```
class Atributo:
    def __init__(self, ataque=0, defesa=0, hp=0, estamina=0):
        self.__atributos = {
            'ataque': ataque,
            'defesa': defesa,
            'hp': hp,
            'estamina': estamina
        }

    @property
    def atributos(self):
        return self.__atributos

    @atributos.setter
    def atributos(self, novos_atributos):
        if isinstance(novos_atributos, dict):
            self.__atributos.update(novos_atributos)
        else:
            raise ValueError("Atributos devem ser passados em um dicionário.")

from personagem import Personagem
from boss import Boss

class Batalha():
    def __init__(self, personagem, boss, batalha_final, finalizada=False, turno=0):
        if isinstance(personagem, Personagem):
            self.__personagem = personagem
        else:
            return
        if isinstance(boss, Boss):
            self.__boss = boss
        else:
            return
        self.__batalha_final = batalha_final
        self.__finalizada = finalizada
        self.__turno = turno



    @property
    def personagem(self):
        return self.__personagem

    @personagem.setter
    def personagem(self, personagem: Personagem):
        if isinstance(personagem, Personagem):
            self.__personagem = personagem
```

```
    @property
    def boss(self):
        return self.__boss

    @boss.setter
    def boss(self, boss: Boss):
        if isinstance(boss, Boss):
            self.__boss = boss

    @property
    def batalha_final(self):
        return self.__batalha_final

    @batalha_final.setter
    def batalha_final(self, batalha_final):
        self.__batalha_final = batalha_final

    @property
    def finalizada(self):
        return self.__finalizada

    @finalizada.setter
    def finalizada(self, finalizada):
        self.__finalizada = finalizada

    @property
    def turno(self):
        return self.__turno

    @turno.setter
    def turno(self, turno):
        self.__turno = turno

from batalha import Batalha
from batalhaView import BatalhaView

class BatalhaController():
    def __init__(self, batalha: Batalha):
        self.__batalha = batalha
        self.__tela = BatalhaView()

    def realizar_turno(self, acao_personagem):
        personagem = self.__batalha.personagem
        boss = self.__batalha.boss

        if acao_personagem == 1:
            personagem.atacar(boss)
            self.__tela.mostra_mensagem(f"{personagem.nome} atacou {boss.nome}!")
```

```
    elif acao_personagem == 2:
        personagem.defender()
        self.__tela.mostra_mensagem(f"{personagem.nome} se defendeu!")
    elif acao_personagem == 3:
        pass
    elif acao_personagem == 4:
        personagem.usar_habilidade(boss)
        self.__tela.mostra_mensagem(f"{personagem.nome} usou uma habilidade!")

    if not self.__batalha.finalizada:
        boss.realizar_acao(personagem)

    self.__batalha.turno += 1

def verificar_vencedor(self):
    if self.__batalha.boss.atributos['hp'] <= 0:
        self.__batalha.finalizada = True
        return "vitória"
    elif self.__batalha.personagem.atributos['hp'].valor <= 0:
        self.__batalha.finalizada = True
        return "derrota"
    return

def iniciar_batalha(self):
    while not self.__batalha.finalizada:
        acao_personagem = self.__tela.tela_opcoes()
        self.realizar_turno(acao_personagem)

        resultado = self.verificar_vencedor()
        if resultado == "vitória":
            self.__tela.mostra_resultado("Você venceu!")
        elif resultado == "derrota":
            self.__tela.mostra_resultado("Você foi derrotado!")

            class BatalhaView:
def tela_opcoes(self):
    print("-------- BATALHA ----------")
    print("Escolha a ação:")
    print("1 - Atacar")
    print("2 - Defender")
    print("3 - Usar Item")
    print("4 - Usar Habilidade")

    opcao = int(input("Escolha a ação: "))
    return opcao

def escolher_habilidade(self):
    #colocar a logica do personagem escolher a habilidade
```

```
        pass

    def mostra_resultado(self, mensagem):
        print(mensagem)

    def mostra_mensagem(self, msg):
        print(msg)

from atributo import Atributo

class Boss:
    def __init__(self, nome, dificuldade, nivel_requerido, ataque, defesa, hp, estamina,
diretor=False):
        super().__init__(ataque, defesa, hp, estamina)
        self.__nome = nome
        self.__dificuldade = dificuldade
        self.__nivel_requerido = nivel_requerido
        self.__diretor = diretor
        self.atributos = {
            'ataque': ataque,
            'defesa': defesa,
            'hp': hp,
            'estamina': estamina
        }


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

    @nome.setter
    def nome(self, nome):
        self.__nome = nome

    @property
    def dificuldade(self):
        return self.__dificuldade

    @dificuldade.setter
    def dificuldade(self, dificuldade):
        self.__dificuldade = dificuldade

    @property
    def nivel_requerido(self):
        return self.__nivel_requerido

    @nivel_requerido.setter
    def nivel_requerido(self, nivel_requerido):
```

```
        self.__nivel_requerido = nivel_requerido

    @property
    def diretor(self):
        return self.__diretor

    @diretor.setter
    def diretor(self, diretor):
        self.__diretor = diretor

    @property
    def atributos(self):
        return self.__atributos

    @atributos.setter
    def atributos(self, atributos: Atributo):
        self.__atributos = atributos

        from boss import Boss
from bossView import BossView

class BossController:
    def __init__(self):
        self.__bosses = []
        self.__bossView = BossView()

    def cadastrar_boss(self):
        dados_boss = self.__bossView.pega_dados_boss()
        boss = Boss(dados_boss)
        self.__bossView.mostra_mensagem(f"Boss {self.__boss.nome} cadastrado com
sucesso!")

        dados_personagem = self.__personagemView.pega_dados_personagem()
        personagem_existente =
self.pega_personagem_por_nome(dados_personagem["nome"])
        if personagem_existente is None:
            classe = dados_personagem["classe"]
            personagem = Personagem(dados_personagem["nome"],
dados_personagem["nivel"], dados_personagem["experiencia"], None, None,
dados_personagem["classe"])
            self.__personagens.append(personagem)
            self.__personagemView.mostra_mensagem(f"Personagem
{dados_personagem['nome']} cadastrado com sucesso!")
        else:
            self.__personagemView.mostra_mensagem(f"O personagem
{dados_personagem['nome']} já existe!")

    def mostrar_atributos(self):
```

```
        if self.__boss:
            atributos = self.__boss.mostrar_atributos()
            self.__bossView.mostra_atributos(atributos)
        else:
            self.__bossView.mostra_mensagem("Nenhum boss cadastrado.")


class BossView:
    def pega_dados_boss(self):
        print("----------CADASTRO BOSS---------")
        nome = input("Nome: ")
        dificuldade = int(input("Dificuldade: "))
        nivel_requerido = int(input("Nível Requerido: "))
        diretor = input("É o diretor? (S/N): ")
        diretor = False if diretor == "N" else True if diretor == "S" else None
        ataque = dificuldade * 2
        defesa = (dificuldade * 2) + 5
        hp = (dificuldade**2) + 25
        estamina = dificuldade * 2


        return {
            'nome': nome,
            'dificuldade': dificuldade,
            'nivel_requerido': nivel_requerido,
            'ataque': ataque,
            'defesa': defesa,
            'hp': hp,
            'estamina': estamina,
            'diretor': diretor
        }

    def mostra_atributos(self, atributos):
        print(atributos)

    def mostra_mensagem(self, msg):
        print(msg)

        from atributo import Atributo

class ClassePersonagem(Atributo):
    def __init__(self, nome_classe, evolucao=0, ataque=0, defesa=0, hp=0, estamina=0):
        super().__init__(ataque, defesa, hp, estamina)
        self.__nome_classe = nome_classe
        self.__evolucao = evolucao


    @property
```

```
    def nome_classe(self):
        return self.__nome_classe

    @nome_classe.setter
    def nome_classe(self, nome_classe):
        self.__nome_classe = nome_classe

    @property
    def evolucao(self):
        return self.__evolucao

    @evolucao.setter
    def evolucao(self, evolucao):
        self.__evolucao = evolucao

        from quiz import Quiz

class Curso(Quiz):
    def __init__(self, nome, nivel_requerido, xp_ganho, setor, dificuldade, realizado=False,
acertos: Quiz=0):
        self._nome = nome
        self._nivel_requerido = nivel_requerido
        self._xp_ganho = xp_ganho
        self._setor = setor
        self._dificuldade = dificuldade
        self._realizado = realizado
        self._acertos = acertos


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

    @nome.setter
    def nome(self, nome):
        self._nome = nome

    @property
    def nivel_requerido(self):
        return self._nivel_requerido

    @nivel_requerido.setter
    def nivel_requerido(self, nivel_requerido):
        self._nivel_requerido = nivel_requerido

    @property
    def xp_ganho(self):
        return self._xp_ganho
```

```
    @xp_ganho.setter
    def xp_ganho(self, xp_ganho):
        self._xp_ganho = xp_ganho

    @property
    def setor(self):
        return self._setor

    @setor.setter
    def setor(self, setor):
        self._setor = setor

    @property
    def dificuldade(self):
        return self._dificuldade

    @dificuldade.setter
    def dificuldade(self, dificuldade):
        self._dificuldade = dificuldade

    @property
    def realizado(self):
        return self._realizado

    @realizado.setter
    def realizado(self, realizado):
        self._realizado = realizado

    @property
    def acertos(self):
        return self._acertos

    @acertos.setter
    def acertos(self, acertos):
        self._acertos = acertos

    def realizar_quiz(self, gabarito):
        pass

    from curso import Curso
from cursoView import CursoView

class CursoController():
    def __init__(self):
        self.cursos = []
        self.__cursoView = CursoView()
```

```
    def cadastrar_curso(self):
        dados_curso = self.__cursoView.pega_dados_curso()
        print(dados_curso)
        curso = Curso(dados_curso["nome"], dados_curso["nivel_requerido"],
dados_curso["xp_ganho"], dados_curso["setor"], dados_curso["dificuldade"],
dados_curso["realizado"])
        self.cursos.append(curso)
        print(self.cursos)
        self.__cursoView.mostra_mensagem(f"O curso {dados_curso["nome"]} foi cadastrado
com sucesso \n")


    def alterar_curso(self):
        self.__cursoView.mostra_cursos(self.cursos)
        nome = self.__cursoView.seleciona_curso()
        curso = None
        for c in self.cursos:
            if c.nome == nome:
                curso = c
                break

        if curso is None:
            self.__cursoView.mostra_mensagem(f"Curso com o nome {nome} não foi
encontrado \n")



        novos_dados = self.__cursoView.pega_dados_curso()

        curso.nome = novos_dados["nome"]
        curso.nivel_requerido = novos_dados["nivel_requerido"]
        curso.xp_ganho = novos_dados["xp_ganho"]
        curso.setor = novos_dados["setor"]
        curso.dificuldade = novos_dados["dificuldade"]
        curso.realizado = novos_dados["realizado"]

        self.__cursoView.mostra_mensagem(f"O curso foi alterado com sucesso \n")


    def excluir_curso(self):
        self.__cursoView.mostra_cursos(self.__cursos)
        nome = self.__cursoView.seleciona_curso()
        curso = None
        for c in self.cursos:
            if c.nome == nome:
                self.cursos.remove(c)
        self.__cursoView.mostra_mensagem(f"O curso {c.nome} foi removido com sucesso \n")

class CursoView():
```

```
    def pega_dados_curso(self):
        print("---DADOS DO CURSO---")
        ##Aqui tem que realizar diversos testes pra cada variável.
        nome = input("Digite o nome do curso: ")
        nivel_requerido = input("Qual o nível requerido do curso? ")
        xp_ganho = input("Quanto de xp esse curso vale? ")
        setor = input("Esse curso é para qual setor? ")
        dificuldade = input("Qual a dificuldade do curso? ")
        print("\n")
        realizado = False
        return {"nome": nome, "nivel_requerido": nivel_requerido, "xp_ganho": xp_ganho,
"setor": setor, "dificuldade": dificuldade, "realizado": realizado}



    def mostra_mensagem(self,mensagem):
        print(mensagem)



    def mostra_cursos(self,cursos):
        print("----LISTA DE CURSOS---- \n")
        print(cursos)
        for curso in cursos:
            print(f"Nome: {curso.nome}, Nível Requerido: {curso.nivel_requerido}, XP Ganhado:
{curso.xp_ganho}, "
                  f"Setor: {curso.setor}, Dificuldade: {curso.dificuldade}, Realizado:
{curso.realizado} \n")

    def seleciona_curso(self):
        nome = input("Qual curso você quer selecionar? ")
        ##Aqui tem que testar se o curso está na lista, senão raise exception
        return nome

from setor import Setor

class Dungeon(Setor):
    def __init__(self, nome, nivel_requerido, xp_ganho, dificuldade, conquistada=False):
        self._nome = nome
        self._nivel_requerido = nivel_requerido
        self._xp_ganho = xp_ganho
        self._dificuldade = dificuldade
        self._conquistada = conquistada



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

