Relatório

Projeto de Programação Orientada a Objetos – 2020

Jogo de Xadrez

Projeto 1

Nome: Gustavo de Jesus Rodrigues Silva

RA: 771021

Sumário:

Classe Peca:

A Classe peca é uma classe abstrata responsável pelas peças do jogo de xadrez, ela é a classe mãe de todos os tipos de peças do jogo.

A classe guarda atributos gerais de todas as peças do jogo:

```
private String cor;
private boolean emJogo;
private String representacao;
```

- O atributo de tipo String, cor, guarda a cor daquela peça, se ela é preta ou branca.
- O atributo de tipo boolean, emJogo, guarda true se a peça está em jogo, e false se a peça não está em jogo.
- O atributo String, representação, guarda o desenho de cada peça.

A classe tem seus métodos getters e setters normais para o manuseamento dos seus atributos:

```
public String getCor() {
    return this.cor;
}

public void setCor(String cor) {
    this.cor = cor;
}

public boolean getEmJogo() {
    return this.emJogo;
}

public void setEmJogo(boolean emJogo) {
    this.emJogo = emJogo;
}


public String getRepresentacao() {
    return this.representacao;
}

public void setRepresentacao(String representacao) {
    this.representacao = representacao;
}
```

Além desses métodos, a classe tem um método abstrato que checa a movimentação de cada tipo de peça do jogo, seu retorno é false se o movimento checado é inválido e true se é válido.

```
public abstract boolean checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino);
```

Classe Peao:

A Classe Peao é responsável por setar sua cor e representação, e por fazer a checagem do movimento da peça peão.

A setagem da cor e representação do peão é feita já no seu construtor:

```
public Peao(String cor){
    // Cria a peça fazendo com que a situação seja que ela esta em jogo
    setCor(cor);
    // Seta a representação do peao
    if (cor.equals("BRANCA")) {
        setRepresentacao("♟");
    } else setRepresentacao("♟");
}
```

A checagem para ver se o movimento passado pelo jogador é válido, é feito pelo seguinte método:

```
//chega se o movimento do peao é valido
public boolean checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino) {
    if(this.getRepresentacao().equals("♟")){

        if ((colunaDestino == colunaOrigem + 1) && (linhaDestino == linhaOrigem)) {
            return true;
        } else if (linhaOrigem == linhaDestino && colunaOrigem == colunaDestino) {
            return false;
        } else if (colunaDestino == colunaOrigem + 1 && ((linhaDestino == linhaOrigem + 1) || (linhaDestino == linhaOrigem -1))){
            return true;
        } else if(colunaOrigem == 1){
            if ((colunaDestino == colunaOrigem + 2) && (linhaDestino == linhaOrigem)){
                return true;
            }
        } else return false;
    } else {

        if ((colunaDestino == colunaOrigem - 1) && (linhaDestino == linhaOrigem)) {
            return true;
        } else if (linhaOrigem == linhaDestino && colunaOrigem == colunaDestino) {
            return false;
        } else if (colunaDestino == colunaOrigem - 1 && ((linhaDestino == linhaOrigem + 1) || (linhaDestino == linhaOrigem -1))){
            return true;
        } else if (colunaOrigem == 6){
            if ((colunaDestino == colunaOrigem - 2) && (linhaDestino == linhaOrigem)){
                return true;
            }
        } else return false;
    }
    return false;
}
```

Primeiramente no método checamos se a peça é branca ou preta, pois a orientação do peão é diferente para cada uma dessas cores, depois checamos se o jogador está querendo andar para o lado ou andar para a mesma casa que ele já está, se um desses casos acontecer retorna falso, depois checamos se ele está fazendo algum dos movimentos originais do peão, se sim, retornamos valido, logo depois checamos o caso de ser o primeiro movimento do peão, se for o peão pode andar duas casas, então retornamos válido.

