1) Questão 1 - Sistema de Classificação e Gerenciamento de Estoque de Grãos (CRUD)

Você está desenvolvendo um sistema para classificar e gerenciar o estoque de diferentes tipos de grãos em um armazém. Este sistema deve permitir:

1. Representar diferentes tipos de grãos (milho, soja, trigo).

2. Definir características comuns a todos os grãos (nome, lote, quantidade).

3. Permitir características específicas para certos tipos de grãos (umidade para milho, teor de proteína para soja, tipo para trigo).

4. Implementar operações básicas de CRUD (Criar, Ler, Atualizar, Deletar) para os grãos no estoque.

5. Garantir que informações básicas do grão (nome e lote) sejam sempre válidas (não vazios) e a quantidade seja não negativa.

6. Demonstrar o uso de herança para compartilhar características comuns entre os grãos.

7. Aplicar polimorfismo para exibir informações específicas de cada tipo de grão.

8. Implementar tratamento de exceções para lidar com a criação de grãos com dados inválidos e operações inválidas.

Tarefas:

1. Crie uma **classe abstrata** chamada Grao com os atributos protegidos _nome (string), _lote (string) e _quantidade (float ou inteiro). O construtor desta classe deve receber o nome, o lote e a quantidade, e levantar uma exceção do tipo ValueError caso os dados fornecidos sejam inválidos (nome ou lote vazios, quantidade negativa). Defina métodos para acessar os atributos (get_nome, get_lote, get_quantidade) e um método para modificar a quantidade (set_quantidade). Declare um método **abstrato** chamado classificar_qualidade que retorne uma string com a classificação do grão. Implemente um método exibir_info que imprima o nome, o lote e a quantidade do grão.

Crie três **classes concretas** que herdem de Grao:

Milho: Adicione um atributo protegido _umidade (float) com validação no construtor (deve estar entre 0 e 1). Implemente o método classificar_qualidade baseado no valor de _umidade (ex: abaixo de 0.1 bom, entre 0.1 e 0.15 regular, acima ruim). Sobrescreva o método exibir_info para incluir a umidade.

Soja: Adicione um atributo protegido _teor_proteina (float) com validação no construtor (deve ser positivo). Implemente o método classificar_qualidade baseado no _teor_proteina (ex: acima de 0.4 alto, entre 0.3 e 0.4 médio, abaixo baixo). Sobrescreva o método exibir_info para incluir o teor de proteína.

Trigo: Adicione um atributo protegido _tipo (string) com validação no construtor (não pode ser vazio). Implemente o método classificar_qualidade baseado no _tipo (ex: "Mole", "Duro", "Semiduro"). Sobrescreva o método exibir_info para incluir o tipo.

adicionar_grao(grao): Adiciona um objeto Grao ao estoque.

visualizar_estoque(): Exibe as informações de todos os grãos no estoque, incluindo a classificação de qualidade. Se o estoque estiver vazio, levante uma exceção personalizada EstoqueVazioError.

buscar_grao(lote): Busca um grão no estoque pelo seu lote. Retorna o objeto Grao se encontrado, ou None caso contrário.

atualizar_quantidade(lote, nova_quantidade): Busca um grão pelo lote e atualiza sua quantidade. Se o grão não for encontrado, levante uma exceção personalizada GraoNaoEncontradoError. Se a nova quantidade for negativa, levante um ValueError.

remover_grao(lote): Busca um grão pelo lote e remove-o do estoque. Se o grão não for encontrado, levante uma exceção personalizada GraoNaoEncontradoError.

1. Crie uma classe chamada ArmazemDeGraos para gerenciar o estoque. Esta classe deve ter um atributo privado _estoque (uma lista ou dicionário para armazenar os objetos Grao). Implemente os seguintes métodos:

2. Crie duas classes de exceção personalizadas que herdem de Exception: EstoqueVazioError e GraoNaoEncontradoError.

3. No bloco principal do seu código, crie uma instância de ArmazemDeGraos. Tente adicionar diferentes tipos de grãos ao armazém (com dados válidos e inválidos, demonstrando o tratamento de ValueError). Tente visualizar o estoque (tratando a exceção EstoqueVazioError). Busque, atualize a quantidade e remova grãos do estoque (tratando a exceção GraoNaoEncontradoError). O Tratamento de Exceção deve ser realizado nas classes!

