Projeto Chatbot ou Boardgames

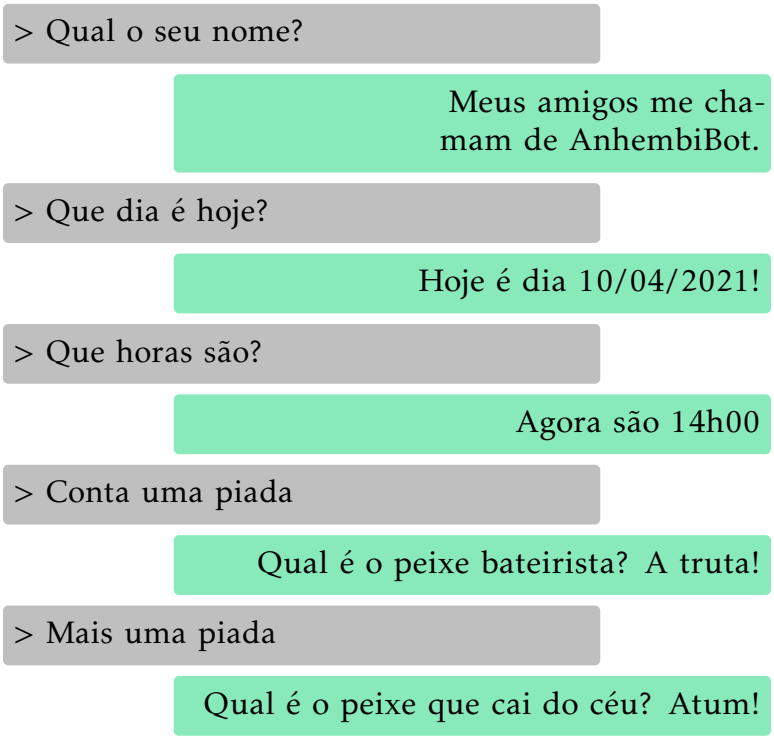
Programação Orientada a Objetos

1 Instruções

- A atividade ficará aberta no Blackboard até a data limite de entrega;
- A atividade deverá ser entregue exclusivamente pelo Blackboard;
- Os alunos que não tiverem acesso ao Blackboard deverão entrar em contato com o professor;
- A submissão da atividade deve ser feita em único arquivo zip. Este arquivo deve ter o nome completo de um dos integrantes do grupo, sem caracteres especiais, separado por underlines, ex: Joao_da_Silva.zip. Este zip deve conter todos os arquivos .java do seu projeto e um **arquivo .txt com o nome e RA de todos os integrantes do grupo**.
- Caso seu código não compile ou apresente erros de execução (crash, loop infinito, etc.) a nota daquele exercício será automaticamente é zerada;
- A atividade deve ser feita em grupos de **3 a 5 alunos**. Caso dois grupos submetam códigos iguais ou **muito semelhantes**, será considerado plágio. Consequentemente, ambos os alunos receberão zero na atividade inteira;
- Escolha **um dos temas** para implementar!

2 Tema: Chatbot

Neste projeto vocês deverão desenvolver um mini-chatbot que converse com o usuário via saída padrão utilizando os conceitos de programação orientada a objetos.



2.1 Classes Obrigatórias

As classes a seguir devem ser obrigatoriamente implementadas com todos os atributos e métodos descritos. Contudo, vocês podem e devem fazer outros atributos e métodos auxiliares para desenvolver o chatbot.

2.1.1 Classe abstrata Resposta

- Métodos:
  - `public abstract boolean verifica(String entrada)`
    - verifica se a entrada fornecida contém a palavra-chave esperado.
  - `public abstract String produz()`
    - produz a resposta do robô.

2.1.2 Classe Chatbot

- Métodos:
  - `public adiciona(Resposta r)`
    - adiciona uma resposta ao conjunto de respostas que aquele robô "conhece".
  - `public processar(String texto)`
    - verifica qual das respostas que ele "conhece" se encaixa naquele texto, em ordem, e responde adequadamente.

2.1.3 Classe RespostaSimples

Herda de Resposta

- Métodos:
  - O construtor recebe por parâmetro a palavra-chave que aciona a resposta, e um String que será o que o robô irá responder.

2.1.4 Classe RespostaAleatoria

Herda de Resposta

- Métodos:
  - O construtor recebe por parâmetro a palavra-chave que aciona a resposta, e uma coleção de Strings contendo as possíveis respostas. Na hora de produzir uma resposta, esta deve ser sorteada a partir da coleção fornecida.

2.1.5 Classe RespostaData

Herda de Resposta

- Métodos:
  - O construtor recebe por parâmetro a palavra-chave que aciona a resposta. Na hora de produzir uma resposta, deve retornar apenas a data atual.