Classe Cavalo:

A Classe Cavalo é responsável por setar sua cor e representação, e por fazer a checagem do movimento da peça cavalo.

A setagem da cor e representação do cavalo é feita já no seu construtor:

```
public Cavalo(String cor){
    // Cria a peça fazendo com que a situação seja que ela esta em jogo
    setCor(cor);
    // Seta a representação da cavalo
    if (cor.equals("BRANCA")) {
        setRepresentacao("♘");
    } else setRepresentacao("♞");
}
```

A checagem para ver se o movimento passado pelo jogador é válido, é feito pelo seguinte método:

```
// Checa se o movimento do cavalo é válido
public boolean checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino) {

    if (Math.abs(linhaOrigem - linhaDestino) == 2 && Math.abs(colunaOrigem - colunaDestino) == 1) {
        return true;
    } else if (Math.abs(linhaOrigem - linhaDestino) == 1 && Math.abs(colunaOrigem - colunaDestino) == 2) {
        return true;
    } else
        return false;
}
```

No caso do cavalo o método é mais simples pois só precisamos ver se ele está fazendo um movimento em L (o movimento normal do cavalo) se ele está fazendo esse movimento retornamos válido, se não retornamos falso.

Classe Torre:

A Classe Torre é responsável por setar sua cor e representação, e por fazer a checagem do movimento da peça torre.

A setagem da cor e representação da torre é feita já no seu construtor:

```
public Torre(String cor){
    //Cria a peça fazendo com que a situação seja que ela esta em jogo
    setCor(cor);

    //Seta a representação da torre
    if(cor.equals("BRANCA")){
        setRepresentacao("♖");
    } else setRepresentacao("♜");

}
```

A checagem para ver se o movimento passado pelo jogador é válido, é feito pelo seguinte método:

```
//Checa se o movimento da torre é valido
public boolean checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino){
    if((linhaOrigem == linhaDestino) && (colunaOrigem != colunaDestino)){
        return true;
    } else if((linhaOrigem != linhaDestino) && (colunaOrigem == colunaDestino)){
        return true;
    } else return false;
}
```

A verificação do movimento da torre é simples, basta vermos se a torre continuou na sua linha e trocou de coluna, ou se trocou de coluna e permaneceu na sua mesma linha de origem, se isso acontecer retornamos valido, se não retornamos falso.

Classe Bispo:

A Classe Bispo é responsável por setar sua cor e representação, e por fazer a checagem do movimento da peça bispo.

A setagem da cor e representação do bispo é feita já no seu construtor:

```
public Bispo(String cor){
    // Cria a peça fazendo com que a situação seja que ela esta em jogo
    setCor(cor);
    // Seta a representação da torre
    if (cor.equals("BRANCA")) {
        setRepresentacao("♗");
    } else setRepresentacao("♝");
}
```

A checagem para ver se o movimento passado pelo jogador é válido, é feito pelo seguinte método:

```
//Checa se o movimento do Bispo é valido
public boolean checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino) {

    if(linhaOrigem == linhaDestino && colunaOrigem == colunaDestino){
        return false;
    }else if ((Math.abs(linhaOrigem - colunaOrigem) == Math.abs(linhaDestino - colunaDestino)) ||  (linhaOrigem + colunaOrigem == linhaDestino + colunaDestino)) {
        return true;
    } else
        return false;
}
```

Para a checagem se a movimentação do bispo temos somente que ver se ele não está indo para a mesma casa que ele está, se isso acontecer retornamos falso, verificarmos se ele está andando em diagonal, se ele está andando em uma das suas diagonais retornamos valido.

Classe Dama:

A Classe Dama é responsável por setar sua cor e representação, e por fazer a checagem do movimento da peça dama(rainha).

