PO24CP - Programação Orientada a Objetos
Profa. Luciene de Oliveira Marin
lucienemarin@utfpr.edu.br

Trabalho #02 - Campeonato de Apostas em Jogos de Dados

Data de entrega [Turma 4CP]: **30/11/2023 (até 23h59)**, via moodle.
Data de entrega [Turma 4CPE]: **30/11/2023 (até 23h59)**, via moodle.

- ✔ O Trabalho #02 deve ser a extensão do Trabalho #01 em termos de código e equipe.
- ✍ Serão descontados **2 pontos por dia de atraso**.
- ✘ Em caso de **cópia** de código, **os alunos envolvidos** terão nota igual a **zero** na média final da disciplina.

Desenvolva um simulador de campeonato de apostas em jogos de dados que permita aos jogadores presentes realizar apostas em dois tipos de jogos de dados, o jogo de azar (Anexo 1) e o jogo general (Anexo 2).[1] As regras a seguir definem como o campeonato irá funcionar:

- O campeonato permitirá no máximo **dez** jogadores (humano/máquina) e cada jogador poderá realizar até **dez apostas** nos jogos de sua escolha, enquanto houver saldo suficiente para tal.
- Para a execução de uma rodada de apostas é necessário ao menos um jogador participante, com saldo disponível (maior que R$ 0,00) para realização de aposta (até no máximo dez) em jogos de sua escolha (um jogo por aposta).
- A cada rodada do campeonato, será perguntado a cada jogador sobre o valor a ser apostado e para qual jogo. Na sequência será realizada a rodada do jogo e a vez será passada ao próximo jogador.
- Para cada jogador serão armazenados os resultados de até **dez** jogos realizados (do tipo general ou de azar) com seus respectivos valores de aposta.
  - O resultado do jogo de azar compreende se o jogador ganhou ou perdeu determinada jogada, de acordo com as regras do jogo (Anexo 1).
  - O resultado do jogo general compreende os valores obtidos para as 13 jogadas. O jogador **ganha a aposta** se a soma dos valores das jogadas de 1 a 12 **for maior que o dobro** do valor obtido na jogada 13 (aleatória).
- Quando o jogador é inserido no campeonato, o mesmo inicia com o saldo de R$ 100,00. No início de cada rodada, o jogador indicará o quanto deseja apostar. Se o jogador **ganhar**, este receberá a mesma quantia que apostou, caso contrário ele perderá a quantia que apostou, por exemplo:

[1] Conforme implementado no trabalho prático #01.

– O jogador inicialmente tinha R$ 100,00, apostou R$ 10,00 e ganhou. Então o novo saldo do jogador é de R$ 110,00. Em um segundo momento o jogador se sentiu confiante e apostou R$ 50,00 e perdeu, logo seu saldo ficou em R$ 60,00;

- Poderão ser executadas n rodadas de apostas desde que sejam cumpridas os condições para cada jogador (saldo suficiente e não ultrapassar mais que dez apostas).
- Em cada rodada, para cada jogo escolhido, deverão ser contabilizadas a quantidade que cada face de cada dado já fora sorteada.
- A aplicação deverá produzir diferentes relatórios de saldos, extratos de resultado dos jogos e estatística das faces sorteadas, por tipo de jogador, por tipo de jogo, por rodadas e por total de campeonato.

