|  |
| --- |
| Universidade do Minho |
| GeoCachingPOO |
| Trabalho Prático |
|  |
| **6/6/2015** |
|  |

Rui Pereira

a67747

Nuno Oliveira

a67649

Rafael Antunes

a67711



|  |
| --- |
| Programação Orientada a Objetos  Licenciatura em Engenharia Informática |

Índice

[1. Introdução 3](#_Toc421387690)

[2. Arquitetura de Classes 4](#_Toc421387691)

[I. View 4](#_Toc421387692)

[II. Business 4](#_Toc421387693)

[III. Data 5](#_Toc421387694)

[IV. Exceptions 5](#_Toc421387695)

[3. Estruturas de Dados 6](#_Toc421387696)

[3.1 Utilizadores 6](#_Toc421387697)

[3.2 Caches 6](#_Toc421387698)

[I. Cache Tradicional 7](#_Toc421387699)

[II. Cache Mistério 7](#_Toc421387700)

[III. Multi-Cache 8](#_Toc421387701)

[IV. MicroCache 8](#_Toc421387702)

[V. Cache Evento 8](#_Toc421387703)

[*3.3* *TravelBugs* 8](#_Toc421387704)

[3.4 Atividades 8](#_Toc421387705)

[4. Principais Decisões Tomadas 10](#_Toc421387706)

[5. *Screenshots* da Aplicação 11](#_Toc421387707)

[6. Classe de Testes 13](#_Toc421387708)

[7. Conclusões e Críticas 14](#_Toc421387709)

# Introdução

Neste trabalho prático da disciplina de Programação Orientada a Objetos (POO) foi proposto o desenvolvimento de uma aplicação que simule o registo e descoberta de caches. Esta aplicação deve alem permitir o registo de diferentes tipos de caches com categorias diferentes, conter um sistema de utilizadores que podem assinar as caches criadas, adicionar outros utilizadores como amigos, e ver as suas atividade realizadas. Esta aplicação deve ser desenvolvida na linguagem JAVA.

# Arquitetura de Classes

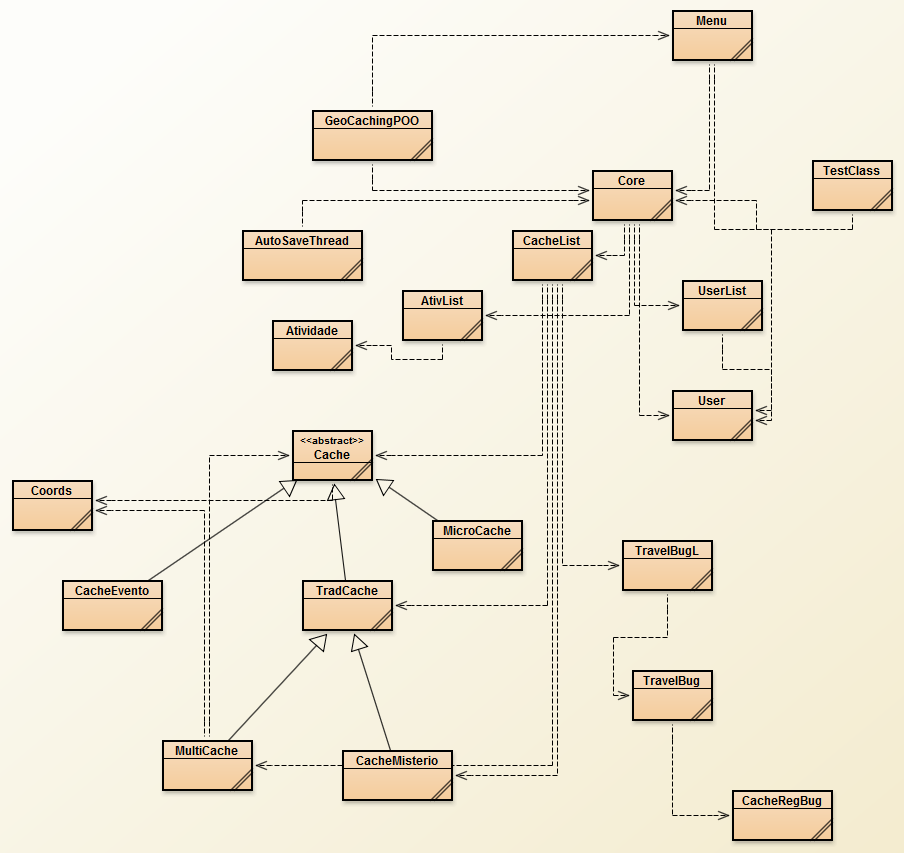


Figura Imagem projeto BlueJ (sem packages)

A aplicação esta dividida em 4 packages:

### View

A package *View* contem apenas uma classe *Menu,* que contem todos os menus da aplicação.

