Escola Profissional
BENTO DE JESUS CARAÇA
DELEGAÇÃO DE LISBOA

RELATÓRIO

Programação de Sistemas Informáticos

CURSO TÉCNICO DE PROGRAMAÇÃO E GESTÃO DE SISTEMAS INFORMÁTICOS

Nome do Aluno: Andreia Barros e Temóteo Dinis

Nº Aluno: L.2493 e L.2494

25/10/2025

O relatório encontra-se em condições para ser apresentado

_____

Ciclo de Formação 2023/2026

Ano Letivo 2025/2026

Escola Profissional
BENTO DE JESUS CARAÇA
DELEGAÇÃO DE LISBOA

ÍNDICE

AGRADECIMENTOS

Agradecemos ao professor e aos colegas pelo apoio durante o desenvolvimento deste projeto. Este trabalho ajudou-nos a compreender melhor a Programação Orientada a Objetos em Python.

INTRODUÇÃO

Este trabalho tem como objetivo desenvolver um sistema para um hotel utilizando a linguagem Python. O projeto procura representar as principais entidades e interações do ambiente hoteleiro, como funcionários e hóspedes, aplicando os conceitos de herança, polimorfismo e classes abstratas. A proposta visa demonstrar a importância da programação orientada a objetos na organização e consistência dos dados de um sistema de gestão hoteleira.

Escola Profissional
BENTO DE JESUS CARAÇA
DELEGAÇÃO DE LISBOA

TEMA DO TRABALHO

O trabalho consiste em desenvolver um sistema para um hotel em Python. O objetivo principal foi modelar as entidades e interações cruciais de um hotel, definindo claramente as funções de diferentes cargos (Funcionários) e as operações essenciais (Hóspedes). A estrutura baseia-se em uso de heranças simples, herança múltipla, polimorfismo e classes abstratas para garantir a consistência, a organização e a integridade dos dados operacionais.

CÓDIGO E EXPLICAÇÕES

Importação das classes (Estrutura do hotel)

```python
# main.py

from hospede import Hospede
from gerente import Gerente
from recepcionista import Recepcionista
from tecnicomanutencao import TecnicoManutencao
from tecnicorecepcao import TecnicoRecepcao
from quarto import QuartoSimples, QuartoLuxo
```

Esta parte do código faz a importação das classes de outros ficheiros (módulos). Colocamos moldes para: Hóspedes e vários tipos de Funcionários (gerente, recepcionista, técnico de manutenção e técnico de recepção). Colocamos moldes para: Quartos (Simples e Luxo), que têm características e preços diferentes, o main.py usa essas classes para criar os objetos do sistema.

Inicialização do sistema (Listas)

```python
lista_hospedes = []
lista_quartos = []
lista_funcionarios = []
```

Criação de listas vazias para guardar todos os hóspedes, quartos e funcionários. Depois, durante o programa, vou adicionando objetos a estas listas. O uso de listas é um exemplo de estrutura de dados.

Criação de quartos e funcionários

Quartos:

```
print("=== Quartos Disponíveis ===")
q1 = QuartoSimples(101)
q2 = QuartoLuxo(102)
q3 = QuartoSimples(103)
lista_quartos.extend([q1, q2, q3])

for q in lista_quartos:
    q.mostrar_informacoes()
```

Criação de três quartos, dois simples e um de luxo. O método .extend() serve para adicionar vários itens de uma vez só à lista, em vez de estar a usar vários appends. Depois, com o for, percorro a lista e chamo o método mostrar informações() de cada quarto. Este método foi sobrescrito (polimorfismo) para mostrar informações diferentes conforme o tipo do quarto.

Funcionários:

```
print("\n=== Funcionários ===")
g1 = Gerente("Gloria", 1, 40, 2000.0)
r1 = Recepcionista("Paula", 2, 28, 1000.0, "Noite")
t1 = TecnicoManutencao("Lucas", 3, 35, 1200.0, "Elétrica")
tr1 = TecnicoRecepcao("Ismael", 3, 32, 1400.0, "Tarde", "Hidráulica")

lista_funcionarios.extend([g1, r1, t1, tr1])

for f in lista_funcionarios:
    f.mostrar_informacoes()
```

Criação de quatro funcionários diferentes: gerente, recepcionista, técnico e técnico-recepção com nomes, IDs, idades, salários e especialidades.O técnico-recepção é especial porque tem herança múltipla, herda tanto de recepcionista quanto de técnico-manutenção. Todos têm o método mostrar informações(), mas cada um mostra dados diferentes, isto é polimorfismo. Depois uso novamente o for para percorrer a lista e mostrar tudo.