2.1.6 Classe RespostaHora

Herda de Resposta

- Métodos:
  - O construtor recebe por parâmetro a palavra-chave que aciona a resposta. Na hora de produzir uma resposta, deve retornar apenas a hora atual.

2.1.7 Classe RespostaContador

Herda de Resposta

- Métodos:
  - O construtor recebe por parâmetro a palavra-chave que aciona a resposta. Na hora de produzir uma resposta, deve retornar quantas vezes aquela resposta foi acionada.

2.2 Simulação

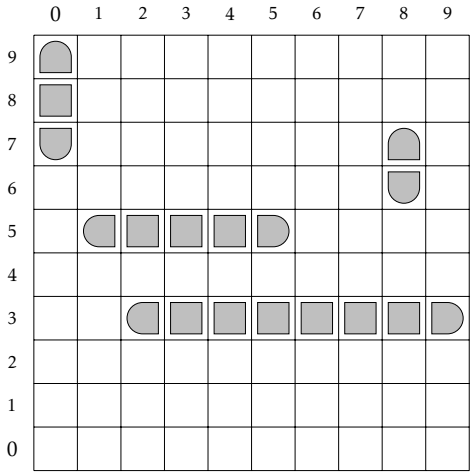
Segue um exemplo mostrando uma possível implementação do chatbot em que está sendo simulado uma conversa com um usuário.

```java
import java.util.ArrayList;
import java.util.Scanner;

public class Main {

  public static void main(String[] args) {

    Chatbot bot = new Chatbot();

    // criando as respostas
    Resposta rs1 = new RespostaSimples("nome", "Meus amigos me chamam de AnhembiBot.");
    Resposta rs2 = new RespostaData("dia");
    Resposta rs3 = new RespostaHora("hora");
    Resposta rs4 = new RespostaContador("conta");

    // criando as piadas
    ArrayList<String> piadas = new ArrayList<>();
    piadas.add("Qual é o peixe baterista? A truta!");
    piadas.add("Qual é o peixe que cai do céu? Atum!");
    Resposta rs5 = new RespostaAleatoria("piada", piadas);

    //adicionando as respostas ao chatbot
    bot.adiciona(rs1);
    bot.adiciona(rs2);
    bot.adiciona(rs3);
    bot.adiciona(rs4);
    bot.adiciona(rs5);

    System.out.println("------------------------------------");
    System.out.println("Iniciando o chatbot");
    System.out.println("------------------------------------");
    Scanner entrada = new Scanner(System.in);
    String texto;
    //loop infinito para ficar lendo as perguntas do usuario
    while (true) {
      System.out.print("> ");
      texto = entrada.nextLine();
      System.out.println(bot.processar(texto));
    }
  }
}
```

```
------------------------------------
Iniciando o chatbot
------------------------------------
> Qual o seu nome?
Meus amigos me chamam de AnhembiBot.
> Que dia é hoje?
Hoje é dia 10/04/2021!
> Que horas são?
Agora são 14h00
> Vamoz fazer uma conta
Ok! A conta atualmente está em 1
> Atualize a conta
Ok! A conta atualmente está em 2
> Mais uma conta
Ok! A conta atualmente está em 3
> Conte uma piada
Qual é o peixe baterista? A truta!
> Conte outra piada
Qual é o peixe baterista? A truta!
> Mais uma piada
Qual é o peixe que cai do céu? Atum!
>
```

3 Tema: Batalha Naval

A implementação de jogos é uma maneira simples de visualizar os benefícios da modelagem orientada a objetos. Neste projeto vocês deverão implementar um jogo de batalha naval utilizando programação orientada a objetos.



Vocês devem modelar um tabuleiro de batalha naval, juntamente com os navios e as tentativas de acertá-los. Nesse exemplo não há jogadores, apenas navios tentando acertar uns aos outros.

O jogo deverá ser simulado colocando ao menos um de cada tipo de embarcação diferente no tabuleiro (mais detalhes nas definições das classes). No seu jogo você precisa:

- Colocar os barcos no tabuleiro respeitando as bordas e espaços ocupados por outras embarcações.
- Simular uma tentativa de bombardear um alvo, mostrando na tela “O <nome da embarcação> foi destruído!”, “água!” ou “tiro inválido!”.
- Mudar a condição da embarcação para detruída quando atingida.
- Mostrar o jogo atualmente.

3.1 Classes Obrigatórias

As classes a seguir devem ser obrigatoriamente implementadas com todos os atributos e métodos descritos. Contudo, vocês podem e devem fazer outros atributos e métodos auxiliares para desenvolver o jogo.