### Business

A package *Business* contem 4 classes:

* **GeoCachingPOO –** Classeprincipal da aplicação que inicializa as classes *Core, AutoSaveThread.*
* **Core –** classe responsável por receber os pedidos dos Menus e pedir a informação às estruturas de dados.
* **AutoSaveThread –** Classe corre uma segunda *thread*, responsável por gravar o estado da aplicação em ficheiro.
* **TestClass –** Classe que contem uma função de povoamento das estruturas de dados.

### Data

A package *Data* contem todas as estruturas de dados utilizadas pela aplicação. Estas estruturas estão responsáveis por organizar e armazenar:

* **Users** – lista de utilizadores da aplicação.
* **Caches –** 5 tipos diferentes de Caches
* **Actividade –** Lista de atividade realizadas pelos utilizadores
* **TravelBugs –** Objetos armazenados nas caches que mantem o registo das caches em que foram armazenados.

A organização destas estruturas de dados será explicada no capítulo *4 Estruturas de Dados*

### *Exceptions*

A package *Exceptions* contem todas as exceções que são utilizadas na aplicação.

Nota: Esta aplicação, apesar de aplicar alguns conceitos de programação em três camadas, não utiliza o modelo na sua totalidade (ex.: A camada de Negócios conhece a classe utilizador).

# Estruturas de Dados

A aplicação possui 3 grandes estruturas de dados independentes entre si, Utilizadores, Caches e Atividades, que são inicializadas pelo objeto da classe Core criado quando se inicia a aplicação, e uma estrutura de dados que contem a lista de *TravelBugs* que está inserida na estrutura de dados das Caches.

## Utilizadores

Os dados dos utilizadores são guardados na classe *UserList* num *HashMap*, em que as *keys* são o identificador do utilizador, que neste caso é o *email* de utilizador não podendo assim ser alterado, e os *values* são os objetos da classe *User* que contem a informação do utilizador.

A classe *User* possui os seguintes dados relativos ao utilizador:

* Endereço de *email* (que funciona também como identificador)
* *Password*
* Nome
* Morada
* Data de Nascimento (armazenada num *LocalDate*)
* Pontuação (pontos adquiridos com a assinatura de Caches)
* Lista de Amigos
* Lista de Pedidos de amizade

A lista de amigos é um *HashMap<String, User>* que contem o identificador e o apontador para o utilizadores que foram adicionados como amigos.

A lista de pedidos de amizade é similar a lista de amigos, mas com os utilizadores que aguardam uma resposta ao pedido de amizade.

## Caches

Os dados das caches são guardados na classe *CacheList*, que tem um HashMap em que as *keys* são o identificador da cache e os *values* são os objetos da classe *Cache* que contem a informação do utilizador. Esta cache contem também um objeto da classe TravelBugL, que contem a estrutura de dados que armazena a informação relativa aos *Travel Bugs*.

A classe *Cache* é uma classe abstrata que possui os seguintes dados relativos a uma cache:

* Referência da Cache
* Coordenadas (armazenadas num objeto da classe *Coords*)
* Criador (email do utilizador que criou a cache)
* Descrição da Cache
* Dificuldade da Cache (Valor numérico utilizado para calcular pontuação)
* Lista de Assinantes
* Número de *Reports*

A lista de assinantes é um *HashSet* de *Strings* que correspondem ao *email* dos utilizadores que assinaram a cache.

Esta classe é estendida por 5 classes correspondentes a 5 tipos diferentes de caches com características diferentes, obedecendo ao seguinte esquema:

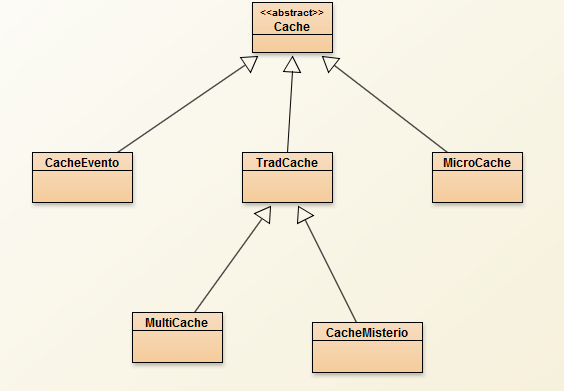


Figura Esquema BlueJ relativo à herança da classe Cache

### Cache Tradicional

A cache Tradicional, definida pela classe *TradCache,* é uma cache em que é possível armazenar tesouros e *TravelBugs*. A classe contem as seguintes variáveis de instância:

* Lista de Tesouros
* Lista de *TravelBugs*

A lista de Tesouros é um *HashSet* de *Strings* que correspondem a um tesouro. Este tesouro é codificado apenas por esta String não sendo registadas as suas características ou se passou por outras caches.