A setagem da cor e representação da Dama é feita já no seu construtor:

```
public Dama(String cor){
    // Cria a peça fazendo com que a situação seja que ela esta em jogo
    setCor(cor);
    // Seta a representação da Dama
    if (cor.equals("BRANCA")) {
        setRepresentacao("♕");
    } else setRepresentacao("♛");
}
```

A checagem para ver se o movimento passado pelo jogador é válido, é feito pelo seguinte método:

```
//checa se o movimento da dama é válido
public boolean checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino) {
    if (linhaOrigem == linhaDestino && colunaOrigem == colunaDestino) {
        return false;
    } else if ((linhaOrigem == linhaDestino) && (colunaOrigem != colunaDestino)) {
        return true;
    } else if ((linhaOrigem != linhaDestino) && (colunaOrigem == colunaDestino)) {
        return true;
    } else if ((Math.abs(linhaOrigem - colunaOrigem) == Math.abs(linhaDestino - colunaDestino)) || (linhaOrigem + colunaOrigem == linhaDestino + colunaDestino)) {
        return true;
    } else
        return false;
}
```

A checagem da movimentação da Dama é a junção das checagens da torre e do bispo pois a movimentação original da dama é justamente a junção da movimentação das duas peças.

Classe Rei:

A Classe Rei é responsável por setar sua cor e representação, e por fazer a checagem do movimento da peça rei.

A setagem da cor e representação do rei é feita já no seu construtor:

```
public Rei(String cor) {
    // Cria a peça fazendo com que a situação seja que ela esta em jogo
    setCor(cor);
    // Seta a representação dao Rei
    if(cor.equals("BRANCA")){
        setRepresentacao("♔");
    } else setRepresentacao("♚");

}
```

A checagem para ver se o movimento passado pelo jogador é válido, é feito pelo seguinte método:

```
//Checa se o movimento do rei é válido
public boolean checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino) {
    if ((Math.abs(linhaOrigem - linhaDestino) == 1) && (Math.abs(colunaOrigem - colunaDestino) == 0)){
        return true;
    } else if ((Math.abs(linhaOrigem - linhaDestino) == 0) && (Math.abs(colunaOrigem - colunaDestino) == 1)){
        return true;
    } else if((Math.abs(linhaOrigem - linhaDestino) == 1) && (Math.abs(colunaOrigem - colunaDestino) == 1)){
        return true;
    } else return false;
}
```

A checagem de movimento do rei é bem simples pois ele pode andar para todos os lados, basta verificar se ele se movimentou somente uma casa, se ele andar somente uma casa retornamos valido, se ele não andar ou andar mais casas retornamos falso.

Classe Jogador:

A Classe Jogador é responsável por guardar suas peças, seu nome, e a cor de suas peças.

```
private String nome;
private String cor;
private Peca[] pecas = new Peca[16];
```

- O atributo de tipo string, nome, armazena o nome do jogador.
- O atributo de tipo string, cor, armazena a cor das peças do jogador.
- O atributo de tipo Peca, pecas, é um vetor que guarda as 16 peças que o jogador pode ter.

A classe tem seus métodos getters e setters normais para o manuseamento dos seus atributos:

```
public String getNome() {
    return this.nome;
}

private void setNome(String nome) {
    this.nome = nome;
}

public String getCor() {
    return this.cor;
}

private void setCor(String cor) {
    this.cor = cor;
}

public Peca[] getPecas() {
    return this.pecas;
}

private void setPecas(Peca[] pecas) {
    this.pecas = pecas;
}
```

Seus setters são privados pois seus atributos não precisam e não devem ser modificados por outra classe. O seu construtor é o responsável por fazer as setagens necessárias:

```
public Jogador(String nome, String cor, Peca[] pecas){
    setNome(nome);
    setCor(cor);
    setPecas(pecas);
}
```

Classe Posicao:

A classe Posicao é responsável pelas posições do tabuleiro, deve informar a cor da posição, se ela está ocupada ou não, e se sim, qual peça está a ocupando. Além disso ela deve guardar sua linha e coluna.

```
private boolean ocupada;
private String cor;
private Peca peca;
private int linha;
private char coluna;
```

- O atributo de tipo boolean, ocupada, armazena true se a posição está ocupada e false se a posição está desocupada.
- O atributo de tipo string, cor, armazena qual é a cor da posição.
- O atributo de tipo Peca, peca, armazena a peça que está na posição.
- Os atributos int e char, linha, coluna, armazena respectivamente a onde se localiza aquela posição no tabuleiro.