Registo de Hóspedes:

```
print("\n=== Registo de Hóspedes ===")
h1 = Hospede("Joshua", 25, 3)
h2 = Hospede("Laercio", 40, 2)
h3 = Hospede("Taynara", 22, 1)

r1.registrar_hospede(h1, lista_hospedes)
r1.registrar_hospede(h2, lista_hospedes)
tr1.registrar_hospede(h3, lista_hospedes)
```

Criação de três hóspedes e o método registrar hospede da Rececionista (r1) ou técnico Receção (tr1) é usado para adicioná-los à lista hospedes. Isso demonstra que são os funcionários que executam a ação de registar, e não o hóspede sozinho.

Atribuir Quartos:

```
print("\n=== Atribuir Quartos ===")
h1.atribuir_quarto(q1)
h2.atribuir_quarto(q2)
h3.atribuir_quarto(q3)
```

Cada hóspede recebe um quarto através do método atribuir quarto. Este método também marca o quarto como ocupado. É um exemplo de interação entre objetos — o hóspede "liga-se" ao quarto.

Lista de hóspedes:

```
print("\n=== Lista de Hóspedes ===")
r1.listar_hospedes(lista_hospedes)
```

O recepcionista mostra todos os hóspedes registrados, usando um for para percorrer a lista. Este método usa a estrutura de repetição (loop for) e mostra as informações com o mostrar informações() de cada hóspede.

Registo de reparos:

```
print("\n=== Registo de Reparos ===")
t1.registar_reparo("Troca de lâmpada no quarto 102")
tr1.registar_reparo("Porta quebrada no corredor principal")
```

Aqui os técnicos registam reparos.O Técnico Recepção herda o método registar reparo da classe Técnico Manutenção. É outro exemplo de herança múltipla e polimorfismo, porque as mensagens são diferentes para cada tipo de técnico.

Checkout e Contas:

```
print("\n=== Checkout e Contas ===")
for h in lista_hospedes[:]:
    total = h.calcular_conta()
    print(f"Conta total de {h.nome}: {total:.2f}€")
    h.checkout()
    lista_hospedes.remove(h)
```

Nesta parte o sistema calcula quanto cada hóspede deve pagar e depois faz o checkout. O método calcular conta multiplica o preço da diária pelos dias de estadia. Uso lista hóspede[:] para fazer uma cópia da lista, e assim posso remover hóspedes enquanto percorro a lista sem erro. É um bom exemplo de manipulação de listas e ciclos for.

Relatório do Gerente:

```
print("\n=== Relatório do Gerente ===")
g1.gerar_relatorio(lista_funcionarios)
```

O gerente gera um relatório com todos os funcionários do hotel. Este método percorre a lista e chama o mostrar informações() de cada funcionário novamente, polimorfismo.

Class Pessoa:

```python
from abc import ABC, abstractmethod

class Pessoa(ABC):
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    @property
    @abstractmethod
    def nome(self):
        pass

    @nome.setter
    @abstractmethod
    def nome(self):
        pass

    @property
    @abstractmethod
    def idade(self):
        pass

    @idade.setter
    @abstractmethod
    def idade(self):
        pass

    @abstractmethod
    def mostrar_informacoes(self):
        pass
```

A classe Pessoa (ABC) é um módulo obrigatório (Classe Abstrata). Ela não pode criar objetos, mas força todas as classes que dela herdam (como Funcionário) a implementar certos métodos (como mostrar informações). Isso garante que todas as entidades do sistema se comportem de maneira consistente.

CONCLUSÃO

O desenvolvimento deste projeto foi uma excelente oportunidade para colocarmos em prática os conhecimentos adquiridos sobre Programação Orientada a Objetos em Python.

Através da criação de várias classes e do uso de herança simples e múltipla, polimorfismo e classes abstratas, conseguimos compreender melhor como estruturar um sistema de forma organizada e funcional.

O uso de listas, ciclos e módulos demonstrou também a importância da modularidade e da reutilização de código.

Este trabalho contribuiu significativamente para o nosso crescimento na área da programação e mostrou a utilidade da Programação Orientada a Objetos na resolução de problemas do mundo real.