Exemplos de Saída:

Adicionando grãos ao armazém...

Erro ao criar grão: O nome não pode estar vazio.

Erro ao criar grão: O lote não pode estar vazio.

Erro ao criar grão: A quantidade não pode ser negativa.

Erro ao criar grão Milho: A umidade deve estar entre 0 e 1.

Erro ao criar grão Soja: O teor de proteína deve ser positivo.

Erro ao criar grão Trigo: O tipo não pode estar vazio.

--- Estoque Inicial ---

Nome: Milho Variedade A, Lote: 2025-M1, Quantidade: 150.0 kg, Umidade: 0.08, Qualidade: Bom

Nome: Soja Orgânica, Lote: 2025-S1, Quantidade: 200.0 kg, Teor de Proteína: 0.42, Qualidade: Alto

Nome: Trigo Comum, Lote: 2025-T1, Quantidade: 300.0 kg, Tipo: Mole, Qualidade: Mole

Buscando grão com lote 2025-S1...

Grão encontrado: Nome: Soja Orgânica, Lote: 2025-S1, Quantidade: 200.0 kg, Teor de Proteína: 0.42, Qualidade: Alto

Atualizando quantidade do lote 2025-M1 para 180.0 kg...

Quantidade atualizada com sucesso.

--- Estoque Após Atualização ---

Nome: Milho Variedade A, Lote: 2025-M1, Quantidade: 180.0 kg, Umidade: 0.08, Qualidade: Bom

Nome: Soja Orgânica, Lote: 2025-S1, Quantidade: 200.0 kg, Teor de Proteína: 0.42, Qualidade: Alto

Nome: Trigo Comum, Lote: 2025-T1, Quantidade: 300.0 kg, Tipo: Mole, Qualidade: Mole

Removendo grão com lote 2025-T1...

Grão removido com sucesso.

--- Estoque Final ---

Nome: Milho Variedade A, Lote: 2025-M1, Quantidade: 180.0 kg, Umidade: 0.08, Qualidade: Bom

Nome: Soja Orgânica, Lote: 2025-S1, Quantidade: 200.0 kg, Teor de Proteína: 0.42, Qualidade: Alto

Tentando remover grão inexistente...

Erro: Grão com lote '2025-X1' não encontrado no estoque.

Tentando visualizar estoque vazio...

Erro: O estoque de grãos está vazio.

2) Questão 2 – Modelagem de Corpos Celestes

Você está desenvolvendo um sistema para modelar diferentes tipos de corpos celestes no universo. Este sistema deve:

1. Representar diferentes tipos de corpos celestes (planetas, estrelas, satélites naturais, supernovas, buracos negros, cometas).

2. Definir características comuns a todos os corpos celestes (nome, massa).

3. Permitir características específicas para certos tipos de corpos celestes (diâmetro para planetas, tipo espectral e luminosidade para estrelas, corpo orbitado para satélites, magnitude absoluta para supernovas, raio de Schwarzschild para buracos negros, período orbital para cometas).

4. Implementar ações como orbitar (para planetas, satélites e cometas), emitir luz (para estrelas e supernovas), e colapsar (para supernovas formando buracos negros).

5. Garantir que informações básicas do corpo celeste (nome) seja sempre válida (não vazio) e a massa seja positiva.

6. Demonstrar o uso de herança em múltiplos níveis para compartilhar características e especializar comportamentos.

7. Aplicar polimorfismo para exibir informações específicas de cada tipo de corpo celeste e simular suas ações.

8. Implementar tratamento de exceções para lidar com a criação de corpos celestes com dados inválidos e tentativas de orbitar/interagir de forma inválida.

Tarefas:

1. Crie uma **classe abstrata** chamada CorpoCeleste com os atributos públicos nome (string) e massa (float). O construtor desta classe deve receber o nome e a massa, e levantar uma exceção do tipo ValueError caso o nome seja vazio ou a massa não seja positiva. Declare um método **abstrato** chamado exibir_info.

2. Crie uma **classe concreta** Planeta que herda de CorpoCeleste. Adicione um atributo público diametro (float) com validação no construtor (deve ser positivo). Implemente o método exibir_info para mostrar o nome, a massa e o diâmetro. Crie um método orbitar(corpo) que recebe outro CorpoCeleste como argumento e retorna uma string indicando que o planeta está orbitando o corpo fornecido.