A classe tem seus métodos getters e setters normais para o manuseamento dos seus atributos.

```
public boolean getOcupada() {
    return this.ocupada;
}

public void setOcupada(boolean ocupada) {
    this.ocupada = ocupada;
}

public String getCor() {
    return this.cor;
}

private void setCor(String cor) {
    this.cor = cor;
}

public Peca getPeca() {
    //Se a posiçao estiver ocupada retorna a peça que está nela;
    if(getOcupada()){
        return this.peca;
        //Se não tiver peça naquela posicao retorna null;
    } else return null;
}

public void setPeca(Peca peca) {
    this.peca = peca;
}

private void setLinha(int linha) {
    this.linha = linha;
}

private void setColuna(char coluna) {
    this.coluna = coluna;
}
```

Os Setters da cor, linha e coluna da posição são privados pois não precisam e não devem ser modificados por outra classe. O seu construtor é o responsável por fazer as setagens necessárias:

```
//Cria as posições com sua cor
public Posicao(int lin, char colun, String cor){
    setLinha(lin);
    setColuna(colun);
    setCor(cor);
    setOcupada(false);
}
```

O setOcupada é chamado no construtor pois quando construímos uma posição ela ainda está sem uma peça ocupando a mesma, então setamos como false;

Classe Tabuleiro:

A Classe Tabuleiro é responsável por criar ou carregar um tabuleiro, colocar posições nele, colocar as peças no tabuleiro, fazer a movimentação das peças no tabuleiro, checar se o caminho da movimentação está livre antes de fazê-la, pela captura de peças. Ele também é responsável por imprimir a situação do tabuleiro e salvar a situação do mesmo.

```
private Posicao tab[][];
```

- O atributo do tipo Posicao, tab, é uma matriz de posições que representa o tabuleiro;

A classe tem somente um método guetter:

```
// retorna posição do tabuleiro
public Posicao getTab(int i, int j) {
    try {
        return this.tab[i][j];
    } catch (IndexOutOfBoundsException e3) {
        return null;
    }
}
```

O método retorna uma posição da matriz que representa o tabuleiro para podermos acessá-la.

O construtor da Classe é responsável por criar todas as posições setando sua linha, coluna e sua cor. O construtor também é responsável por verificar seleção do jogador de iniciar um novo jogo ou carregar o último jogo salvo, com isso o tabuleiro deve carregar o ultimo tabuleiro ou criar um novo tabuleiro;

```
// Inicia o tabuleiro criando as posições e setando se a posição é branca ou
// preta, também carrega ou faz um tabuleiro.
public Tabuleiro(Peca[] brancas, Peca[] pretas, String Selecao) {
    // Aloca as posiçoes do tabuleiro
    this.tab = new Posicao[8][8];
    int contador = 1;
    // Cria as posições do tabuleiro determinado se a posição é uma casa preta ou
    // branca do chadrez
    for (int i = 0; i < 8; i++) {
        if (contador == 0) {
            contador = 1;
        } else
            contador = 0;
        for (int j = 0; j < 8; j++) {
            if (contador == 1) {
                tab[i][j] = new Posicao(i, converter(j), "PRETA");
                contador--;
            } else {
                tab[i][j] = new Posicao(i, converter(j), "BRANCA");
                contador++;
            }
        }
    }
    //Verifica se é para fazer um novo tabuleiro ou Carregar um ja salvo
    if (Selecao.equals("NOVO")) {
        // Assim que cria o tabuleiro e arruma as peças em seu devido lugar para o
        // inicio do jogo;
        arrumarPecas(brancas);
        arrumarPecas(pretas);
    } else {
        // Carrega o tabuleiro de um jogo antigo
        carregarTabuleiro(brancas, pretas);
    }
}
```

