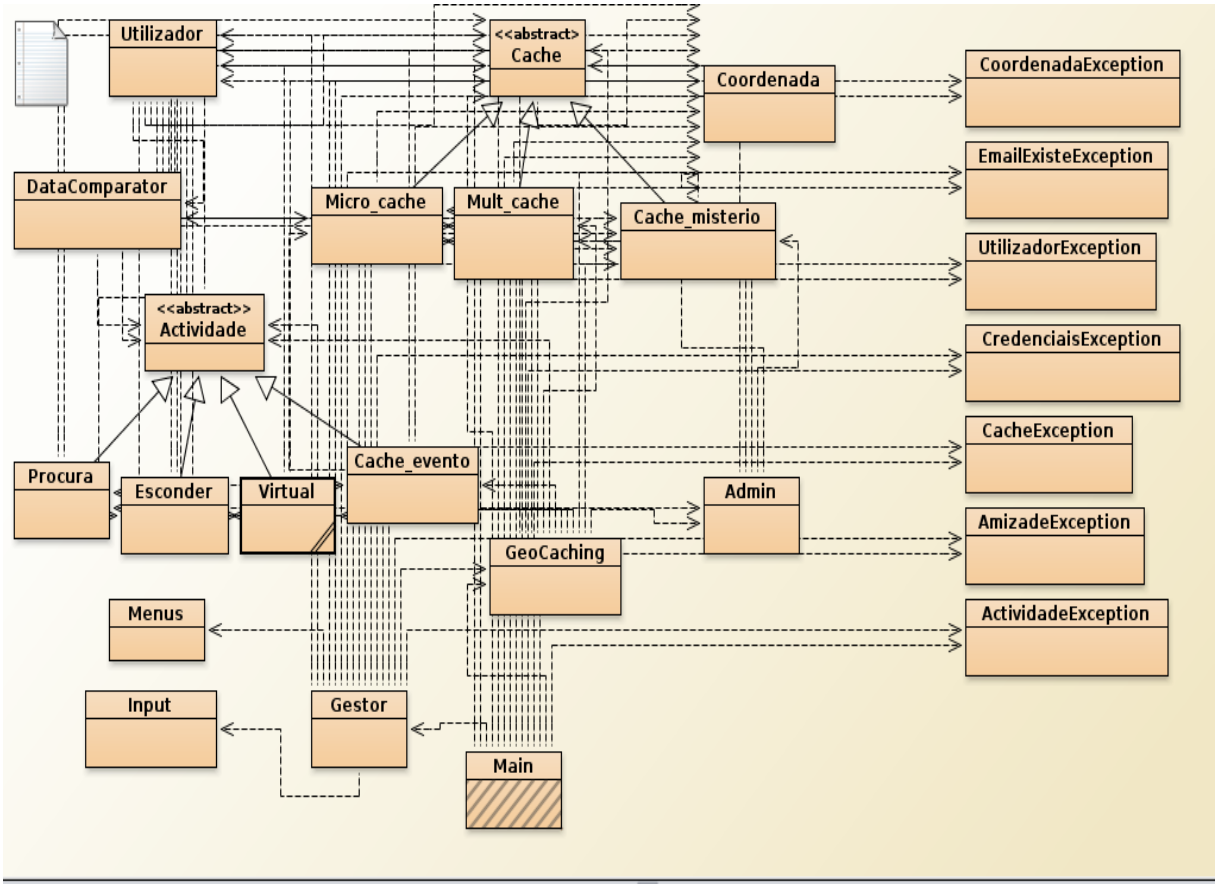


Figura 1 Diagrama do BlueJ

Resultados

Classe Main

Nesta classe estará a estrutura do nosso programa principal, desta forma, encontram-se definidos os métodos de leitura e escrita da informação que se encontra no sistema. Mais propriamente, possibilita carregar a informação das caches e dos utilizadores.

Classe Menu

Esta classe foi criada para facilitar a organização da informação a ser processada. Para que seja de fácil manuseamento interagir com o sistema, de seguida mostramos como estão organizados os menus.

Para começar o menu principal tem como funções registar um novo utilizador, fazer login de um utilizador já registado, carregar dados, abrir a conta de administrador e fechar a aplicação, sendo que cada uma dessas opções tem a si associado um número que é pressionado através do teclado. Por outro lado quando já estamos dentro da nossa conta temos uma série de opções, tais como ver os

nossos dados pessoais, ver as atividades, gravar os dados, ver as caches, assim como gerir os pedidos de amizade.