A lista de *TravelBugs* é um *HashSet* de *Strings* que correspondem à referência do *TravelBug* (ver 3. TravelBugs).

A classe *TradCache* é estendida por dois tipos de caches: a Cache Mistério e a *Multi-Cache*.

### Cache Mistério

A cache mistério, definida pela classe *CacheMisterio*, é uma cache Tradicional que contem um puzzle que deve ser resolvido para encontrar a cache. A classe contem as seguintes variáveis de instância:

* Descrição do puzzle
* Pontos Extra (Pontos de dificuldade extra relacionados com a dificuldade do puzzle)

### Multi-Cache

A *Multi-Cache*, definida pela classe *MultiCache,* é uma cache tradicional que possui uma lista de coordenadas intermedias que tem de ser descobertas para se descobrir a coordenada seguinte. A classe contem as seguintes variáveis de instância:

* Pontos Intermédios
* Pontos Extra (Pontos de dificuldade extra relacionados com a dificuldade dos pontos intermédios)

### MicroCache

A *MicroCache*, definida pela classe *MicroCache*, é uma cache que ao contrário das caches Tradicionais é de pequenas dimensões, não sendo possível armazenar objetos. Esta classe apenas possui a variável relativa à dificuldade extra, que por pré definição toma o valor de 2, e não pode ser alterado.

### Cache Evento

Uma cache evento, definida pela classe *CacheEvento*, é uma cache codifica um evento. A classe contem as seguintes variáveis de instância:

* Lista de Organizadores do Evento
* Data de Realização do Evento
* Pontos Extra (Bónus de pontuação atribuído por participar no evento)

A lista de Organizadores é um *HashSet* com os *emails* dos utilizadores que estão a organizar o evento para alem do criador do mesmo.

## *TravelBugs*

Os dados relativos aos TravelBugs estão guardados num objeto da classe TravelBugL que contem um HashMap em que as *keys* são as referências dos TravelBugs criados, e os *values* são os objetos da classe *TravelBug* que contem a informação relativa a um *TravelBug*.

A classe *TravelBug* possui as seguintes os seguintes dados:

* Referência/Descrição do *TravelBug*
* Lista de Registos de caches por onde passou
* Cache onde se encontra atualmente

A lista de registos é um *HashSet* de Objetos da classe *CacheRegBug* que tem os seguintes dados:

* Referência de uma cache
* Data em que o *TravelBug* foi colocado na cache

## Atividades

O registo das atividades realizadas pelos utilizadores é armazenado num objeto da classe *AtivList* que contem um *ArrayList* de objetos da classe *Atividade*.

A classe *Atividade* contem os seguintes dados:

* Data de realização da Atividade
* Endereço de *email* do utilizador que a realizou
* Descrição da atividade

A descrição da atividade é gerada através dos métodos de inserção de dados no registo. Estes métodos fornecem a informação da atividade que foi executada e gera a *String* que descreve a atividade. A data de realização e o *email* do utilizador são utilizados para os métodos de seleção de atividades por lista de utilizadores e para organizar por ordem cronológica.

# Principais Decisões Tomadas

A aplicação foi desenhada do ponto de vista da gestão de uma rede de caches e não como o jogo de descoberta de caches. Não é necessária a apresentação de uma prova para assinar a cache e informações que normalmente estão escondidas como as coordenadas de uma cache Mistério ou Multi-Cache estão visíveis mesmo sem resolver o puzzle ou descobrir pontos intermédios.

O clima tem apenas 3 níveis, sendo 1 bom tempo, 2 tempo com algumas condições adversas (Ceu muito nublado, chuva leve, *etc*) e 3 mau tempo (Chuva intensa, Noite cerrada, *etc*). Este valor é calculado aleatoriamente cada vez que uma cache é assinada e contribui para a pontuação recebida pela assinatura da cache.

A pontuação atribuída pela assinatura da cache é a soma do valor de dificuldade, a pontuação extra (se aplicável), e do clima. Esta pontuação adicionada a pontuação total do utilizador.

Não foi previsto ou implementado o perfil de administrador da aplicação, por isso, só o criador da cache pode efetuar alterações á mesma e qualquer utilizador pode criar qualquer tipo de cache. A cache no entanto será removida automaticamente se for reportada com imprópria 10 vezes.

As atividades realizadas pelos utilizadores são guardadas em formato de *String*. Esta *String* é criada no momento em que a atividade é realizada, e está associada a um utilizador e à data de realização da mesma. As atividades do próprio e dos utilizadores na sua lista de amigos podem ser apresentadas a pedido do utilizador, ordenadas por ordem cronológica.

Os eventos são criados sob a forma de uma cache, possuindo a data em que se realiza o evento e as coordenadas do mesmo. A assinatura desta cache considera-se como marcação de presença no evento. O evento não pode ser assinado numa data posterior a data do evento. É possível também ver todos os eventos que serão realizados posteriormente à data de procura.

