Instituto Politcénico de Viana do Castelo

Escola Superior de Tecnologia e Gestão

Programação 2

“Gestão de uma stand de automóveis”

Trabalho realizado por:

Samuel Gomes nº23434

José Lima nº23442

Docentes: Professores António Cruz e Ricardo Castro

Índice

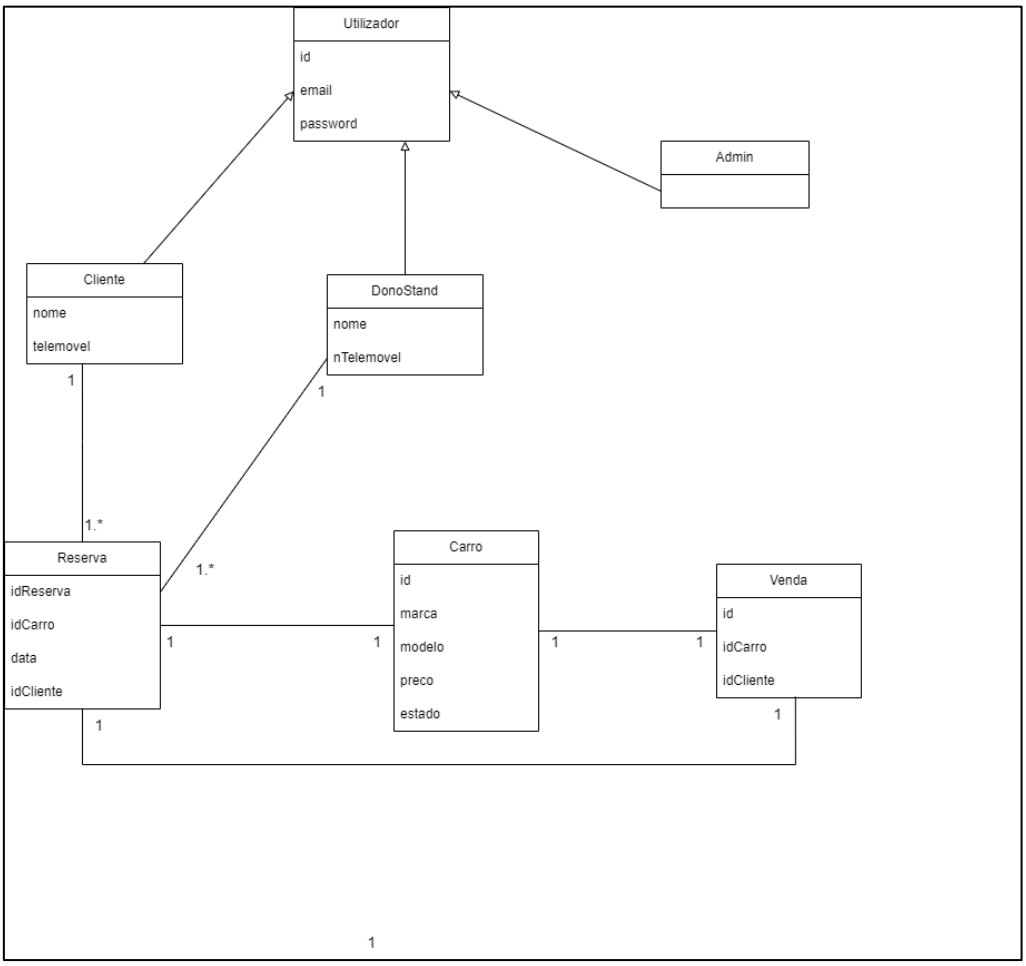
Introdução

Neste trabalho da unidade curricular de Programação 2, foi nos proposto a criação de uma aplicação para gestão de uma stand de automóveis. Esta aplicação terá 3 tipos de utilizadores: um administrador, clientes e um dono da stand. Cada um destes, após efetuar o seu registo e login, terá o seu próprio menu onde poderão realizar varias tarefas como por exemplo para o caso do admin, listar todos os utilizadores.

Objetivos

O principal objetivo deste trabalho será criar uma aplicação utilizando Java como linguagem de programação e Java Swing como ferramenta para criar interfaces gráficas tentando cumprir ao máximo com os requisitos propostos tornando a aplicação eficiente e friendly-user.

