

PROGRAMAÇÃO ORIENTADA A OBJETOS

Sistema de Gestão de Hotel

Professor: Artur Marques

Aluno:

Diogo Madeira - 220000904

Diogo Trigueiros - 220000941

Ano Letivo: 2025-2026

Índice

Índice de Figuras.....	3
Resumo	4
Introdução	4
Materiais e Métodos	4
Desenvolvimento	4
Estruturação do Projeto.....	4
Funcionalidades do Administrador.....	6
Funcionalidades do Cliente.....	7
Gestão de Reservas	8
Persistência de Dados	9
Códigos Importantes	10
Resultados	14
Conclusão	14

Índice de Figuras

Figura 1 – Estrutura das pastas do projeto	5
Figura 2 – Menu do Administrador	6
Figura 3 – Criação de reserva (Administrador)	6
Figura 4 – Menu do Cliente	7
Figura 5 – Criação de reserva (Cliente)	7
Figura 6 – Listagem de reservas pelo Cliente	8
Figura 7 – Ordenação de reservas	8
Figura 8 – Ficheiro JSON	9
Figura 9 – Classe abstrata do Utilizador	10
Figura 10 – Enum tipo de Utilizador	10
Figura 11 – Comparador para ordenar as reservas	11
Figura 12 – Instanciação do Gson na Base de Dados	11
Figura 13 – Filtro de ver as reservas	12
Figura 14 – Filtro de ver reservas por data	12
Figura 15 – Ordenar reservas por nome	12
Figura 16 – Ordenar reservas por nº de Hóspede	12

Resumo

O presente trabalho tem como objetivo o desenvolvimento de um sistema de gestão de hotel, implementado em Java, no âmbito da unidade curricular de Programação Orientada a Objetos. A aplicação permite a gestão de utilizadores e reservas, distinguindo claramente as funcionalidades disponíveis para administradores e clientes.

Introdução

Este projeto consiste no desenvolvimento de uma aplicação de gestão de hotel, cujo principal objetivo é permitir a administração eficiente de clientes e reservas.

Materiais e Métodos

A aplicação foi desenvolvida em Java, utilizando a biblioteca Gson para leitura e escrita de ficheiros JSON. Foram seguidas boas práticas de Programação Orientada a Objetos.

Desenvolvimento

De seguida descreve-se a implementação do sistema, bem como a sua estrutura interna e funcionalidades.

Estruturação do Projeto

Organização modular do código com separação clara de responsabilidades.

```
P00/  
|  
├─ src/  
|   └─ poo/  
|       ├── Main.java  
|       ├── Hotel.java  
|       ├── BaseDeDados.java  
|       ├── Utilizador.java  
|       ├── Cliente.java  
|       ├── Admin.java  
|       ├── Reserva.java  
|       ├── ReservaComparators.java  
|       ├── TipoUtilizador.java  
|       └── Validador.java  
|  
|   └─ dependencies/  
|       ├── gson-2.13.1.jar  
|       ├── build.ps1  
|       └── run.ps1  
|  
├─ clientes.json  
├─ reservas.json  
└─ README.md
```

Figura 1 – Estrutura das pastas do projeto

Funcionalidades do Administrador

O administrador possui permissões alargadas.

```
==== Admin ====
1 - Criar reserva
2 - Ver todas as reservas
3 - Editar reserva
4 - Excluir reserva
5 - Filtrar por n° de quarto
6 - Filtrar por data
7 - Ordenar por data
8 - Ordenar por n° de quarto (numérico)
9 - Ordenar por nome
10 - Ordenar por n° hóspedes
11 - Ordenar por estado de pagamento
12 - Terminar sessão
13 - Listar clientes
14 - Remover reservas duplicadas
Opção:
```

Figura 2 – Menu do Administrador

```
Opção:
1

Nome do hóspede: testar
Número do quarto: 10
Contacto: 919191919
Data (DD/MM/AAAA): 27/12/2025
Reserva criada!
```

Figura 3 – Criação de reserva (Administrador)

Funcionalidades do Cliente

O cliente pode criar e consultar as suas próprias reservas.

```
===== Login =====
1 - Admin
2 - Cliente
3 - Sair
Opção:
2

És cliente registado? (s/n): n
O teu nome:
Miguel
Queres registar-te com o NIF? (s/n): s
NIF (somente números): 241991367
Registo efetuado. Agora podes autenticar com o teu NIF no próximo login.

===== Cliente (Miguel) =====
1 - Criar a minha reserva
2 - Ver as minhas reservas
3 - Pagar reserva
4 - Terminar sessão
```

Figura 4 – Menu do Cliente

```
Número do quarto: 8
Contacto: 914246678
Data (DD/MM/AAAA): 31/12/2025
Reserva criada!
```

Figura 5 – Criação de reserva (Cliente)

