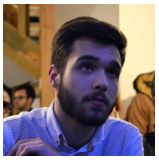
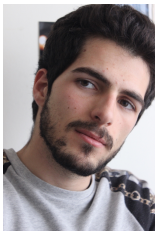
Relatório do Trabalho Prático de Programação Orientada a Objetos Grupo11 POO2017

Elísio Freitas Fernandes {55617}, Daniel Gonçalves Martins {73175}, and Nuno José Ribeiro da Silva {78879}

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
e-mail: {a55617, a73175, a78879}@uminho.pt



Daniel Gonçalves Martins {73175}



Elísio Freitas Fernandes {55617}

Nuno José Ribeiro da Silva {78879}

Resumo Serve o presente como relatório do projeto elaborado no âmbito da unidade curricular de Programação Orientada a Objetos. Projeto esse no qual se previa a elaboração de um programa para a empresa UMeR, o qual fosse capaz de garantir a prestação continua do seu serviço. Os requisitos de tal programa incluem a criação e manutenção da base de dados que inclui os dados dos clientes, bem como as informações dos seus colaboradores e das suas viaturas, utilizadas para a prestação do serviço em questão. Mais ainda, está incluída a criação de uma interface de interação com os utilizadores com várias opções, como: criação de conta; posterior acesso e atualização dos dados da conta; consulta de histórico de serviços requeridos/prestados.

1 Introdução

Com o presente documento pretende-se apresentar resumidamente o trabalho prático elaborado pelo grupo, como proposta de resolução do enunciado apresentado, no seguimento da UC de programação Orientada aos Objetos. As necessidades da empresa UMeR foram abordadas e tratadas usando java, temos recorrido maioritariamente ao Atom como editor de texto e o Terminal como compilador, bem como o ***** e a consola do linux para windows. Irão ser apresentadas as classes elaboradas e a várias escolhas especificas feitas em cada uma delas, a sua hierarquia e sua razão de ser.

2 Tipos de dados usados

No geral, as estruturas de dados usadas para o armazenamento dos dados de clientes, veículos e viagens foram os *Maps* - verificar, por exemplo, **Figura 1**. Dada a sua eficácia em termos de armazenamento ordenado de dados/rapidez na procura, à sua conveniência, uma vez que permite identificar cada objecto unicamente através de um dado único. Queríamos garantir a unicidade de dados, pois não haveria razão para termos duas entradas de registo para o mesmo utilizador, assim, usamos o email do mesmo como *key* de identificação no *Map* e assim garantimos que o mesmo *User* não se regista duas vezes. Poupamos memória e reduzimos dados duplicados nas bases de dados da empresa. O mesmo se dá para os *Vehicle*, *Trips* e para a associação de *Driver*/*Vehicle*.

```
public class UMeR implements Serializable
{
    private boolean isLogged;
    private String loggedUserEmail;
    private Map<String, User> userList;
    private Map<String, Vehicle> vehicleList;
    private Map<Integer, Trip> tripList;
    private Map<String, String> driverVehicle;
    private int tripNumber;
```

Figura 1. Variáveis da classe UMeR

Por outro lado, usamos também dados do tipo *List* em situações como na classe *User*, para guardar os ids das viagens efetuadas, uma vez que o tipo de dados é só um tipo de dados primitivo.

2.1 Metodologias de implementação dos tipos de dados

A nível da declaração das variáveis de instância foi usado sempre um tipo de dados com um grande nível de generalização. Com tal pretendemos facilitar a alteração dos tipos de dados específicos usados. Por exemplo, ao usar a declaração do tipo *Map* invés de *HashMap* permitimos que no futuro fosse fácil alterar esse tipo para um *TreeMap* por exemplo, mantendo no entanto a mesma intenção de permitir uma gestão eficiente e organizada da informação.

Pela mesma razão declaramos as variáveis de instância como *List*, permitindo depois a escolha do tipo específico de *List* pretendida - verificar, por exemplo, **Figura 2** .

por fotos dos dados usados e justificar o uso de maps e lists generalista - abstração

implementacao

falaar da abstracao de dados . utilizacao de treemaps quando se quer ordenar algo.

```
public abstract class User implements Comparable<User>, Serializable {

    private String name;
    private Address address;
    private LocalDate birthday;
    private String email;
    private String password;
    private List<Integer> tripHistory;
    private double totalTripCost;
```

Figura 2. Variáveis da classe User

3 Métodos

Em cada classe existem métodos que permitem a interação da empresa com a mesma, permitindo a criação, de instâncias de classe, atribuição de valores ás várias variáveis através dos *set<nome da variável>*, bem como obtenção dos valores contidos nas mesmas, através dos métodos *get<nome da variável>*

3.1 One more...

4 Arquitetura

Nesta proposta foram utilizados vários tipos de classes com diferentes atributos. Desde classes concretas a classes abstratas, classes com e sem implementações de interfaces e subclasses de classes abstratas.
Por questões de necessidade de gravação do estado da aplicação, todas as classes tem em comum a implementação da interface *Serializable*, oferecendo, assim, um método simples de gravação dos objetos com todos os seus estados atuais em ficheiro, para posterior consulta.

4.1 Classe UMeR

Utiliza Maps, e tipos primitivos para guardar os seus dados. Possui um registo de toda a informação necessária para o bom funcionamento da empresa. Contém quatro estruturas de dados Map onde se encontram guardados os dados de: 1. User como *value* e email como *key*, para guardar todos os utilizadores registados na empresa, quer clientes, quer condutores ; 2. Vehicle como *value* e licensePlate como *key*, para guardar a informação da viaturas ao serviço da empresa; 3. Trip como *value* e id de viagem como *key*, para manter um histórico de todas as viagens efetuadas através da empresa; 4. Email como *value* e licensePlate como *key*, para manter um registo do veículo associado a cada condutor. Existe ainda a variável isLogged, do tipo *boolean*, que indica se existe um utilizador "logado". Se sim, então o seu e-mail estará presente na variável loggedUserEmail, do tipo *String*. Por último, existe na classe uma variável tripNumber, do tipo *Integer*, responsável por garantir a sequencialização dos identificadores únicos da viagem, utilizados no Map das Trips.

Foram aglomerados todos os dados do tipo *User*, respetivamente, todos os dados de tipo *Trip* e *Vehicle*, num só *Map*, uma vez que eram compatíveis, de maneira a ser possível efetuar operações sobre todos os utilizadores de uma só vez, bem como facilitar a adição de novos tipo de utilizadores.

5 Hierarquia

Genericamente, como pode ser visto na figura ******, temos a ligação

6 Metodologia

programa inicia na main e chama as outras classes

7 Manual de utilização

fotos e indicações de navegação
According to Table 3...

(a) Delay and jiiter	(b) Delay and loss
(c) Delay and throughput	(d) Jitter and loss
(e) Jitter and throughput	(f) Loss and throughput

Figura 3. Tabela exemplo.

8 Conclusions

Neste trabalho...

Referências

1. Zadeh, L.: Fuzzy sets (1965)
2. Nguyen, H., Walker, E.: First course in fuzzy logic. Boca Raton: Chapman and Hall/CRC Press (1999)