No caso de criar um novo jogo, o construtor do tabuleiro vai receber as peças pretas e brancas dos jogadores e mandá-las uma de cada vez para o método arrumarPecas.

```java
// essa função coloca cada peça em seu lugar;
private void arrumarPecas(Peca[] pecas) {
    // coloca as peças brancas no seu lugar inicial no tabuleiro;
    if (pecas[0].getCor() == "BRANCA") {
        for (int i = 0; i < 8; i++) {
            colocarPeca(i, 1, pecas[i]);
        }
        colocarPeca(0, 0, pecas[8]);
        colocarPeca(7, 0, pecas[9]);
        colocarPeca(1, 0, pecas[10]);
        colocarPeca(6, 0, pecas[11]);
        colocarPeca(2, 0, pecas[12]);
        colocarPeca(5, 0, pecas[13]);
        colocarPeca(3, 0, pecas[14]);
        colocarPeca(4, 0, pecas[15]);
    } else {
        // Coloca as peças pretas no seu luagar inial no tabuleiro;
        for (int i = 0; i < 8; i++) {
            colocarPeca(i, 6, pecas[i]);
        }
        colocarPeca(0, 7, pecas[8]);
        colocarPeca(7, 7, pecas[9]);
        colocarPeca(1, 7, pecas[10]);
        colocarPeca(6, 7, pecas[11]);
        colocarPeca(2, 7, pecas[12]);
        colocarPeca(5, 7, pecas[13]);
        colocarPeca(3, 7, pecas[14]);
        colocarPeca(4, 7, pecas[15]);

    }
}
```

O método arrumarPecas vai receber um vetor com todas as peças do jogador, ele antes de tudo verifica se as peças são brancas ou pretas, com isso ele coloca as peças na sua posição inicial no tabuleiro com o método colocarPeca, enviando para esse método o local onde é pqra colocar a peça e a onde a peça deve ser colocada.

Obs: Esse vetor de peças é criado na classe Jogo, e irei explicar como ele é acessado quando for explicar a classe.

```java
// Função para colocar uma peça numa posição do tabuleiro;
private void colocarPeca(int i, int j, Peca peca) {
    getTab(i, j).setPeca(peca);
    getTab(i, j).setOcupada(true);
    peca.setEmJogo(true);
}
```

O método colocarPeca recebe dois int que significam a linha e coluna de uma posição do tabuleiro, e recebe uma peça. A função desse método é colocar a peça na posição informada.

Primeiramente ele seta a peça em sua posição, seta a posição como ocupada, e seta a peça colocando-a em jogo.

Continuando a falar do construtor da classe, no caso de o jogador querer carregar um jogo, o construtor irá chamar o método carregar Tabuleiro mandando para ele as peças brancas e pretas:

```java
//Função que carrega o tabuleiro de um jogo antigo
private void carregarTabuleiro(Peca[] brancas, Peca[] pretas) {

    String[][] tabEmString = new String[8][8];

    //Numeração de casa peça para o acesso do vetor de peças
    int peaob = 0, peaop = 0, torreb = 8, torrep = 8, cavalob = 10, cavalop = 10;
    int bispob = 12, bispop = 12, dama = 14, rei = 15;


    try {
        //Carrega o arquivo
        FileInputStream arquivo = new FileInputStream("Tabuleiro.txt");
        InputStreamReader input = new InputStreamReader(arquivo);
        BufferedReader br = new BufferedReader(input);
        String representacao;

        //Faz a leitura do arquivo
        for (int i = 0; i < 8; i++) {
            representacao = br.readLine();
            for (int j = 0; j < 8; j++) {
                tabEmString[i][j] = Character.toString(representacao.charAt(j));

            }
        }
```

Vamos dividir esse método em duas partes, primeiramente o método cria uma matriz de string para armazenar os dados que iremos pegar do arquivo, logo depois criamos constantes para usar como interadores no vetor de

peças. Após isso ele abre arquivo Tabuleiro.txt que é criado pelo método salvarTabuleiro.