Gestão de Reservas

As reservas podem ser listadas e ordenadas.

```
2
===== AS MINHAS RESERVAS =====
Nome: Miguel | Quarto: 8 | Contacto: 914246678 | Data: 31/12/2025 | Hóspedes: 1 | Pago: não
===== Cliente (Miguel) =====
```

Figura 6 – Listagem de reservas pelo Cliente


```
===== RESERVAS POR DATA =====  
Nome: Diogo | Quarto: 1 | Contacto: 914246617 | Data: 01/01/2000 | Hóspedes: 1 | Pago: sim  
Nome: ppp | Quarto: 90 | Contacto: 987654321 | Data: 12/12/2012 | Hóspedes: 1 | Pago: não  
Nome: tu | Quarto: 56 | Contacto: 123456789 | Data: 24/02/2025 | Hóspedes: 1 | Pago: não  
Nome: vvv | Quarto: 56 | Contacto: 123456789 | Data: 26/02/2025 | Hóspedes: 1 | Pago: não  
Nome: tu | Quarto: 24 | Contacto: 123456789 | Data: 19/11/2025 | Hóspedes: 1 | Pago: sim  
Nome: tu | Quarto: 26 | Contacto: 123456789 | Data: 12/12/2025 | Hóspedes: 1 | Pago: sim  
Nome: bot | Quarto: 22 | Contacto: 123456789 | Data: 18/12/2025 | Hóspedes: 0 | Pago: não  
Nome: testar | Quarto: 10 | Contacto: 919191919 | Data: 27/12/2025 | Hóspedes: 1 | Pago: não  
Nome: Miguel | Quarto: 8 | Contacto: 914246678 | Data: 31/12/2025 | Hóspedes: 1 | Pago: não  
Nome: laranja | Quarto: 1 | Contacto: 914246616 | Data: 07/08/2029 | Hóspedes: 0 | Pago: não
```

Figura 7 – Ordenação de reservas

Persistência de Dados

Os dados são armazenados em ficheiros JSON como podemos ver abaixo.

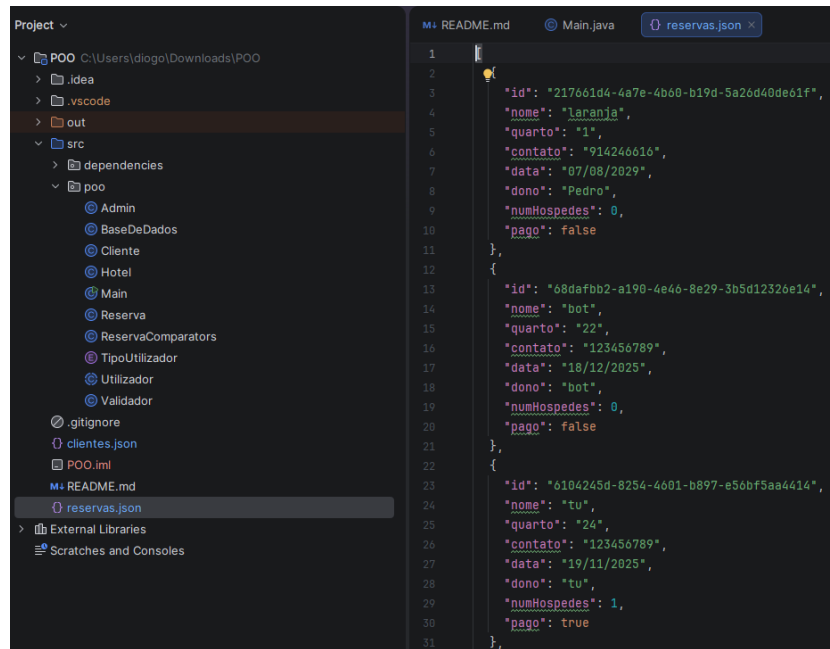


Figura 8 – Ficheiro JSON

Códigos Importantes

```

package poo;

public abstract class Utilizador { 2 usages 2 inheritors  Madeira23 +1

    protected String nome; 11 usages
    protected TipoUtilizador tipo; 2 usages

    public Utilizador(String nome, TipoUtilizador tipo) { 4 usages  Madeira23
        this.nome = nome;
        this.tipo = tipo;
    }

    public String getNome() { return nome; } 6 usages  Madeira23
    public TipoUtilizador getTipo() { return tipo; } no usages  Madeira23

    public abstract void mostrarMenu(Hotel hotel); no usages 2 implementations  Diogo Trigueiros
}

```

Figura 9- Classe abstrata do Utilizador

```
public enum TipoUtilizador { 7 usages  🧑 Madeira23 +1
    ADMIN, 1 usage
    CLIENTE 3 usages
}
```

