Relatório Projeto 2 e 3

Caio Fernando Peres - 769298

Neste projeto foi implementado um jogo de xadrez em java.
Foram utilizados todos os conceitos de orientação a objetos cabíveis.
Neste documento consta um relatório do desenvolvimento de todas as classes e métodos.

Descrição das classes:

- Classe Abstrata Peca:
    - Possui atributos inteiros capturada e cor.
    - Possui métodos abstratos checaMovimento() e desenho(). Tais métodos devem ser implementados em todas as peças pois são diferentes em todas elas.
    - Possui métodos getCapturada(), setCapturada() e getCor(). No setCapturada() foi feito um tratamento em caso de exceção.
    - O construtor da classe atribui a cor recebida ao atributo da classe no momento da inicialização do objeto filho. Foi feito um tratamento caso ocorra uma exceção.
    - Não pode ser instanciada, apenas servir como progenitora, e precisa ter os métodos abstratos implementados.

- Classes de peças específicas:

    - Todas herdam da classe Peca.
    - O método checaMovimento() é diferente em todas as classes.
    - O método desenho() é o mesmo em todas as classes, apenas trocando a letra que será mostrada com a respectiva primeira letra do nome da Classe. Em maiusculo caso a cor de sua peça seja 1 e minusculo caso seja 2. A cor da peça é acessada através do método getCor() da classe mãe. desenho() precisa ser implementado individualmente em cada peça.

    - Rei:
        - Possui atributos herdados capturada e cor.
        - Possui a implementação dos métodos abstratos checaMovimento() e desenho().
        - Possui métodos herdados setCapturada(), getCapturada() e getCor().
        - O método checaMovimento() é específico da classe, pois checa se o movimento compreende em qualquer direção, andando apenas uma casa.
        - Da mesma forma, o método desenho() mostrará na tela o desenho específico da peça.
        - O construtor chama o construtor da classe mãe.

    - Dama:
        - Possui atributos herdados capturada e cor.
        - Possui a implementação dos métodos abstratos checaMovimento() e desenho().
        - Possui métodos herdados setCapturada(), getCapturada() e getCor().
        - O método checaMovimento() é específico da classe, pois checa se o movimento compreende em qualquer direção, quantas casas quiser.
        - Da mesma forma, o método desenho() mostrará na tela o desenho específico da peça.
        - O construtor chama o construtor da classe mãe.

    - Bispo:
        - Possui atributos herdados capturada e cor.

    - Possui métodos implementados checaMovimento() e desenho().
    - Possui a implementação dos métodos abstratos checaMovimento() e desenho().
    - O método checaMovimento() é específico da classe, pois checa se o movimento compreende nas diagonais, quantas casas quiser.
    - Da mesma forma, o método desenho() mostrará na tela o desenho específico da peça.
    - O construtor chama o construtor da classe mãe.

- Cavalo:
    - Possui atributos herdados capturada e cor.
    - Possui a implementação dos métodos abstratos checaMovimento() e desenho().
    - Possui métodos herdados setCapturada(), getCapturada() e getCor().
    - O método checaMovimento() é específico da classe, pois checa se o movimento compreende em L, com um número específico de casas.
    - Da mesma forma, o método desenho() mostrará na tela o desenho específico da peça.
    - O construtor chama o construtor da classe mãe.

- Torre:
    - Possui atributos herdados capturada e cor.
    - Possui a implementação dos métodos abstratos checaMovimento() e desenho().
    - Possui métodos herdados setCapturada(), getCapturada() e getCor().
    - O método checaMovimento() é especifico da classe, pois checa se o movimento compreende na horizontal e na vertical, quantas casas quiser.
    - Da mesma forma, o método desenho() mostrará na tela o desenho específico da peça.
    - O construtor chama o construtor da classe mãe.

- Peao:
    - Possui atributos herdados capturada e cor.
    - Possui a implementação dos métodos abstratos checaMovimento() e desenho().
    - Possui métodos herdados setCapturada(), getCapturada() e getCor().