Diagrama de classes



Especificação das classes


```java
public class Utilizador implements Serializable {

    private String id;
    private String email;
    private String password;
```

```java
public class Cliente extends Utilizador {

    private String nome;
    private int nTelemovel;
```

```java
public class Admin extends Utilizador implements Serializable {

    public Admin(String email, String password) {
        super(email, password);
    }

}
```

```java
public class DonoStand extends Utilizador{
    private String nome;
    private int nTelemovel;
```

```java
public class Carro implements Serializable{
    private String id;
    private String marca;
    private String modelo;
    private int preco;
    private int estado; //0-desativado, 1-reservado, 2-disponivel
```

```java
public class Reserva implements Serializable{
    String idReserva;
    String idCarro;
    String idCliente;
    String data;
```

```java
public class Venda implements Serializable{
    private String id;
    private String idCarro;
    private String idCliente;
```

Métodos das classes

1. Classe: BLLLogin

```
    public int Registo(String email, String password, String nome, String telemovel){
        //verificar se é um numero telefonico
        int telemovelint = Integer.parseInt(telemovel.trim());
        if(telemovel.length()==9){
                //verificar se é email
                final Pattern EMAIL_REGEX = Pattern.compile("[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:.[a-z0-9!#$%&'

                if(EMAIL_REGEX.matcher(email).matches()){
                    //verificar se email ja existe
                    boolean existe = false;
                    ArrayList<Utilizador> users = dados.getAllUsers();
                    for(Utilizador aux : users){
                        if(aux.getEmail().equals(email)){
                                existe=true;
                        }
                    }

                    if(existe==false){
                        Cliente registo = new Cliente(email, password, nome, telemovelint);
                        dados.addUser(registo);
                        return 0;
                    }else{
                        return 1;
                    }

                }else{
                   return 2;
                }
            }else{
                return 3;
            }
    }
}
```

Esta função diz respeito ao registo de clientes e vai verificar 2 coisas, se o email introduzido pelo mesmo é valido e já existe, e também valida o número de telemóvel. Se o número de telemóvel for valido e o email ainda não estiver guardado este vai guardar os dados introduzidos e vai ser adicionado um novo cliente, caso contrário não vai ser permitido a criação de um novo cliente.

```
public boolean Login(String email, String password){

    ArrayList<Utilizador> users = dados.getAllUsers();
    for(Utilizador aux : users){
        if(aux.getEmail().equals(email)){
            if(aux.getPassword().equals(password)){
                if(aux instanceof Cliente){
                    new MenuCliente(aux).setVisible(true);
                    return true;
                }else if(aux instanceof DonoStand){
                    new MenuDono(aux).setVisible(true);
                    return true;
                }else if(aux instanceof Admin){
                    new MenuAdmin(aux).setVisible(true);
                    return true;
                }
                return false;
            }
        }
    }
    return false;
}
```

Nesta função acontece a validação dos dados de login. Apos introduzir o email e password é verificado se estes já pertencem algum utilizador e se for o caso avança para o painel correspondente ao tipo de utilizador. Se os dados não corresponderem a nenhum utilizador existente ou a password associada ao email colocado estiver errada é retornada uma mensagem de erro.

2. Classe: BLLMenuAdmin

```
public int addDono(String email, String password, String nome, String telemovel){
    //verificar se é um numero telefonico
    int telemovelint = Integer.parseInt(telemovel.trim());
    if(telemovel.length()==9){
            //verificar se é email
            final Pattern EMAIL_REGEX = Pattern.compile("[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:.[a-z0-9!#$%&'*+/=?^_`{|}~-

            if(EMAIL_REGEX.matcher(email).matches()){
                //verificar se email ja existe
                boolean existe = false;
                ArrayList<Utilizador> users = dados.getAllUsers();
                for(Utilizador aux : users){
                    if(aux.getEmail().equals(email)){
                            existe=true;
                    }
                }

                if(existe==false){
                    DonoStand dono = new DonoStand(email, password, nome, telemovelint);
                    dados.addUser(dono);
                    JOptionPane.showMessageDialog(null, "Dono do Stand adicionado com sucesso!");
                    return 0;
                }else{
                    return 1;
                }

            }else{
               return 2;
            }
        }else{
            return 3;
        }
}
```

```
public int addAdmin(String email, String password){
            //verificar se é email
            final Pattern EMAIL_REGEX = Pattern.compile("[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:.[a

            if(EMAIL_REGEX.matcher(email).matches()){
                //verificar se email ja existe
                boolean existe = false;
                ArrayList<Utilizador> users = dados.getAllUsers();
                for(Utilizador aux : users){
                    if(aux.getEmail().equals(email)){
                            existe=true;
                    }
                }

                if(existe==false){
                    Admin admin = new Admin(email, password);
                    dados.addUser(admin);
                    JOptionPane.showMessageDialog(null, "Admin adicionado com sucesso!");
                    return 0;
                }else{
                    return 1;
                }

            }else{
               return 2;
            }
}
```

Estas funções estão presentes no menu do admin e desempenham o mesmo papel que a função “Registo” acima, mas neste caso para um dono da Stand e para um novo admin, respetivamente.

3. Classe: BLLMenuDono

```
public int addCarro(String marca, String modelo, String preco){
    //verificar se é um numero telefonico
    int precoint = Integer.parseInt(preco.trim());
    if(precoint>0){
                    Carro car = new Carro(marca, modelo, precoint);
                    dados.addCarro(car);
                    JOptionPane.showMessageDialog(null, "Carro adicionado com sucesso!");
                    return 0;
        }else{
            return 1;
        }
}
```

```
public boolean alterarEstado(String id, int estado){

    ArrayList<Carro> carros = dados.getAllCarros();

    for (int i = 0; i < carros.size(); i++) {
        Carro aux = carros.get(i);
        if (aux.getId().equals(id)) {
            aux.setEstado(estado);
        }
    }

    try (FileOutputStream fos = new FileOutputStream("carros.dat");
        ObjectOutputStream oos = new ObjectOutputStream(fos)) {
        oos.writeObject(carros);
        return true;
    } catch (IOException e) {
        System.out.println("Erro ao guardar no ficheiro: " + e.getMessage());
        return false;
    }
}
```

```
public boolean cancelarReserva(String id_reserva, String id_Carro){
    dados.removeReserva(id_reserva);
    alterarEstado(id_Carro, 2);
    return true;
}

public boolean concluirVenda(String id_reserva, String id_cliente,String id_Carro){
    dados.removeReserva(id_reserva);
    alterarEstado(id_Carro, 0);
    Venda venda = new Venda(id_Carro, id_cliente);
    dados.addVenda(venda);
    return true;
}

public boolean concluirVenda(String id_cliente,String id_Carro){
    alterarEstado(id_Carro, 0);
    Venda venda = new Venda(id_Carro, id_cliente);
    dados.addVenda(venda);
    return true;
}
```

4. Classe: BLLCliente

```
public void efetuarReserva(String idCarro, String idCliente, String data){
    Reserva reserva = new Reserva(idCarro, idCliente, data);
    d.addReserva(reserva);
    dono.alterarEstado(idCarro, 1);

    JOptionPane.showMessageDialog(null, "Carro reservado com sucesso!");
}
```

Esta função permite ao Cliente selecionar um veículo listado e efetuar uma nova reserva. Posteriormente o estado do veículo escolhido vai ser alterado e deixa de pertencer à lista de veículos à venda.

5. Classe: dados

```
public ArrayList<Utilizador> getAllUsers(){

    ArrayList<Utilizador> users = new ArrayList<>();

    try (FileInputStream fis = new FileInputStream("utilizador.dat");
         ObjectInputStream ois = new ObjectInputStream(fis)) {
        users = (ArrayList<Utilizador>) ois.readObject();
        System.out.println("ArrayList read from file: " + users);
    } catch (IOException | ClassNotFoundException e) {
        System.out.println("Error reading from file: " + e.getMessage());
    }

    return users;
}
```

Nesta função vamos buscar a lista de utilizadores criados. Esta é do tipo ArrayList e está guardada no ficheiro “utilizador.dat”.

```
    public void addUser(Utilizador user){

        ArrayList<Utilizador> users = getAllUsers();

        users.add(user);

        try (FileOutputStream fos = new FileOutputStream("utilizador.dat");
            ObjectOutputStream oos = new ObjectOutputStream(fos)) {
            oos.writeObject(users);
            System.out.println("Users guardados com sucesso");
        } catch (IOException e) {
            System.out.println("Erro ao guardar no ficheiro: " + e.getMessage());
        }


    }


}
```

Esta função adiciona um novo utilizador à lista “users”. O novo utilizador é adicionado e é retornada a mensagem “Users guardados com sucesso”.

```
    public boolean removeUser(String id){
        System.out.println("cheguei aqui");
        ArrayList<Utilizador> users = getAllUsers();

        for (int i = 0; i < users.size(); i++) {
            Utilizador aux = users.get(i);
            if (aux.getId().equals(id)) {
                users.remove(aux);
            }
        }

        try (FileOutputStream fos = new FileOutputStream("utilizador.dat");
            ObjectOutputStream oos = new ObjectOutputStream(fos)) {
            oos.writeObject(users);
            return true;
        } catch (IOException e) {
            System.out.println("Erro ao guardar no ficheiro: " + e.getMessage());
            return false;
        }
    }
```