```
    @nome.setter
    def nome(self, nome):
        self._nome = nome

    @property
    def nivel_requerido(self):
        return self._nivel_requerido

    @nivel_requerido.setter
    def nivel_requerido(self, nivel_requerido):
        self._nivel_requerido = nivel_requerido

    @property
    def xp_ganho(self):
        return self._xp_ganho

    @xp_ganho.setter
    def xp_ganho(self, xp_ganho):
        self._xp_ganho = xp_ganho

    @property
    def dificuldade(self):
        return self._dificuldade

    @dificuldade.setter
    def dificuldade(self, dificuldade):
        self._dificuldade = dificuldade

    @property
    def conquistada(self):
        return self._conquistada

    @conquistada.setter
    def conquistada(self, conquistada):
        self._conquistada = conquistada

        from dungeonView import DungeonView
from dungeon import Dungeon
from setorController import SetorController

class DungeonController():
    def __init__(self):
        self.__dungeons = []
        self.__dungeonView = DungeonView()
        self.__setorController = SetorController()

    def cadastrar_dungeon(self):
        dados_dungeon = self.__dungeonView.pega_dados_dungeon()
```

```
        dificuldade = self.__setorController.calcular_media_dificuldades
        dungeon = Dungeon(dados_dungeon["nome"], dados_dungeon["nivel_requerido"],
dados_dungeon["xp_ganho"], dados_dungeon[dificuldade], dados_dungeon["status"])

        for _ in range(dados_dungeon["n_setores"]):
            setor = self.__setorController.adicionar_setor()
            dungeon.setores.append(setor)

        self.__dungeons.append(dungeon)
        self.__dungeonView.mostra_mensagem(f"A dungeon {dados_dungeon["nome"]} foi
cadastrada com sucesso")

class DungeonView():
    def pega_dados_dungeon(self):
        print("------DADOS DA DUNGEON------")
        nome = input("Digite o nome da dungeon: ")
        n_setores = input("Digite o número de setores: ")
        return {"nome": nome, "n_setores": n_setores}


    def mostra_mensagem(self,mensagem):
        print(mensagem)

from sistemaController import SistemaControllerr

def main():
    sistema = SistemaControllerr()
    while True:
        sistema.iniciar()

if __name__ == "__main__":
    main()from classePersonagem import ClassePersonagem
from pocao_hp import PocaoHP
from pocao_est import PocaoEstamina

class Personagem():
    def __init__(self, nome, nivel, experiencia, nome_classe: str, pocao_hp: PocaoHP=None,
pocao_est: PocaoEstamina=None, pontos_disponiveis=10):
        self.__nome = nome
        self.__nivel = nivel
        self.__experiencia = experiencia
        self.__classe_personagem = ClassePersonagem(nome_classe=nome_classe)

        self.__pocao_hp = pocao_hp if pocao_hp else PocaoHP(quant=3)
        self.__pocao_est = pocao_est if pocao_est else PocaoEstamina(quant=3)

        self.__pontos_disponiveis = pontos_disponiveis
```

```
@property
def nome(self):
    return self.__nome

@nome.setter
def nome(self, nome):
    self.__nome = nome

@property
def nivel(self):
    return self.__nivel

@nivel.setter
def nivel(self, nivel):
    self.__nivel = nivel

@property
def experiencia(self):
    return self.__experiencia

@experiencia.setter
def experiencia(self, experiencia):
    self.__experiencia = experiencia

@property
def classe_personagem(self):
    return self.__classe_personagem

@classe_personagem.setter
def classe_personagem(self, classe_personagem: ClassePersonagem):
    self.__classe_personagem = classe_personagem

@property
def pocao_hp(self):
    return self.__pocao_hp

@pocao_hp.setter
def pocao_hp(self, pocao_hp: PocaoHP):
    self.__pocao_hp = pocao_hp

@property
def pocao_est(self):
    return self.__pocao_est

@pocao_est.setter
def pocao_est(self, pocao_est: PocaoEstamina):
    self.__pocao_est = pocao_est
```

```
    @property
    def pontos_disponiveis(self):
        return self.__pontos_disponiveis

    @pontos_disponiveis.setter
    def pontos_disponiveis(self, pontos):
        self.__pontos_disponiveis = pontos
from personagemView import PersonagemView
from personagem import Personagem
import time


class PersonagemController:
    def __init__(self):
        self.__personagens = []
        self.__personagemView = PersonagemView()

    @property
    def personagens(self):
        return self.__personagens

    def pega_personagem_por_nome(self, nome: str):
        for personagem in self.__personagens:
            if personagem.nome == nome:
                return personagem
        return None

    def cadastrar_personagem(self, nome, nivel=1, experiencia=0, nome_classe=""):
        if self.pega_personagem_por_nome(nome) is not None:
            raise ValueError(f"O personagem {nome} já existe!")


        personagem = Personagem(
            nome=nome,
            nivel=nivel,
            experiencia=experiencia,
            nome_classe=nome_classe
        )
        self.__personagens.append(personagem)
        return personagem

    def mostrar_status(self, personagem: Personagem):
        status = {
            'nome': personagem.nome,
            'nivel': personagem.nivel,
            'experiencia': personagem.experiencia,
            'ataque': personagem.classe_personagem.atributos['ataque'],
```