    - O método checaMovimento() é especifico da classe, pois checa se o movimento compreende 2 casas apenas na primeira jogada, ou 1 casa no resto do jogo, ou na diagonal (apenas para comer outra peça, mas isso quem checa é o tabuleiro).
    - Da mesma forma, o método desenho() mostrará na tela o desenho específico da peça.
    - O construtor chama o construtor da classe mãe.
    - É o único que utiliza a cor para a checagem do movimento.

- Classes para o funcionamento do jogo:

    - Gerenciador
        - Possui apenas o método main.
        - Inicia o jogo com os nomes dos jogadores, ao instanciar um objeto Jogo, e gerencia as jogadas.
        - Trata exceções e solicita novas entradas ao jogador.

    - Tabuleiro
        - Possui o atributo posicoes do tipo Posicao, em forma de matriz de posições, instanciada com tamanho fixo de 8x8 posições.
        - Possui construtor para instanciar posições para a matriz de posições, bem como inicializar tais posições com linha, coluna (em caracter) e cor. O construtor também chamará um método para inicializar as peças recebidas por parâmetro. Caso o usuário opte por carregar a partir de um arquivo, a inicialização das peças não acontece para que possa ser inicializado a partir do arquivo.
        - Possui método inicializaPosicoes() que irá inicializar as posições do tabuleiro a partir de uma chamada do construtor.
        - Possui método inicializaPecas() que irá inicializar as peças no tabuleiro a partir de uma chamada do construtor e com o conjunto de peças recebido da classe Jogo. Caso o usuário opte por carregar o jogo salvo do arquivo, esse método não será chamado.
        - Possui método checaMovimento() que verifica se o movimento é válido, checando se pertence aos limites do tabuleiro e se a posição de destino é a mesma da origem. Também faz verificações se a peça na posição de origem é nula, se a peça é do próprio jogador que efetuou a jogada, se o movimento da peça é valido, chamando o método da peça especifica e se o caminho está livre para as peças que não pulam.

- Possui método checaCaminho() que verifica se o caminho está livre, sendo usado em conjunto com a verificação do movimento da peça.
- Possui método checaXeque(), que verifica se o rei adversário está em xeque após a jogada. Se estiver, chama o método consegueResolver() para ver se é possivel resolver o xeque. Caso não seja, é um caso de xeque-mate.
- Possui método checaXequeSimulacao(), que verifica se uma posição simulada está em xeque (no alcance de alguma peça). Pode ser usada para o rei ou para outra peça.
- Possui método consegueResolver(), que verifica se o xeque pode ser resolvido através de varias operações algébricas para identificar se existe uma opção para resolver o xeque através de movimentação do Rei, se a peça que está dando xeque pode ser comida por alguma outra ou se existe alguma peça que pode deter o ataque entrando na frente.
- Possui método podiaResolver(), que verifica se a peça que o usuário quis mudar, poderia ter resolvido o xeque antes de efetuar o movimento. Esse método foi necessário pois o planejamento do método checaXeque() envolveu a identificação do xeque após a jogada. Portanto, caso o usuário fizesse uma jogada com a peça que era a única que poderia salvar o rei do ataque, seria um xeque-mate. Dessa forma, esse método é utilizado na classe Jogo para fazer uma verificação antes e depois. Caso ele poderia ter resolvido, a jogada é retornada para que o usuário possa resolver o xeque.
- Possui método getDesenho(), que retorna o desenho da peça caso a posição dada como parâmetro esteja dentro dos limites do tabuleiro e exista uma peça. Caso não exista uma peça, é retornado 0. Caso esteja fora dos limites do tabuleiro, é considerado uma exceção e também tratada retornando 0.
- Possui método setPeca(), que coloca uma peça em determinada posicao do tabuleiro, a partir dos parâmetro do movimento. Caso o movimento envolva comer uma peça, retorna a peça comida. Caso não coma, retorna null. Dessa forma o Jogo consegue saber o que aconteceu e colocar a peça comida como capturada. Já foi checado a validade dos argumentos antes desse método ser chamado.
- Possui outro método chamado setPeca(), que insere uma peça recebida como parâmetro em determinada posição do tabuleiro, com sua linha e coluna também recebida por parâmetro.
- Possui método desenhar() que desenha na tela o tabuleiro atual.