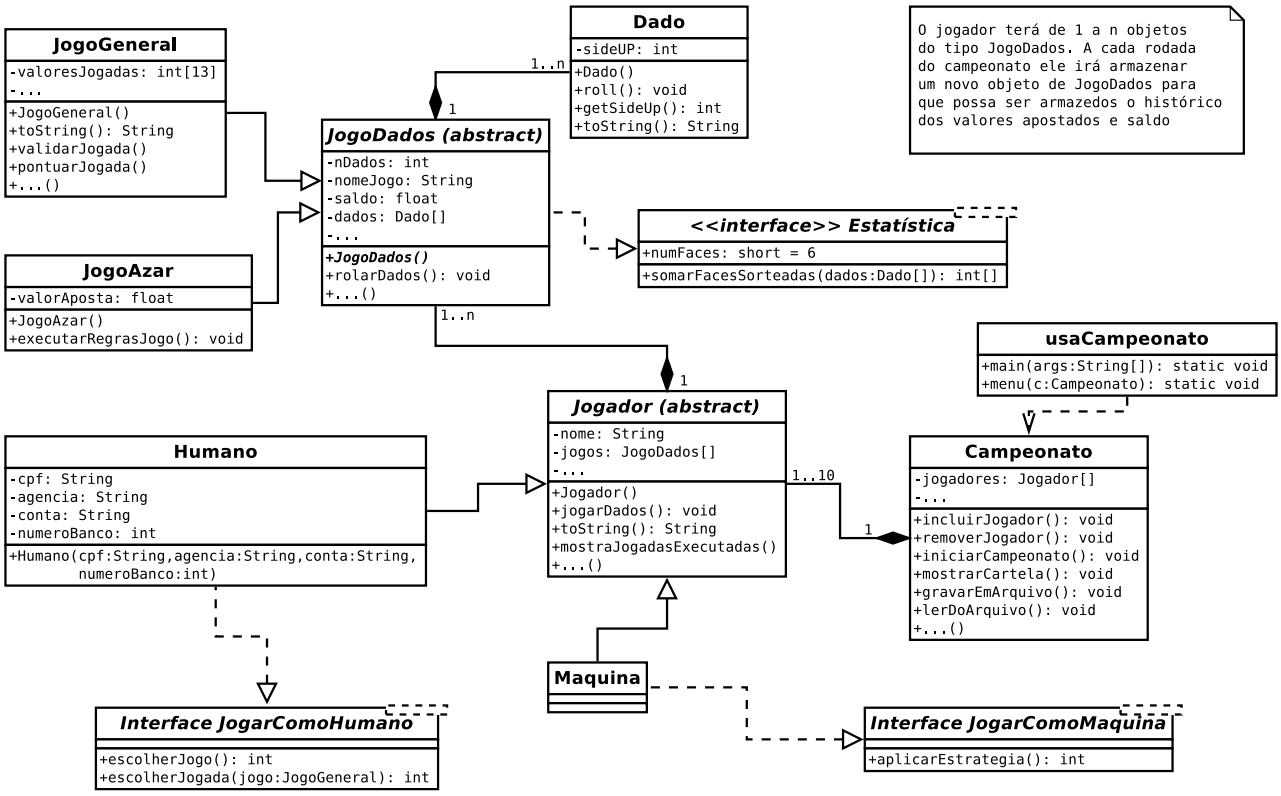

Figura 1: Esboço do Diagrama UML a ser seguido.

Com base no detalhamento anterior, faça:

1. Descreva o diagrama UML das classes do simulador de *Campeonato de Apostas em Jogos de Dados*, tomando como modelo o esboço apresentado na Figura 1 (gerar o arquivo pdf do diagrama).
2. Com base no diagrama UML (Fig. 1), desenvolva um aplicativo Java com um menu iterativo que permita ao usuário simular o campeonato, com no máximo dez jogadores (humanos ou máquinas):
   (a) Incluir jogador (solicitar nome e tipo [humano ou máquina?])
   (b) Remover jogador (pelo nome)

(c) Executar rodada(s) de aposta(s) [para cada jogador (apto), solicitar o valor da aposta e o jogo escolhido. executar o jogo, e ao final mostrar seu resultado (ganho ou perda) e o saldo resultante da jogada.]

(d) Imprimir saldo(s):
   - para todos os jogadores
   - apenas para os jogadores humanos
   - apenas para os jogadores máquinas

(e) Imprimir extratos dos resultados (valores das jogadas [jogo general], valor apostado, ganho ou perda) dos jogos, na(s) aposta(s) feitas pelos jogadores
   - para o jogo general, [para o jogo de azar] e [para ambos]
     * para todos os jogadores
     * apenas para os jogadores humanos
     * apenas para os jogadores máquinas

(e) Imprimir estatísticas [Exibe quantas vezes cada face de cada dado já fora sorteada]:
   - por jogador
   - por jogos escolhidos por cada jogador
   - total por jogos
   - total do campeonato

(f) Gravar os dados do campeonato em arquivo

(g) Ler os dados do campeonato em arquivo

(h) Sair da aplicação

Anexo 1: Jogo de Azar

Um jogo de azar faz uso de **dois dados** e possui a seguinte regra: O jogador lança os dois dados:

- Se a soma das faces dos dados for 7 ou 11 o jogador **ganha**;
- Se a soma for 2, 3 ou 12 o jogador **perde**;
- Se a soma obtida no primeiro lançamento de dados não for qualquer um dos valores acima, esta soma será tratada como o valor a ser buscado pelo jogador nos lançamentos subsequentes, ou seja, o jogador só irá ganhar se ele conseguir novamente atingir a soma obtida com o primeiro lançamento.

