

EXERCÍCIO - PROGRAMAÇÃO ORIENTADA A OBJETOS

PROF. CARLOS EDUARDO BATISTA

HERANÇA, SOBRECARGA DE OPERADORES E VARIÁVEIS E FUNÇÕES DE CLASSE

UFPB - 2024.1

Contexto e Objetivo

O objetivo do exercício é implementar um **Sistema de Gerenciamento de Campeonato de Futebol**. Esse sistema deve explorar os conceitos de **herança pública, protegida e privada, herança múltipla e virtual, sobrecarga de operadores, composição e variáveis e métodos de classe**.

Requisitos do Sistema

O sistema deve ser capaz de:

1. **Cadastrar times:** Cada time deve ter um nome, um técnico responsável e uma lista de jogadores.
2. **Cadastrar jogadores:** Jogadores devem ser associados a um time, ter uma posição (goleiro, zagueiro, meio-campo, atacante), idade e número de gols marcados.
3. **Cadastrar técnicos:** Técnicos devem ser responsáveis por um time e ter experiência (em anos).
4. **Cadastrar juízes:** Juízes devem ter experiência e devem ser alocados em jogos específicos.
5. **Cadastrar `tecnico_jogador`:** Técnico que também joga, combinando características de `tecnico` e `jogador`.
6. **Cadastrar e gerenciar jogos:** Jogos entre times devem registrar resultados e atualizar as estatísticas de pontuação dos times e jogadores.
7. **Gerenciar as tabelas de classificação:** Mostrar a tabela de classificação de times ordenados por pontuação e de jogadores ordenados pelo número de gols.
8. **Comparar jogadores e times:** Usar sobrecarga de operadores para comparar jogadores pelo número de gols e times pela pontuação.
9. **Usar variáveis e métodos de classe para gerenciar contadores globais.**

Diagrama UML



- Atributos:

- **nome** (string): Nome completo da pessoa.
- **idade** (int): Idade da pessoa.
- **contador** (static int): variável da classe que conta todas as pessoas.

- Métodos:

- Construtor que inicializa os atributos com valores padrão.
- Gets e sets para os atributos privados.
- Método `virtual void exibir_informacoes()` (virtual puro) para exibir as informações da pessoa (exibido em classes derivadas).
- Sobrecarga do operador `==` para comparar pessoas com base no `nome` e `idade`.

- Atributos:

- **salario** (double): Salário do membro do clube.

- Métodos:

- Construtor que inicializa os atributos.
- Gets e sets para os atributos privados.
- Método `virtual double calcular_salario() = 0` (método puramente virtual).
- Sobrecarga do operador `<<` para exibir informações do membro do clube.

- Atributos:

- `posicao` (string): Posição do jogador (goleiro, zagueiro, meio-campo, atacante).
- `gols_marcados` (int): Número total de gols marcados pelo jogador.

- Métodos:

- Construtor que inicializa os atributos.
- Gets e sets para os atributos privados.
- Implementação de `calcular_salario()` que retorna `salario + (gols_marcados * 100)`.
- Sobrecarga do operador `<` para comparar jogadores pelo número de gols.

4. Classe `tecnico` (Deriva de `membro_clube` com Herança Pública)

- Atributos:

- `experiencia` (int): Anos de experiência como técnico.

- Métodos:

- Construtor que inicializa os atributos.
- Gets e sets para os atributos privados.
- Implementação de `calcular_salario()` que retorna `salario + (experiencia * 200)`.

5. Classe `tecnico_jogador` (Herança Múltipla de `jogador` e `tecnico` com Herança Virtual)

- Atributos:

- `bonus_duplo_papel` (double): Bônus por atuar como técnico e jogador ao mesmo tempo.

- Métodos:

- Construtor que inicializa os atributos.
- Gets e sets para os atributos privados.
- Implementação de `calcular_salario()` que retorna `jogador::calcular_salario() + tecnico::calcular_salario() + bonus_duplo_papel`.
- Sobrecarga do operador `<<` para exibir informações detalhadas do `tecnico_jogador`.

6. Classe `juiz` (Deriva de `pessoa` com Herança Pública)

- Atributos:

- `partidas_arbitradas` (int): Número total de partidas arbitradas.