- Posicao
    - Possui os atributos inteiros linha e cor, e o atributo de caracter coluna, todos declarados como final. Possui também um atributo do tipo Peca.
    - Como foram declarados como final, não foi possivel utilizar if ou try, para tratar uma possivel exceção, pois é mostrado um erro de compilação por estarmos alterando mais de uma vez uma variável final. Entretanto houveram verificações antes dessa chamada, portanto isso não deve ser um problema.
    - Possui um construtor que inicializará os atributos com os valores recebidos.
    - Possui métodos getPeca() e setPeca(). O setPeca() deve aceitar peça nula na posicao, então não havia necessidade de tratar uma possível exceção.

- Jogo
    - Possui os atributos jogador1 e jogador2 do tipo Jogador, os atributos inteiros vez e estado, o atributo tabuleiro do tipo Tabuleiro e um vetor do tipo Peca de 32 posições, que é o conjunto de peças de Jogo.
    - Possui duas possibilidades de construtores. Ambos irão inicializar as pecas a partir de um método chamado inicializarPecas(), e o tabuleiro com o vetor de pecas, e os jogadores com os nomes e o conjunto de pecas, a vez começando com 1 pois o branco sempre começa no xadrez e estado com 0. Porém um deles irá criar um novo jogo recebendo os nomes dos jogadores como parâmetro e o outro a partir de um arquivo salvo. Nesse caso, utilizará os métodos carregarPecasArq() para carregar as peças que estão em jogo e insereTabuleiroArq() para inserir de fato no tabuleiro. Caso o arquivo não exista, será criado um novo jogo normalmente. Foram tratadas as exceções cabíveis.
    - Possui o método jogada(), que fará a jogada do jogador a partir do programa principal. Nesse método, é preciso converter as linhas recebidas para índices reais de vetores, uma vez que o tabuleiro não começa com a linha 0. Além disso, também é preciso converter os caracteres recebidos das colunas para valores reais de índice, e nesse processo já é feita a verificação dos valores das colunas. Foram tratadas as exceções cabíveis. Depois, checa o movimento da peça através do método checaMovimento() do tabuleiro. Caso o movimento seja válido,

também é feito a verificação se o movimento poderia ter resolvido o xeque antes da jogada, através do método podiaResolver(). Essa verificação é necessária pois, como foi descrito na classe Tabuleiro, o planejamento do método checaXeque() envolveu a identificação do xeque após a jogada. Portanto, aqui está sendo feita uma verificação antes e depois. Após isso, é inserido a peça da jogada no tabuleiro através do método setPeca() do tabuleiro. O retorno do método é guardado e verificado se é nulo. Caso não seja, a peça é então procurada no conjunto de peças do Jogo, que é o mesmo conjunto de peças dos Jogadores. Após isso, é feito a checagem dos xeques da vez e do adversário, e feita verificações para saber se o xeque foi resolvido, se o jogador se colocou em xeque ou xeque-mate ou se o jogo acabou com o xeque-mate. Caso seja xeque-mate, o jogo é finalizado. Caso o movimento tenha sido inválido, é retornado ao programa principal mantendo a vez do jogador para que jogue novamente. Ao final do método, o estado muda de acordo com a checagem acima, troca a vez do jogador, salva e desenha o tabuleiro no console chamando o método desenhar do tabuleiro.