O que encontramos em Tabuleiro.txt:

Logo depois de abrir o arquivo, o método irá colocar cada caractere da matriz que existe no arquivo na matriz de string para podermos comparar onde cada peça estava quando o jogador quis salvar o jogo para continuar depois.

```
//Coloca as peças no tabuleiro a onde estava marcado no arquivo;
for(int i = 0; i < 8; i++){
    for(int j = 0; j < 8; j++){
        if(tabEmString[i][j].equals("♟")){
            colocarPeca(i, j, brancas[peaob]);
            peaob++;
        } else if (tabEmString[i][j].equals("♟")){
            colocarPeca(i, j, pretas[peaop]);
            peaop++;
        } else if(tabEmString[i][j].equals("♖")){
            colocarPeca(i, j, brancas[torreb]);
            torreb++;
        } else if(tabEmString[i][j].equals("♜")){
            colocarPeca(i, j, pretas[torrep]);
            torrep++;
        } else if(tabEmString[i][j].equals("♘")){
            colocarPeca(i, j, brancas[cavalob]);
            cavalob++;
        } else if(tabEmString[i][j].equals("♞")){
            colocarPeca(i, j, pretas[cavalop]);
            cavalop++;
        } else if(tabEmString[i][j].equals("♗")){
            colocarPeca(i, j, brancas[bispob]);
            bispob++;
        } else if(tabEmString[i][j].equals("♝")){
            colocarPeca(i, j, pretas[bispop]);
            bispop++;
        } else if(tabEmString[i][j].equals("♕")){
            colocarPeca(i, j, brancas[dama]);
        } else if(tabEmString[i][j].equals("♛")){
            colocarPeca(i, j, pretas[dama]);
        } else if(tabEmString[i][j].equals("♔")){
            colocarPeca(i, j, brancas[rei]);
        } else if(tabEmString[i][j].equals("♚")){
            colocarPeca(i, j, pretas[rei]);
        }
    }
}
```

Nessa parte checamos onde cada peça estava e colocamos a peça no tabuleiro com o método colocarPeca. Assim carregamos o nosso tabuleiro da memória.

- Nesse método temos um tratamento de exceção, a exceção tratada é o erro de caso de não conseguirmos abrir o arquivo, ou de ter um acesso inválido no arquivo (Capturar uma posição nula sem nada), se a exceção acontecer fazemos um novo tabuleiro colocando as peças em sua posição original.

```
} catch (Exception e) {
    System.out.println("Erro ao Carregar Tabuleiro");
    System.out.println("Não se preocupe, faremos um novo tabuleiro pra você");
    arrumarPecas(brancas);
    arrumarPecas(pretas);
}
```

O tabuleiro é salvo em um arquivo pelo método salvarTabuleiro:

```
//Salva o Tabuleiro num Arquivo
public Boolean salvarTabuleiro() {

    //Carrega o arquivo onde salvamos o Tabuleiro;
    File arquivo = new File("Tabuleiro.txt");

    try {
        //Se o arquivo não existir criamos ele;
        if (!arquivo.exists()) {
            arquivo.createNewFile();
        }

        FileWriter fw = new FileWriter(arquivo, false);

        //Escreve no arquivo a representação da peça a onde ela está no tabuleiro e escreve x a onde não existe peça;
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++) {

                if (this.tab[i][j].getOcupada()) {
                    Posicao casa = this.tab[i][j];
                    fw.write(casa.getPeca().getRepresentacao());
                } else
                    fw.write("x");
            }
            fw.write("\n");
        }
        fw.close();

        return true;
    } catch (IOException ex) {
        return false;
    }
}
```

O método salva uma matriz demarcando onde cada peça estava e um x nas posições vazias.

O método é Boolean pois devemos informar se o tabuleiro foi salvo ou se ocorreu um erro. O método tem um tratamento de exceção retornando false se qualquer erro de carregar ou de escrever no arquivo for detectado, se ocorrer tudo bem retorna true.

A parte de movimentação das peças é feita pelo método moverPeca:

```
// função que move uma peça
public Boolean moverPeca(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino, Jogador jogador) {
    // verifica se o jogador escolheu uma peça que é sua;
    if (getTab(linhaOrigem, colunaOrigem).getOcupada()
            && getTab(linhaOrigem, colunaOrigem).getPeca().getCor() == jogador.getCor()) {
        // Se o movimento for válido realizamos o mesmo;
        if (validarMovimento(linhaOrigem, colunaOrigem, linhaDestino, colunaDestino,
                getTab(linhaOrigem, colunaOrigem).getPeca())) {
            // Colcoa a peça na sua posição de Destino;
            colocarPeca(linhaDestino, colunaDestino, getTab(linhaOrigem, colunaOrigem).getPeca());
            // Tira a peça da sua posição Original;
            limpaCasa(linhaOrigem, colunaOrigem);
            // Retorna true pois o movimento aconteceu;
            return true;
        } else
            System.out.println("Esse movemento é inválido, tente novamente!!");
    } else
        System.out.println("Essa peça não é sua, tente novamente!!");
    // retorna false se o movimento não aconteceu;
    return false;
}
```

O método recebe a movimentação, a linha e coluna de onde está a peça para ser movimentada e a linha e coluna de onde o jogador quer movimentar a peça.

Primeiramente o método checa se tem uma peça na quela posição e se tiver, verifica se a peça é do jogador que está fazendo a movimentação. Depois disso ele vê se aquele movimento é válido com o método validarMovimento, se o movimento for válido ele coloca a peça na posição de destino com o método de colocarPeca e limpa a casa de posição original com o método limpaCasa.

```
// função que limpa uma casa tirando a peça de lá;
private void limpaCasa(int i, int j) {
    getTab(i, j).setOcupada(false);
    getTab(i, j).setPeca(null);
}
```

A função limpaCasa somente desocupa a casa e tira a peça que está lá.

A validação feita pelo método validarMovimento é feita da seguinte foma:

```
// Função que Valida um movimento;
private boolean validarMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino, Peca p) {
    // Se o movimento da peça foi válido e se o seu caminho estar livre retorna
    // true;
    if (p.checaMovimento(linhaOrigem, colunaOrigem, linhaDestino, colunaDestino)
            && checaCaminho(linhaOrigem, colunaOrigem, linhaDestino, colunaDestino, p))
        return true;
    // Se o movimento não for válido retorna false;
    return false;
}
```

Primeiramente o método checa se o movimento que o usuário está querendo fazer com a peça é válido chamando o método checaMovimento implementado em cada uma das casas, e logo depois verifica se o caminho da movimentação está livre com o método checaCaminho, se as verificações forem válidas retorna true, se forem falsas retorna false.

