Sexto mini Exercício Programa

Nome: Bruno Sesso NUSP: 8536002
Nome: Gustavo Estrela de Matos NUSP: 8536051

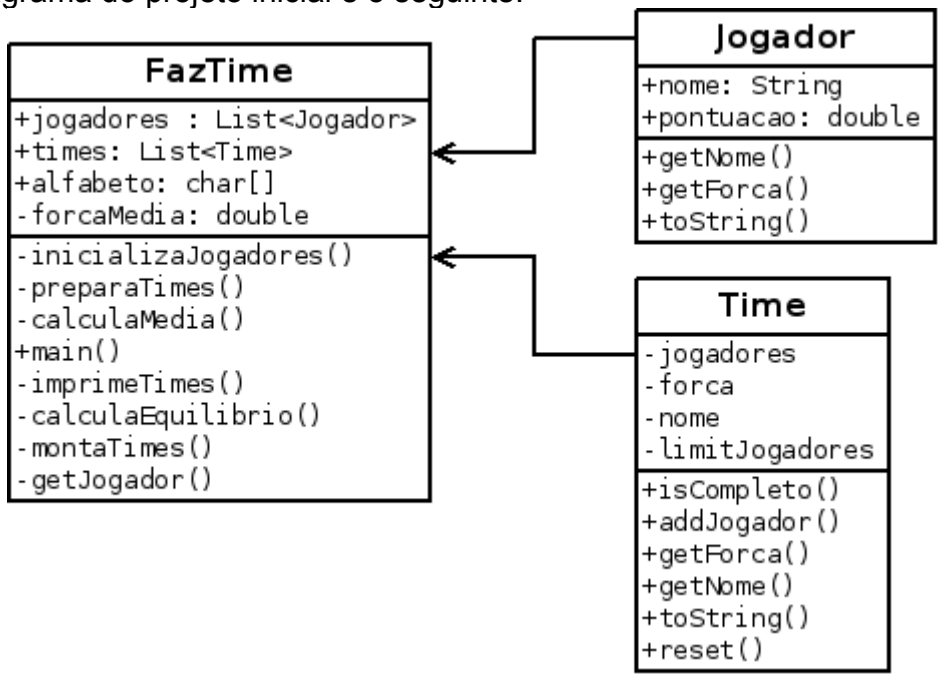
1. Introdução

Para este mini-ep, foi escolhido um código em Java que resolve o problema de montar times equilibrados para uma partida de futebol. Esse código foi encontrado no GitHub ao acaso após pesquisar projetos pelo palavra-chave “futebol” e ordenar por “menos estrelas”, para garantir que não estava escolhendo um projeto muito “famoso” o que, em tese, implicaria a escolha de um projeto mais “redondo” e “acabado”. Esse projeto pode ser acessado por este link.

2. O código inicial

2.1 Diagrama de classes

O diagrama do projeto inicial é o seguinte:



As classes acima possuem as seguintes responsabilidades:

- *FazTime* – Essa classe é responsável por criar de fato um time, com o auxílio das classes *Jogador* e *Time.*
- *Jogador* – Essa classe é a representação um jogador de um time.
- *Time* – Essa classe é responsável por representar um time.

2.2 Mudanças imediatas

O primeiro erro que pretendo contornar nesse projeto é a maneira como os jogadores são colocados a disposição para a classe FazTime. No código atual, os jogadores foram colocados na lista *jogadores* de forma *hard-coded*, ou seja, se conhecermos um grupo de amigos novos e quisermos colocar eles no nosso algoritmo, teremos que abrir nossa classe FazTime e mudar lá os jogadores que são colocados na lista, o que não é bom, pois implica em constantes mudanças em uma classe, mesmo que não estruturais.

3 Proposta de mudanças

O projeto escolhido, aparentemente, serviu para organizar jogos do seu

desenvolvedor com seus amigos, isto é, estamos lidando com um projeto pequeno e que muito provavelmente não passaria por mudanças. Mesmo assim, o desenvolvedor foi cuidadoso e escreveu seu código de maneira modularizada. Aproveitando isso, com o intuito de colocar em prática pelo menos um dos padrões de desenvolvimentos vistos em aula, resolvi propor o desafio de ampliação do projeto.

O desafio é fazer este código receber informações mais detalhadas sobre jogadores, como peso, altura, envergadura; e a partir disso ser capaz de montar equipes equilibradas de acordo com os seguintes esportes variados, como futebol, beisebol, basquete e vólei; e, além disso, deixar o código modularizado o suficiente tal que seja fácil acrescentar funcionalidades como calcular times para um esporte diferente.

4. Mudanças

Tentamos, ao máximo, não mudar o código que já estava feito, mas como nossa proposta era expandir o programa e fazer dele um gerador de times mais geral, tivemos que criar algumas classes e modificar algumas das que já estavam prontas.

4.1 Entrada

Para realizar a leitura e instanciação foi criada a classe Leitor. A entrada passou a ser feita pela entrada padrão, ao invés de ser *hard-coded* como estava na versão original. O arquivo de entrada possui dados do jogador e o esporte para o qual estamos querendo montar um time.