- Possui método salvar(), que salva o jogo a cada jogada feita, em um arquivo chamado jogosalvo.txt. Foi tratada a exceção cabível (exceção no arquivo).
- Possui método carregarPecasArq() que carrega todas as peças em jogo e coloca valor de capturada nas peças que não estavam no arquivo salvo. Foi tratada a exceção cabível.
- Possui método insereTabuleiroArq() que de fato insere as peças que foram carregadas pelo método anterior no tabuleiro. Foi tratada a exceção cabível.
- Possui método setCapturada(), que auxilia o método setPeca() do tabuleiro. Caso o método setPeca() retorne uma peça que foi capturada, é então chamado o método setCapturada() para que defina o status de capturada à peca.
- Possui método estático limparTela(), que é uma forma de limpar a tela dos terminais do Windows ou Linux, como forma de refinar a interação com o usuário.
- Possui método getNomeJogador() que retorna o nome do jogador da vez.
- Possui método setVez() que coloca a vez de acordo com um intervalo válido, tratando a exceção que poderia ocorrer.
- Possui método setEstado(), que atribui um estado para o atributo estado da classe, tratando a exceção que poderia ocorrer.

- Jogador
    - Possui atributo nome do tipo String.
    - Possui um vetor de pecas do tipo Peca.
    - Possui um construtor que atribui uma string para o nome, e seleciona qual das metades do vetor de peças recebido como parâmetro pela classe Jogo irá atribuir para o jogador.
    - Possui método getNome(), que retorna o nome do jogador.
    - Possui método setNome(), que define um nome para o atributo nome.

Exceções:

Considerando os testes feitos até o presente momento, não se conhece condições que possam causar um erro que não está sendo tratado. O tratamento consiste em apresentar uma mensagem de erro para o usuário, na maioria dos casos. Caso seja um erro que impossibilite a continuação do programa (falta de memória, impossibilitando a alocação de novos objetos), o programa irá fechar ou ficar impossibilitado de exercer suas funções normalmente. Caso seja detectado que os arquivos estão corrompidos ou ausentes, será criado um novo jogo.

Em comparação com o Projeto 1, foi substituido a maioria dos tratamentos que não utilizavam exceções por exceções.

Das condições que podem causar um erro tratado ocasionado pelo usuário são:

- Entradas que extrapolam os limites do tabuleiro
- Coluna como número
- Linha como caracter
- Movimento que retorna para a mesma posição
- Corrompimento do arquivo propositadamente por parte do usuário

Instruções

Para utilizar o programa aqui presente, é preciso compilar os arquivos .java em .class. Para isso, é possivel utilizar o Prompt de Comando do Windows ou o Terminal do Linux, com o java devidamente instalado e ir até a pasta raiz do pacote, que é a pasta main/java.

Estando na pasta java, é só utilizar o comando:

javac com/mycompany/projeto1/*.java

para compilar e

java com.mycompany.projeto1.Gerenciador

para rodar o programa.

Ou então, é possivel compilar dentro da pasta dos arquivos .java, que é a pasta projeto1, porém o comando ficará diferente:

javac *.java para compilar

e

java -cp ../../.. com.mycompany.projeto1.Gerenciador

para rodar o programa.

Funcionamento:

A interface é simples: no inicio é perguntado ao usuário se ele deseja carregar um jogo salvo, e então ele precisa responder s ou n para que o programa continue. Caso seja respondido que deseja carregar um jogo salvo porém não haja um jogo salvo na pasta raiz, ele irá criar um novo jogo.

A partir disso é só inserir as jogadas no estilo: linhaorigem colunaorigem linhadestino colunadestino.

Exemplo: 2 a 4 a.

O jogo salva automaticamente a cada jogada.

Para sair é só fechar a janela, ou ctrl+C no terminal.

Discussão

Até onde os testes conseguiram mostrar, o projeto está completo. Foram feitos testes exaustivos para verificar todas as condições de erro do programa.
Todas as funcionalidades de um jogo de Xadrez, com exceção dos movimentos especiais, foram implementadas. Porém nem todas foram implementadas da melhor forma possível, tendo em vista que foi gasto mais tempo em algumas funcionalidades do que em outras, e o final teve que ser mais corrido. No caso do xeque-mate, foi investido bastante tempo e, como mencionado na descrição das classes, ele foi planejado de um jeito que não condizia com todos os requisitos posteriores, porém já era um pouco tarde para mudar. Portanto no caso do xeque-mate é possível fazer um planejamento melhor dos métodos e deixa-los mais compactos e tratar de mais casos dentro do mesmo método do que como está no presente momento.

Diagrama de Classes: