Universidade do Minho

Licenciatura em Engenharia Informática

Programação Orientada aos Objetos - 2014/2015

GeoPOO - Grupo 4

6 de Junho de 2015

Duarte Duarte
a61001/pg27715

Fábio Gomes
a61065/pg27752

Hélder Gonçalves
a61084/pg28505

Índice

Introdução

No âmbito da Unidade Curricular de Programação Orientada aos Objetos, da Licenciatura em Engenharia Informática, este relatório pretende apresentar o trabalho que nos foi proposto e todo o trabalho efetuado, bem como as decisões que foram tomadas e as suas dificuldades.

O trabalho prático consiste no desenvolvimento de uma aplicação denominada de **GeoPOO**, que realize a simulação das funcionalidades do site *geocaching.com* permitindo aos utilizadores registarem-se, encontrar e criar novas Caches, por exemplo.

Este *jogo* funciona muito resumidamente da seguinte forma, há utilizadores registados que submetem novas caches para o sistema, um Reviewer toma conta dela e faz a sua análise e indica ao owner o que ele deve fazer para melhorar. Quando estiver tudo bem será publicada para que novos jogadores a possam encontrar. Uma cache contém basicamente uma coordenada, informação, tipo e dicas sobre onde ela está escondida.

Para a realização desta aplicação foram considerados vários tipos de *Cache*, Utilizadores e Atividades. As especificações de cada um serão explicados posteriormente.

Serão explicadas as nossas decisões sobre a implementação das Classes, divisão dos *packages* e o porquê das suas escolhas.

Com a realização deste trabalho pretendemos consolidar os conhecimentos adquiridos ao longo das aulas da unidade curricular de POO, desenvolver a linguagem de programação, reter métodos eficazes e acima de tudo aprender/melhorar/consolidar algo que será importante para o nosso futuro na área da Engenharia: o saber trabalhar em equipa.

Package - Activity

Apenas presente a Classe Activity.

Activity

Uma Atividade é referente a um acontecimento especial, será visível ao utilizador e aos seus amigos, pode ter um dos seguintes tipos:

```
	public enum Type {
	NEW_CACHE, FOUND_CACHE, DIDNT_FIND_CACHE, ARCHIVED_CACHE, DISABLED_CACHE,
ENABLED_CACHE, UPDATED_LOG_TYPE, FRIENDS_WITH, NOT_FRIENDS_WITH, REV_NOTE, NOTE
}
```

Dependendo do acontecimento, é escolhido o Tipo respectivo. Por exemplo, quando um Utilizador encontra uma Cache, a Atividade será `FOUND_CACHE`, se uma nova Cache for Publicada então estamos na presença de uma `NEW_CACHE`. Além do Tipo temos também associada a data em que o acontecimento ocorreu, uma Cache, 2 Utilizadores e um Log. Consoante o tipo dela serão preenchidos os campos respectivos.

```
	private GregorianCalendar date;
	private Type type;
	private Cache cache;
	private UserAbstract user1, user2;
	private Log log;
```

Se pretendermos saber se a Atividade é sobre o Utilizador *user*, então podemos invocar o método **about**:

```
public boolean about(UserAbstract user) {
	if (this.user1.equals(user)) {
		return true;
	}
	if (this.user2 != null) {
		if (this.user2.equals(user)) {
		return true;
		}
	}
	return false;
}
```

Ou então o **aboutWithCache** para procurarmos também se a cache da Atividade pertence ao *user* ou é um *Reviewer*.