Esta função permite ao admin remover utilizadores.

```java
public ArrayList<Carro> getAllCarros(){

    ArrayList<Carro> carros = new ArrayList<>();

    try (FileInputStream fis = new FileInputStream("carros.dat");
         ObjectInputStream ois = new ObjectInputStream(fis)) {
        carros = (ArrayList<Carro>) ois.readObject();
    } catch (IOException | ClassNotFoundException e) {
        System.out.println("Error reading from file: " + e.getMessage());
    }

    return carros;
}
```

Esta função permite listar todos os carros, independentemente do seu estado atual.

```java
public void addCarro(Carro carro){

    ArrayList<Carro> carros = getAllCarros();

    carros.add(carro);

    try (FileOutputStream fos = new FileOutputStream("carros.dat");
        ObjectOutputStream oos = new ObjectOutputStream(fos)) {
        oos.writeObject(carros);
    } catch (IOException e) {
        System.out.println("Erro ao guardar no ficheiro: " + e.getMessage());
    }
}
```

Esta função permite adicionar um novo veiculo à lista “carros”.

```java
public boolean removeCarro(String id){
    ArrayList<Carro> carros = getAllCarros();

    for (int i = 0; i < carros.size(); i++) {
        Carro aux = carros.get(i);
        if (aux.getId().equals(id)) {
            carros.remove(aux);
        }
    }

    try (FileOutputStream fos = new FileOutputStream("carros.dat");
        ObjectOutputStream oos = new ObjectOutputStream(fos)) {
        oos.writeObject(carros);
        return true;
    } catch (IOException e) {
        System.out.println("Erro ao guardar no ficheiro: " + e.getMessage());
        return false;
    }
}
```

Nesta função conseguimos remover um veículo existente na lista “carros”. Após introduzir o id que queremos remover vamos percorrera lista ate ser encontrado e caso seja remove.

```java
public ArrayList<Reserva> getAllReservas(){

    ArrayList<Reserva> reservas = new ArrayList<>();

    try (FileInputStream fis = new FileInputStream("reservas.dat");
         ObjectInputStream ois = new ObjectInputStream(fis)) {
        reservas = (ArrayList<Reserva>) ois.readObject();
    } catch (IOException | ClassNotFoundException e) {
        System.out.println("Error reading from file: " + e.getMessage());
    }

    return reservas;
}
```

Esta função permite ao dono da stand listar todas as reservas criadas por clientes. Estas reservas estão guardas num ArrayList “reservas”.

```
    public ArrayList<Venda> getAllVendas(){

        ArrayList<Venda> vendas = new ArrayList<>();

        try (FileInputStream fis = new FileInputStream("vendas.dat");
             ObjectInputStream ois = new ObjectInputStream(fis)) {
            vendas = (ArrayList<Venda>) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error reading from file: " + e.getMessage());
        }

        return vendas;
    }
}
```

Esta função permite ao dono da stand listar todas as vendas efetuadas. As vendas estão guardas num ArrayList “vendas”.

```
    public void addVenda(Venda venda){

        ArrayList<Venda> vendas = getAllVendas();

        vendas.add(venda);

        try (FileOutputStream fos = new FileOutputStream("vendas.dat");
            ObjectOutputStream oos = new ObjectOutputStream(fos)) {
            oos.writeObject(vendas);
        } catch (IOException e) {
            System.out.println("Erro ao guardar no ficheiro: " + e.getMessage());
        }
    }
```

Dificuldades no trabalho

Neste trabalho as principais dificuldades que enfrentamos encontraram se na adaptação dos métodos criados às interfaces. Trabalhamos com java swing que era uma ferramenta com a qual já tínhamos trabalhado, mas pela dificuldade de interpretação do enunciado e uma arquitetura de solução pouco clara, eficiente e rápida levamos mais tempo do que era esperado para conseguir por a aplicação a funcionar.

Conclusão

Com este trabalho penso que conseguimos alcançar os objetivos definidos e concluir praticamente todos requisitos mínimos pedidos. Conseguimos ainda adicionar alguns requisitos extra com a implementação de interfaces gráficas e uso do git para desenvolver o trabalho.

Podemos assim concluir que este trabalho refletiu aquilo que aprendemos e aplicamos durante o semestre e permitiu nos ganhar muitas bases para os próximos desafios que se seguem.