3.1.1 BatalhaNaval

- Atributos:
  - `Tabuleiro tabuleiro`
    - Tabuleiro contendo as embarcações.
- Métodos
  - `public void imprimir()`
    - Imprime as informações do jogo
  - `public void atirar(Embarcacao atirador, int linhaAlvo, int colunaAlvo)`
    - Tenta atirar em um alvo no tabuleiro. Se o alvo for o próprio atirador ou estiver fora do tabuleiro, mostrar "tiro inválido!".
    - Se o atirador for uma embarcação já destruída, mostrar "A embarcação está destruída, ela não pode atirar mais!".
    - Se não houver nenhuma embarcação no local atingido, mostrar "água!".
  - `public BatalhaNaval(Tabuleiro tabuleiro)`
    - Construtor que recebe um objeto tabuleiro.

3.1.2 Tabuleiro

- Atributos:
  - `char[][] tabu`
    - Matriz representando o tabuleiro onde as embarcações estão posicionadas. Uma embarcação é representada pela sua primeira letra pela letra 'd' caso já tenha sido afundada.
  - `ArrayList<Embarcacao> barcos`
    - Lista contendo todos os objetos do tipo Embarcacao.

- Métodos:
  - `public void adicionar(Embarcacao barco)`
    - Adiciona a embarcação ao tabuleiro. Caso a embarcação fique em cima de outra ou fique fora do tabuleiro, não permita que ela seja adicionada e mostre na tela: "Posição inválida!"
  - `public void atualizarNoTabuleiro(Embarcacao barco)`
    - Método para atualizar o tabuleiro quando um barco é adicionado no tabuleiro ou atingido.
  - `public boolean posicaoValida(Embarcacao barco)`
    - Verifica se a posição em que a embarcação está sendo inserida é válida.
  - `public ArrayList<Embarcacao> getEmbarcacoes()`
    - Retorna ArrayList de embarcações no tabuleiro.
  - `public Tabuleiro()`
    - Construtor do tabuleiro que não recebe parâmetros.

3.1.3 Embarcacao (classe abstrata)

- Atributos:
  - `ArrayList<Integer> linhasOcupadas`
    - Números das linhas do tabuleiro ocupadas pelo objeto.
  - `ArrayList<Integer> colunasOcupadas`
    - Números das colunas do tabuleiro ocupadas pelo objeto.
  - `boolean destruido`
    - Atributo que verifica se a embarcação já foi atingida.
- Métodos:
  - `abstract void destruir()`
    - Muda o valor da variável destruida para true, atualiza o tabuleiro e mostra uma mensagem com o nome da embarcação destruída.
  - `boolean estaDestruida()`

- Retorna se essa embarcação já foi atingida.

- `Embarcacao(char orientacao, int linhaInicial, int colunaInicial)`
  - Construtor para a classe embarcação.
  - orientacao ('h' ou 'v') - determina se a embarcação está na horizontal ou na vertical no tabuleiro
  - linhaInicial - primeira linha ocupada no tabuleiro de cima para baixo pela embarcação.
  - colunaInicial - primeira coluna ocupada no tabuleiro da esquerda para a baixo pela embarcação.

3.1.4 Submarino

Herda da classe Embarcação. Embarcação com comprimento 2 no tabuleiro.

- Métodos:
  - `void destruir()`
    - Muda o valor da variável para destruída e mostra na tela "O submarino foi destruído!".
  - `String toString()`
    - Retorna "Submarinos".
  - `Submarino(char orientacao, int linhaInicial, int colunaInicial)`

3.1.5 Navio

Herda da classe Embarcação. Embarcação com comprimento 3 no tabuleiro.

- Métodos:
  - `void destruir()`
    - Muda o valor da variável para destruída e mostra na tela "O navio foi destruído!".
  - `String toString()`
    - Retorna "Navio".
  - `Navio(char orientacao, int linhaInicial, int colunaInicial)`

3.1.6 PortaAviao

Herda da classe Embarcação. Embarcação com comprimento 4 no tabuleiro.

- Métodos
  - `void destruir()`
    - Muda o valor da variável para destruída e mostra na tela "O porta aviões foi destruído!".
  - `String toString()`
    - Retorna "Porta aviões".
  - `PortaAviao(char orientacao, int linhaInicial, int colunaInicial)`

3.2 Simulação

Segue um exemplo mostrando uma possível implementação do jogo em que está sendo simulado uma batalha naval. Nele, cada tipo de embarcação é representada pela sua primeira letra, enquanto as embarcações destruídas são representadas pela letra 'd'.