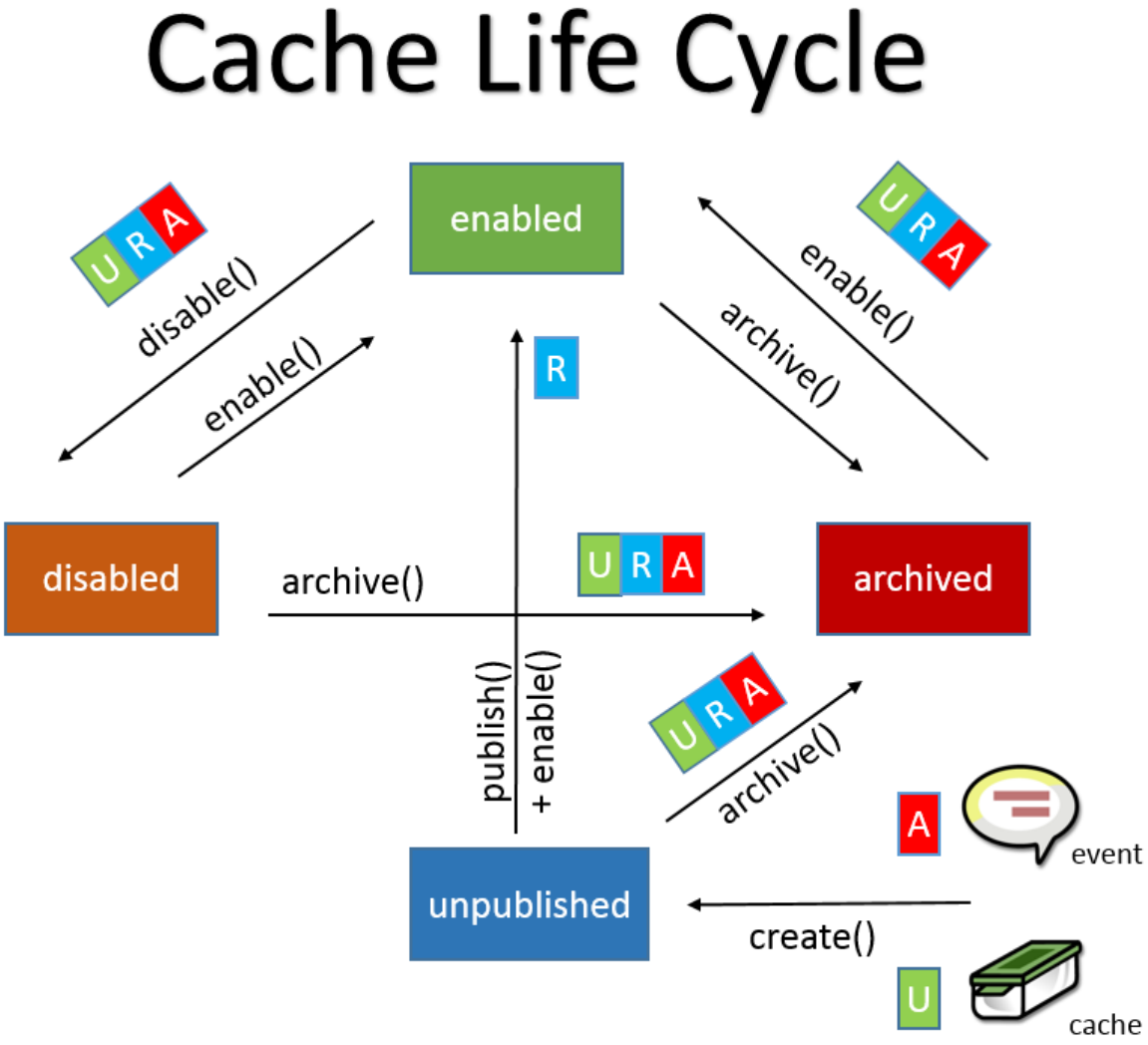
```
public boolean aboutWithCache(UserAbstract user) {

	if (this.user1.equals(user)) {
		return true;
	}
	if (this.user2 != null) {
		if (this.user2.equals(user)) {
		return true;
		}
	}
	if (this.cache != null) {
		if (cache.getOwner().equals(user) &&
			(this.user1.getRole() == UserAbstract.Role.REVIEWER)) {
		return true;
		}
	}
	return false;}
```

Baseado no Tipo o **toString()** vai também variar, como podemos ver neste excerto:

```
public String toString() {
	String res = "";
	switch (this.type) {
		case FOUND_CACHE:
		res = this.user1.getName() + " Found \""
				+ this.cache.getCacheTitle() + "\"";
		break;
		case DIDNT_FIND_CACHE:
		res = this.user1.getName() + " didn't Find \""
				+ this.cache.getCacheTitle() + "\"";
		break;
		case REV_NOTE:
		res = this.user1.getName() + " posted a Reviewer Note
					on \"" + this.cache.getCacheTitle() + "\"";
		break;
...}
```

Package - Caches

Aqui estão representados todos os tipos de Cache e o Log. Uma Cache tem um Ciclo de Vida como o seguinte diagrama demonstra:



Tudo começa com a criação da Cache, apenas os *User* podem criá-las ficando associados a elas como *Owner* (donos). Os Administradores podem apenas criar Eventos. As Caches e Eventos passarão para o estado *Unpublished*, para que um *Reviewer* possa analisar as informações delas e caso estejam em condições passarão para o estado *Enabled*, mas apenas o Reviewer associado a ela pode fazer *publish*, ficando assim ligado a ela. Quando está em *Enabled* a Cache/Evento já fica visível para todos os outros Utilizadores. A partir daí todas as alterações ao Estado referentes a um *User* são referentes apenas ao *Owner*. Só podemos Desativar Caches que estão *Enabled*, o fim de vida de uma Cache é o Estado *Archived*.

Cache

A classe Abstrata que dará origem a todas as outras, incluindo Eventos. Como referido no diagrama acima, a Cache tem 4 Estados:

```
public enum Status {

	UNPUBLISHED, ENABLED, DISABLED, ARCHIVED;
}
```

Definimos ainda 6 Tipos (mais o Default):

```
public enum Type {
	EARTH, EVENT, LETTERBOX, MYSTERY, MULTI, TRADITIONAL, DEFAULT;}
```

E o resto das variáveis de instância:

```
private GregorianCalendar publishDate; // Date cache was published
private GregorianCalendar creationDate; // Date cache was created
private String cacheID; // Cache ID number
private boolean premiumOnly; // Cache Premium
private String description; // Cache description
private Status cacheState; // Cache Status
private String cacheTitle; // Cache name
private UserAbstract owner; // Who placed the cache
private int cacheSize; // Type of container
private float difficulty; // How difficult is it to find the cache
private Position position;
private String hint; // Hints to find the cache
private TreeSet<Log> cache_Logs; // Cache logs
private Reviewer reviewer = null; // Reviewer responsible
private Data data = null; // System Data
```

Temos 2 datas, a de criação e a de publicação. Identificador único da Cache, por exemplo *GC5QR3G0*, a referência a ser visível apenas a membros Premium. Descrição, Título e Estado. A referência ao *Owner*, o tamanho dela, varia de 1 a 5 (Micro, Small, Regular, Large, Other). A dificuldade dela, de 1.0 a 5.0, a sua Posição/Coordenadas incluindo a dificuldade do terreno. Uma pista, lista de Logs, quem é o Reviewer associado e uma referência aos Dados para a inserção de Atividades.