Exemplo:

```
Primeiro lançamento:
2 e 5 = 7
Jogador ganhou!  [encerra-se a jogada]
...
Nova jogada
Primeiro lançamento:
2 e 6 = 8
```

```
Número a ser buscado:  8
Segundo lançamento:
1 e 3 = 4
Terceiro lançamento:
5 e 2 = 7
Quarto lançamento:
3 e 5 = 8
Jogador ganhou!  [encerra-se a jogada]
...
Nova jogada
Primeiro lançamento:
6 e 3 = 9
Número a ser buscado:  9
Segundo lançamento:
5 e 5 = 10
Terceiro lançamento:
2 e 2 = 4
Quarto lançamento:
2 e 1 = 3
Jogador perdeu!  [encerra-se a jogada]
```

Anexo 2: Jogo General

Nesta segunda parte do trabalho, o jogo general seguirá como implementado no trabalho prático #01. Portanto, fará uso de **cinco dados** e o objetivo do jogo é marcar o maior número de pontos, em **treze jogadas**, o que compreende **uma rodada** do jogo. Assim, por meio da combinação dos treze arremessos, com cada resultado relacionado, por escolha do jogador, a uma das seguintes possibilidades de jogada:

Jogada de 1: um certo número de dados (de 0 a 5) marcando o número 1; sendo que a jogada vale mais pontos conforme a quantidade de dados que marcarem o número 1. Por exemplo: 1-1-1-4-5 vale 3 pontos.

Jogadas de 2, 3, 4, 5 e 6: correspondentes à jogada de 1 para os demais números. Por exemplo: 3-3-4-4-5 vale 6 pontos se for considerada uma jogada de 3; ou 8 pontos se for considerada uma jogada de 4; ou ainda 5 pontos se for uma jogada de 5.

Trinca (T): três dados marcando o mesmo número. Vale a soma dos 5 dados. Exemplo: 4-4-4-5-6 vale 23 pontos.

Quadra (Q): quatro dados marcando o mesmo número. Vale a soma dos 5 dados. Exemplo: 1-5-5-5-5 vale 21 pontos.

Full-hand (F) ou Full-house: uma trinca e um par (exemplo: 2-2-2-6-6). Vale 25 pontos para qualquer combinação.

Seqüência alta (S+): 2-3-4-5-6. Vale 30 pontos.

Seqüência baixa (S-): 1-2-3-4-5. Vale 40 pontos.

General (G): cinco dados marcando o mesmo número (por exemplo: 4-4-4-4-4). Vale 50 pontos.

Jogada aleatória (X) : qualquer combinação. Vale a soma dos 5 dados. Por exemplo: 1-4-4-5-6 vale 20 pontos.

Ao final dos 13 arremessos de dados, o jogador ganha o jogo se a soma dos valores das jogadas de 1 a 6, trinca, quadra, full-hand, sequências alta e baixa e general (jogadas de 1 a 12) for maior que o dobro do valor obtido na jogada aleatória (jogada 13).

Avaliação:

O trabalho será avaliado em função da:

- Correção (o aplicativo cumpre com as exigências?);
- Documentação (o aplicativo está devidamente comentado, **principalmente quanto às ocorrências de herança, herança múltipla e polimorfismo?**);
- Paradigma orientado a objetos (o aplicativo está seguindo os **princípios da programação OO: -encapsulamento, -associação de classes (composição ou agregação, herança, herança múltipla), -polimorfismo, -cada classe executando suas operações específicas, o esboço do diagrama UML proposto foi seguido?**).

  Observação: **não utilizar a API** (*Application Programmig Interface*) de estruturas de dados (**Coleções**), por exemplo, **ArrayList**, **LinkedList**, **HashSet**, **TreeSet**, etc da linguagem Java (ou qualquer outra escolhida).
- Modularidade (o aplicativo está bem estruturado onde necessário, com métodos (funções) parametrizados?);
- Robustez (o aplicativo não trava em tempo de execução?).

Detalhamento de itens a serem avaliados:

Item a ser cumprido	Atendeu?
Respeitar o princípio do encapsulamento de dados	
Usar modificadores de acesso adequados (**private** e **public**)	
Criar métodos "getters e setters" que forem necessários	
Criar métodos construtores parametrizados	
Fazer **sobrecarga** de pelo menos um método (qualquer um)	
Criar associação entre classes (Agregação ou Composição)	
O aplicativo não deve travar em tempo de execução	
Seguir o diagrama UML apresentado	
Não utilizar classes da API de coleções (estrutura de dados) da linguagem Java (ou outra escolhida)	
Ter pelo menos um atributo `final`	
Fazer uso da palavra reservada `this`	
Fazer uso do operador **instanceof**	
Ter pelo menos um atributo `static`	
Fazer uso de **classe abstrata**	
Fazer uso de **interface**	
Fazer uso do conceito de **herança, herança múltipla e polimorfismo**	
Não utilizar o modificador `protected`	
Apresentar o diagrama UML da aplicação	