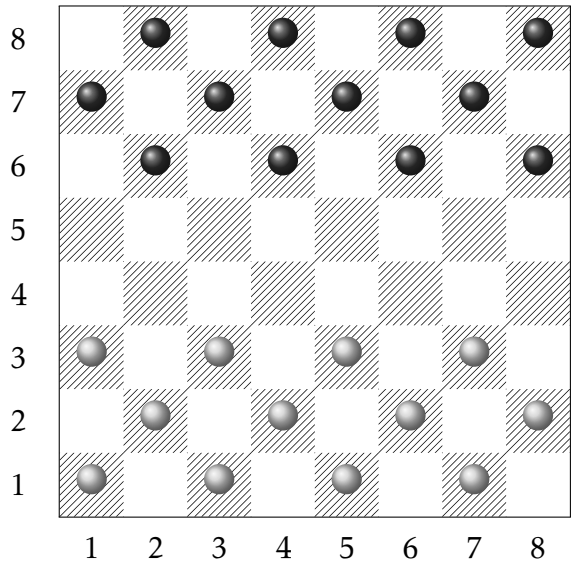
```java
public class Main {
    public static void main (String[] args) {
        // criando tabuleiro
        Tabuleiro tabuleiro = new Tabuleiro();

        // criando os barcos
        Embarcacao submarino = new Submarino('h', 4, 1);
        Embarcacao portaAviao = new PortaAviao('v', 0, 2);
        Embarcacao navio = new Navio('h', 4, 5);

        // adicionando os barcos
        tabuleiro.adicionar(submarino);
        tabuleiro.adicionar(navio);
        tabuleiro.adicionar(portaAviao);

        BatalhaNaval bn = new BatalhaNaval(tabuleiro);

        bn.imprimir();
        // submarino tenta atirar em (3,4)
        bn.atirar(navio, 3, 4);
        // navio tenta atirar em (0,0)
        bn.atirar(submarino, 0, 0);
        // portaAviao tenta atirar em (10,0)
        bn.atirar(portaAviao, 10, 0);
        // portaAviao tenta atirar em (4,1)
        bn.atirar(navio, 4, 1);
        bn.imprimir();
        // submarino tenta atirar em (4,1)
        bn.atirar(submarino, 4, 1);
        // portaAviao tenta atirar em (0,2)
        bn.atirar(portaAviao, 0, 2);
        bn.imprimir();
    }
}
```

```
| |0|1|2|3|4|5|6|7|8|9|
|0| | |p| | | | | | |
|1| | |p| | | | | | |
|2| | |p| | | | | | |
|3| | |p| | | | | | |
|4| |s|s| | |n|n|n| | |
|5| | | | | | | | | |
|6| | | | | | | | | |
|7| | | | | | | | | |
|8| | | | | | | | | |
|9| | | | | | | | | |
-------------------------------------------------------
Navio atirando na posição (3, 4):
água
-------------------------------------------------------
Submarino atirando na posição (0, 0):
água
-------------------------------------------------------
Porta avioes atirando na posição (10, 0):
tiro inválido!
-------------------------------------------------------
Navio atirando na posição (4, 1):
O submarino foi destruído!
| |0|1|2|3|4|5|6|7|8|9|
|0| | |p| | | | | | |
|1| | |p| | | | | | |
|2| | |p| | | | | | |
|3| | |p| | | | | | |
|4| |d|d| | |n|n|n| | |
|5| | | | | | | | | |
|6| | | | | | | | | |
|7| | | | | | | | | |
|8| | | | | | | | | |
|9| | | | | | | | | |
-------------------------------------------------------
Submarino atirando na posição (4, 1):
A embarcação está destruída, ela não pode atirar mai
-------------------------------------------------------
Porta avioes atirando na posição (0, 2):
tiro inválido!
| |0|1|2|3|4|5|6|7|8|9|
|0| | |p| | | | | | |
|1| | |p| | | | | | |
|2| | |p| | | | | | |
|3| | |p| | | | | | |
|4| |d|d| | |n|n|n| | |
|5| | | | | | | | | |
|6| | | | | | | | | |
|7| | | | | | | | | |
|8| | | | | | | | | |
|9| | | | | | | | | |
```

Neste exemplo, há um submarino, um navio e um porta aviões. Os dois primeiros estão posicionados na horizontal e o último na vertical. O navio e o submarino atiram na água, já o porta aviões tenta atirar fora do tabuleiro, o que provoca um tiro inválido. Na segunda rodada, o navio atira e destrói o submarino; esse responde com um tiro inválido, uma vez que já foi destruído; por último o porta aviões tenta atirar em si mesmo, o que também provoca um tiro inválido.

4 Tema: Damas

Neste projeto vocês deverão implementar um jogo de damas utilizando os conceitos de programação orientada a objetos.



O objetivo é que vocês consigam modelar as classes descritas a seguir de tal forma que seja possível simular uma partida de damas considerando o ponto de vista de um usuário de uma biblioteca de damas.

O jogo pode ser simulado utilizando somente duas peças de um jogo de damas clássico. Seu jogo deve ser capaz de:

- Movimentar as peças respeitando as regras de movimentação de damas clássico.
- Verificar se o movimento é válido.
- Eliminar uma peça ao fazer um movimento que elimine a peça do adversário.
- Mostrar o status do jogo.