A lista de Logs é um TreeSet, pois queremos que eles sejam ordenados pelos mais recentes.

Em termos de métodos associados, temos o inevitável logCache que pega num Log e num Utilizador e faz o respectivo Log.

```
public boolean logCache(UserAbstract user, Log log)
```

Se for um *FOUND_IT* será incrementado 1 ao total de Caches encontradas do *user*.

Mas têm que ser feitas verificações, por exemplo um Utilizador não pode logar um Found It numa cache Unpublished. Seguindo a seguinte Matriz sobre “Pode Ver / Logar ?”:

Cache State	Owner/Reviewer		Regular User	
	View	Write	View	Write
Archived	Yes	Notes	Yes	Notes
Enabled	Yes	Yes	Yes	Notes
Disabled	Yes	Notes	Yes	Notes
Unpublished	Yes	RNotes	No	No

Alguns métodos úteis foram adicionados, tais como o *hasFound(User user)* que informa se o *user* já tem um *log* do tipo *FOUND_IT* nesta cache.

O `public TreeSet<Log> getFriendsLogs(User user)`, que devolve os *logs* dos amigos do *user*. Tem ainda 5 métodos para Listing/toString para serem usados conforme a situação, se estamos a ver os detalhes totais da Cache, apenas os seus logs ou então só uma pequena informação para listagem.

Tipos de Cache

Traditional

Este é o tipo original de geocache e o mais simples. Estas caches serão um container numas coordenadas indicadas. Os tamanhos variam, mas todas elas devem conter um logbook para registo dos visitantes. Containers maiores podem poder albergar itens para troca.

Earth

Esta cache é bastante simples, deve servir para encontrar um sítio especial onde se possam aprender dados importante sobre o Planeta Terra. Deve incluir notas educativas além das coordenadas. Visitantes devem poder observar como o nosso Planeta é moldado por processos geológicos, como gerimos os seus recursos e como os cientistas recolhem dados. Tipicamente para logar uma *Earth Cache* devemos enviar os dados ao *Owner* para ele validar que de facto observamos a localização.

Para efeitos tecnológicos o envio de dados ao *Owner* não será contabilizado, portanto a validação do log ser feita nele mesmo. Sendo portanto igual a uma *Traditional*.

Multi

Este tipo de geocache involve 2 ou mais localizações, em que a localização final contém o container físico com um logbook dentro. Podem existir muitas variações mas tipicamente começamos no primeiro *Stage*, e temos que calcular coordenadas seguintes baseados em objetos no GZ.

Inclui uma lista de *Stages*, um *Stage* é uma combinação de um texto e o índice.

Letterbox

Letterboxing é outra forma de caça ao tesouro que inclui pistas em vez de coordenadas. É usual percorrer uma série de passos, como numa Multi, de forma a chegar ao final. Em cada Stage recebemos pistas sobre como ir para o próximo.

Tal como na Multi, esta inclui Stages, mas são não-opcionais.

Mystery

Normalmente inclui puzzles ou enigmas que devem ser resolvidos antes de visitar a zona da cache. Quando resolvidos, é obtida a coordenada final e uma ajuda.

Portanto adicionamos uma Position, *finalPos*, e uma descrição, *finalText*, que serão reveladas apenas se o Utilizador introduzir as coordenadas correctas no nosso verificador:

```
public boolean checkCoord(Position coord) {
      return (this.getFinalPos().getLati() == coord.getLati() &&
             this.getFinalPos().getLongi() == coord.getLongi());}
```

Event

Ficando um pouco alterada do que na realidade é, a Cache Evento é constituído por uma Data em que um conjunto de caches são escolhidas e no final da atividade conseguir obter mais pontos irá vencer o Evento.

O processo inicial é o mesmo que uma Cache normal, vai para a lista Unpublished até que um Reviewer lhe pegue. Depois é que vai para as enabledEvents. Depois de terminado o Evento passará para a pastEvents.

É constituído basicamente por duas datas, a do Evento em si e a data de término de inscrições, depois dessa data não será possível inscreverem-se. Ainda um limite máximo de participantes.

Chegado o dia do Evento é possível simulá-lo para obter as classificações por pontos dos Users participantes.

```
private HashMap<String, UserAbstract> participants;
private HashMap<String, Integer> points;
private HashMap<String, Cache> caches;
private GregorianCalendar dateEvent, dateEndAplications;
private int maxParticipants;
```

Log

Um Log é um registo de uma visita a uma Cache, temos os seguintes tipos: `FOUND_IT, DNF, NEEDS_MAINTENANCE, REVIEWER_NOTE, NEEDS_ARCHIVING, NOTE`

Juntando isso a uma data, o texto do Log e o Utilizador interveniente. Como temos isto num *TreeSet<Log>* então é preciso termos um *compareTo* devido ao *implements Comparable*.

```
public int compareTo(Object o) {
      return -((Log) o).getDate().compareTo(this.getDate());}
```

Package - User

Aqui estão criados os 3 tipos de Utilizador, *User*, *Reviewer* e *Admin*. Incluindo 2 *Interfaces* e o *UserAbstract* que serve de base para os 3 tipos.

