Trabalho 2 (M2)

Considere que você precisa desenvolver um programa para manipular dados de jogadores, times e jogos de futebol. Sua solução deve compreender tanto o diagrama de classes quanto o projeto implementado em Java.

Cada jogador tem uma nota de 0 a 100 que corresponde a sua habilidade. Entretanto, o cálculo da habilidade varia de acordo com a posição do jogador e com o valor de seus atributos, os quais também variam de 0 a 100. Um goleiro possui os atributos altura e reflexos. Considere que o atributo altura é a altura real do jogador em centímetros normalizada para um valor entre 0 e 100. Para isso, a altura máxima a ser considerada é 210 centímetros. Desta forma, qualquer jogador com 210 centímetros ou mais de altura terá a nota 100. Um jogador de 181 centímetros terá a nota 86 (considerando o arredondamento HALF_EVEN). Um defensor possui os atributos cobertura e desarme. Um atacante possui os atributos velocidade e técnica. O cálculo da habilidade é dado pelas seguintes fórmulas:

- Goleiro: altura * 4 + reflexos * 6
- Defensor: cobertura * 6 + desarme * 4
- Atacante: velocidade * 4 + técnica * 6

O programa deve controlar a quantidade de gols feita por cada jogador. Este valor é acumulativo, pois um jogador pode fazer gols em diferentes partidas. Além do nome do jogador, o programa deve calcular sua idade atual. Cada jogador tem um número de camisa único dentro de seu time (um time não pode ter dois jogadores com um mesmo número de camisa).

Um time possui um nome e jogadores que podem ser adicionados ou removidos. Um time completo é formado por 5 jogadores, sendo 1 goleiro, 2 defensores e 2 atacantes. Além disso, o programa deve controlar os resultados obtidos pelo time (vitórias, empates e derrotas). Ao final de cada partida, os resultados de cada time devem ser atualizados. Uma vitória vale 3 pontos, empate vale 1 e derrota não pontua.

Uma partida é composta de dois times, onde um é o time da casa e o outro é o visitante. Além da data da partida, o programa deve guardar o placar da partida (quantos gols o time da casa fez e quantos gols o time visitante fez). Desta forma, será possível saber quem venceu a partida ou se houve empate.

O sistema deve permitir cadastrar várias partidas e depois simular a execução destas partidas, atualizando estatísticas de times e jogadores. Na simulação, considere a pontuação total das habilidades de cada time (quem tem a maior pontuação, tem mais chance de sair vitorioso). Além disso, se o time que estiver jogando em casa tem uma chance adicional. Você pode definir sua própria fórmula, mas ela deve estar explicada no código (javadoc).

Para a validação, crie casos de teste para avaliar o programa desenvolvido. Não há a necessidade de criar uma interface com o usuário. Tudo deve ser demonstrado pelos casos de teste. Para facilitar, considere a redefinição do método toString().

Testes esperados:

- Cadastrar vários times.
- Cadastrar vários jogadores.
- Classificar os times de acordo com a habilidade total de seus jogadores.
- Classificar os times de acordo com suas estatísticas. Para ordenar, considere total de pontos, número de vitórias, número de empates e saldo de gols.
- Adicionar jogadores em um time.

Trabalho 2 (M2)

- Remover jogadores de um time.
- Cadastrar várias partidas.
- Simular a execução das partidas.
- Avaliar se a simulação foi executada corretamente.

Orientações gerais

1. A pasta do projeto deve ser compactada (formato zip). O arquivo compactado deve ser renomeado antes de ser colocado na Intranet. O formato para o nome do arquivo deve ser "T<n>_<Nome1>_<Nome2>.zip", onde:
   - <n> é o número do trabalho solicitado
   - <Nome1> é o nome de um dos alunos (não precisa ser completo, mas o suficiente para identificá-lo na turma)
   - <Nome2> é o nome do outro aluno participante

   Os projetos devem ter rigorosamente esta nomenclatura, pois a ferramenta do Material Didático não renomeia o arquivo baixado com o nome dos alunos que fizeram o upload. Além disso, facilitará a identificação de qual projeto pertence a qual equipe, quando carregados no IntelliJ. Exemplos para nomes válidos:
   - T2_TonyStark_ SteveRogers.zip
   - T2_JessicaJones_NatashaRomanova.zip

2. Este trabalho não é apenas de implementação. Você deve projetar as classes que você precisará e quais operações serão necessárias. Não tente escrever o código diretamente. Você deve entregar também a modelagem UML completa da solução. A modelagem deve ser feita obrigatoriamente na ferramenta Bouml. Adicione o projeto da ferramenta no arquivo compactado que corresponde à entrega.

3. Depois de decidir quais classes você precisa, implemente uma de cada vez (considerando as classes simples primeiro). Realize testes para assegurar que ela está funcionando antes de prosseguir para a próxima classe. Eventualmente, você perceberá a necessidade de ajustes no seu design e não será necessário modicar todo o código. Siga a ideia: projete um pouco, codifique um pouco, teste um pouco. Repita.

4. Escreva um código legível e utilize comentários (javadoc) para documentar como as classes e respectivas operações devem ser utilizadas. Lembre-se que o código deve ser de todos. A qualidade e pertinência dos comentários será considerada como critério de avaliação.

5. Implemente validações sempre que pertinente, utilizando exceções. Você pode criar uma classe Validator para validações comuns. Implemente, pelo menos, uma classe de exceção para tratamento de informações inválidas.

6. Observe os princípios da orientação a objetos. Classes e operações com baixa coesão, métodos muitos longos ou com muitos parâmetros afetarão negativamente a avaliação do seu trabalho.