4.1 Classes Obrigatórias

As classes a seguir devem ser obrigatoriamente implementadas com todos os atributos e métodos descritos. Contudo, vocês podem e devem fazer outros atributos e métodos auxiliares para desenvolver o jogo.

4.1.1 Damas

- Atributos:
  - `Tabuleiro tabuleiro`
  - `Jogador jogador1;`
  - `Jogador jogador2;`
- Métodos
  - `public void adicionar(Tabuleiro tabuleiro)`
  - `public void adicionar(Jogador jogador);`
  - `public void imprimir()`
    - Imprime as informações do jogo
  - `public void jogar(Jogador jogador, String nomePeca, String posicao)`
    - Faz uma movimentação no jogo. Ao final da execução deste método uma peça terá movido de lugar ou deverá ter uma mensagem informando a razão do movimento não ter sido realizado.
  - `public Damas(Tabuleiro t, Jogador j1, Jogador j2)`
    - Construtor que recebe um objeto tabuleiro e dois jogadores.

4.1.2 Tabuleiro

- Atributos:
  - `ArrayList<Peca> pecas`
    - Lista contendo objetos do tipo peça
- Métodos:

- `public void adicionar(Peca p)`
- `public boolean isMovimentoValido(posicao)`
  - Verifica se o movimento realizado obedece os limites físicos do tabuleiro.
- `public void imprimirConfiguracaoAtual()`
  - Imprime o status atual do jogo. Deve ser mostrado as posições atuais de cada peça.
- `public boolean isMovimentoEliminatorio(Peca p, String posicao)`
  - Verifica se o movimento eliminou a peça do adversário.
- `public void remover(Peca p)`
  - Remove a peça do jogo.

4.1.3 Peça (classe abstrata)

- Atributos:
  - `String nome;`
  - `String cor;`
  - `double posicaoAtual;`
- Métodos:
  - `public void atualizarPosicao(String posicao)`
  - `public abstract boolean isMovimentoValido(String posicao)`
    - Método abstrato para verificar se a posição solicitada é válida dado as regras de movimentação daquela peça.
  - `public Peca(String nome, String cor, String posicao)`
    - Construtor que recebe um nome, uma cor e uma posição inicial.

4.1.4 PecaSimples

Herda da classe Peca. Deve implementar o método isMovimentoValido(String posicao).

4.1.5 Dama

Herda da classe Peca. Deve implementar o método isMovimentoValido(String posicao).

4.1.6 Jogador

- Atributos:
  - `String nome;`
  - `String corPeca;`
  - `double pontuacao;`
- Métodos:
  - `public ArrayList<Pecas> listarPecasAdquiridas()`
    - Retorna uma lista contendo as peças do adversário que foram adquiridas ao longo da partida.
  - `public Jogador(String nome, String corPeca)`
    - Construtor que recebe um nome e uma cor de peça.

4.1.7 Main

- métodos:
  - `public static void main(String[] args)`
    - Deve ser criado todos os objetos necessários para simular uma partida e mostrar a execução de alguns movimentos do jogo.

4.2 Simulação

Segue um exemplo mostrando uma possível implementação do jogo em que está sendo simulado uma partida de damas. Vocês podem desenhar o tabuleiro de outra forma, não precisa ser igual ao da figura. Sendo assim, você deve mostrar através da saída padrão que o seu jogo está funcionando.