```
            'defesa': personagem.classe_personagem.atributos['defesa'],
            'hp': personagem.classe_personagem.atributos['hp'],
            'estamina': personagem.classe_personagem.atributos['estamina'],
            'pontos_disponiveis': personagem.pontos_disponiveis,
            'pocoes_hp': personagem.pocao_hp.quant,
            'pocoes_est': personagem.pocao_est.quant
        }
        self.__personagemView.mostrar_status(status)

    def upar_atributos(self, personagem: Personagem):
        if personagem.pontos_disponiveis > 0:
            atributo_escolhido = self.__personagemView.escolher_atributo()
            pontos = self.__personagemView.pega_quantidade_pontos()
            if pontos <= personagem.pontos_disponiveis:
                if atributo_escolhido in personagem.classe_personagem.atributos:
                    personagem.classe_personagem.atributos[atributo_escolhido] += pontos
                    personagem.pontos_disponiveis -= pontos
                    self.__personagemView.mostra_mensagem(f"Atributo {atributo_escolhido}
aumentado em {pontos} pontos!")
                else:
                    self.__personagemView.mostra_mensagem("Atributo inválido.")
            else:
                self.__personagemView.mostra_mensagem("Pontos insuficientes.")
        else:
            self.__personagemView.mostra_mensagem("Você não tem pontos disponíveis para
distribuir.")

    def upar_nivel(self, personagem: Personagem):
        if personagem.experiencia >=
self.__calcular_exp_para_proximo_nivel(personagem.nivel):
            personagem.nivel += 1
            personagem.experiencia = 0
            personagem.pontos_disponiveis += 5
            self.__personagemView.mostra_mensagem(f"{personagem.nome} subiu para o nível
{personagem.nivel}!")
        else:
            self.__personagemView.mostra_mensagem(f"{personagem.nome} não tem
experiência suficiente para subir de nível.")

    def __calcular_exp_para_proximo_nivel(self, nivel_atual):
        return 100 + (10 * nivel_atual)

    def usar_item(self, personagem: Personagem):
        tipo_item = self.__personagemView.escolher_item()
        if tipo_item == 1 and personagem.pocao_hp and personagem.pocao_hp.quant > 0:
            personagem.classe_personagem.atributos['hp'] += personagem.pocao_hp.valor
            personagem.pocao_hp.quant -= 1
```

```
            self.__personagemView.mostra_mensagem(f"{personagem.nome} usou Poção de
HP!")
        elif tipo_item == 2 and personagem.pocao_est and personagem.pocao_est.quant > 0:
            personagem.classe_personagem.atributos['estamina'] +=
personagem.pocao_est.valor
            personagem.pocao_est.quant -= 1
            self.__personagemView.mostra_mensagem(f"{personagem.nome} usou Poção de
Estamina!")
        else:
            item_nomes = {
                1: "Poção de HP",
                2: "Poção de Estamina"
            }
            item_nome = item_nomes.get(tipo_item, "Item desconhecido")
            self.__personagemView.mostra_mensagem(f"{personagem.nome} não tem
{item_nome} disponível!")

    def usar_habilidade(self, classe_personagem: Personagem):
        #colocar a logica em que o jogador escolhe qual habilidade ele quer usar,
        #podendo escolher entre 3 habilidades, que tem efeitos diferentes no inimigo.
        pass

    import os
class PersonagemView():

    def mostrar_status(self, dados_personagem):
        os.system('cls' if os.name == 'nt' else 'clear')
        print("-------- STATUS ----------")
        print(f"Nome: {dados_personagem['nome']}")
        print(f"Nível: {dados_personagem['nivel']}")
        print(f"Experiência: {dados_personagem['experiencia']}")
        print(f"Ataque: {dados_personagem['ataque']}")
        print(f"Defesa: {dados_personagem['defesa']}")
        print(f"HP: {dados_personagem['hp']}")
        print(f"Estamina: {dados_personagem['estamina']}")
        print(f"Pontos disponíveis para distribuir: {dados_personagem['pontos_disponiveis']}")
        print(f"Poções de HP: {dados_personagem['pocoes_hp']}")
        print(f"Poções de Estamina: {dados_personagem['pocoes_est']}")

    def escolher_atributo(self):
        os.system('cls' if os.name == 'nt' else 'clear')
        print("-------- UPAR ATRIBUTOS ----------")
        print("Escolha o atributo para aumentar:")
        print("1 - Ataque")
        print("2 - Defesa")
        print("3 - HP")
        print("4 - Estamina")
        opcao = int(input("Digite o número do atributo: "))
```

```
        atributos = {1: "ataque", 2: "defesa", 3: "hp", 4: "estamina"}
        return atributos.get(opcao, None)

    def pega_quantidade_pontos(self):
        print("-------- UPAR ATRIBUTOS ----------")
        pontos = int(input("Quantos pontos deseja aplicar? "))
        return pontos

    def escolher_item(self):
        os.system('cls' if os.name == 'nt' else 'clear')
        print("Escolha o item para usar:")
        print("1 - Poção de HP")
        print("2 - Poção de Estamina")
        return int(input("Digite o número do item: "))

    def escolher_habilidade(self):
        os.system('cls' if os.name == 'nt' else 'clear')
        print("Escolha a habilidade:")
        print("1 - hab1")
        print("2 - hab2")
        print("3 - hab3")
        #Colocar logica para que as habilidades mostradas sejam somente aquelas que a
classe tem, tipo, trainee só tem
        # a primeira habilidade, enquanto o estagiario tem a primeira e a segunda, e o CLT tem
a primeira, segunda e a terceira.
        return int(input("Digite o número da habilidade: "))

    def mostra_mensagem(self, msg):
        os.system('cls' if os.name == 'nt' else 'clear')
        print("****************************************")
        print(msg)
        print("****************************************")

        class Pocao:
    def __init__(self, nome, valor):
        self.__nome = nome
        self.__valor = valor


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

    @nome.setter
    def nome(self, nome):
        self.__nome = nome

    @property
```

```
    def valor(self):
        return self.__valor

    @valor.setter
    def valor(self, valor):
        self.__valor = valor

        from pocao  import Pocao

class PocaoEstamina(Pocao):
    def __init__(self, quant):
        super().__init__("Poção de Estamina", 5)
        self.__quant = quant


    @property
    def quant(self):
        return self.__quant

    @quant.setter
    def quant(self, quant):
        self.__quant = quant

        from pocao  import Pocao

class PocaoHP(Pocao):
    def __init__(self, quant):
        super().__init__("Poção de HP", 10)
        self.__quant = quant


    @property
    def quant(self):
        return self.__quant

    @quant.setter
    def quant(self, quant):
        self.__quant = quant

        class Quiz:
    def __init__(self, setor, pergunta, respostas, resposta_correta):
        self._setor = setor
        self._pergunta = pergunta
        self._respostas = respostas
        self._resposta_correta = resposta_correta
        self._selecionada = None
        self._gabaritou_miga = False
```

```
@property
def setor(self):
    return self._setor

@setor.setter
def setor(self, setor):
    self._setor = setor

@property
def pergunta(self):
    return self._pergunta

@pergunta.setter
def pergunta(self, pergunta):
    self._pergunta = pergunta

@property
def respostas(self):
    return self._respostas

@respostas.setter
def respostas(self, respostas):
    self._respostas = respostas

@property
def resposta_correta(self):
    return self._resposta_correta

@resposta_correta.setter
def resposta_correta(self, resposta_correta):
    self._resposta_correta = resposta_correta

@property
def selecionada(self):
    return self._selecionada

@selecionada.setter
def selecionada(self, selecionada):
    self._selecionada = selecionada

@property
def gabaritou_miga(self):
    return self._gabaritou_miga

@gabaritou_miga.setter
def gabaritou_miga(self, gabaritou_miga):
    self._gabaritou_miga = gabaritou_miga
```

```
    def responder(self, resposta_selecionada):
        pass

    from quiz import Quiz
from quizView import QuizView
from cursoView import CursoView

class QuizController():
    def __init__(self):
        self.__quizrh = {
    "1": {
        "a": "Aumentar o número de funcionários",
        "b": "Desenvolver e gerir talentos na organização",
        "c": "Reduzir os custos operacionais",
        "pergunta": "Qual é o principal objetivo da gestão de recursos humanos?",
        "resposta": "b"
    },
    "2": {
        "a": "Contratar candidatos externos",
        "b": "Promover funcionários atuais para novas posições",
        "c": "Realizar entrevistas com pessoas de fora da empresa",
        "pergunta": "O que é recrutamento interno?",
        "resposta": "b"
    },
    "3": {
        "a": "Aumento de salário",
        "b": "Rotatividade de funcionários",
        "c": "Recrutamento de novos colaboradores",
        "pergunta": "O que significa 'turnover' em RH?",
        "resposta": "b"
    },
    "4": {
        "a": "Ajudar na contratação de novos funcionários",
        "b": "Implementar e avaliar programas de treinamento",
        "c": "Monitorar o cumprimento de normas de segurança",
        "pergunta": "Qual é a principal função de um analista de treinamento e
desenvolvimento?",
        "resposta": "b"
    },
    "5": {
        "a": "Marketing voltado para consumidores",
        "b": "Estratégia para engajar os funcionários dentro da empresa",
        "c": "Aumentar as vendas por meio de campanhas internas",
        "pergunta": "O que significa 'endomarketing'?",
        "resposta": "b"
    },
    "6": {
```

