

# Trabalho Prático

## Cartolitos CF

Élvis Júnior, 24/1038700  
Gustavo Alves, 24/1020779  
Pedro Marcinoni, 24/1002396  
Grupo 1

<sup>1</sup>Dep. Ciência da Computação – Universidade de Brasília (UnB)  
CIC0197 - Técnicas de Programação I

elvismirandajr@gmail.com, gusfring.a@gmail.com, pedroextrer@gmail.com

**Abstract.** *This document describes the final project of the course "Técnicas de Programação I" at Universidade de Brasília. The project is a desktop application that allows users to manage their fantasy football teams, providing features such as team creation, player selection, and match simulation. The application is built following object-oriented programming principles and utilizes design patterns to ensure maintainability and scalability.*

**Resumo.** *Este documento descreve o projeto final da disciplina Técnicas de Programação I da Universidade de Brasília. O projeto é uma aplicação desktop que permite aos usuários gerenciar suas equipes de futebol fantasy, oferecendo recursos como criação de equipes, seleção de jogadores e simulação de partidas. A aplicação é construída seguindo princípios de programação orientada a objetos e utiliza padrões de design para garantir manutenibilidade e escalabilidade.*

## 1. Descrição do Problema

Como bons brasileiros, quem não gosta de futebol? Foi pensando nisso que a emissora mais famosa do país, a Rede Globo, criou a Cartolitos CF, o novo Fantasy Game do momento, que reflete em tempo real os resultados estatísticos da Copa do Mundo de Clubes, o mais novo fenômeno do futebol. Veja também [1, 2].

Disponível para computador, o aplicativo permite que os usuários escalem seus times comprando jogadores dos 32 clubes do torneio com “cartoletas” — a moeda virtual do jogo — e pontue de acordo com o desempenho real desses atletas em campo na rodada. Assim, os participantes podem comparar seus resultados com os amigos em uma liga estilo tiro-curto, na qual se considera apenas uma única rodada.

Para colocar esse projeto em prática, a empresa designou três estudantes de Ciência da Computação — Elvis Miranda, Gustavo Alves e Pedro Marcinoni — para desenvolver um programa capaz de auxiliar na logística de gestão de qualquer liga de tiro-curto, servindo depois como base para o funcionamento integral do aplicativo.

## 2. Definição das regras de negócio

1. Função do programa:
  - (a) Cadastrar usuários, clubes, jogadores e equipes de usuários
  - (b) Simula estatísticas e pontuações conforme regras de negócios
  - (c) era resultados da liga e rankings
2. Identificação: Cada usuário tem ID único, nome, e-mail e senha.
3. Orçamento: Cada usuário começa com 150 cartoletas.
4. Escalação:
  - (a) O time do usuário deve ter formação 4-3-3, ou seja, 1 goleiro, 4 defensores, 3 meio-campistas e 3 atacantes.
  - (b) O time do usuário não pode ter jogadores duplicados.
5. Capitão: Cada time deve ter um capitão, que recebe o dobro de pontos.
6. Estatísticas dos jogadores:
  - (a) Cada jogador possui um preço base e uma nota (overall) que varia de 0 a 100.
  - (b) Cada confronto (casa/fora) usa as notas de cada jogador para ajustar as probabilidades de contribuir com gols, assistências, desarmes, etc.
7. Pontuação:
  - (a) Goleiros:
    - Defesa de pênalti: +7 pontos
    - Defesa: +1.5 pontos
    - Gol sofrido: -1 ponto
    - Não sofrer gol: +5 pontos
  - (b) Defensores:
    - Desarme: +1.5 pontos
    - Falta cometida: -0.5 pontos
    - Gol contra: -5 pontos
    - Cartão amarelo: -1 pontos
    - Cartão vermelho: -3 pontos
  - (c) Meio-campistas:
    - Assistência: +5 pontos
    - Gol: +8 pontos
    - Falta cometida: -0.5 pontos
    - Desarme: +1.5 pontos
  - (d) Atacantes:
    - Gol: +8 pontos
    - Assistência: +5 pontos
    - Finalização: +0.8 pontos
    - Falta cometida: -0.5 pontos
8. Simulação de partidas: O administrador do programa pode simular partidas entre os times cadastrados, gerando estatísticas e pontuações para cada jogador.
9. Resultados da liga: O programa deve gerar resultados da liga, mostrando a pontuação total de cada time e o ranking dos usuários.