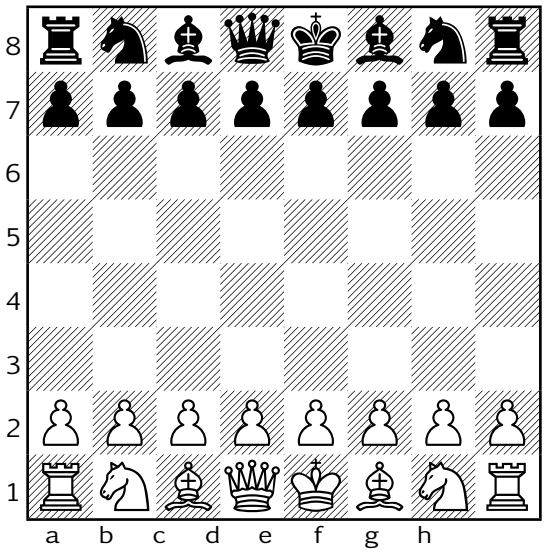
```java
public class Main {
  public static void main(String[] args) {
    // Configurando o tabuleiro
    Tabuleiro tabuleiro = new Tabuleiro();

    // criando as pecas
    Peca dama1 = new Dama("d1", "branca", "51");
    Peca pecaSimples1 = new PecaSimples("p1", "branca", "33");
    Peca dama2 = new Dama("D1", "preta", "17");
    Peca pecaSimples2 = new PecaSimples("P1", "preta", "66");

    // adicionando as pecas ao tabuleiro
    tabuleiro.adicionarPeca(dama1);
    tabuleiro.adicionarPeca(pecaSimples1);
    tabuleiro.adicionarPeca(dama2);
    tabuleiro.adicionarPeca(pecaSimples2);

    // criando os jogadores
    Jogador jogador1 = new Jogador("Fulano", "branca");
    Jogador jogador2 = new Jogador("Ciclano", "preta");

    // criando o jogo
    Damas damas = new Damas(tabuleiro, jogador1, jogador2);

    // imprimindo a configuracao inicial
    System.out.println("----------------------------");
    System.out.println("Configuracao inicial");
    System.out.println("----------------------------");
    damas.imprimir();

    // simulando uma partida
    damas.jogar(jogador1, "d1", "73");
    damas.imprimir();
    damas.jogar(jogador2, "P1", "75");
    damas.imprimir();
  }
}
```

```
----------------------------
Configuracao inicial
----------------------------
Jogador: Ciclano -- pecas: preta
     1    2    3    4    5    6    7    8
  +----+----+----+----+----+----+----+----+
8 |    |    |    |    |    |    |    |    | 8
  +----+----+----+----+----+----+----+----+
7 | D1 |    |    |    |    |    |    |    | 7
  +----+----+----+----+----+----+----+----+
6 |    |    |    |    |    | P1 |    |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    |    |    |    |    |    | 5
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    |    |    |    |    | 4
  +----+----+----+----+----+----+----+----+
3 |    |    | p1 |    |    |    |    |    | 3
  +----+----+----+----+----+----+----+----+
2 |    |    |    |    |    |    |    |    | 2
  +----+----+----+----+----+----+----+----+
1 |    |    |    |    | d1 |    |    |    | 1
  +----+----+----+----+----+----+----+----+
     1    2    3    4    5    6    7    8
Jogador: Fulano -- pecas: branca


--------------------------------------
movimento: Fulano  peca: d1  posicao: 73
--------------------------------------
Jogador: Ciclano -- pecas: preta
     1    2    3    4    5    6    7    8
  +----+----+----+----+----+----+----+----+
8 |    |    |    |    |    |    |    |    | 8
  +----+----+----+----+----+----+----+----+
7 | D1 |    |    |    |    |    |    |    | 7
  +----+----+----+----+----+----+----+----+
6 |    |    |    |    |    | P1 |    |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    |    |    |    |    |    | 5
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    |    |    |    |    | 4
  +----+----+----+----+----+----+----+----+
3 |    |    | p1 |    |    |    | d1 |    | 3
  +----+----+----+----+----+----+----+----+
2 |    |    |    |    |    |    |    |    | 2
  +----+----+----+----+----+----+----+----+
1 |    |    |    |    |    |    |    |    | 1
  +----+----+----+----+----+----+----+----+
     1    2    3    4    5    6    7    8
Jogador: Fulano -- pecas: branca


--------------------------------------
movimento: Ciclano  peca: P1  posicao: 75
--------------------------------------
Jogador: Ciclano -- pecas: preta
     1    2    3    4    5    6    7    8
  +----+----+----+----+----+----+----+----+
8 |    |    |    |    |    |    |    |    | 8
  +----+----+----+----+----+----+----+----+
7 | D1 |    |    |    |    |    |    |    | 7
  +----+----+----+----+----+----+----+----+
6 |    |    |    |    |    |    |    |    | 6
  +----+----+----+----+----+----+----+----+
5 |    |    |    |    |    |    | P1 |    | 5
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    |    |    |    |    | 4
  +----+----+----+----+----+----+----+----+
3 |    |    | p1 |    |    |    | d1 |    | 3
  +----+----+----+----+----+----+----+----+
2 |    |    |    |    |    |    |    |    | 2
  +----+----+----+----+----+----+----+----+
1 |    |    |    |    |    |    |    |    | 1
  +----+----+----+----+----+----+----+----+
     1    2    3    4    5    6    7    8
Jogador: Fulano -- pecas: branca
```

Neste exemplo, as peças brancas estão representadas por letras minúsculas e as peças pretas por letras maiúsculas. Ambos os jogadores possuem uma peça simples e uma dama. Inicialmente, mostra-se o tabuleiro em sua configuração inicial e logo em seguida o jogador 1 faz um movimento o qual move a sua dama para a posição 73. Em seguida, o jogador 2 faz seu lance ao movimentar a sua peça simples para a posição 75.

5 Tema: Xadrex

Neste projeto vocês deverão implementar um jogo de xadrez utilizando os conceitos de programação orientada a objetos.



O objetivo é que vocês consigam modelar as classes descritas a seguir de tal forma que seja possível simular uma partida de xadrez considerando o ponto de vista de um usuário de uma biblioteca de xadrez.

