Escola Profissional
BENTO DE JESUS CARAÇA
DELEGAÇÃO DE LISBOA

Ano Letivo _____/_____

Ficha de Trabalho

Curso GPSI **Turma** **Ano** **Data**

Disciplina PSI **Módulo** 11

Nome do aluno **Número**

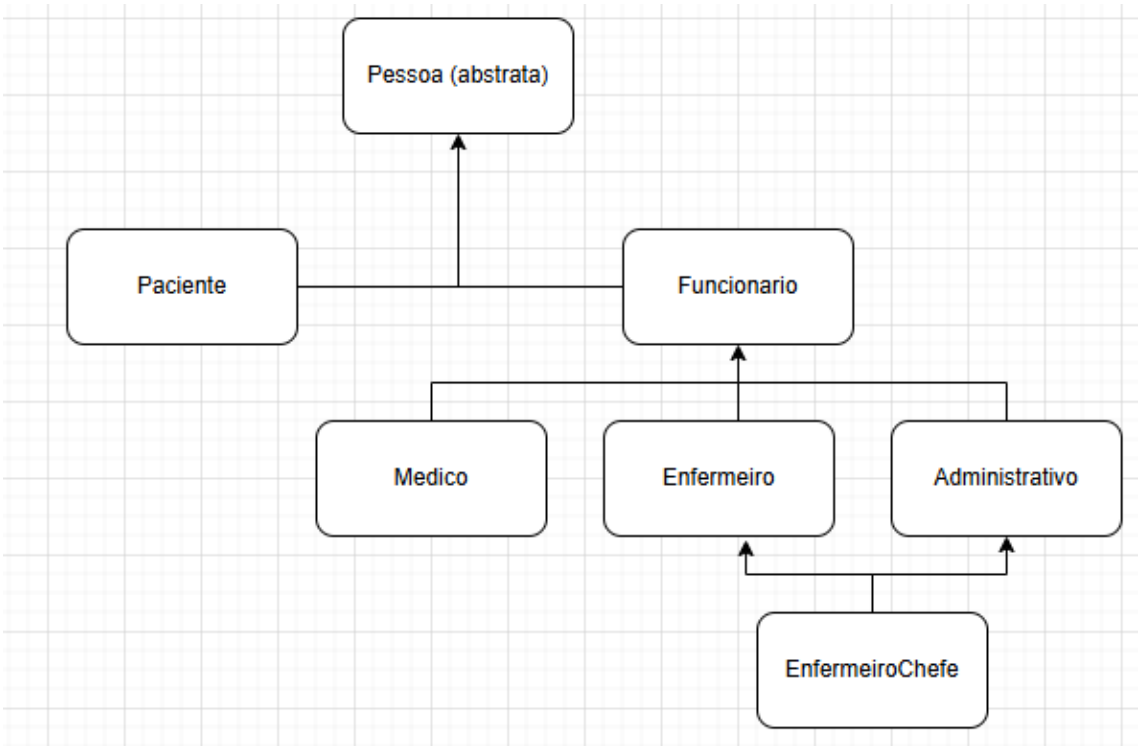
Professor Breno Sousa **Avaliação**

1º Trabalho Avaliativo

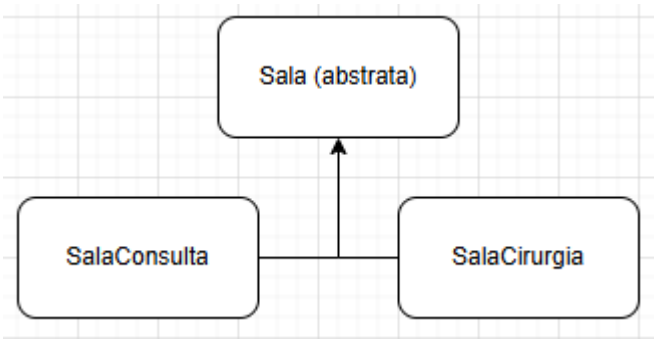
1. Você e sua equipa foram contratados para desenvolver um sistema para um hospital. A aplicação deve ser desenvolvida em Python. É solicitado o uso de heranças simples, herança múltiplas, polimorfismo e classes abstratas. O uso de módulos é obrigatório. Lembre-se, sempre que for preciso, a equipa deve fazer uso de estrutura de dados (por exemplo, listas, dicionários, tuplas, etc.) e laços de repetição (for, while, etc.). Em resumo, o hospital precisa de um sistema para gerir:
    - Registar pacientes e funcionários.;
    - Gerir consultas e salas;

Em seguida, apresenta-se o modelo das heranças e funções solicitadas.

Estrutura Geral das Classes:



Estrutura de Salas:



Sugestão de funções para cada classe. Porém, sempre que necessário, adicionar novas funções ou classes.

Classe Pessoa (Abstrata):

- __init__(self, nome, idade): inicializa nome e idade.
- @property nome: retorna o nome da pessoa.
- @nome.setter: define o nome, apenas se não for vazio.
- @property idade: retorna a idade da pessoa.
- @idade.setter: define a idade, apenas se for positiva.
- exibir_informacoes(self): método abstrato que deverá ser implementado nas subclasses.