Figura 10- Enum tipo de Utilizador

```
import java.util.Comparator;

public class ReservaComparators { 3 usages  🧑 Diogo Trigueiros

    public static Comparator<Reserva> byRoomNumberNumeric() { 1 usage  🧑 Diogo Trigueiros
        return ( Reserva a, Reserva b) -> {
            try {
                int na = Integer.parseInt(a.getQuarto());
                int nb = Integer.parseInt(b.getQuarto());
                return Integer.compare(na, nb);
            } catch (Exception e) {
                return a.getQuarto().compareTo(b.getQuarto());
            }
        };
    }

    public static Comparator<Reserva> byNumGuests() { return Comparator.comparingInt(Reserva::getNumHospedes); }

    public static Comparator<Reserva> byPaidStatus() { return Comparator.comparing(Reserva::isPago).reversed(); }
```

Figura 11- Comparator para ordenar as reservas

```

public class BaseDeDados { 4 usages  Diogo Trigueiros
    private static final String FICHEIRO_RESERVAS = "reservas.json"; 2 usages
    private static final Gson gson = new GsonBuilder().setPrettyPrinting().create(); 4 usages

    public static void guardarReservas(ArrayList<Reserva> reservas) {...}

    public static ArrayList<Reserva> carregarReservas() { 1 usage  Diogo Trigueiros
        try (Reader r = new FileReader(FICHEIRO_RESERVAS)) {
            ArrayList<Reserva> lista = gson.fromJson(r, new TypeToken<ArrayList<Reserva>>(){}.getType());  Diogo Trigueiros
            return (lista != null) ? lista : new ArrayList<>();
        } catch (Exception e) {
            return new ArrayList<>();
        }
    }

    // ===== Clientes persistence =====
    private static final String FICHEIRO_CLIENTES = "clientes.json"; 2 usages

    public static void guardarClientes(ArrayList<Cliente> clientes) { 1 usage  Diogo Trigueiros
        try (Writer w = new FileWriter(FICHEIRO_CLIENTES)) {
            gson.toJson(clientes, w);
        } catch (IOException e) {
            System.out.println("Erro ao guardar clientes: " + e.getMessage());
        }
    }

    public static ArrayList<Cliente> carregarClientes() { 1 usage  Diogo Trigueiros
        try (Reader r = new FileReader(FICHEIRO_CLIENTES)) {
            ArrayList<Cliente> lista = gson.fromJson(r, new TypeToken<ArrayList<Cliente>>(){}.getType());  Diogo Trigueiros
            return (lista != null) ? lista : new ArrayList<>();
        } catch (Exception e) {
            return new ArrayList<>();
        }
    }
}

```

Figura 12 – Instanciação do Gon na Base de Dados

```

public void verPorNumeroDeQuarto() { 2 usages  Diogo Trigueiros
    System.out.print("Número do quarto: ");
    String quarto = scanner.nextLine();

    String finalQuarto = quarto;

    var lista = reservas.stream()
        .filter(Reserva r -> r.getQuarto() != null &&
            r.getQuarto().equals(finalQuarto))
        .collect(Collectors.toList());

    if (lista.isEmpty()) {
        System.out.println("Sem reservas para este quarto.\n");
        return;
    }

    imprimirLista(titulo: "RESERVAS DO QUARTO " + quarto, lista);
}

```

Figura 13 – Filtro de ver as reservas

```

public void verPorDia() { 2 usages  🧑 Diogo Trigueiros
    System.out.print("Data (DD/MM/AAAA): ");
    String data = scanner.nextLine();
    data = Validador.validarEFormatarData(data);

    if (data == null) {
        System.out.println("Data inválida.\n");
        return;
    }

    String finalData = data;

    var lista = reservas.stream()
        .filter( Reserva r -> r.getData() != null &&
            r.getData().equals(finalData))
        .collect(Collectors.toList());

    if (lista.isEmpty()) {
        System.out.println("Sem reservas para esta data.\n");
        return;
    }

    imprimirLista( titulo: "RESERVAS EM " + data, lista);
}

```

Figura 14 - Filtro de ver reservas por data

```

public void verOrdenadoPorNome() { 1 usage  🧑 Diogo Trigueiros
    List<Reserva> ord = new ArrayList<>(reservas);
    ord.sort(Comparator.comparing(Reserva::getNome, String.CASE_INSENSITIVE_ORDER));
    imprimirLista( titulo: "RESERVAS POR NOME", ord);
}

```

Figura 15 – Ordenar reservas por nome

```
public void verOrdenadoPorNumHospedes() { 1 usage Diogo Trigueiros
    List<Reserva> ord = new ArrayList<>(reservas);
    ord.sort(ReservaComparators.byNumGuests());
    imprimirLista( titulo: "RESERVAS POR N° HÓSPEDES", ord);
}
```

Figura 16 – Ordenar reservas por nº de Hóspede

Resultados

Os resultados demonstram o correto funcionamento do sistema nas perspectivas de administrador e cliente, sendo possível reservar , cancelar e pagar cada reserva.

Conclusão

O projeto permitiu consolidar os conhecimentos adquiridos em Programação Orientada a Objetos, apresentando uma solução organizada e extensível para um software na vida real.