|  |
| --- |
| Universidade do Minho |
| GeoCachingPOO |
| Trabalho Prático de POO – Grupo 43 |
|  |
|  |
|  |

Rui Pereira

a67747

Nuno Oliveira

a67649

Rafael Antunes

a67711



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

Table of Contents

[2. Introdução 3](#_Toc421375996)

[3. Arquitetura de Classes 4](#_Toc421375997)

[I. View 4](#_Toc421375998)

[II. Business 4](#_Toc421375999)

[III. Data 5](#_Toc421376000)

[IV. Exceptions 5](#_Toc421376001)

[4. Estruturas de Dados 6](#_Toc421376002)

[1. Utilizadores 6](#_Toc421376003)

[2. Caches 6](#_Toc421376004)

[I. TravelBugs 7](#_Toc421376005)

[II. Cache Tradicional 7](#_Toc421376006)

[5. Screenshots da Aplicação 8](#_Toc421376007)

[6. Conclusões e Criticas 9](#_Toc421376008)

# 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 actividade realizadas. Esta aplicação deve ser desenvolvida na linguagem JAVA.

# Arquitetura de Classes

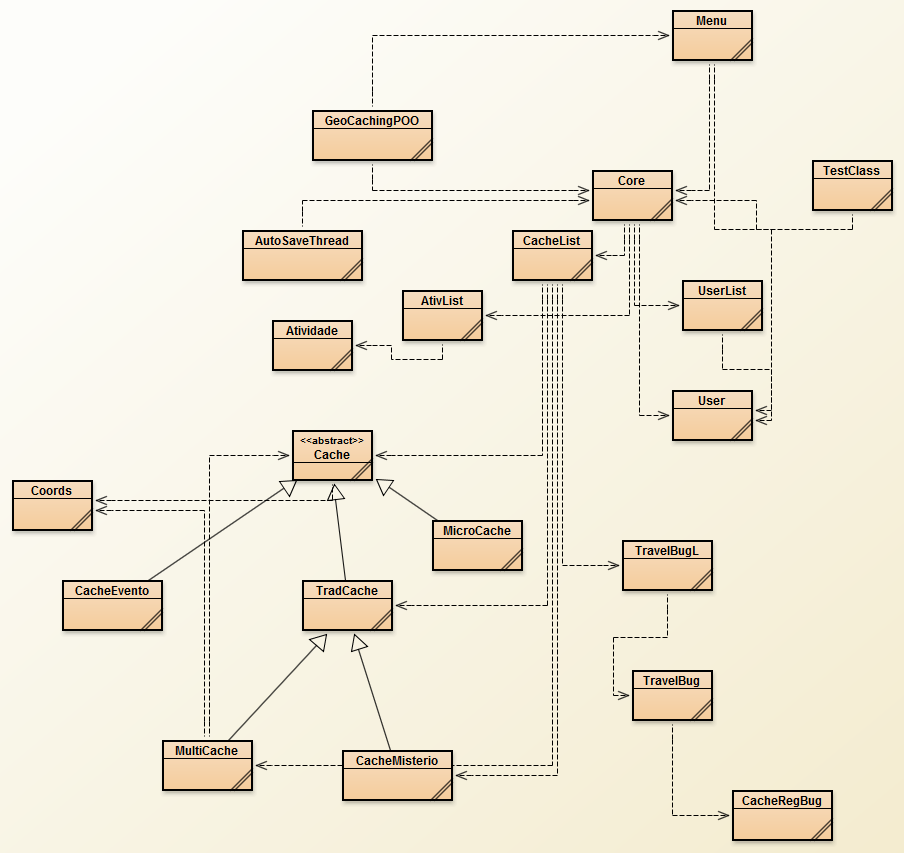


Figura 1 Imagem projeto BlueJ (sem packages)

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 –** ClasseMain 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 –** utilizadores da aplicação.
* **Caches –** 5 tipos diferentes de Caches
* **Actividade –** Lista de actividade 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:

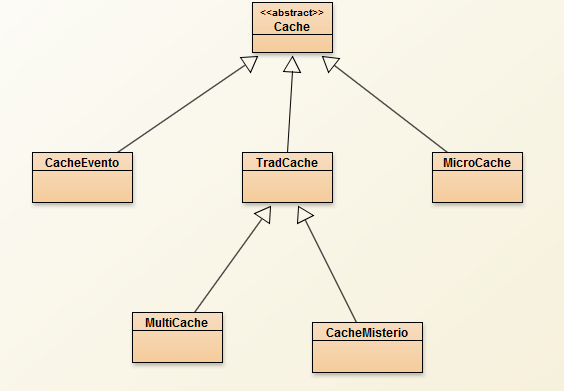


Figura 2 Esquema BlueJ relativo à herança da classe Cache

### 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*.

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 contrario 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
* E a descrição da atividade

# Screenshots da Aplicação

# Conclusões e Criticas