UserInterface

Esta Interface indica os 4 métodos obrigatórios a serem implementados, *getRole* para nos indicar qual é o Tipo/Role do Utilizador presente e 3 tipos de *toString* para diferentes ocasiões.

```
public interface UserInterface {

	public Role getRole();

	public String toStringTotal();

	@Override
	public String toString();

	public String toStringOthers();
}
```

BasicCacheMethodsInterface

Esta Interface indica os 4 métodos obrigatórios também a serem implementados para a interação com as Caches, *disableCache* para poder desativar Caches, *archiveCache* para arquivar Caches, *enableCache* para ativar Caches e *logCache* para poder logar Caches, .

```
public interface BasicCacheMethodsInterface {

	public boolean disableCache(Cache c);

	public boolean archiveCache(Cache c);

	public boolean enableCache(Cache c);

	public boolean logCache(Log l, Cache c);
}
```

UserAbstract

Esta classe Abstrata será o ponto de partida para os Tipos de Utilizador, aqui estão definidos todos os métodos e variáveis de instância que todos eles vão possuir em comum. Começando pelo *enum Role*:

```
public enum Role {

	ADMIN, REVIEWER, USER, DEFAULT;

	@Override
	public String toString() {
		switch (this) {
		case ADMIN:
				return "Administrator";
		case REVIEWER:
				return "Reviewer";
		case USER:
				return "User";
		case DEFAULT:
				return "";
		default:
			throw new IllegalArgumentException();
		}}}
```

Default, é para indicar esta classe. As variáveis de instância são as seguintes:

```
	private final String email;
	private byte[] password;
	private String name;
	private String gender;
	private String address;
	private GregorianCalendar birthDate;
	private boolean premium;
	private Data data = null;
```

O *email* será o nosso identificador de Utilizadores no sistema, o ID portanto, há ainda a usual *password* para *login* no sistema, contém ainda Nome, Género, Morada, Data de Nascimento, uma indicação sobre o seu estatuto de Premium e uma referência aos Dados.

Este estatuto Premium faz com que apenas sejam visíveis Caches Premium se o User também fôr, estas Caches tendem a ser especiais e não devem ser publicadas para o Utilizador comum. Para passar a Premium é apenas necessário adquirir o estatudo e a sua revogação também é possível.

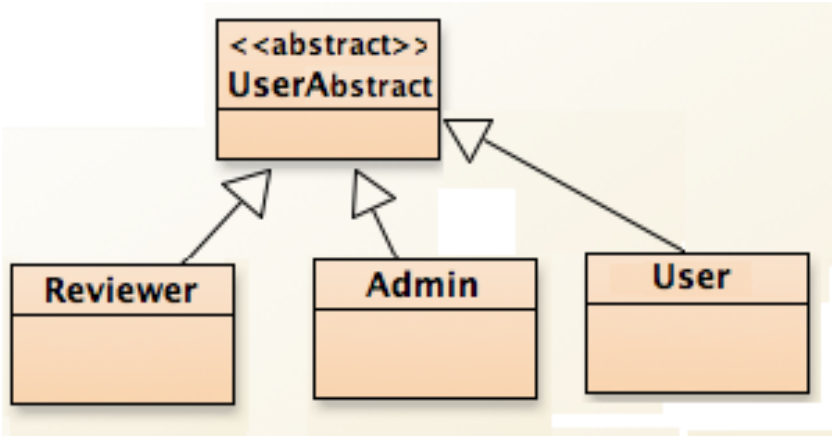
As Passwords são guardadas em SHA-256 para melhor segurança, por isso estarem em `byte[]`, seguindo o seguinte método:

```
private byte[] getHash(String password) { //SHA-256
	byte byteData[] = null;
	try {
		MessageDigest md = MessageDigest.getInstance("SHA-256");
		md.update(password.getBytes());

		byteData = md.digest();

	} catch (NoSuchAlgorithmException ex) {
		return null;
	}
	return byteData;
}
```

Portanto para efectuar o login comparamos os campos email e password com os guardados nos Dados.

```
public boolean login(String email, String password) {
	return this.getEmail().equals(email) &&
			Arrays.equals(this.getPassword(), getHash(password));
	}
```

Definido o UserAbstract, utilizamos a seguinte estrutura para incluir os restantes:



User

Este é o Utilizador comum com a inclusão das Caches criadas, dos Amigos e do Total de Caches encontradas. Estão em HashMap pois assim podemos aceder directamente a uma Cache específica, via CacheID, ou Utilizador/Amigo, via e-mail.

```
private HashMap<String, Cache> caches = null;
private HashMap<String, User> friends = null;
private int totalFound;
```

Como esta classe implementa o *BasicCacheMethodsInterface*, temos de definir os métodos por ela indicados. Tal como o *createCache*.

Com o seguinte método, o User poderá criar uma Cache. Apenas os User's podem fazê-lo. Veremos mais à frente que os Admin fazem Eventos, um tipo especial de Cache.

```
public boolean createCache(Cache cache) {
HashMap<String, Cache> map=super.getData().getAllCachesAndUnpublished();
if (map == null) {
      return false;
}

if (map.containsKey(cache.getCacheID()) == true) {
      return false;
}
cache.setCacheID(cache.genID(6));
cache.setOwner(this);
cache.setCacheState(Cache.Status.UNPUBLISHED);
super.getData().getUnpublishedCaches().put(cache.getCacheID(), cache);
this.caches.put(cache.getCacheID(), cache);

return true;
}
```

Já para os disableCache, archiveCache e enableCache tem que ser feita a verificação se o *User* é o *Owner*, caso contrário não poderá fazer a ação.

```
if (c.getOwner().equals(this) == false) {
      return false;}
```

As verificações estão de acordo com o Cache *Life Cycle*, apresentado anteriormente. Em cada ação é feita uma nova Atividade se tal for preciso, da seguinte forma. Caso de Arquivamento:

```
Activity act = new Activity(new GregorianCalendar(),
                            Activity.Type.ARCHIVED_CACHE, cache, this);
super.getData().addActivity(act);
```

Identifcamos o Tipo de Atividade, a cache respectiva e o Utilizador que fez a ação. Depois é adicionada às Actividades com o método *addActivity()*.

Reviewer

Classe referente ao Utilizador Reviewer, a função principal é tratar da qualidade das caches, ver se está tudo em condições, publicar novas caches, arquivar ou desativar caso necessário… Este tipo não faz logs de descoberta, como o *User*, apenas tem uma lista de Caches atribuídas:

```
TreeSet<Cache> assignedCaches;
```

Como fica responsável pela cache que lhe calhou, tem que ter o cuidado de orientar o *User* para que a Cache esteja nas condições desejadas. Só ele, o *Reviewer*, é que pode fazer Publish da Cache portanto tem responsabilidade de verificar a sua qualidade.

Para obter Cache deve invocar o método :

```
public Cache giveMeCache()
```

que lhe irá devolver uma Cache, incluindo Eventos, caso hajam na lista de Unpublished.

Admin

A tarefa do Administrador é bastante simples, pode ver todas as Caches, efetuar operações sobre elas e criar Reviewers e outros Administradores.

Quando pretende fazer um Log, apenas não pode fazer um FOUND_IT ou DNF porque não entra nas atividades de descoberta de caches. Não pode Publicar uma Cache pois tem que ser um Reviewer a ser o responsável por ela, depois dessa etapa poderá fazer as tarefas de controlo à vontade.

Como referido, é o único capaz de criar Caches Evento tendo à sua disposição o único menu criado para o efeito.

Visto que o Admin tem Eventos associados, foi adicionado um HashMap com as referências a eles. HashMap porque é directo o acesso ao Evento que pretendemos.

```
private HashMap<String, Event> events;
```

Adicionamos um método CreateEvent que vai tratar da Criação deles.

Package - dataCreation

Aqui são criados de forma completamente aleatória Caches, Countries (países) e ainda Users (utilizadores). Estas classes são utilizadas para povoar o programa sem que para isso tenha de ser utilizada a forma manual de fazê-lo.

CachesData

Esta classe permite criar de forma fácil e aleatória caches.

```
generateRandomCache();
```

Este método gera de forma aleatória uma cache recorrendo a uma posição também gerada de forma aleatória. É atribuída uma dificuldade, e são criadas todas as estruturas auxiliares necessárias a qualquer tipo de cache que seja criada.

CountriesData

Esta classe permite criar de forma fácil e aleatória países (countries).

```
getRandomPosition();
```

Este método retorna uma posição aleatória em qualquer parte do mundo recorrendo aos métodos que se seguem.

```
getCountryFromEurope(Boolean generateDiff);
getCountryFromNorthAmerica(Boolean generateDiff);
getCountryFromSouthAmerica(Boolean generateDiff);
getCountryFromAfrica(Boolean generateDiff);
getCountryFromAsia(Boolean generateDiff);
getCountryFromOceania(Boolean generateDiff);
```

Estes métodos retornam uma posição relativa ao continente a que pertencem. Utilizando as variáveis de instancia são de forma aleatória retornadas capitais da Europa se corresponder ao primeiro método acima declarado, da América do norte se for o segundo, América do sul, África, Ásia e por fim Oceania.

```
getCountryByName(String country, String continent);
```

Este método retorna uma posição relativa a um país que tiver o nome do primeira argumento e que esteja no continente que corresponde ao segundo argumento.

UsersData

Esta classe permite criar de forma fácil e aleatória utilizadores (users).

```
populateOceanie();
populateAsia();
populateAfrica();
populateOceanie();
populateSouthAmerica();
populateNorthAmerica();
populateEurope();
```

Estes métodos permitem popular os continentes com países destes e cada país é representado numa classe CountryInContinent que contém três ArrayList, um para os nomes próprios masculinos, outro para os nomes próprios femininos e por fim um com os apelidos. Todos os nomes e apelidos são reais e correspondentes à nacionalidade que os “adopta”.

```
getRandomUser(boolean male);
```

Este método devolve um utilizador completamente aleatório de um continente aleatório utilizando os métodos que se seguem. O argumento deste método permite apenas escolher o sexo do utilizador a ser retornado.

```
getUserFromEurope(boolean male);
getUserFromNorthAmerica(boolean male);
getUserFromSouthAmerica(boolean male);
getUserFromAfrica(boolean male);
getUserFromAsia(boolean male);
getUserFromOceanie(boolean male);
```

Cada um destes métodos devolve um utilizador completamente a um país que pertença ao continente ao qual o métodos corresponde de forma aleatória apenas dando a possibilidade de escolher o sexo que é passado no argumento.

```
newUser();
```

Este devolve um utilizador de forma completamente aleatório sendo que é impossível contribuir para escolher qualquer parâmetro.