Dessa forma, a classe **Pessoa** garante a reutilização de código e facilita a manutenção, permitindo que funcionalidades comuns sejam implementadas uma única vez e herdadas pelas demais classes do sistema.

### 3.1.1. Sublasse Usuario

A subclasse **Usuario** representa um usuário do sistema, sendo uma especialização da classe abstrata `Pessoa`. Ela adiciona atributos e comportamentos específicos relacionados à participação do usuário em ligas e à administração dessas ligas.

- **Atributos principais:**

- `cartoletas`: representa o saldo virtual do usuário, utilizado para comprar jogadores.
- `tipo`: indica o tipo do usuário, podendo ser comum ou administrador de liga.
- `timeUsuario`: armazena o time montado pelo usuário.
- `liga`: referência à liga da qual o usuário faz parte.

- **Métodos principais:**

- `editarNomeLiga(String novoNome)`: permite ao administrador de liga alterar o nome da liga.
- `addUsuarioLiga(Usuario usuario)`: permite ao administrador adicionar um novo usuário à liga.
- `removeUsuarioLiga(Usuario usuario)`: permite ao administrador remover um usuário da liga.
- `entrarLiga(Liga liga)`: permite ao usuário entrar em uma liga, caso ainda não participe de nenhuma.
- `sairLiga(Liga liga)`: permite ao usuário sair da liga atual.
- `toString()`: retorna uma representação textual do usuário, exibindo seu ID e nome.

Esses métodos garantem as principais operações de gerenciamento de ligas e participação dos usuários, respeitando as regras de negócio do sistema.

### 3.1.2. Sublasse Admin

A subclasse **Admin** representa o administrador do sistema, sendo uma especialização da classe abstrata `Pessoa`. Ela é responsável por operações administrativas, como gerenciamento de usuários e simulação de partidas.

- **Atributos principais:**

- `usuarioDAO`: objeto responsável pelo acesso e manipulação dos dados dos usuários no banco de dados.

- **Métodos principais:**

- `Admin(int id, String nome, String senha, Connection conn)`: construtor que inicializa o administrador e o objeto de acesso a dados.
- `simular()`: executa a simulação das partidas do sistema.
- `resetarSimulacao()`: reseta os dados da simulação realizada.
- `deleteUsuario(Usuario usuario)`: remove um usuário do sistema a partir de um objeto `Usuario`.
- `deleteUsuario(int id)`: remove um usuário do sistema a partir do seu identificador.

- `isAdmin()`: sobrescreve o método da classe `Pessoa` para indicar que esta instância é de um administrador.

Esses métodos permitem ao administrador controlar usuários e simulações, garantindo a manutenção e o funcionamento correto do sistema.

### 3.2. Classe `TimeUsuario`

A classe `TimeUsuario` representa o time montado por um usuário, sendo responsável por gerenciar a escalação, o capitão, o cálculo de pontuação e o controle de validade do time para simulação.

- **Atributos principais:**

- `usuario`: referência ao usuário dono do time.
- `jogadores`: conjunto de jogadores escalados no time.
- `pontuacao`: pontuação total do time após a simulação.
- `preco`: valor total do time, somando o preço de todos os jogadores.
- `capitao`: jogador escolhido como capitão, que tem sua pontuação dobrada.
- `valido`: indica se a escalação está válida para simulação (time completo e capitão definido).
- `contG`, `contZ`, `contM`, `contA`: contadores de jogadores por posição (goleiro, zagueiro, meia e atacante).

- **Métodos principais:**

- `TimeUsuario(Usuario usuario)`: construtor que inicializa um time vazio para o usuário.
- `TimeUsuario(Usuario usuario, Set<Jogador> jogadores, Jogador jogcapitao)`: construtor que inicializa o time já com jogadores e capitão.
- `calcularPontuacao()`: calcula a pontuação total do time, considerando o capitão.
- `calcularPreco()`: calcula o preço total do time.
- `addJogador(Jogador jogador)`: adiciona um jogador ao time, respeitando as regras de quantidade e orçamento.
- `removeJogador(Jogador jogador)`: remove um jogador do time.
- `setCapitao(Jogador jogador)`: define o capitão do time, caso o jogador esteja escalado.
- `removeCapitao()`: remove o capitão do time.
- `removerTodosJogadores()`: remove todos os jogadores do time.
- `imprimirTime()`: imprime no console as informações do time, incluindo jogadores, pontuação e capitão.
- `getGoleiro()`, `getZagueiros()`, `getMeias()`, `getAtacantes()`: retornam os jogadores do time por posição.
- `getTodosOsEscalados()`: retorna todos os jogadores do time, organizados por posição.