O jogo deverá ser simulado utilizando ao menos duas peças de xadrez clássico. Seu jogo deve ser capaz de:

- Movimentar as peças respeitando as regras de movimentação do xadrez clássico.
- Verificar se o movimento é válido
- Eliminar uma peça ao fazer um movimento que elimine a peça do adversário.
- Mostrar o status do jogo

5.1 Classes Obrigatórias

As classes a seguir devem ser obrigatoriamente implementadas com todos os atributos e métodos descritos. Contudo, vocês podem e devem fazer outros atributos e métodos auxiliares para desenvolver o jogo.

5.1.1 Xadrez

- Atributos:
  - `Tabuleiro tabuleiro`
  - `Jogador jogador1;`
  - `Jogador jogador2;`
- Métodos
  - `public void adicionar(Tabuleiro tabuleiro)`
  - `public void adicionar(Jogador jogador);`
  - `public void imprimir()`
    - Imprime as informações do jogo
  - `public void jogar(Jogador jogador, String nomePeca, String posicao)`
    - Faz uma movimentação no jogo. Ao final da execução deste método uma peça terá movido de lugar ou deverá ter uma mensagem informando a razão do movimento não ter sido realizado.
  - `public Xadrez(Tabuleiro t, Jogador j1, Jogador j2)`
    - Construtor que recebe um objeto tabuleiro e dois jogadores.

5.1.2 Tabuleiro

- Atributos:
  - `ArrayList<Peca> pecas`
    - Lista contendo objetos do tipo peça
- Métodos:

- `public void adicionar(Peca p)`
- `public boolean isMovimentoValido(posicao)`
  - Verifica se o movimento realizado obedece os limites físicos do tabuleiro.
- `public void imprimirConfiguracaoAtual()`
  - Imprime o status atual do jogo. Deve ser mostrado as posições atuais de cada peça.
- `public boolean isMovimentoEliminatorio(Peca p, String posicao)`
  - Verifica se o movimento eliminou a peça do adversário.
- `public void remover(Peca p)`
  - Remove a peça do jogo.

5.1.3 Peça (classe abstrata)

- Atributos:
  - `String nome;`
  - `String cor;`
  - `double posicaoAtual;`
- Métodos:
  - `public void atualizarPosicao(String posicao)`
  - `public abstract boolean isMovimentoValido(String posicao)`
    - Método abstrato para verificar se a posição solicitada é válida dado as regras de movimentação daquela peça.
  - `public Peca(String nome, String cor, String posicao)`
    - Construtor que recebe um nome, uma cor e uma posição inicial.

5.1.4 Jogador

- Atributos:
  - `String nome;`

- `String corPeca;`
- `double pontuacao;`

- Métodos:
  - `public ArrayList<Pecas> listarPecasAdquiridas()`
    - Retorna uma lista contendo as peças do adversário que foram adquiridas ao longo da partida.
  - `public Jogador(String nome, String corPeca)`
    - Construtor que recebe um nome e uma cor de peça.

5.1.5 Main

- métodos:
  - `public static void main(String[] args)`
    - Deve ser criado todos os objetos necessários para simular uma partida e mostrar a execução de alguns movimentos do jogo.

5.2 Classes opcionais

Não será necessário implementar todas as peças de um jogo de xadrez clássico. Porém, **ao menos duas peças devem ser implementas**. Lembrem-se que cada peça possui sua regra de movimentação.

5.2.1 Peão

Herda da classe Peca. Deve implementar o método isMovimentoValido(String posicao).

5.2.2 Bispo

Herda da classe Peca. Deve implementar o método isMovimentoValido(String posicao).

5.2.3 Torre

Herda da classe Peca. Deve implementar o método isMovimentoValido(String posicao).

5.2.4 Cavalo

Herda da classe Peca. Deve implementar o método isMovimentoValido(String posicao).

5.2.5 Rainha

Herda da classe Peca. Deve implementar o método isMovimentoValido(String posicao).

5.2.6 Rei

Herda da classe Peca. Deve implementar o método isMovimentoValido(String posicao).