```
      "a": "Reduz o tempo de contratação",
      "b": "Traz novas ideias e diversidade para a empresa",
      "c": "Evita a necessidade de treinamento",
      "pergunta": "Qual é a vantagem do recrutamento externo?",
      "resposta": "b"
   },
   "7": {
      "a": "Medir o nível de satisfação do cliente",
      "b": "Analisar o desempenho dos funcionários em relação às suas metas",
      "c": "Avaliar a qualidade dos produtos da empresa",
      "pergunta": "O que é a avaliação de desempenho?",
      "resposta": "b"
   },
   "8": {
      "a": "Lista de promoções automáticas para funcionários",
      "b": "Estrutura de progressão profissional dentro da empresa",
      "c": "Um curso oferecido para novos funcionários",
      "pergunta": "O que é plano de carreira?",
      "resposta": "b"
   },
   "9": {
      "a": "Que os funcionários recebam bônus",
      "b": "Que a empresa esteja em conformidade com as regulamentações trabalhistas",
      "c": "Que todos os funcionários tenham seguro de saúde",
      "pergunta": "O que o departamento de RH deve garantir em relação às leis
trabalhistas?",
      "resposta": "b"
   },
   "10": {
      "a": "Benefícios fixos para todos os funcionários",
      "b": "Benefícios que podem ser personalizados de acordo com as necessidades dos
funcionários",
      "c": "Um plano de cortes de benefícios para reduzir custos",
      "pergunta": "O que é uma política de benefícios flexíveis?",
      "resposta": "b"
   },
   "11": {
      "a": "Apenas comunicar a demissão ao funcionário",
      "b": "Oferecer suporte e realizar o desligamento de maneira correta e respeitosa",
      "c": "Realizar o pagamento imediato das verbas rescisórias",
      "pergunta": "Qual é a função do RH em uma demissão?",
      "resposta": "b"
   },
   "12": {
      "a": "Processo de alternância de turnos",
      "b": "Rotação de funcionários entre diferentes funções ou departamentos",
      "c": "Sistema de avaliação de desempenho",
      "pergunta": "O que é 'job rotation'?",
```

```
    "resposta": "b"
  },
  "13": {
    "a": "Que o novo colaborador conheça o ambiente e as políticas da empresa",
    "b": "Que o colaborador inicie imediatamente suas atividades sem treinamento",
    "c": "Que o colaborador participe de entrevistas com seus colegas",
    "pergunta": "O que o RH deve garantir na integração de novos colaboradores?",
    "resposta": "a"
  },
  "14": {
    "a": "Condições físicas do ambiente de trabalho",
    "b": "Percepção dos funcionários sobre o ambiente e cultura da empresa",
    "c": "Temperatura média nos escritórios",
    "pergunta": "O que é clima organizacional?",
    "resposta": "b"
  },
  "15": {
    "a": "Avaliação das competências técnicas do candidato",
    "b": "Avaliação de como o candidato reage em situações específicas",
    "c": "Perguntas sobre a vida pessoal do candidato",
    "pergunta": "O que é uma entrevista comportamental?",
    "resposta": "b"
  },
  "16": {
    "a": "Avaliar funcionários apenas por resultados numéricos",
    "b": "Focar no desenvolvimento das habilidades necessárias para o cargo",
    "c": "Treinar apenas os funcionários novos",
    "pergunta": "O que é gestão por competências?",
    "resposta": "b"
  },
  "17": {
    "a": "Habilidades técnicas adquiridas ao longo da carreira",
    "b": "Competências comportamentais e emocionais, como comunicação e trabalho em
equipe",
    "c": "Habilidades manuais específicas para o trabalho",
    "pergunta": "O que são soft skills?",
    "resposta": "b"
  },
  "18": {
    "a": "Aconselhamento sobre questões pessoais",
    "b": "Processo de orientação para desenvolver o potencial dos funcionários",
    "c": "Recrutamento de novos talentos",
    "pergunta": "O que significa 'coaching' no contexto de RH?",
    "resposta": "b"
  },
  "19": {
    "a": "Garantir que todos os funcionários tenham os mesmos benefícios",
```

```
        "b": "Promover políticas e práticas que incentivem a diversidade e inclusão na
empresa",
        "c": "Aumentar a quantidade de benefícios para minorias",
        "pergunta": "Qual é o papel do RH na diversidade e inclusão?",
        "resposta": "b"
    },
    "20": {
        "a": "Avaliação do funcionário feita apenas por seu supervisor direto",
        "b": "Avaliação feita pelo funcionário, colegas, subordinados e supervisores",
        "c": "Feedback dado em reuniões semanais",
        "pergunta": "O que é feedback 360 graus?",
        "resposta": "b"
    }
}

        self.__quizfin = {
    "1": {
        "a": "Planejar o crescimento da empresa",
        "b": "Controlar receitas e despesas",
        "c": "Realizar auditorias fiscais",
        "pergunta": "Qual é a principal função do setor financeiro?",
        "resposta": "b"
    },
    "2": {
        "a": "Fluxo de caixa",
        "b": "Orçamento de capital",
        "c": "Relatório de sustentabilidade",
        "pergunta": "Qual ferramenta financeira projeta entradas e saídas de dinheiro?",
        "resposta": "a"
    },
    "3": {
        "a": "Aumentar as vendas",
        "b": "Reduzir o custo dos produtos",
        "c": "Maximizar o valor para os acionistas",
        "pergunta": "Qual é o objetivo principal de uma empresa em termos financeiros?",
        "resposta": "c"
    },
    "4": {
        "a": "Investimento em ações",
        "b": "Controle do orçamento",
        "c": "Gestão do passivo",
        "pergunta": "Qual é uma das principais atividades da tesouraria em uma empresa?",
        "resposta": "a"
    },
    "5": {
        "a": "Lançamento de ações",
        "b": "Captação de recursos",
        "c": "Gerenciamento de caixa",
```

```
    "pergunta": "O que envolve a emissão de títulos de dívida corporativa?",
    "resposta": "b"
  },
  "6": {
    "a": "Dinheiro necessário para operações diárias",
    "b": "Recursos destinados à expansão",
    "c": "Reservas para pagamento de impostos",
    "pergunta": "O que significa 'capital de giro'?",
    "resposta": "a"
  },
  "7": {
    "a": "Registro de ações",
    "b": "Relatório financeiro anual",
    "c": "Demonstração do resultado",
    "pergunta": "Qual documento demonstra a lucratividade de uma empresa?",
    "resposta": "c"
  },
  "8": {
    "a": "Auditoria externa",
    "b": "Projeção de vendas",
    "c": "Previsão de fluxo de caixa",
    "pergunta": "Qual ferramenta ajuda a prever as necessidades futuras de caixa de uma
empresa?",
    "resposta": "c"
  },
  "9": {
    "a": "Depreciação",
    "b": "Amortização",
    "c": "Desvalorização cambial",
    "pergunta": "Qual termo descreve a redução do valor de um ativo ao longo do tempo?",
    "resposta": "a"
  },
  "10": {
    "a": "Análise SWOT",
    "b": "Análise de liquidez",
    "c": "Análise de alavancagem",
    "pergunta": "Qual métrica mede a capacidade de uma empresa de pagar suas dívidas
de curto prazo?",
    "resposta": "b"
  },
  "11": {
    "a": "Valor presente líquido",
    "b": "Taxa interna de retorno",
    "c": "Margem bruta",
    "pergunta": "Qual método é utilizado para avaliar o retorno de um investimento?",
    "resposta": "a"
  },
  "12": {
```

```
        "a": "Auditoria interna",
        "b": "Controle de estoques",
        "c": "Gestão de riscos",
        "pergunta": "Qual departamento é responsável pela prevenção de fraudes financeiras?",
        "resposta": "a"
    },
    "13": {
        "a": "Pagamentos a fornecedores",
        "b": "Provisão de garantias bancárias",
        "c": "Gestão de custos indiretos",
        "pergunta": "Qual é uma função básica do contas a pagar?",
        "resposta": "a"
    },
    "14": {
        "a": "Taxa de crescimento sustentável",
        "b": "Margem de lucro operacional",
        "c": "Faturamento bruto",
        "pergunta": "Qual indicador mede a eficiência da operação de uma empresa?",
        "resposta": "b"
    },
    "15": {
        "a": "Análise horizontal",
        "b": "Análise de cenários",
        "c": "Análise vertical",
        "pergunta": "Qual técnica compara itens de demonstrações financeiras ao longo do tempo?",
        "resposta": "a"
    },
    "16": {
        "a": "Déficit financeiro",
        "b": "Superávit de caixa",
        "c": "Endividamento total",
        "pergunta": "Qual termo indica quando as receitas excedem as despesas?",
        "resposta": "b"
    },
    "17": {
        "a": "Empréstimo bancário",
        "b": "Crédito rotativo",
        "c": "Captação de capital de terceiros",
        "pergunta": "Qual é uma forma de financiamento de curto prazo?",
        "resposta": "b"
    },
    "18": {
        "a": "Índice de solvência",
        "b": "Índice de liquidez corrente",
        "c": "Índice de giro do ativo",
```

```
        "pergunta": "Qual índice mede a capacidade de uma empresa em cumprir suas
obrigações de longo prazo?",
        "resposta": "a"
    },
    "19": {
        "a": "Distribuição de dividendos",
        "b": "Liquidação de dívidas",
        "c": "Captação de recursos via IPO",
        "pergunta": "O que ocorre quando uma empresa decide pagar seus acionistas?",
        "resposta": "a"
    },
    "20": {
        "a": "Capital de terceiros",
        "b": "Empréstimo interbancário",
        "c": "Reserva de lucros",
        "pergunta": "Qual termo refere-se ao dinheiro emprestado por uma empresa?",
        "resposta": "a"
    }
}
        self.__quizmark = {
    "1": {
        "a": "Divulgar produtos ou serviços",
        "b": "Criar embalagens de produtos",
        "c": "Gerenciar os recursos financeiros",
        "pergunta": "Qual é o principal objetivo do marketing?",
        "resposta": "a"
    },
    "2": {
        "a": "Aumentar a produção",
        "b": "Criar valor para o cliente",
        "c": "Desenvolver estratégias de vendas",
        "pergunta": "Qual é a função central de uma estratégia de marketing?",
        "resposta": "b"
    },
    "3": {
        "a": "Pesquisa de mercado",
        "b": "Auditoria financeira",
        "c": "Gestão de pessoas",
        "pergunta": "Qual ferramenta é usada para entender as necessidades do
consumidor?",
        "resposta": "a"
    },
    "4": {
        "a": "Propor melhorias no produto",
        "b": "Aumentar o engajamento nas redes sociais",
        "c": "Identificar oportunidades de mercado",
        "pergunta": "O que é o papel da pesquisa de mercado?",
        "resposta": "c"
```

```
},
"5": {
    "a": "Monitoramento do ROI de campanhas",
    "b": "Controle de estoque",
    "c": "Análise de custos de produção",
    "pergunta": "O que o marketing de desempenho visa medir?",
    "resposta": "a"
},
"6": {
    "a": "Analisar o mercado",
    "b": "Aumentar o número de funcionários",
    "c": "Gerenciar a cadeia de suprimentos",
    "pergunta": "Qual é uma responsabilidade do gerente de marketing?",
    "resposta": "a"
},
"7": {
    "a": "Estratégia de distribuição de produtos",
    "b": "Planejamento de marketing",
    "c": "Definição de metas financeiras",
    "pergunta": "Qual é a função do plano de marketing?",
    "resposta": "b"
},
"8": {
    "a": "Gestão de marca",
    "b": "Controle de qualidade",
    "c": "Gerenciamento de vendas",
    "pergunta": "Qual é a principal função do branding?",
    "resposta": "a"
},
"9": {
    "a": "Analisar a concorrência",
    "b": "Aumentar a produção",
    "c": "Gerar leads",
    "pergunta": "Qual é o objetivo da análise de concorrência?",
    "resposta": "a"
},
"10": {
    "a": "Criar uma imagem positiva da marca",
    "b": "Gerenciar a logística de produtos",
    "c": "Controlar a qualidade dos produtos",
    "pergunta": "Qual é o principal objetivo das relações públicas?",
    "resposta": "a"
},
"11": {
    "a": "Oferecer um desconto",
    "b": "Aumentar o engajamento e interações",
    "c": "Vender diretamente ao cliente",
    "pergunta": "Qual é o objetivo do marketing de conteúdo?",
```

```
        "resposta": "b"
    },
    "12": {
        "a": "Alinhar produto e preço",
        "b": "Definir preço baseado na demanda",
        "c": "Criar campanhas para fidelizar clientes",
        "pergunta": "Qual é a função da precificação baseada em valor?",
        "resposta": "b"
    },
    "13": {
        "a": "Reduzir custos operacionais",
        "b": "Promover o produto de forma criativa",
        "c": "Controlar a produção em massa",
        "pergunta": "Qual é o papel da propaganda no mix de marketing?",
        "resposta": "b"
    },
    "14": {
        "a": "Segmentar o público-alvo",
        "b": "Definir metas de crescimento",
        "c": "Gerenciar a distribuição do produto",
        "pergunta": "Qual é a primeira etapa em uma campanha de marketing?",
        "resposta": "a"
    },
    "15": {
        "a": "Maximizar o lucro",
        "b": "Satisfazer necessidades dos consumidores",
        "c": "Reduzir o tempo de produção",
        "pergunta": "Qual é o objetivo principal de uma estratégia de posicionamento de
marca?",
        "resposta": "b"
    },
    "16": {
        "a": "Aumentar as vendas",
        "b": "Promover a sustentabilidade",
        "c": "Gerar leads e oportunidades",
        "pergunta": "Qual é o papel do marketing digital?",
        "resposta": "c"
    },
    "17": {
        "a": "Fortalecer a reputação da marca",
        "b": "Reduzir custos de distribuição",
        "c": "Realizar auditorias periódicas",
        "pergunta": "Qual é o principal objetivo do marketing de influência?",
        "resposta": "a"
    },
    "18": {
        "a": "Monitorar as tendências de mercado",
        "b": "Reduzir custos de produção",
```

```
        "c": "Aumentar a produção interna",
        "pergunta": "Qual é uma função importante da análise de dados de marketing?",
        "resposta": "a"
    },
    "19": {
        "a": "Aumentar a percepção de valor",
        "b": "Criar novas oportunidades de emprego",
        "c": "Desenvolver novos produtos",
        "pergunta": "Qual é o objetivo do marketing emocional?",
        "resposta": "a"
    },
    "20": {
        "a": "Aumentar o volume de vendas",
        "b": "Entender as necessidades e desejos dos clientes",
        "c": "Distribuir produtos em novos mercados",
        "pergunta": "Qual é o principal foco do marketing centrado no cliente?",
        "resposta": "b"
    }
}
        self.__quizti = {
    "1": {
        "a": "Comprimir arquivos para economizar espaço",
        "b": "Estabelecer políticas de controle de acesso",
        "c": "Executar rotinas de backup automaticamente",
        "pergunta": "Qual é a função de um sistema de gestão de identidade e acesso (IAM)?",
        "resposta": "b"
    },
    "2": {
        "a": "Dividir a rede em sub-redes menores para melhorar a performance",
        "b": "Conectar dispositivos de rede em diferentes locais geográficos",
        "c": "Criar conexões seguras entre clientes e servidores",
        "pergunta": "Qual é a principal função do subnetting em redes de computadores?",
        "resposta": "a"
    },
    "3": {
        "a": "Identificar vulnerabilidades em software",
        "b": "Monitorar a integridade de arquivos no servidor",
        "c": "Verificar o tráfego de rede para padrões de ataque",
        "pergunta": "O que faz um sistema de detecção de intrusões (IDS)?",
        "resposta": "c"
    },
    "4": {
        "a": "Segurança física dos datacenters",
        "b": "Implementação de políticas de segurança de software",
        "c": "Segurança em ambientes virtualizados",
        "pergunta": "Qual é o maior desafio relacionado à segurança na computação em
nuvem?",
        "resposta": "c"
```

```
},
"5": {
    "a": "Reduzir o tempo de resposta do sistema",
    "b": "Otimizar o uso de hardware físico",
    "c": "Segregar aplicações para evitar interferência mútua",
    "pergunta": "Qual é o principal benefício da virtualização de servidores?",
    "resposta": "b"
},
"6": {
    "a": "Faz uma cópia idêntica dos dados em tempo real",
    "b": "Armazena os dados em diferentes locais geográficos",
    "c": "Garante a criptografia dos dados transmitidos pela rede",
    "pergunta": "O que é a replicação de dados em ambientes de T.I.?",
    "resposta": "a"
},
"7": {
    "a": "Utiliza várias instâncias de um sistema operacional",
    "b": "Permite que múltiplos usuários acessem a mesma aplicação simultaneamente",
    "c": "Distribui automaticamente a carga de trabalho entre servidores",
    "pergunta": "O que é balanceamento de carga em um ambiente de servidores?",
    "resposta": "c"
},
"8": {
    "a": "Controlar as permissões de arquivos em redes Linux",
    "b": "Gerenciar dispositivos de armazenamento em massa",
    "c": "Automatizar a instalação de software",
    "pergunta": "Para que serve o comando 'chmod' em sistemas Linux?",
    "resposta": "a"
},
"9": {
    "a": "Atualiza o kernel do sistema operacional",
    "b": "Protege contra falhas de hardware",
    "c": "Monitora o desempenho dos aplicativos em tempo real",
    "pergunta": "O que faz um sistema de alta disponibilidade (HA)?",
    "resposta": "b"
},
"10": {
    "a": "Encaminhar pacotes de dados entre redes diferentes",
    "b": "Controlar o fluxo de dados dentro de uma rede",
    "c": "Proteger a rede de ataques distribuídos",
    "pergunta": "Qual é a função de um roteador em uma rede?",
    "resposta": "a"
},
"11": {
    "a": "Analisar logs de sistemas para encontrar falhas",
    "b": "Atualizar drivers de dispositivos automaticamente",
    "c": "Aplicar patches de segurança em servidores",
    "pergunta": "Qual é a função do software de gerenciamento de vulnerabilidades?",
```

```
    "resposta": "c"
},
"12": {
    "a": "Oferecer suporte a interfaces de programação",
    "b": "Servir de intermediário para requisições entre cliente e servidor",
    "c": "Fornecer acesso remoto a aplicações",
    "pergunta": "O que é um servidor proxy reverso?",
    "resposta": "b"
},
"13": {
    "a": "Auditar transações financeiras em sistemas de ERP",
    "b": "Controlar o acesso a recursos compartilhados em rede",
    "c": "Monitorar o tráfego de rede em busca de padrões suspeitos",
    "pergunta": "Qual é o papel de um administrador de segurança da informação?",
    "resposta": "c"
},
"14": {
    "a": "Controlar o uso de recursos do sistema para evitar sobrecarga",
    "b": "Registrar atividades de usuários para fins de auditoria",
    "c": "Proteger sistemas contra malware e ataques de rede",
    "pergunta": "O que é um sistema de gerenciamento de logs?",
    "resposta": "b"
},
"15": {
    "a": "Prevenir acessos não autorizados a sistemas de armazenamento",
    "b": "Garantir que os dados estejam disponíveis em casos de falhas",
    "c": "Permitir que aplicações acessem bancos de dados de forma eficiente",
    "pergunta": "Qual é a função de um sistema de backup e recuperação?",
    "resposta": "b"
},
"16": {
    "a": "Gerenciar pacotes de dados em uma rede distribuída",
    "b": "Garantir a compatibilidade entre diferentes sistemas operacionais",
    "c": "Distribuir cópias de dados entre diversos servidores",
    "pergunta": "O que faz um sistema de replicação de dados?",
    "resposta": "c"
},
"17": {
    "a": "Monitorar a saúde dos sistemas e redes",
    "b": "Controlar a largura de banda utilizada pelos dispositivos",
    "c": "Registrar transações em tempo real para auditorias",
    "pergunta": "O que faz um sistema de monitoramento de rede?",
    "resposta": "a"
},
"18": {
    "a": "Controlar a comunicação entre os serviços e clientes de um aplicativo",
    "b": "Gerenciar permissões de arquivos no servidor",
    "c": "Criar ambientes de teste para aplicações em desenvolvimento",
```

```
        "pergunta": "Qual é a função de um load balancer em sistemas de TI?",
        "resposta": "a"
    },
    "19": {
        "a": "Proteger a rede contra ataques de negação de serviço (DDoS)",
        "b": "Evitar que dispositivos ultrapassem os limites de largura de banda",
        "c": "Gerenciar a distribuição de endereços IP dentro de uma rede",
        "pergunta": "O que faz um firewall de próxima geração (NGFW)?",
        "resposta": "a"
    },
    "20": {
        "a": "Reduzir a latência de redes distribuídas",
        "b": "Realizar a verificação de vulnerabilidades no código de aplicações",
        "c": "Fornecer acesso seguro a redes remotas",
        "pergunta": "Qual é o objetivo principal de uma VPN (Virtual Private Network)?",
        "resposta": "c"
    }
}
        self.__quizvendas = {
    "1": {
        "a": "Analisar a performance dos concorrentes",
        "b": "Identificar necessidades e desejos dos clientes",
        "c": "Desenvolver campanhas de marketing",
        "pergunta": "Qual é o principal objetivo da prospecção de vendas?",
        "resposta": "b"
    },
    "2": {
        "a": "Estratégia para fechar vendas rapidamente",
        "b": "Relação de longo prazo com o cliente",
        "c": "Focar em transações únicas e de alto valor",
        "pergunta": "O que caracteriza a abordagem de 'vendas consultivas'?",
        "resposta": "b"
    },
    "3": {
        "a": "Construir confiança com o cliente",
        "b": "Oferecer descontos imediatos para aumentar vendas",
        "c": "Encerrar rapidamente as negociações",
        "pergunta": "Qual é a principal vantagem da técnica de 'rapport' em vendas?",
        "resposta": "a"
    },
    "4": {
        "a": "Aumentar o ticket médio por cliente",
        "b": "Reduzir custos operacionais de venda",
        "c": "Focar apenas em novos clientes",
        "pergunta": "O que é 'upselling'?",
        "resposta": "a"
    },
    "5": {
```

```
    "a": "Facilitar a aquisição de novos clientes",
    "b": "Fidelizar os clientes existentes",
    "c": "Reduzir o ciclo de vendas",
    "pergunta": "Qual é o principal objetivo de uma estratégia de retenção de clientes?",
    "resposta": "b"
},
"6": {
    "a": "Oferecer produtos complementares",
    "b": "Realizar vendas cruzadas entre setores",
    "c": "Reduzir o ciclo de vendas",
    "pergunta": "O que significa 'cross-selling' em vendas?",
    "resposta": "a"
},
"7": {
    "a": "Oferecer várias opções para o cliente escolher",
    "b": "Personalizar a abordagem de vendas com base no comportamento do cliente",
    "c": "Utilizar sempre a mesma estratégia de vendas",
    "pergunta": "Qual é a importância da personalização no processo de vendas?",
    "resposta": "b"
},
"8": {
    "a": "Oferecer descontos para atrair novos clientes",
    "b": "Prever vendas futuras com base em dados históricos",
    "c": "Focar em vendas de curto prazo",
    "pergunta": "O que é 'previsão de vendas'?",
    "resposta": "b"
},
"9": {
    "a": "Clientes que compram esporadicamente",
    "b": "Clientes que recomendam a marca para outras pessoas",
    "c": "Clientes que apenas usam cupons de desconto",
    "pergunta": "O que são 'clientes promotores'?",
    "resposta": "b"
},
"10": {
    "a": "Negociação em uma única reunião",
    "b": "Relacionamento contínuo com base na confiança",
    "c": "Fechamento de venda focado no produto",
    "pergunta": "O que caracteriza a venda baseada em relacionamento?",
    "resposta": "b"
},
"11": {
    "a": "Oferecer mais produtos durante o atendimento ao cliente",
    "b": "Estabelecer metas de curto prazo para fechar vendas",
    "c": "Concentrar-se em vendas imediatas ao invés de fidelização",
    "pergunta": "Qual é o foco de uma estratégia de 'venda agressiva'?",
    "resposta": "c"
},
```

```
"12": {
    "a": "Registrar dados de clientes para campanhas futuras",
    "b": "Estabelecer parâmetros para o atendimento ao cliente",
    "c": "Focar em prospectos que já demonstraram interesse",
    "pergunta": "O que é 'lead scoring' no processo de vendas?",
    "resposta": "c"
},
"13": {
    "a": "Proporcionar uma experiência única ao cliente",
    "b": "Manter contato regular para novas oportunidades de venda",
    "c": "Finalizar a venda e encerrar o contato com o cliente",
    "pergunta": "Qual é o papel do 'follow-up' no processo de vendas?",
    "resposta": "b"
},
"14": {
    "a": "Desenvolver argumentos de venda baseados em dados concretos",
    "b": "Oferecer descontos progressivos com o aumento do volume",
    "c": "Diminuir o tempo entre o primeiro contato e a venda",
    "pergunta": "O que caracteriza a técnica de 'proposta de valor'?",
    "resposta": "a"
},
"15": {
    "a": "Segmentar o público de acordo com o comportamento de compra",
    "b": "Oferecer um produto mais caro do que o inicialmente desejado",
    "c": "Estender o ciclo de vendas para maiores negociações",
    "pergunta": "Qual é o principal foco da técnica de 'qualificação de leads'?",
    "resposta": "a"
},
"16": {
    "a": "Concentrar-se em resolver uma objeção do cliente",
    "b": "Oferecer um desconto imediato para finalizar a venda",
    "c": "Fechar a venda mesmo sem resolver todas as objeções",
    "pergunta": "Qual é o objetivo principal de lidar com objeções em vendas?",
    "resposta": "a"
},
"17": {
    "a": "Vender com base nas características técnicas do produto",
    "b": "Focar nos benefícios e valor que o produto traz ao cliente",
    "c": "Vender apenas para clientes que procuram preços baixos",
    "pergunta": "O que é uma abordagem de venda focada em benefícios?",
    "resposta": "b"
},
"18": {
    "a": "Segmentação demográfica e geográfica de clientes",
    "b": "Fechamento de venda baseado em demonstrações técnicas",
    "c": "Uso de dados e métricas para otimizar a performance de vendas",
    "pergunta": "O que caracteriza a 'venda baseada em dados'?",
    "resposta": "c"
```

```
    },
    "19": {
        "a": "Construção de uma relação de confiança antes de propor a venda",
        "b": "Fornecer múltiplas alternativas de produtos para o cliente escolher",
        "c": "Oferecer soluções sem customização ao cliente",
        "pergunta": "Qual é o princípio central da 'venda consultiva'?",
        "resposta": "a"
    },
    "20": {
        "a": "Foco na retenção de clientes antigos",
        "b": "Abertura de novos mercados e segmentos",
        "c": "Aumentar o volume de vendas por cliente",
        "pergunta": "Qual é o principal objetivo de uma estratégia de 'market share'?",
        "resposta": "b"
    }
}

        self.__quizView = QuizView()
        self.__cursoView = CursoView()

    def realizar_quiz(self):
        self.__cursoView.mostra_cursos(self.__cursoController.cursos)
        nome_curso = self.__cursoView.seleciona_curso()
        for curso in self.__cursoController.cursos:
            if curso.nome == nome_curso:
                setor = curso.setor
                dificuldade = curso.dificuldade

        if setor == "RH":
            pontos = self.__quizView.comeca_quiz(dificuldade, setor, self.__quizrh)


        elif setor == "Financeiro":
            pontos = self.__quizView.comeca_quiz(dificuldade, setor, self.__quizfin)

        elif setor == "Marketing":
            pontos = self.__quizView.comeca_quiz(dificuldade, setor, self.__quizmark)


        elif setor == "T.I":
            pontos = self.__quizView.comeca_quiz(dificuldade, setor, self.__quizti)


        elif setor == "Vendas":
            pontos = self.__quizView.comeca_quiz(dificuldade, setor, self.__quizvendas)

    import os
import random
```

```
class QuizView():

    def mostra_mensagem(self, mensagem):
        print(mensagem)

    def limpar_terminal(self):
        # Limpa o terminal dependendo do sistema operacional
        os.system('cls' if os.name == 'nt' else 'clear')

    def comeca_quiz(self, dificuldade, setor, quiz):
        self.limpar_terminal()
        print(f"Bem-vindo ao quiz de {setor}! ##É necessário gabaritar para ganhar
experiência## \n Responda as perguntas abaixo:\n")


        pontos = 0
        perguntas = []
        while len(perguntas) != int(dificuldade):
            numero = random.randint(1,20)
            if numero not in perguntas:
                perguntas.append(numero)


        for i in range(len(perguntas)):
            pergunta = quiz[str(perguntas[i])]["pergunta"]
            a = quiz[str(perguntas[i])]["a"]
            b = quiz[str(perguntas[i])]["b"]
            c = quiz[str(perguntas[i])]["c"]
            resposta_correta = quiz[str(perguntas[i])]["resposta"]


            print(f"{(perguntas[i])}. {pergunta}")
            print()
            print(f"a) {a}")
            print(f"b) {b}")
            print(f"c) {c}")


            resposta_usuario = input("Digite a sua resposta (a, b ou c): ").lower()


            if resposta_usuario == resposta_correta:
                print("Resposta correta!\n")
                print()
                pontos += 1  # Incrementar a pontuação
            else:
                print(f"Resposta incorreta. A resposta correta era '{resposta_correta}'.\n")
                print()
```

```
        if pontos == len(perguntas):
            print("Parabéns, você gabaritou o quiz! \n")

        return pontos



    from personagem import Personagem

class Ranking(Personagem):
    def __init__(self, tipo, id, personagem:Personagem):
        self.__tipo = tipo
        self.__id = id
        self.__personagens = list[personagem] ## sabemos que não é assim!