CountryInContinente

Esta classe encontra-se “dentro” da classe usersData pois serve apenas para organizar os nomes de forma lógica dentro de cada país.

```
String country;
ArrayList<String> namesM;
ArrayList<String> namesF;
ArrayList<String> surnames;
```

As variaveis de instancia servem para guardar o nome do país, a lista de nomes do sexo masculino, lista de nomes do sexo feminino e por fim os apelidos.

```
getRandomName(String gender);
```

Este método retorna um nome do país a que corresponde em função do sexo passado por argumento.

Package - Base

Data

Esta é a nossa “Base de Dados”, contém HashMaps para as caches publicadas, desativadas, não publicadas e arquivadas, também para Eventos ativos e acabados além de todos os Utilizadores registados. Temos depois 2 TreeMaps pois é importante tê-los ordenados, para Atividades e Posições.

```
	private HashMap<String, Cache> enabledCaches = null;
	private HashMap<String, Cache> disabledCaches = null;
	private HashMap<String, Cache> unpublishedCaches = null;
	private HashMap<String, Cache> archivedCaches = null;

	private HashMap<String, Event> enabledEvents = null;
	private HashMap<String, Event> pastEvents = null;

	private HashMap<String, UserAbstract> allUsers = null;
	private TreeMap<GregorianCalendar, Activity> allActivities = null;

	private TreeMap<String, Position> allPositions = null;
```

Por vezes para a listagem nos Menus é importante termos em ArrayList para fazer *get(index)* por isso temos métodos que fazem essa “conversão”, tais como:

```
ArrayList<Activity> getActivitiesArray(UserAbstract user, int total) {
	int i = 0;
	ArrayList<Activity> array = new ArrayList<Activity>();

	for (Activity c : this.getAllActivities().values()) {
		if (c.aboutWithCache(user)) {
		array.add(c);
		i++;
		if (i >= total) {
			break;
		}
		}}
	return array;}
```

Geocaching

É a nossa *main*, esta parte é a que faz toda a ligação da API com o Utilizador. Os menus, apresentados na secção *Utilização da Aplicação*, estão todos definidos lá, de forma a que apenas se faça a invocação dos métodos correspondentes.

Persistencia dos Dados

Gravar o estado e carregar o estado do sistema resume-se a apenas guardar a classe instanciada *Data*. Foi concebida com esse intuito, fica tudo muito mais fácil, para quando pretendermos procurar informações e mais concretamente para gravar para ficheiro. Obviamente que para tal foi preciso que fosse adicionado o *implements Serializable* a todas as classe por ela envolvidas.

Utilities

```
getValueToDifficulty();
```

Este método devolve um float entre 0.5 e 5.0 com incrementos de 0.5 entre eles. Este método é utilizado pelas outras classes para gerar dificuldades para as caches e para as posições.

```
fromNameToEmail(String name, String emailDomain);
```

Este método converte um nome passado por argumento e um dominio também ele passado por argumento num email.

```
firstLetterCapital(String st);
```

Este método retorna a string passada por argumento com a primeira letra em maiúscula.

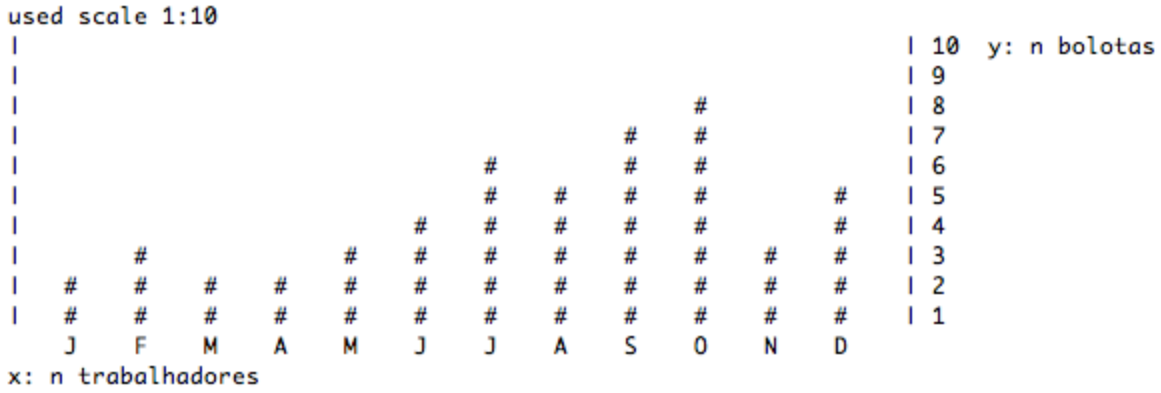
Statistics

```
graphicCacheCreationByYear(Data data);
graphicCacheCreationByMonth(Data data);
graphicCacheCreationByDay(Data data);
```

Os métodos acima geram gráficos relativos à criação de caches nos ultimos 50 anos, ano e mês.

```
graphic2D(ArrayList<Integer> points, ArrayList<String> listX, int columns, int
limit, String xcaption, String ycaption, boolean graph);
```

Este método gera um gráfico em função das coordenadas X,Y passadas por argumento, o numero de colunas a serem representadas, o limite maximo do eixo dos YY e as legendas para os eixos. O aspecto do gráfico gerado é apresentado na imagem que se segue.




```
getValuesForStatisticsInMonths(HashMap<String, Cache> caches);
```