Classe Paciente:

- __init__(self, nome, idade, numero_utente): inicializa paciente com nome, idade e número de utente.
- @property numero_utente: retorna o número de utente.
- adicionar_registro(self, descricao): adiciona uma entrada ao histórico médico.
- mostrar_historico(self): exibe o histórico médico do paciente.
- exibir_informacoes(self): mostra nome, idade e número de utente.

Classe Funcionario:

- __init__(self, nome, idade, cargo, salario): inicializa dados do funcionário.
- @property salario: retorna o salário atual.
- @salario.setter: atualiza o salário, impedindo valores negativos.
- mostrar_informacoes(self): exibe nome, cargo e salário.
- aplicar_aumento(self, percentual): aumenta o salário com base em um percentual.

Classe Medico:

- __init__(self, nome, idade, salario_base, especialidade): inicializa médico com especialidade e salário base.
- @property especialidade: retorna a especialidade do médico.
- @especialidade.setter: define a especialidade, desde que não seja vazia.
- adicionar_paciente(self, paciente): adiciona paciente à lista de atendidos.
- listar_pacientes(self): mostra todos os pacientes do médico.
- calcular_pagamento(self): retorna salário base + valor fixo por paciente atendido.
- exibir_informacoes(self): exibe nome, especialidade e número de pacientes.

Classe Enfermeiro:

- __init__(self, nome, idade, salario_base, turno): inicializa enfermeiro com turno (dia/noite).
- @property turno: retorna o turno atual.
- @turno.setter: define o turno apenas se for "dia" ou "noite".
- adicionar_paciente(self, paciente): adiciona paciente sob cuidado.
- listar_pacientes(self): exibe pacientes sob responsabilidade.
- calcular_pagamento(self): retorna salário base + adicional conforme o turno.
- exibir_informacoes(self): exibe nome, turno e total de pacientes.

Classe Administrativo:

- __init__(self, nome, idade, salario_base, setor): inicializa funcionário administrativo com setor e salário base.
- @property setor: retorna o setor de atuação.
- @setor.setter: define o setor apenas se for válido.
- registrar_horas(self, horas): acumula horas trabalhadas.
- calcular_pagamento(self): retorna salário base + valor por hora registrada.
- exibir_informacoes(self): mostra nome, setor e total de horas trabalhadas.

Classe EnfermeiroChefe:

- __init__(self, nome, idade, salario_base, turno, setor, bonus_chefia): inicializa o híbrido com dados de enfermeiro e administrativo.
- @property bonus_chefia: retorna o bônus adicional.
- @bonus_chefia.setter: define o bônus apenas se for positivo.
- calcular_pagamento(self): combina o pagamento de enfermeiro e administrativo + bônus de chefia.
- exibir_informacoes(self): mostra nome, turno, setor e pacientes sob cuidado.

Classe Sala (abstrata)

- __init__(self, numero, capacidade): inicializa a sala com número e capacidade máxima.
- @property numero: retorna o número da sala.
- @numero.setter: define o número apenas se for positivo.
- @property capacidade: retorna a capacidade da sala.
- @capacidade.setter: define capacidade apenas se for maior que zero.
- detalhar_sala(self): método abstrato para descrição da sala.

Classe SalaConsulta (herda de Sala)

- __init__(self, numero, capacidade, medico_responsavel): inicializa sala de consulta com médico e pacientes.
- @property medico_responsavel: retorna o médico responsável.
- @medico_responsavel.setter: define o médico apenas se for uma instância válida de Médico.
- agendar_consulta(self, paciente): adiciona paciente à lista de consultas.
- detalhar_sala(self): exibe número, capacidade e nome do médico responsável.

Classe SalaCirurgia (herda de Sala)

- __init__(self, numero, capacidade): inicializa sala de cirurgia com número e capacidade.
- adicionar_equipamento(self, equipamento): adiciona equipamento à lista.
- detalhar_sala(self): exibe número, capacidade e lista de equipamentos.

Classe Consulta (OPCIONAL):

- __init__(self, medico, paciente, data, tipo): cria a ligação entre médico e paciente com data e tipo (rotina, emergência, etc.).
- @property tipo: retorna o tipo da consulta.
- @tipo.setter: define o tipo apenas se for string válida.
- exibir_detalhes(self): mostra informações do médico, paciente e tipo de consulta.

Critérios de avaliação:

- A solução desenvolvida deve acompanhar um relatório do desenvolvimento;
- Cada aluno será avaliado individualmente.
- Nota = (Nota do Projeto * 0,85) + (Nota de Participação * 0,15)
- O aluno que demonstrar comprometimento e assiduidade durante as aulas do módulo terá uma adição de até 10%. Assim sendo:
- Nota Final = Nota + (Nota * 0,1)

ALERTA:

- Está proibido o uso de qualquer ferramenta de geração de código que faça uso de inteligência artificial.
- Caso algum integrante da dupla não participe do desenvolvimento do projeto, poderá receber uma penalização de até 50% na nota final. Para isso, o professor deve ser notificado sobre o problema. **Nota Final = Nota - (Nota * 0,5)**
- Se não houver comunicação, todos os membros da dupla poderão ser penalizados em até 50% da nota final. No entanto, se a comunicação for feita antes da metade do tempo previsto para a realização do projeto, apenas o aluno que não estiver participando será penalizado. **Nota Final = Nota - (Nota * 0,5)**
- **Total de horas para o desenvolvimento: 7 horas**