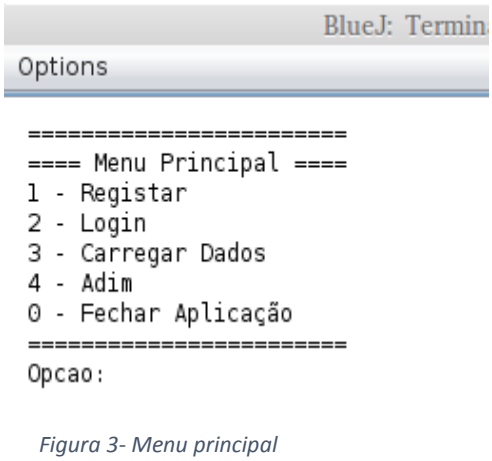


Figura 3- Menu principal

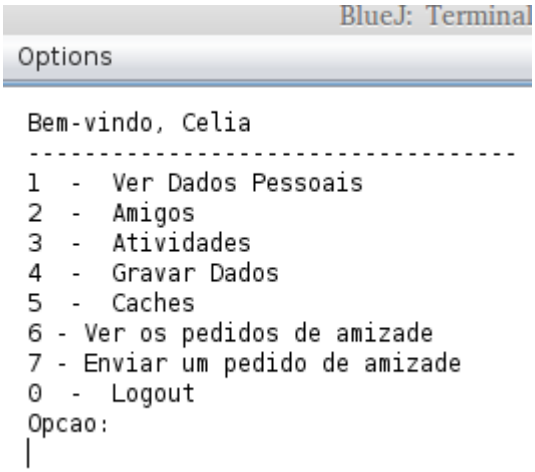
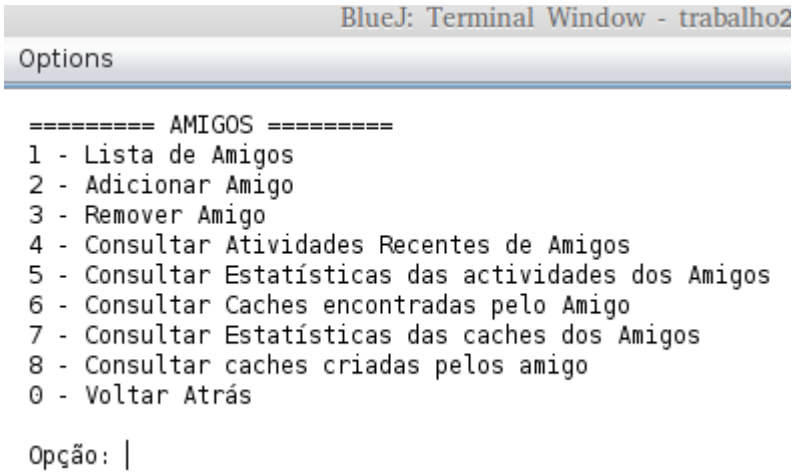
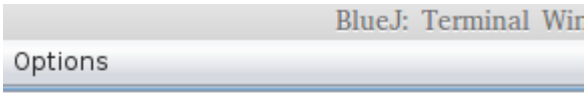


Figura 2 - Menu após se fazer o login

Quando escolhemos a opção AMIGOS, podemos ver quem são os nossos amigos, adicionar ou remover, e consultar as estatísticas deles.



Quando escolhemos a opção Caches, podemos consultar as caches tanto criadas como encontradas, ou reportar uma cache.




```
=================
=======Caches=======
1  -  Consultar caches encontradas
2  -  Consultar estatistica das caches
3 -  Report abuse cache
4 -  Consultar caches  criada
5  - Consultar caches do sistema
0  -   Voltar Atrás

Opção:
```

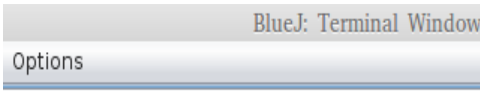
Quando escolhemos Consultar as Caches encontradas podemos ver qual o tipo de cache que foi encontrada.

```
Opção:
1
=======Consultar Caches encontradas===============

ID -    Tipo de Cache -     Nome da Cache
1     CacheMisterio                          r


2 - Remover cache
1 - Ver Detallhes
0 - Volta atras
Opcao:
```

As ATIVIDADES também podem ser consultadas, tanto as mais recentes como a lista completa das atividades, assim como registar uma atividade, ou consultar as estatisticas.




```
========= ATIVIDADES =========
1 - Registar Atividade
2 - Consultar Atividades Mais Recentes
3 - Consultar Lista Completa Atividades
4 - Estatísticas
0 - Voltar Atrás

Opção:
```

```
Opção:
2
-------------------------------------------------------------
ID - Tipo       -       Duração       -       Data
-------------------------------------------------------------
1   Lazer            20                  10/jan/2014
2   Lazer            20                  1/fev/2014
3   Esconder cache           200                 12/mar/2014
4   Lazer            20                  3/abr/2014
5   Lazer            20                  2/mai/2014
6   Lazer            20                  1/jun/2014
7   Lazer            20                  10/jul/2014
-------------------------------------------------------------



1 - Ver Detalhes
0 - Voltar Atrás

Opcao:
```

Conclusão

Neste projeto um dos objetivos 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 abstrata Cache e o que é característica 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. A interface final, menu, oferece ao utilizador uma utilização fácil e prática fase as funcionalidades que eram esperadas do programa.