3. Crie uma **classe abstrata** Estelar que herda de CorpoCeleste. Adicione um atributo público luminosidade (float) com validação no construtor (deve ser positivo). Declare um método abstrato emitir_luz.

4. Crie uma **classe concreta** Estrela que herda de Estelar. Adicione um atributo público tipo_espectral (string) com validação no construtor (não pode ser vazio). Implemente o

método exibir_info para mostrar o nome, a massa, a luminosidade e o tipo espectral. Implemente o método emitir_luz retornando uma string indicando que a estrela está emitindo luz com base na sua luminosidade e tipo espectral.

1. Crie uma **classe concreta** Supernova que herda de Estelar. Adicione um atributo público magnitude_absoluta (float). Implemente o método exibir_info para mostrar o nome, a massa, a luminosidade e a magnitude absoluta. Implemente o método emitir_luz retornando uma string indicando a intensa emissão de luz da supernova. Adicione um método colapsar() que retorna uma string indicando que a supernova colapsou.

2. Crie uma **classe concreta** BuracoNegro que herda de CorpoCeleste. Adicione um atributo público raio_schwarzschild (float) com validação no construtor (deve ser positivo). Implemente o método exibir_info para mostrar o nome, a massa e o raio de Schwarzschild.

3. Crie uma **classe concreta** SateliteNatural que herda de CorpoCeleste e implementa uma interface Orbitavel (com método orbitar). Adicione um atributo público corpo_orbitado (string) com validação no construtor (não pode ser vazio). Implemente o método exibir_info para mostrar o nome, a massa e o corpo orbitado. Implemente o método orbitar(corpo) retornando uma string indicando que o satélite está orbitando o corpo fornecido.

4. Crie uma **classe concreta** Cometa que herda de CorpoCeleste e implementa a interface Orbitavel. Adicione um atributo público periodo_orbital (float) com validação no construtor (deve ser positivo). Implemente o método exibir_info para mostrar o nome, a massa e o período orbital. Implemente o método orbitar(corpo) retornando uma string indicando que o cometa está orbitando o corpo fornecido com seu período orbital.

5. Crie uma interface chamada Orbitavel com um método abstrato orbitar(corpo).

6. Crie uma função chamada simular_acao_celeste que recebe um objeto CorpoCeleste. Use isinstance para verificar o tipo do corpo celeste e chamar os métodos apropriados (orbitar, emitir_luz, colapsar). Imprima as informações do corpo celeste utilizando o método exibir_info antes de simular a ação.

7. No bloco, crie instâncias de todos os tipos de corpos celestes com dados válidos e inválidos (demonstrando o tratamento de ValueError). Crie uma lista de corpos celestes e chame a função simular_acao_celeste para cada um deles. O Tratamento de Exceção deve ser realizado nas classes!

Exemplos de Saída:

Criando corpos celestes...

Erro ao criar planeta: O diâmetro deve ser positivo.

Erro ao criar estrela: O tipo espectral não pode estar vazio.

Erro ao criar supernova: A luminosidade deve ser positiva.

Erro ao criar buraco negro: O raio de Schwarzschild deve ser positivo.

Erro ao criar satélite: O corpo orbitado não pode estar vazio.

Erro ao criar cometa: O período orbital deve ser positivo.

--- Informações e Simulação ---

Nome: Terra, Massa: 5.972e+24 kg, Diâmetro: 12742.0 km

Terra orbitando Sol.

--- Informações e Simulação ---

Nome: Sol, Massa: 1.989e+30 kg, Luminosidade: 3.828e+26 W, Tipo Espectral: G2V

Sol emitindo luz.

--- Informações e Simulação ---

Nome: SN 1987A, Massa: 3.0e+31 kg, Luminosidade: 1.0e+39 W, Magnitude Absoluta: -16.0

SN 1987A emitindo luz intensamente.

SN 1987A colapsou.

--- Informações e Simulação ---

Nome: Cygnus X-1, Massa: 3.0e+31 kg, Raio de Schwarzschild: 8850.0 m

--- Informações e Simulação ---

Nome: Lua, Massa: 7.3476e+22 kg, Orbitando: Terra

Lua orbitando Terra.

--- Informações e Simulação ---

Nome: Halley, Massa: 2.2e+14 kg, Período Orbital: 76.0 anos

Halley orbitando Sol com um período de 76.0 anos.