Escola Profissional
BENTO DE JESUS CARAÇA
DELEGAÇÃO DE LISBOA

RELATÓRIO

PROGRAMAÇÃO DE SISTEMAS DE INFORMAÇÃO
PROJETO FINAL MODULO 11

Programação orientada de objetos avançados

Professor: Breno Sousa

Nome dos Alunos: Martim Rocha, Tiago Moisão, Simão Mendes
Nº dos Alunos: L2491, L2479, L2477

24/10/2025
O relatório encontra-se em condições para ser apresentado

Ciclo de formatação 2023/2026
Ano Letivo 2025/2026

Introdução

Este projeto foi desenvolvido no contexto da disciplina de Programação e Sistemas Informáticos, com o intuito de aplicar os conhecimentos adquiridos em python e gerenciamento de dados com resolução de problemas técnicos. O tema é o seguinte: A minha equipa foi contratada para desenvolver um sistema para um hospital. A aplicação deve ser desenvolvida em Python. É solicitado o uso de heranças simples, herança múltiplas, polimorfismo e classes abstratas.

Código

1. Pessoa.py — Classe Base

A primeira imagem apresenta a classe Pessoa, que é a base de toda a estrutura do projeto. Nesta classe são definidos atributos genéricos como nome e idade, bem como os respetivos métodos de acesso e modificação (getters e setters, caso existam).

O principal objetivo da classe Pessoa é servir como superclasse para outras entidades que representam pessoas no sistema, como Paciente e Funcionario. Desta forma, evita-se a duplicação de código, garantindo que todos os tipos de pessoas partilham características comuns.

Importância no projeto: fornece a base para a herança e a consistência dos dados de todos os objetos que representam pessoas.

```
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, n):
        pass

    @property
    @abstractmethod
    def idade(self):
        pass

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

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

2. Funcionario.py — Herança e Especialização

A segunda imagem mostra a classe Funcionario, que herda de Pessoa. Esta classe adiciona novos atributos relevantes ao contexto hospitalar, como o cargo e o salário, permitindo representar qualquer trabalhador do hospital.

Além disso, podem existir métodos que tratam de operações comuns aos funcionários, como calcular salários, mostrar informações completas ou alterar o cargo.

Importância no projeto: a classe Funcionario funciona como uma camada intermédia entre Pessoa e as subclasses mais específicas (Medico, Enfermeiro, Administrativo). Permite reaproveitar comportamentos comuns a todos os profissionais de saúde e administrativos.

```
from Pessoa import Pessoa

class Funcionario(Pessoa):
    def __init__(self, nome, idade, cargo, salario):
        super().__init__(nome, idade)
        self.cargo= cargo
        self.salario= salario


    @property
    def nome(self):
        return self._nome

    @nome.setter
    def nome(self, n):
        if n == "":
            print("Nome não pode ser vazio!")
        else:
            self._nome = n

    @property
    def idade(self):
        return self._idade

    @idade.setter
    def idade(self, i):
        if i >= 0:
            self._idade = i
        else:
            print("Idade não pode ser negativa!")

    @property
    def salario(self):
        return self._salario

    @salario.setter
    def salario(self, s):
        if s >= 0:
            self._salario = s
```

3. Medico.py — Classe Específica de Profissional de Saúde

A terceira imagem corresponde à classe Medico, uma subclasse de Funcionario. Aqui, o programador introduz atributos e métodos específicos de um médico, como a especialidade, lista de pacientes ou funções clínicas.

Esta classe exemplifica bem o conceito de especialização — um médico é um funcionário, mas com responsabilidades e dados distintos. A herança garante que o médico também tem acesso a todos os métodos e propriedades da classe Funcionario e Pessoa.

Importância no projeto: representa um papel concreto dentro do hospital, sendo uma peça essencial na simulação da equipa médica.

```
from Pessoa import Pessoa

class Funcionario(Pessoa):
    def __init__(self, nome, idade, cargo, salario):
        super().__init__(nome, idade)
        self.cargo= cargo
        self.salario= salario


    @property
    def nome(self):
        return self._nome

    @nome.setter
    def nome(self, n):
        if n == "":
            print("Nome não pode ser vazio!")
        else:
            self._nome = n

    @property
    def idade(self):
        return self._idade

    @idade.setter
    def idade(self, i):
        if i >= 0:
            self._idade = i
        else:
            print("Idade não pode ser negativa!")

    @property
    def salario(self):
        return self._salario

    @salario.setter
    def salario(self, s):
        if s >= 0:
            self._salario = s
```

4. EnfermeiroChefe.py — Exemplo de Herança Hierárquica

A quarta imagem mostra a classe EnfermeiroChefe, que herda de Enfermeiro. Esta classe representa um profissional de enfermagem com responsabilidades acrescidas, como coordenar equipas, supervisionar procedimentos e garantir o bom funcionamento da unidade.

No código, é provável que esta classe adicione métodos para gestão de outros enfermeiros ou controlo de turnos. Demonstra claramente como a herança pode ser usada para criar níveis hierárquicos dentro da mesma profissão.

Importância no projeto: mostra a flexibilidade da orientação a objetos, permitindo criar subclasses especializadas sem reescrever código.