Este método obtém as estatísticas compiladas por meses em função das caches passadas como argumento. E gera um histograma em que o eixo dos XX são os meses do ano.

```
getValuesForStatisticsInDays(HashMap<String, Cache> caches);
```

Este método obtém as estatísticas compiladas por dias em função das caches passadas como argumento. E gera um histograma em que o eixo dos XX são os dias do mês.

```
getValuesForStatisticsInYears(HashMap<String, Cache> caches);
```

Este método obtém as estatísticas compiladas por anos (ultimos 50) em função das caches passadas como argumento. E gera um histograma em que o eixo dos XX são os anos.

```
topTenCacheFinders(HashMap<String, UserAbstract> users);
```

Este método retorna os dez utilizadores que mais caches encontraram até ao momento.

```
topTenCacheCreators(HashMap<String, Cache> caches);
```

Este método retorna os dez utilizadores que mais caches criaram até ao momento.

```
monthStatistics(Data data, User user, GregorianCalendar dateFinnish, boolean withGraph);
```

Este método retorna as estatísticas relativas ao ultimo mês para um determinado utilizador e é possível escolher se retorna um gráfico ou não.

```
yearStatistics(Data data, User user, GregorianCalendar dateFinnish, boolean withGraph);
```

Este método retorna as estatísticas relativas ao ultimo ano para um determinado utilizador e é possível escolher se retorna um gráfico ou não.

ToTop

Esta classe permite guardar uma contagem para gerar um top.

```
int count;
String name;

compareTo(ToTop o)
```

Este método é o unico relevante nesta classe. Este método compara primeiro pelo valor da variável de instancia count e só depois por ordem alfabética da variavel name.

Package - Meteo

Meteo

Esta classe guarda uma meteorologia correspondente a uma altura do ano.

```
autoMeteo();
```

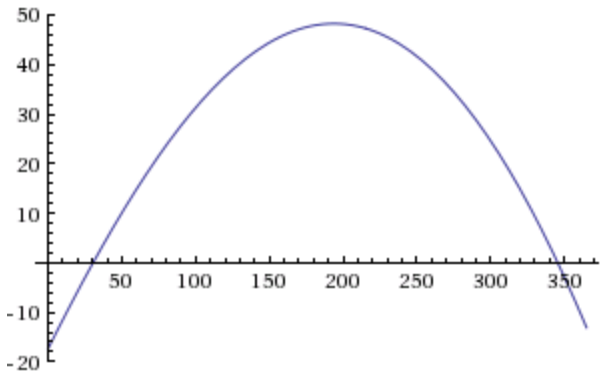
Este método gera uma meteorologia random sendo indiferente ao local.

```
autoMeteo(Position position, GregorianCalendar date);
```

Este método recorrendo à posição e à data que são passados por argumento estima a metereologia nesse local em tal dia.

```
staticMeteo(int day, Position position);
```

Recorrendo a uma função determinista este método utiliza a posição e o dia do ano para obter uma meteorologia. Usando a seguinte equeação por nós estimada:



-16.4793+0.590559*x-0.00140614*x^2-1.02857×10^-7*x^3-1.09144×10^-9*x^4

Obtida pela regressão quártica dos seguintes pontos:
{0,-10},{5,-5},{100,5},{210,80},{265,5},{360,-10},{365,-5},{415,-55},{-50,-55}

MeteoOnline

Esta classe tem uma particularidade ela usa meteorologias reais para uma determinada posição ou gera ela de forma estimada.

```
getOnlineWeather(Position position);
```

Este método devolve em função da posição passada por argumento uma meteorologia recorrendo à internet, ela faz isto com a ajuda de um JAR incorporado neste projecto.

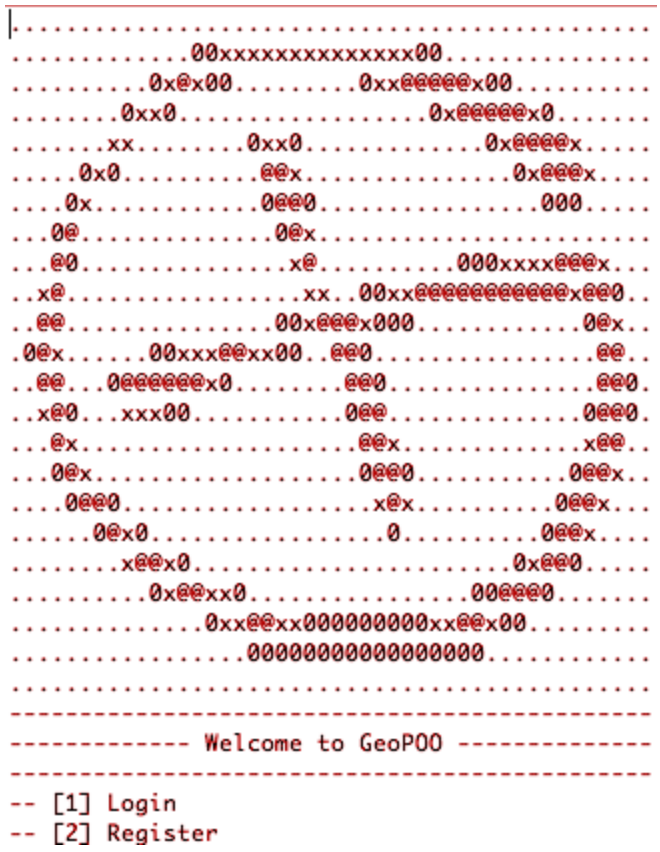
```
getOnlineWeather(Position position, GregorianCalendar date);
```

Este método devolve em função da posição passada por argumento uma meteorologia recorrendo à internet mas com a particularidade de se escolher a data em que se pretende essa meteorologia.