Esses métodos permitem ao usuário montar, visualizar e gerenciar seu time, garantindo que a escalação siga as regras do sistema e esteja pronta para a simulação das partidas.

### 3.3. Classe Liga

A classe **Liga** representa uma liga do sistema, permitindo o agrupamento de usuários para competições e gerenciamento de rankings.

- **Atributos principais:**

- `id`: identificador único da liga.
- `nroUsuarios`: quantidade de usuários participantes da liga.
- `nome`: nome da liga.
- `senha`: senha de acesso à liga.
- `usuarios`: lista de usuários que participam da liga.

- **Métodos principais:**

- `Liga(int id, String nome, String senha)`: construtor que inicializa a liga com identificador, nome e senha.
- `removeAll()`: remove todos os usuários da liga.
- `addUsuario(Usuario usuario)`: adiciona um usuário à liga e incrementa o número de participantes.
- `removeUsuario(Usuario usuario)`: remove um usuário da liga e decrementa o número de participantes.
- `gerarRanking()`: gera e retorna uma lista dos times dos usuários, ordenada pela pontuação (ranking da liga).
- `exibirRanking(List<TimeUsuario> times)`: exibe no console o ranking dos times da liga.
- `toString()`: retorna uma representação textual da liga, mostrando seu nome.

Esses métodos permitem o gerenciamento dos participantes e o acompanhamento do desempenho dos times dentro de uma liga, proporcionando a competição entre os usuários.

### 3.4. Classe Simulacao

A classe **Simulacao** é responsável por gerenciar o processo de simulação das partidas entre clubes, além de controlar o estado das simulações e o relacionamento com as ligas e usuários do sistema.

- **Atributos principais:**

- `ocorreu`: indica se a simulação já foi realizada.
- `partidas`: conjunto de partidas que serão simuladas.
- `ligaDAO`: objeto responsável pelo acesso e manipulação dos dados das ligas no banco de dados.
- `usuarioDAO`: objeto responsável pelo acesso e manipulação dos dados dos usuários no banco de dados.

- **Métodos principais:**

- `InicializarConexoes(Connection conn)`: inicializa os objetos de acesso a dados com a conexão ao banco.
- `gerarPartidasAleatorias(List<Clube> clubes)`: sorteia clubes em partidas aleatórias, garantindo que todos participem.

- `addPartida(Clube clubeCasa, Clube clubeFora)`: adiciona uma partida entre dois clubes, se ambos ainda não estiverem em outra partida.
- `simular()`: executa a simulação das partidas e calcula a pontuação dos jogadores e times das ligas.
- `resetar()`: reseta o estado da simulação, restaurando os dados dos jogadores e times.
- `trocarClubes(Clube clube1, Clube clube2)`: troca clubes entre partidas diferentes.
- `clear()`: limpa todas as partidas da simulação e libera os clubes para novas partidas.

Esses métodos permitem o controle completo do ciclo de simulação das partidas, desde a geração dos confrontos até o cálculo e a reinicialização dos resultados.

### 3.5. Interface Simulavel

A interface **Simulavel** define um contrato para as classes que desejam implementar a funcionalidade de simulação no sistema. Ela garante que qualquer classe que a implemente possua o método necessário para realizar uma simulação.

- **Métodos principais:**

- `simular()`: método abstrato que deve ser implementado pelas classes concretas, responsável por executar a lógica de simulação específica de cada classe.

Dessa forma, a interface **Simulavel** promove a padronização e a reutilização de código, permitindo que diferentes componentes do sistema possam ser simulados de maneira uniforme.

#### 3.5.1. Classe Partida

A classe **Partida**, implementação da interface **Simulavel** representa um confronto entre dois clubes no sistema, sendo responsável por simular o jogo, calcular estatísticas dos jogadores e armazenar os principais eventos da partida.