4.2 Classe Jogador

A classe *Jogador* ganhou os novos atritutos *stats* e *forcaCalculator.* O atributo *stats* é um instancia da nova classe *Stats* e o atributo *forcaCalculator* é uma instancia de alguma classe que segue a interface *ForcaCalculator.*

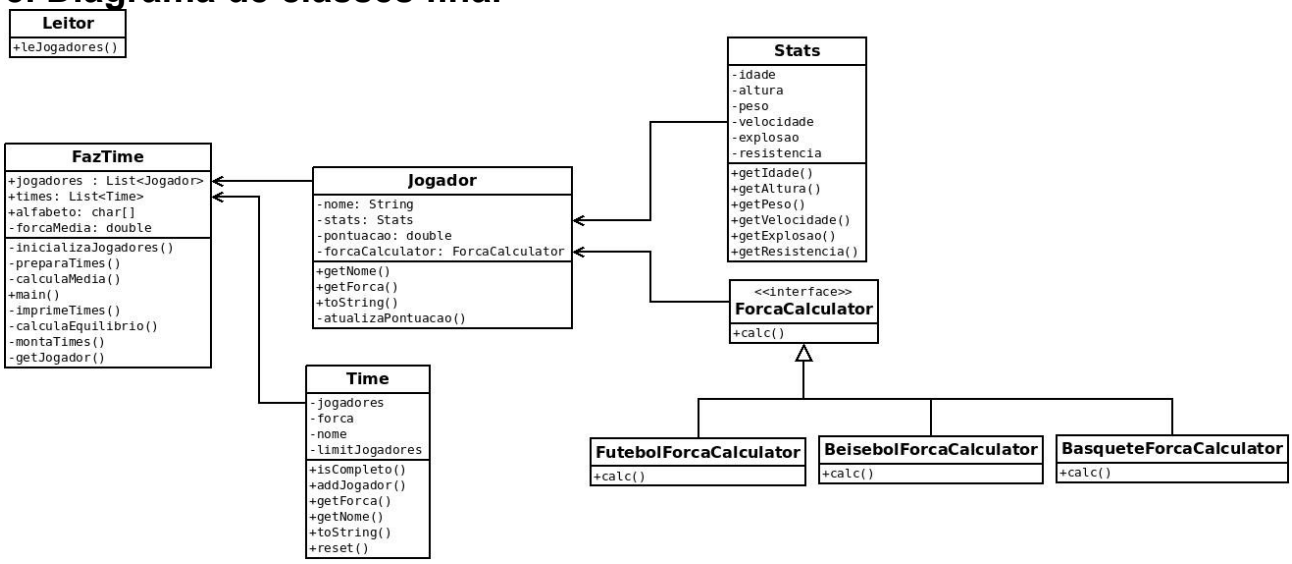
4.3 Classe Stats

A classe *Stats* é responsável por guardar dados do jogador como peso, altura, velocidade, etc.

4.4 Classe ForcaCalculator

Esta classe é uma interface para qualquer outra classe criada que seja uma calculadora de força para um jogador. Essa classe foi criada para ser possível aplicar o padrão Strategy no cálculo da força de um jogador.

5. Diagrama de classes final



6. Padrão de design aplicado

Em meio a expensão do projeto, verificamos que haviam diferenças para o calculo da força de um jogador dependendo do tipo de esporte que estamos considerando. Um jogador que possui grande velocidade e pouca resistência teria, em teoria, uma força maior para o beisebol do que para o futebol.

Para que seja fácil expandir o programa e fazer que ele funcione para diferentes esportes, aplicamos o padrão de projeto *Strategy.* O padrão foi aplicado na função calc da interface *ForcaCalculator.* Assim, para adicionar um novo esporte, por exemplo, basta criar uma nova classe que implementa a interface.

7. Para compilar e rodar o programa

Para compilar o programa, usamos o *javac.* Para compilar, basta estar um diretorio acima do diretorio futebol e rodar o comando:

- *javac futebol/FazTime.java*

Para rodar o programa faça:

- *java futebol.FazTime [esporte] < futebol/entrada.txt*

Conclusões

A maior dificuldade desse mini-ep foi achar um código na internet que fosse possível de se aplicar um padrão de design. Isso aconteceu, muito provavelmente, porque até agora na graduação vimos muito pouco de orientação objeto e portanto vimos e produzimos muito pouco código orientado a objeto.

O código escolhido era bem simples e foi fácil expandi-lo. Essa facilidade ocorreu principalmente porque o código já estava razoavelmente bem fatorado, o que fez com que não precisássemos ver muito partes do código, como a classe *Time*, que não foi modificada.

A aplicação do padrão de projeto era praticamente imediata para a proposta que foi feita na primeira entrega, pois a generalização do programa para mais esportes força o programador a pensar em maneiras de construir o código de maneira que facilite acrescentar um novo esporte, o que na verdade, significa acrescentar uma nova maneira de se calcular a força de um jogador.