Utilização da Aplicação

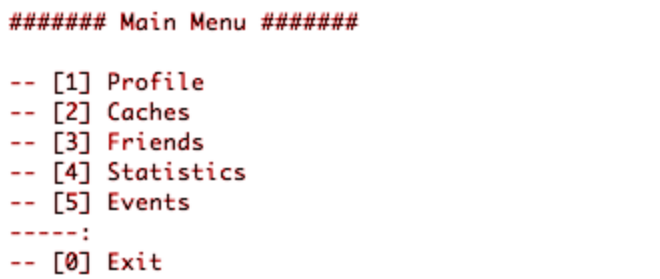
Menu inicial

Quando a aplicação inicia é possível escolher uma de duas opções, entrar para quem já tem um registo efectuado na aplicação (login) ou registar uma nova conta para quem não tem (register).



Login

Se o utilizador escolher a opção de login então será conduzido para o menu principal da aplicação.



Profile

Aqui é o local onde se pode visualizar o perfil do utilizador que se encontra com sessão iniciada.

```
####### My Profile #######

E-Mail - 1
Name - Ulisses
Gender - Male
Address - rua
Birth Date - 24/02/2000
Premium - true
Total Found - 2
```

Activities

Escolhendo esta opção permite ver todas as ultimas 10 actividades realizadas por este utilizador e pelos seus amigos e depois é possível saber mais sobre cada uma delas.

```
####### Ulisses and Friends' Activities Timeline #######

        [  1 ] Uche Villareal is now Friends with Ukra [06/06/2015 09:51:58]
        [  2 ] Ulisses is now Friends with Uche Villareal [06/06/2015 09:51:58]
        [  3 ] Ulisses posted a Note on "Dinossauros" [06/06/2015 09:51:58]
        [  4 ] Richard posted a Reviewer Note on "Em Braga" [06/06/2015 09:51:58]
        [  5 ] Rickon posted a Note on "New in Lisbon" [06/06/2015 09:51:58]
        [  6 ] Ulisses Found "Gualtar - A Primeira" [06/06/2015 09:51:58]
        [  7 ] Ulisses didn't Find "Em Braga" [06/06/2015 09:51:58]
        [  8 ] Uche Villareal Found "Em Braga" [06/06/2015 09:51:58]
        [  9 ] Ulisses Found "Em Braga" [06/06/2015 09:51:58]
        [ 10 ] Ulisses posted a Note on "New in Lisbon" [06/06/2015 09:51:58]
```

Edit Profile

Aqui é possivel editar as informações do perfil.

```
####### Edit Profile #######

E-Mail - 1
Name - Ulisses
Gender - Male
Address - rua
Birth Date - 24/02/2000
Premium - true
Total Found - 2

---- I want to edit :
-- [1] Name
-- [2] Gender
-- [3] Address
-- [4] Birth Date
-- [5] Password
```

Premium Membership

Escolhendo esta opção permite tornar o utilizador membro premium ou anular a subscrição premium.

Caches

Escolhendo esta opção entra-se no menu das caches onde é possivel procurar caches, criar caches, ver as encontradas e as criadas por este utilizador.

```
####### Caches Menu #######

-- [1] Search Caches
-- [2] Create a Cache
-- [3] View Found Caches
-- [4] View Owned Caches
```

Friends

Escolhendo este menu o utilizador consulta a sua lista de amigos e pode ver o perfil de cada um deles.

```
####### Friends #######

        [ 1 ] Uche Villareal (2)
```

Statistics

Escolhendo este menu entra na parte das estatísticas. Aqui é possível ver as estatísticas do utilizador ou as estatísticas globais, ou seja, do conjunto de todos os utilizadores.

```
####### Statistics #######

-- [1] Mine Statistics
-- [2] Global Statistics
```

Mine Statistics

Last month

Aqui é possível ver em forma de um histograma as estatísticas relativas as caches do ultimo mês e do ultimo ano.

```
Ulisses created 0 caches in last month
Ulisses found 1 caches in last month
        GC48567D 'Gualtar - A Primeira' (Traditional) D 4.0 / T 1.5 (02/07/2015 11:
used scale 1:10
|
|
|
|
|
|
|
|
|                         #
|   #    #    #    #    #    #    #    #    #    #    #    #    #    #    #    #
    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16
x: days
```

Last year

Global Statistics

Events

Escolhendo este menu pode se aceder a tudo sobre eventos desde criar eventos, ver calendário de eventos entre outros. (Vista de Administrador, o resto dos Utilizadores não verão os menus 4 e 5)

```
####### Events Menu #######

-- [1] Scheduled Events
-- [2] Today's Events
-- [3] Create an Event
-- [4] View Owned Events
-- [5] View Participated Events
```

Sign Up

Se o utilizador escolher a opção de registo é conduzido para uma parte da aplicação em que pode efectuar um registo utilizando o seu email e uma password à sua escolha.