    @property
    def tipo(self):
        return self.__tipo

    @tipo.setter
    def tipo(self, tipo):
        self.__tipo = tipo

    @property
    def id(self):
        return self.__id

    @id.setter
    def id(self, id):
        self.__id = id

    @property
    def personagens(self):
        return self.__personagens

    @personagens.setter
    def personagens(self, personagens):
        self.__personagens = personagens

    def adicionar_personagem(self, personagem):
        self.__personagens.append(personagem)

    def remover_personagem(self, personagem):
        self.__personagens.remove(personagem)

    def ranking_nivel(self):
```

```
        pass

    def ranking_dungeons(self):
        pass

    def ranking_cursos(self):
        pass

    class Setor:
    def __init__(self, nome, dificuldade):
        self._nome = nome
        self._dificuldade = dificuldade

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

    @nome.setter
    def nome(self, nome):
        self._nome = nome

    @property
    def dificuldade(self):
        return self._dificuldade

    @dificuldade.setter
    def dificuldade(self, dificuldade):
        self._dificuldade = dificuldade

        from setorView import SetorView
from setor import Setor

class SetorController():
    def __init__(self):
        self.__setorView = SetorView()



    def adicionar_setor(self):
        dados_setor = self.__setorView.pega_dados_setor()
        setor = Setor(dados_setor["nome"], dados_setor["dificuldade"])
        return setor



    def calcular_media_dificuldades(self):
        dificuldade = 0
        dificuldade += [setor["dificuldade"] for setor in self.setores.values()]
```

```
class SetorView():

    def pega_dados_setor(self):
        print("--------DADOS DO SETOR--------")
        nome = input("Digite o nome do setor: ")
        dificuldade = input("Qual a dificuldade do setor: ")
        return {"nome": nome, "dificuldade": dificuldade}

class Sistema:
    def __init__(self):
        self.__personagens = []

    @property
    def personagens(self):
        return self.__personagens

    def adicionar_personagem(self, personagem):
        self.__personagens.append(personagem)

    def listar_personagens(self):
        return self.__personagens

import json
import time
import os
from sistema import Sistema
from sistemaView import SistemaView
from personagemController import PersonagemController
from cursoController import CursoController
from quizController import QuizController
from bossController import BossController
from dungeonController import DungeonController
from batalhaController import BatalhaController
from setorController import SetorController


class SistemaControllerr:
    def __init__(self):
        self.__sistema = Sistema()
        self.__sistemaView = SistemaView()
        self.__personagemController = PersonagemController()
        self.__cursoController = CursoController()
        self.__quizController = QuizController()
        self.__bossController = BossController()
        self.__setorController = SetorController()
        self.__dungeonController = DungeonController()
        self.__batalhaController = BatalhaController(self)
        self.__quizController._QuizController__cursoController = self.__cursoController
```

```
    self.__arquivo_personagens = "personagens.json"
    self.carregar_personagens()


@property
def cursoController(self):
    return self.__cursoController

@property
def quizController(self):
    return self.__quizController

@property
def bossController(self):
    return self.__bossController

@property
def setorController(self):
    return self.__setorController

@property
def dungeonController(self):
    return self.__dungeonController

@property
def batalhaController(self):
    return self.__batalhaController

def limpar_terminal(self):
    os.system('cls' if os.name == 'nt' else 'clear')

def salvar_personagens(self):
    personagens_salvar = []
    for personagem in self.__personagemController.personagens:
        personagens_salvar.append({
            'nome': personagem.nome,
            'nivel': personagem.nivel,
            'experiencia': personagem.experiencia,
            'classe': personagem.classe_personagem.nome_classe,
            'pontos_disponiveis': personagem.pontos_disponiveis,
            'pocoes_hp': personagem.pocao_hp.quant,
            'pocoes_est': personagem.pocao_est.quant,
            'atributos': personagem.classe_personagem.atributos
        })

    with open(self.__arquivo_personagens, 'w') as arquivo:
        json.dump(personagens_salvar, arquivo, indent=4)
```

```
        self.__sistemaView.mostrar_mensagem("Personagens salvos com sucesso!")
        time.sleep(2)

    def carregar_personagens(self):
        try:
            with open(self.__arquivo_personagens, 'r') as arquivo:
                personagens_carregados = json.load(arquivo)
                for dados_personagem in personagens_carregados:
                    personagem = self.__personagemController.cadastrar_personagem(
                        nome=dados_personagem['nome'],
                        nivel=dados_personagem['nivel'],
                        experiencia=dados_personagem['experiencia'],
                        nome_classe=dados_personagem['classe']
                    )
                    personagem.pocao_hp.quant = dados_personagem['pocoes_hp']
                    personagem.pocao_est.quant = dados_personagem['pocoes_est']
                    personagem.pontos_disponiveis = dados_personagem['pontos_disponiveis']

personagem.classe_personagem.atributos.update(dados_personagem['atributos'])
                self.__sistemaView.mostrar_mensagem(f"{len(personagens_carregados)}
personagens carregados com sucesso!")
                time.sleep(2)
        except FileNotFoundError:
            self.__sistemaView.mostrar_mensagem("Nenhum arquivo de personagens
encontrado. Iniciando sistema sem personagens.")
            time.sleep(2)

    def iniciar(self):
        self.limpar_terminal()
        while True:
            self.__sistemaView.menu_inicial()
            opcao = self.__sistemaView.pegar_opcao()

            if opcao == '1':
                self.menu_personagem()

            elif opcao == '2':
                self.__dungeonController.cadastrar_dungeon()
                self.iniciar()

            elif opcao == '3':
                self.salvar_personagens()
                self.__sistemaView.mostrar_mensagem("Saindo do sistema...")
                time.sleep(1)
                break

    def menu_principal(self):
        self.limpar_terminal()
```

```
    while True:
        self.__sistemaView.menu_principal()
        opcao = self.__sistemaView.pegar_opcao()

        if opcao == '1':
            self.menu_curso()

        elif opcao == '2':
            self.quiz_controller.realizar_quiz()

        elif opcao == '3':
            self.menu_personagem()

        elif opcao == '4':
            self.menu_dungeons()

        elif opcao == '5':
            self.menu_bosses()

        elif opcao == '6':
            pass

        elif opcao == '7':
            exit()

def menu_personagem(self):
    self.limpar_terminal()
    while True:
        self.__sistemaView.menu_personagem()
        opcao = self.__sistemaView.pegar_opcao()

        if opcao == '1':
            self.cadastrar_personagem()
        elif opcao == '2':
            personagem = self.selecionar_personagem()
            if personagem:
                self.mostrar_status(personagem)
                self.opcoes_personagem(personagem)
        elif opcao == '0':
            self.salvar_personagens()
            self.__sistemaView.mostrar_mensagem("Saindo do sistema...")
            time.sleep(1)
            break
        else:
            self.__sistemaView.mostrar_mensagem("Opção inválida. Tente novamente.")
            time.sleep(1)

def cadastrar_personagem(self):
```

```
        dados_personagem = None
        while dados_personagem is None:
            try:
                dados_personagem = self.__sistemaView.pega_dados_personagem()
                if not dados_personagem['classe']:
                    raise ValueError("Classe inválida!")
                for personagem in self.__sistema.listar_personagens():
                    if personagem.nome == dados_personagem["nome"]:
                        raise ValueError(f"O personagem {dados_personagem['nome']} já existe!")
            except ValueError as e:
                self.__sistemaView.mostrar_mensagem(str(e))
                dados_personagem = None

        personagem = self.__personagemController.cadastrar_personagem(
            nome=dados_personagem["nome"],
            nivel=dados_personagem["nivel"],
            experiencia=dados_personagem["experiencia"],
            nome_classe=dados_personagem["classe"]
        )
        self.__sistema.adicionar_personagem(personagem)
        self.__sistemaView.mostrar_mensagem(f"Personagem {dados_personagem['nome']}
da classe {dados_personagem['classe']} cadastrado com sucesso!")