```
from Enfermeiro import Enfermeiro
from Administrativo import Administrativo

class EnfermeiroChefe(Enfermeiro, Administrativo):
    def __init__(self, nome, idade, salario, turno, setor, bonus_chefia, horas):
        Enfermeiro.__init__(self, nome, idade, salario, turno)
        Administrativo.__init__(self, nome, idade, salario, setor, horas)
        self._bonus_chefia = bonus_chefia

    @property
    def bonus_chefia(self):
        return self._bonus_chefia

    @bonus_chefia.setter
    def bonus_chefia(self, valor):
        if valor > 0:
            self._bonus_chefia = valor
        else:
            print("O bônus tem de ser positivo!")

    def calcular_pagamento(self):
        pagamento_enfermeiro = Enfermeiro.calcular_pagamento(self)
        pagamento_administrativo = Administ
```

5. SalaCirurgia.py — Modelação de Recursos Físicos

A quinta imagem apresenta a classe SalaCirurgia, derivada de Sala. Esta classe é responsável por representar as salas de operações, com características específicas, como tipo de equipamento, capacidade máxima e estado de esterilização.

A existência desta classe mostra que o projeto não se limita a modelar pessoas, mas também recursos físicos do hospital, aplicando os mesmos princípios de abstração e especialização.

Importância no projeto: ilustra a aplicação da POO na modelação de espaços e equipamentos, permitindo distinguir entre salas de consulta e salas de cirurgia.

```
from Sala import Sala

class SalaCirurgia(Sala):
    def __init__(self, numero, capacidade):
        super().__init__(numero, capacidade)
        self.equipamentos = []

    def adicionar_equipamento(self, equipamento):
        if isinstance(equipamento, str) and equipamento.strip():
            self.equipamentos.append(equipamento.strip())
            print(f"Equipamento '{equipamento}' adicionado à sala {self.numero}.")
        else:
            print("Nome de equipamento inválido!")

    def detalhar_sala(self):
        print(f"Sala Nº{self.numero} (Cirurgia)")
        print(f"Capacidade: {self.capacidade}")
        print("Equipamentos disponíveis:")
        if not self.equipamentos:
            print(" - Nenhum equipamento registado.")
        else:
            for i, eq in enumera
```

6. main.py — Núcleo Funcional e Execução do Programa

A última imagem contém o ficheiro main.py, que é o ponto de entrada do programa. Aqui estão definidas listas que armazenam todos os objetos criados (pacientes, médicos, enfermeiros, salas, etc.) e funções que permitem criar novos registos através de entradas do utilizador (input).

Por exemplo, a função criar_paciente() pede ao utilizador o nome, idade e número de utente, cria um objeto Paciente e adiciona-o à lista de pacientes. É provável que o programa principal também apresente um menu que permita ao utilizador escolher que tipo de entidade quer criar ou visualizar.

Importância no projeto: é a peça central da aplicação. Reúne todas as classes e permite interagir com o sistema, demonstrando o funcionamento conjunto de todos os componentes.

```
from Paciente import Paciente
from Medico import Medico
from Enfermeiro import Enfermeiro
from Administrativo import Administrativo
from EnfermeiroChefe import EnfermeiroChefe
from SalaConsulta import SalaConsulta
from SalaCirurgia import SalaCirurgia

pacientes = []
medicos = []
enfermeiros = []
administrativos = []
chefes = []
salas_consulta = []
salas_cirurgia = []


def criar_paciente():
    nome = input("Nome do paciente: ").strip()
    idade = int(input("Idade: "))
    numero = int(input("Número de utente: "))
    p = Paciente(nome, idade, numero)
    pacientes.append(p)
    print(f"Paciente {nome} criado com sucesso!")


def criar_medico():
    nome = input("Nome do médico: ").strip()
    idade = int(input("Idade: "))
    cargo = input("Cargo: ").strip()
    salario = float(input("S
```

Conclusão

O presente projeto demonstra a aplicação prática dos princípios da programação orientada a objetos em Python, através do desenvolvimento de um sistema simples de gestão hospitalar.
Foram criadas várias classes interligadas, representando pessoas, funcionários e salas, permitindo uma estrutura modular e facilmente extensível.
O código apresenta uma hierarquia bem definida, com classes base e subclasses que refletem diferentes papéis e responsabilidades dentro de um ambiente hospitalar.
A implementação do ficheiro main.py integra todas as componentes do sistema, possibilitando a criação e gestão de instâncias de forma interativa.

Em suma, o projeto evidencia uma compreensão sólida dos conceitos de herança, encapsulamento e abstração, aplicados a um contexto realista e funcional.