O estado da aplicação é guardado utilizando *serialization*. Todas as estruturas de dados e o objeto da classe core utilizado pela aplicação sal guardados deste modo.

Foi implementado a funcionalidade de *autosave* na aplicação que guarda o estado da aplicação de 5 em 5 minutos sem a intervenção do utilizador. O *autosave* baseia-se num método com um *sleep* que de 5 em 5 minutos acorda e guarda o estado da aplicação. A classe que contém este método corre numa *thread* à parte, e é fechada quando a aplicação encerra.

# *Screenshots* da Aplicação

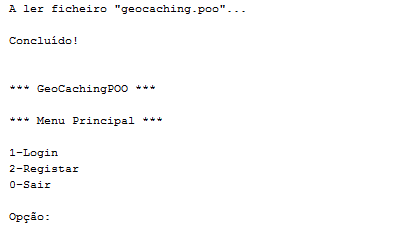


Figura Menu de Entrada da aplicação

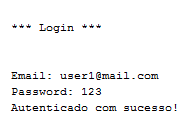


Figura Menu de Login da Aplicação

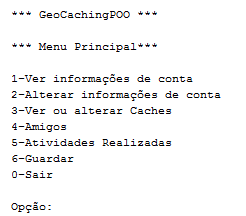


Figura Menu principal da aplicação

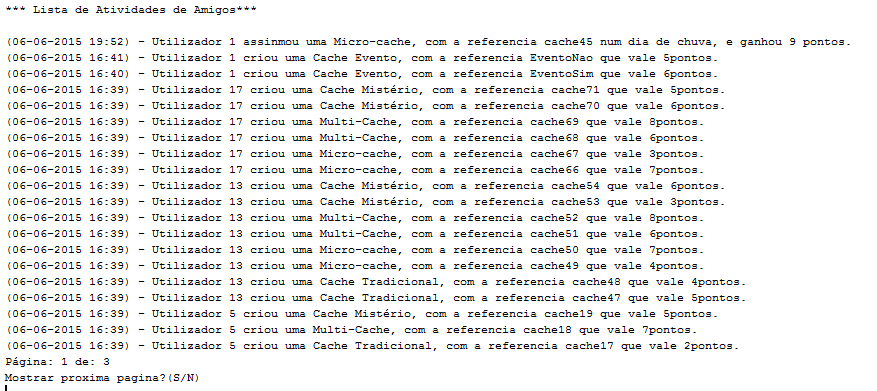


Figura Lista de Atividades dos Amigos de um utilizador

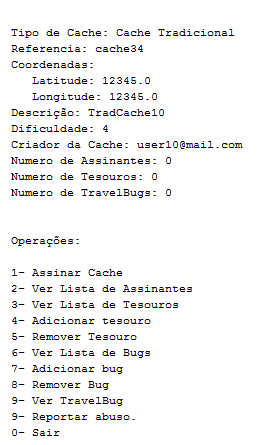


Figura Detalhes de uma cache e menu de operações da mesma

# Classe de Testes

Foi criada uma classe de testes que povoa as estruturas de dados da aplicação com um número definido de utilizadores e caches criadas pelos mesmos. Nesta versão da classe apenas são adicionados utilizadores, caches, e alguns pedidos de amizade, bem como as atividades criadas por consequência da criação dos mesmos. A classe não utilizadas operações de assinar caches, ou operações relativas aos tesouros ou *TravelBugs*. A classe quando utilizada descarta todos os dados da aplicação.

Para correr o povoamento de teste da aplicação deve-se utilizar o valor 3 no menu apresentado na Figura 3.

A função de povoamento vai criar vinte utilizadores com o *email* “user*α*@mail.com” em que *α* é o numero do utilizador que vai de 0 a 20; e com a palavra-passe “123”. Estes dados podem ser utilizados para efetuar o login na aplicação.

# Conclusões e Críticas

O objetivo para ente trabalho era desenvolver uma aplicação em java, que permitisse registar e simular atividades de registo e descoberta de caches, bem como um sistema de interação entre os utilizadores da aplicação. Grande parte dos requisitos foram cumpridos e implementados com sucesso, no entanto devido à conclusão do prazo estipulado para a realização do trabalho não foi possível concluir todos os objetivos.

Melhoramentos possíveis ao trabalho realizado:

* Concluir documentação da aplicação.
* Melhorar função de povoamento para simular as restantes funcionalidades da aplicação.
* Requisitar prova de assinatura de cache para validar assinatura da cache no sistema.
* Adicionar funcionalidade de estatísticas de maneira a ser possível ver quantos pontos um utilizador obteve num espaço de tempo, bem como o número de cache que assinou.
* Ranking de Utilizadores por pontuação.
* …