    def selecionar_personagem(self):
        personagens = self.__personagemController.personagens
        self.__sistemaView.mostrar_personagens(personagens)

        escolha = self.__sistemaView.pegar_personagem_selecionado()
        if escolha.isdigit() and 1 <= int(escolha) <= len(personagens):
            personagem = personagens[int(escolha) - 1]
            return personagem
        else:
            self.__sistemaView.mostrar_mensagem("Escolha inválida.")
            time.sleep(2)
            return None

    def mostrar_status(self, personagem):
        self.__personagemController.mostrar_status(personagem)


    def opcoes_personagem(self, personagem):
        while True:
            self.__sistemaView.mostrar_opcoes_personagem()
            opcao = self.__sistemaView.pegar_opcao()

            if opcao == '1':
                self.mostrar_status(personagem)
            elif opcao == '2':
```

```
            self.__personagemController.upar_atributos(personagem)
        elif opcao == '3':
            self.__personagemController.usar_item(personagem)
        elif opcao == '4':
            self.__personagemController.upar_nivel(personagem)
        elif opcao == '0':
            break
        else:
            self.__sistemaView.mostrar_mensagem("Opção inválida. Tente novamente.")
            time.sleep(2)

    def menu_curso(self):
        self.limpar_terminal()
        while True:
            opcao = input("\nMenu de Cursos:\n1. Cadastrar Curso\n2. Alterar Curso\n3. Excluir
Curso\n4. Voltar\nEscolha uma opção: ")

            if opcao == '1':
                self.__cursoController.cadastrar_curso()
            elif opcao == '2':
                self.__cursoController.alterar_curso()
            elif opcao == '3':
                self.__cursoController.excluir_curso()
            elif opcao == '4':
                self.menu_principal()

    def menu_bosses(self):
        self.limpar_terminal()
        while True:
            opcao = input("\nMenu de Bosses:\n1. Cadastrar Boss\n2. Voltar\nEscolha uma
opção: ")

            if opcao == '1':
                self.__bossController.cadastrar_boss()

            elif opcao == '2':
                self.menu_principal()

    def menu_dungeons(self):
        self.limpar_terminal()
        while True:
            opcao = input("\nMenu de Dungeons:\n1. Cadastrar Dungeon\n2. Ver Dungeons\n3.
Voltar\nEscolha uma opção: ")

            if opcao == '1':
                self.__dungeonController.cadastrar_dungeon()

            elif opcao == '2':
```

```
            self.__dungeonController.ver_dungeons()  # Implementar ainda

        elif opcao == '3':
            self.menu_principal()

from cursoController import CursoController
from quizController import QuizController
from bossController import BossController
from dungeonController import DungeonController
from setorController import SetorController
from personagemController import PersonagemController
from batalhaController import BatalhaController
import os



class SistemaController:
    def __init__(self):
        self.curso_controller = CursoController()
        self.quiz_controller = QuizController()
        self.personagem_controller = PersonagemController()
        self.boss_controller = BossController()
        self.setor_controller = SetorController()
        self.dungeon_controller = DungeonController()
        self.batalha_controller = BatalhaController()
        self.quiz_controller._QuizController__cursoController = self.curso_controller

    def limpar_terminal(self):
        # Limpa o terminal dependendo do sistema operacional
        os.system('cls' if os.name == 'nt' else 'clear')


    def iniciar(self):
        self.limpar_terminal()
        while True:
            opcao = input("\nBem vindo ao RPG do Mercado de Trabalho!!\n Você quer
cadastrar um personagem ou uma empresa?\n1. Cadastrar Personagem\n2. Cadastrar
Empresa\n3. Sair\nEscolha uma opção: ")

            if opcao == '1':
                self.personagem_controller.cadastrar_personagem()
                self.menu_principal()

            elif opcao == '2':
                self.dungeon_controller.cadastrar_dungeon()
                self.iniciar()

            elif opcao == '3':
```

```
                break

    def menu_principal(self):
        self.limpar_terminal()

        while True:
            opcao = input("\nMenu Principal:\n1. Gerenciar Cursos\n2. Realizar Quiz\n3. Gerenciar Personagens\n4. Gerenciar Dungeons\n5. Gerenciar Bosses\n6. Batalhar\n7. Sair\nEscolha uma opção: ")

            if opcao == '1':
                self.menu_curso()

            elif opcao == '2':
                self.quiz_controller.realizar_quiz()

            elif opcao == '3':
                self.menu_personagem()

            elif opcao == '4':
                self.menu_dungeons()

            elif opcao == '5':
                self.menu_bosses()

            elif opcao == '6':
                pass

            elif opcao == '7':
                exit()




    def menu_curso(self):
        self.limpar_terminal()
        while True:
            opcao = input("\nMenu de Cursos:\n1. Cadastrar Curso\n2. Alterar Curso\n3. Excluir Curso\n4. Voltar\nEscolha uma opção: ")

            if opcao == '1':
                self.curso_controller.cadastrar_curso()
            elif opcao == '2':
                self.curso_controller.alterar_curso()
            elif opcao == '3':
                self.curso_controller.excluir_curso()
```

```
            elif opcao == '4':
                self.menu_principal()

    def menu_personagem(self):
        self.limpar_terminal()
        while True:
            opcao = input("\nMenu de Personagens:\n1. Cadastrar Personagem\n2. Ver
Status\n3. Usar Item\n4. Voltar\nEscolha uma opção: ")

            if opcao == '1':
                self.personagem_controller.cadastrar_personagem()

            elif opcao == '2':
                self.personagem_controller.mostrar_status()

            elif opcao == '3':
                self.personagem_controller.usar_item()

            elif opcao == '4':
                self.menu_principal()


    def menu_dungeons(self):
        self.limpar_terminal()
        while True:
            opcao = input("\nMenu de Dungeons:\n1. Cadastrar Dungeon\n2. Ver Dungeons\n3.
Voltar\nEscolha uma opção: ")

            if opcao == '1':
                self.dungeon_controller.cadastrar_dungeon()

            elif opcao == '2':
                self.dungeon_controller.ver_dungeons() ##Implementar ainda

            elif opcao == '3':
                self.menu_principal()


    def menu_bosses(self):
        self.limpar_terminal()
        while True:
            opcao = input("\nMenu de Bosses:\n1. Cadastrar Boss\n2. Voltar\nEscolha uma
opção: ")

            if opcao == '1':
                self.boss_controller.cadastrar_boss()

            elif opcao == '2':
```

```
            self.menu_principal()

import os
import time
class SistemaView:
    def menu_inicial(self):
        print("##########################################")
        print("Bem vindo ao RPG do Mercado de Trabalho!!")
        print("##########################################")
        print("")
        print("Você quer ser um personagem ou uma empresa?")
        print("1 - Personagem")
        print("2 - Empresa")
        print("3 - Nah, sair")
        print("")

    def menu_principal(self):
        print("----------MENU PRINCIPAL---------")
        print("Bem vindo ao RPG do Mercado de Trabalho!!")
        print("1. Gerenciar Cursos")
        print("2. Realizar Quiz")
        print("3. Gerenciar Personagens")
        print("4. Gerenciar Dungeons")
        print("5. Gerenciar Bosses")
        print("6. Batalhar")
        print("7. Sair")
        print("")

    def menu_personagem(self):
        os.system('cls' if os.name == 'nt' else 'clear')
        print("--------- MENU PERSONAGEM ---------")
        print("1 - Cadastrar Personagem")
        print("2 - Selecionar Personagem")
        print("0 - Sair")

    def mostrar_personagens(self, personagens):
        os.system('cls' if os.name == 'nt' else 'clear')
        if not personagens:
            print("################################")
            print(" Nenhum personagem cadastrado!")
            print("################################")
            time.sleep(2)
            return
        print("--------- PERSONAGENS CADASTRADOS ---------")
        print("Selecione um personagem:")
        for idx, personagem in enumerate(personagens, start=1):
            print(f"{idx} - {personagem.nome} - Nível: {personagem.nivel} - Classe:
{personagem.classe_personagem.nome_classe}")
```

```
def pegar_personagem_selecionado(self):
    return input("Digite o número do personagem que deseja selecionar: ").strip()

def pegar_opcao(self):
    return input("Escolha uma opção: ").strip()

def pega_dados_personagem(self):
    print("----------CADASTRO PERSONAGEM---------")
    nome = input("Nome: ")
    if not isinstance(nome, str):
        raise Exception("Nome inválido")

    while True:
        print("-------- CLASSES ----------")
        print("Escolha uma classe:")
        print("1 - CLT (Bom no early game)")
        print("2 - Estagiário (Médio no early, bom no late)")
        print("3 - Trainee (Fraco no early, muito forte no late)")

        opcao = int(input("Digite o número da classe: "))

        if opcao == 1:
            classe = "CLT"
        elif opcao == 2:
            classe = "Estagiario"
        elif opcao == 3:
            classe = "Trainee"
        else:
            print("Classe inválida! Tente novamente.")

            continue

        break

    return {"nome": nome,
            "classe": classe,
            "nivel": 1,
            "experiencia": 0
            }

def mostrar_opcoes_personagem(self):
    print("\n--------- OPÇÕES DO PERSONAGEM ---------")
    print("1 - Mostrar Status")
    print("2 - Aumentar Atributo")
    print("3 - Usar Item")
    print("4 - Upar Nível")
    print("0 - Voltar ao menu principal")
```

```
def mostrar_status(self, status):
    print("-------- STATUS ----------")
    for key, value in status.items():
        print(f"{key.capitalize()}: {value}")

def mostrar_mensagem(self, msg):
    os.system('cls' if os.name == 'nt' else 'clear')
    print("****************************************")
    print(msg)
    print("****************************************")
```