- **Atributos principais:**

- `clubeCasa, clubeFora`: referências aos clubes que disputam a partida.
- `golsClubeCasa, golsClubeFora`: quantidade de gols marcados por cada clube.
- `assistClubeCasa, assistClubeFora`: quantidade de assistências de cada clube.
- `golsPenaltiCasa, golsPenaltiFora`: quantidade de gols de pênalti de cada clube.
- `jogadoresGolCasa, jogadoresGolFora`: listas de jogadores que marcaram gols por cada clube.
- `jogadoresAssistenciaCasa, jogadoresAssistenciaFora`: listas de jogadores que deram assistências por cada clube.

- `cartaoVermelhoClubeCasa`, `cartaoVermelhoClubeFora`: indicam se algum jogador do clube recebeu cartão vermelho.
- `random`: objeto para geração de números aleatórios, utilizado na simulação dos eventos da partida.
- **Métodos principais:**
  - `Partida(Clube clubeCasa, Clube clubeFora)`: construtor que inicializa uma partida entre dois clubes.
  - `simular()`: executa toda a lógica de simulação da partida, gerando estatísticas para os jogadores e clubes.
  - `simularStatusJogador(...)`: simula os eventos individuais de cada jogador (gols, assistências, desarmes, faltas, cartões, etc.).
  - `simularPenalti(...)`: simula a ocorrência de pênaltis, gols e defesas de pênalti na partida.
  - `corrigirGolsAssistencias(...)`: ajusta a relação entre gols e assistências para garantir consistência nos dados da partida.
  - `resetStats()`: reseta todas as estatísticas da partida e dos jogadores envolvidos, preparando para uma nova simulação.
  - `mostrarResumoPartida()`: exibe no console um resumo detalhado dos principais eventos e estatísticas da partida.
  - `poisson(double lambda, Random random)`: calcula um valor aleatório baseado na distribuição de Poisson, utilizado para simular eventos como gols e assistências.
  - `calcularBonusClube(boolean timeCasa)`: calcula bônus de desempenho para o clube, considerando fatores como mando de campo e cartões vermelhos.
  - `getAllJogadores()`: retorna uma lista com todos os jogadores participantes da partida.

Esses métodos e atributos permitem simular partidas de forma realista, atribuindo estatísticas individuais aos jogadores e resultados aos clubes, além de possibilitar o acompanhamento detalhado dos eventos ocorridos durante o jogo.

### 3.6. Classe Jogador

A classe **Jogador** representa um atleta disponível para ser escalado nos times dos usuários, armazenando informações essenciais para a simulação e pontuação.

- **Atributos principais:**
  - `id`: identificador único do jogador.
  - `nome`: nome do jogador.
  - `posicao`: posição em que o jogador atua (goleiro, zagueiro, meia ou atacante).
  - `clube`: referência ao clube ao qual o jogador pertence.
  - `preco`: valor do jogador em cartoletas.
  - `overall`: nota geral do jogador, utilizada para simulação de desempenho.
  - `pontuacao`: pontuação total do jogador após a simulação.
  - `stats`: objeto que armazena as estatísticas detalhadas do jogador na partida.



- **Métodos principais:**

- `Jogador(int id, String nome, Posicao posicao, Clube clube, double preco, double overall)`: construtor que inicializa o jogador com seus dados básicos.
- `getStringPosicao()`: retorna a posição do jogador em formato textual.
- `calcularPontuacao()`: calcula e atualiza a pontuação do jogador com base em suas estatísticas e regras de pontuação do sistema.
- `equals(Object obj)` e `hashCode()`: métodos sobrescritos para garantir a correta comparação e uso do jogador em coleções.

Esses métodos e atributos permitem representar, identificar e calcular o desempenho de cada jogador, sendo fundamentais para a lógica de escalação e simulação das partidas.

### 3.7. Classe Stats

A classe **Stats** representa as estatísticas detalhadas de desempenho de um jogador em uma partida, armazenando informações essenciais para o cálculo da pontuação individual.

- **Atributos principais:**

- `desarmes`: quantidade de desarmes realizados pelo jogador.
- `gols`: quantidade de gols marcados.
- `assistencias`: quantidade de assistências realizadas.
- `sg`: indica se o jogador terminou a partida sem sofrer gols (aplicável a goleiros e zagueiros).
- `finalizacoes`: número de finalizações feitas.
- `defesas`: número de defesas realizadas (goleiros).
- `defesaPenalti`: número de defesas de pênalti.
- `golsContra`: quantidade de gols contra marcados.
- `cartaoVermelho`: indica se o jogador recebeu cartão vermelho.
- `golsSofridos`: quantidade de gols sofridos (goleiros e zagueiros).
- `cartaoAmarelo`: quantidade de cartões amarelos recebidos.
- `faltasCometidas`: número de faltas cometidas.
- `posicao`: posição do jogador na partida.