5.3 Simulação

Segue um exemplo mostrando uma possível implementação do jogo em que está sendo simulado uma partida de xadrez. Vocês podem desenhar o tabuleiro de outra forma, não precisa ser igual ao da figura. Sendo assim, você deve mostrar através da saída padrão que o seu jogo está funcionando.

```
public class Main {
  public static void main(String[] args) {
    // Configurando o tabuleiro
    Tabuleiro tabuleiro = new Tabuleiro();

    // criando as pecas
    Peca cavalo1 = new Cavalo("c", "branca", "21");
    Peca peao1 = new Peao("p", "branca", "12");
    Peca cavalo2 = new Cavalo("C", "preta", "78");
    Peca peao2 = new Peao("P", "preta", "57");

    // adicionando as pecas ao tabuleiro
    tabuleiro.adicionarPeca(cavalo1);
    tabuleiro.adicionarPeca(peao1);
    tabuleiro.adicionarPeca(cavalo2);
    tabuleiro.adicionarPeca(peao2);

    // criando os jogadores
    Jogador jogador1 = new Jogador("Fulano", "branca");
    Jogador jogador2 = new Jogador("Ciclano", "preta");

    // criando o jogo
    Xadrez xadrez = new Xadrez(tabuleiro, jogador1, jogador2);

    // imprimindo a configuracao inicial
    System.out.println("---------------------------");
    System.out.println("Configuracao inicial");
    System.out.println("---------------------------");
    xadrez.imprimir();

    // simulando uma partida
    xadrez.jogar(jogador1, "c", "33");
    xadrez.imprimir();
    xadrez.jogar(jogador2, "P", "55");
    xadrez.imprimir();
  }
}
```

```
---------------------------
Configuracao inicial
---------------------------
Jogador: Ciclano -- pecas: preta
    1   2   3   4   5   6   7   8
  +---+---+---+---+---+---+---+---+
8 |   |   |   |   |   |   | C |   | 8
  +---+---+---+---+---+---+---+---+
7 |   |   |   |   | P |   |   |   | 7
  +---+---+---+---+---+---+---+---+
6 |   |   |   |   |   |   |   |   | 6
  +---+---+---+---+---+---+---+---+
5 |   |   |   |   |   |   |   |   | 5
  +---+---+---+---+---+---+---+---+
4 |   |   |   |   |   |   |   |   | 4
  +---+---+---+---+---+---+---+---+
3 |   |   |   |   |   |   |   |   | 3
  +---+---+---+---+---+---+---+---+
2 | p |   |   |   |   |   |   |   | 2
  +---+---+---+---+---+---+---+---+
1 |   | c |   |   |   |   |   |   | 1
  +---+---+---+---+---+---+---+---+
    1   2   3   4   5   6   7   8
Jogador: Fulano -- pecas: branca


-------------------------------------
movimento: Fulano  peca: c  posicao: 33
-------------------------------------
Jogador: Ciclano -- pecas: preta
    1   2   3   4   5   6   7   8
  +---+---+---+---+---+---+---+---+
8 |   |   |   |   |   |   | C |   | 8
  +---+---+---+---+---+---+---+---+
7 |   |   |   |   | P |   |   |   | 7
  +---+---+---+---+---+---+---+---+
6 |   |   |   |   |   |   |   |   | 6
  +---+---+---+---+---+---+---+---+
5 |   |   |   |   |   |   |   |   | 5
  +---+---+---+---+---+---+---+---+
4 |   |   |   |   |   |   |   |   | 4
  +---+---+---+---+---+---+---+---+
3 |   |   | c |   |   |   |   |   | 3
  +---+---+---+---+---+---+---+---+
2 | p |   |   |   |   |   |   |   | 2
  +---+---+---+---+---+---+---+---+
1 |   |   |   |   |   |   |   |   | 1
  +---+---+---+---+---+---+---+---+
    1   2   3   4   5   6   7   8
Jogador: Fulano -- pecas: branca


-------------------------------------
movimento: Ciclano  peca: P  posicao: 55
-------------------------------------
Jogador: Ciclano -- pecas: preta
    1   2   3   4   5   6   7   8
  +---+---+---+---+---+---+---+---+
8 |   |   |   |   |   |   | C |   | 8
  +---+---+---+---+---+---+---+---+
7 |   |   |   |   |   |   |   |   | 7
  +---+---+---+---+---+---+---+---+
6 |   |   |   |   |   |   |   |   | 6
  +---+---+---+---+---+---+---+---+
5 |   |   |   |   | P |   |   |   | 5
  +---+---+---+---+---+---+---+---+
4 |   |   |   |   |   |   |   |   | 4
  +---+---+---+---+---+---+---+---+
3 |   |   | c |   |   |   |   |   | 3
  +---+---+---+---+---+---+---+---+
2 | p |   |   |   |   |   |   |   | 2
  +---+---+---+---+---+---+---+---+
1 |   |   |   |   |   |   |   |   | 1
  +---+---+---+---+---+---+---+---+
    1   2   3   4   5   6   7   8
Jogador: Fulano -- pecas: branca
```

Neste exemplo, as peças brancas estão representadas por letras minúsculas e as peças pretas por letras maiúsculas. Ambos os jogadores possuem um cavalo e um peão. Inicialmente, mostra-se o tabuleiro em sua configuração inicial e logo em seguida o jogador 1 faz um movimento o qual move o cavalo para a posição 33. Em seguida, o jogador 2 faz seu lance ao movimentar o peão para a posição 55.