- Métodos:

- Construtor que inicializa os atributos.
- Gets e sets para os atributos privados.
- Método `void incrementar_partidas_arbitradas()` que incrementa o número de partidas arbitradas.
- Implementação de `exibir_informacoes()` que exibe o número de partidas arbitradas.

7. Classe `time`

- Atributos:

- `nome` (string): Nome do time.
- `tecnico` (ponteiro para `tecnico`): Técnico do time.
- `jogadores` (vetor de objetos `jogador`): Lista de jogadores do time.
- `pontuacao` (int): Pontuação total do time.

- Métodos:

- Construtor que inicializa os atributos.
- Gets e sets para os atributos privados.
- Método `void adicionar_jogador(const jogador &j)` que adiciona um jogador ao time.
- Método `void registrar_resultado(char resultado)` que atualiza as estatísticas e a pontuação do time com base no resultado ('V' para vitória, 'E' para empate, 'D' para derrota).
- Método `void exibir_informacoes()` que exibe as informações completas do time.
- Sobrecarga do operador `<` para comparar times pela pontuação total.

8. Classe `jogo`

- Atributos:

- `time_casa` (ponteiro para `time`): Time jogando em casa.
- `time_visitante` (ponteiro para `time`): Time visitante.
- `juiz_principal` (ponteiro para `juiz`): Juiz responsável pela partida.
- `gols_time_casa` (int): Gols marcados pelo time da casa.
- `gols_time_visitante` (int): Gols marcados pelo time visitante.

- Métodos:

- Construtor que inicializa os atributos.
- Gets e sets para os atributos privados.
- Método `void registrar_resultado(int gols_casa, int gols_visitante)` que atualiza o resultado do jogo e ajusta a pontuação dos times envolvidos.
- Implementação de `exibir_informacoes()` que exibe o resultado do jogo.

9. Classe `campeonato`

- Atributos:

- `nome_campeonato` (string): Nome do campeonato.
- `times` (vetor de objetos `time`): Lista de times no campeonato.
- `jogos` (vetor de objetos `jogo`): Lista de jogos no campeonato.

- Métodos:

- Construtor que inicializa os atributos.
- Gets e sets para os atributos privados.
- Método `void adicionar_time(const time &t)` que adiciona um time ao campeonato.
- Método `void adicionar_jogo(const jogo &j)` que adiciona um jogo ao campeonato.

- Método `void exibir_classificacao()` que exibe a tabela de classificação dos times ordenados pela pontuação.

Instruções para o Aluno

- Atente para o Manual de Boas Práticas de Programação anexo ao exercício.
- Crie a hierarquia de classes conforme a especificação.
- Use `override` ao implementar métodos virtuais em classes derivadas.
- Sobrecargue os operadores `<<` e `<` para exibir e comparar times e jogadores.
- Use herança virtual para garantir que `tecnico_jogador` herde corretamente os atributos das classes `jogador` e `tecnico`.
- Use ponteiros inteligentes (`std::shared_ptr` ou `std::weak_ptr`) para gerenciar memória dinamicamente, se preferir.

Adicional: Cadastro de Campeonato

- Crie um campeonato chamado "`Campeonato Quadrangular`".
- Cadastre **quatro times** diferentes.
- Cada time deve ter um técnico e, pelo menos, 5 jogadores.
- Cadastre **um tecnico_jogador** para um dos times.
- Cadastre dois juizes.
- Cadastre **todos os jogos** (cada time deve jogar contra todos os outros) e preencha os resultados (gols marcados por cada time).
- Exiba a tabela de classificação final com todos os jogos realizados e as pontuações finais de cada time.

PONTUAÇÃO

O código deverá ser completamente comentado com relação a realização do solicitado. Plágio será punido com a não correção do exercício. O exercício vale 4,0 ponto para a segunda prova.

ENTREGA

A entrega poderá ser feita até as 23h59 do dia 03/10/2024 valendo a pontuação completa (4,0). Pode ser entregue até as 18h do dia 04/10 valendo 2,0 pontos.

O método de entrega poderá ser:

- via email para `bidu@ci.ufpb.br` - título: "[POO-241-E002] Sua matrícula".
- via assignment do github <https://classroom.github.com/a/BdKJBr7l>