- **Métodos principais:**

- `resetStats()`: reseta todas as estatísticas do jogador para os valores iniciais, preparando para uma nova simulação.
- `forEach(Consumer<? super Map.Entry<String, Integer>> action)`: percorre todas as estatísticas, permitindo executar uma ação para cada uma delas (útil para exibição ou processamento).

Esses métodos e atributos permitem registrar, manipular e exibir as estatísticas individuais dos jogadores, sendo fundamentais para o cálculo da pontuação e para a análise de desempenho nas partidas simuladas.

### 3.8. Classe Clube

A classe **Clube** representa um clube de futebol no sistema, armazenando informações sobre seus jogadores, desempenho médio e integração com o banco de dados.

- **Atributos principais:**

- `id`: identificador único do clube.
- `nome`: nome do clube.
- `overAtaque`: média aritmética do overall dos jogadores de ataque do clube.
- `overDefesa`: média aritmética do overall dos jogadores de defesa do clube.
- `jogadores`: conjunto de jogadores pertencentes ao clube.
- `jogA`, `jogD`: quantidade de jogadores de ataque e defesa, respectivamente.
- `partida`: indica se o clube já está escalado para uma partida.
- `clubeDAO`, `jogadorDAO`: objetos responsáveis pelo acesso e manipulação dos dados do clube e dos jogadores no banco de dados.

- **Métodos principais:**

- `Clube(...)`: construtores que inicializam o clube, podendo inserir ou recuperar dados do banco de dados.
- `addJogador(String nomeJogador, Posicao posicao, double preco, double overall)`: adiciona um novo jogador ao clube e ao banco de dados.
- `addJogador(Jogador jogador)`: adiciona ao clube um jogador já existente no banco de dados.
- `removeJogador(Connection conn, Jogador jogador)`: remove um jogador do clube e do banco de dados, se a simulação ainda não ocorreu.
- `removeJogadorById(Connection conn, int idJogador)`: remove um jogador pelo ID, tanto do clube quanto do banco de dados.
- `recalcOverAtaqueAdd/Sub(double overall)` e `recalcOverDefesaAdd/Sub(double overall)`: recalculam a média de overall de ataque ou defesa ao adicionar ou remover jogadores.
- `isAtaque(Posicao posicao)` e `isDefesa(Posicao posicao)`: verificam se uma posição é considerada de ataque ou defesa.
- `toString()`: retorna uma representação textual do clube.

Esses métodos e atributos permitem o gerenciamento completo dos clubes, incluindo a manutenção dos jogadores, atualização das médias de desempenho e integração com o banco de

## 4. Telas

### 4.1. Telas Iniciais

O programa possui dois fluxos principais: o fluxo do usuário e o fluxo do administrador. Cada um possui telas específicas para suas funcionalidades.

## **4.2. Fluxo do usuário**

### **4.2.1. Menu Principal**

### **4.2.2. Ligas**

## **5. Conclusão**

Colocar aqui a conclusão do projeto. O que foi aprendido? O que poderia ser melhorado? Quais as dificuldades encontradas? Quais os próximos passos?

## **Referências**

[Cartola FC ] Cartola FC. Wikipédia. [https://pt.wikipedia.org/wiki/Cartola\\_FC](https://pt.wikipedia.org/wiki/Cartola_FC). [Online; acesso em 01-jun-2025].

[Esporte fantasy ] Esporte fantasy. Wikipédia. [https://pt.wikipedia.org/wiki/Esporte\\_fantasy](https://pt.wikipedia.org/wiki/Esporte_fantasy). [Online; acesso em 01-jun-2025].

## **Referências**

[1] Cartola FC. Disponível em: [https://pt.wikipedia.org/wiki/Cartola\\_FC](https://pt.wikipedia.org/wiki/Cartola_FC). Acesso em: 10 jun. 2024.

[2] Fantasy Sports. Disponível em: [https://en.wikipedia.org/wiki/Fantasy\\_sport](https://en.wikipedia.org/wiki/Fantasy_sport). Acesso em: 10 jun. 2024.