O método checaCaminho checa se o caminho da casa está livre, se o caminho está livre mais a última casa está sendo ocupada por uma cor diferente o método captura a peça com o método capiturarPeca.

```
// fumção que checa se um caminho estpá livre para a movimentação;
private boolean checaCaminho(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino, Peca p) {

    // checa o caso do peão;
    if (p instanceof Peao) {
        // se o destino do peão estiver ocupado por uma peça da sua cor retorna false;
        if ((getTab(linhaDestino, colunaDestino).getOcupada()
                && getTab(linhaDestino, colunaDestino).getPeca().getCor() == p.getCor())) {
            return false;
            // se o peão for andar para frente e seu destino estiver ocupado retorna false;
        } else if ((linhaOrigem == linhaDestino) && getTab(linhaDestino, colunaDestino).getOcupada()) {
            return false;
            // Se o seu destino não estiver ocupado por uma peça da sua mesma cor e se o
            // movimento não foi para frente, capitura a peça da sua diagonal;
            // Se a diagonal estiver desocupado a própria função capiturarPeca retornara
            // false;
        } else if (getTab(linhaDestino, colunaDestino).getOcupada())
            return capiturarPeca(linhaDestino, colunaDestino, p);
        // Checa o Caso do cavalo
    } else if (p instanceof Cavalo) {
        // Se a casa final do Cavalo estiver ocupada por uma peça de cor diferente do
        // mesmo ele capitura a peça;
        if (getTab(linhaDestino, colunaDestino).getOcupada()
                && getTab(linhaDestino, colunaDestino).getPeca().getCor() != p.getCor()) {
            return capiturarPeca(linhaDestino, colunaDestino, p);
            // Se a casa fional do Cavalo estiver sento ocupada por uma peça de cor igual do
            // mesmo retorna false pois o movimento é inválido;
        } else if (getTab(linhaDestino, colunaDestino).getOcupada()) {
            return false;
        }
        // Casos de movimentação Geral;
```

Primeiramente o método checaCaminho verifica os casos de a peça ser um cavalo ou um peão pois suas movimentações são diferentes das outras peças, no caso do peão primeiramente ele checa se sua movimentação para frente é válida checando se o caminho está livre, depois ele checa o

movimento em diagonal do peão deixando-o se mover em diagonal somente se a casa destino estiver sendo ocupada por uma peça de cor diferente, capturando essa peça. O caso do cavalo, que pula as peças do seu caminho, valida o movimento somente se a casa destino estiver desocupada ou se estiver ocupada por uma peça de cor diferente, capturando essa peça.

Assim só nos resta verificar os movimentos das peças restantes que são movimentos em vertical horizontal e diagonal.

```
    // Casos de movimentação Geral;
} else {
    int inicio;
    int fim;
    // Caso a peça se movimentar Horizontalmente;
    if (linhaOrigem == linhaDestino) {
        // Verifica se a Origem ou o destino é maior para criar uma constante para andar;
        // pelas casas no for;
        if (colunaOrigem < colunaDestino) {
            inicio = colunaOrigem;
            fim = colunaDestino;
        } else {
            inicio = colunaDestino;
            fim = colunaOrigem;
        }
        // Checa se alguma casa do caminho horizontal está ocupada;
        for (int j = inicio + 1; j < fim; j++) {
            try {
                // Se estiver ocupada retorna false;
                if (getTab(linhaOrigem, j).getOcupada()) {
                    return false;
                }
            } catch (IndexOutOfBoundsException e1) {
                continue;
            }
        }
        // Caso a peça se mova Verticalmente
    } else if (colunaOrigem == colunaDestino) {
        // Verifica se a Origem ou o destino é maior para criar uma constante para andar
        // pelas casas no for;
        if (linhaOrigem < linhaDestino) {
            inicio = linhaOrigem;
            fim = linhaDestino;
        } else {
            inicio = linhaDestino;
            fim = linhaOrigem;
        }
        // Checa se alguma casa do caminho vertical está ocupada;
        for (int i = inicio + 1; i < fim; i++) {
            try {
                // Se estiver ocupada retorna false;
                if (getTab(i, colunaOrigem).getOcupada()) {
                    return false;
                }
            } catch (IndexOutOfBoundsException e2) {
                continue;
            }
        }
```

Os movimentos em vertical e horizontal são checados observando se a peça se moveu na mesma linha, depois de fazer essa checagem criamos interadores para andar com um for pelas casas do caminho que a peça irá fazer na movimentação, se alguma dessas casas estiver ocupada retornamos false.

Depois do caso dos movimentos em vertical e horizontal, o caso de o movimento ser em diagonal.

```
    // Caso seja um movimento em diagonal;
} else {
    int colunaInicio;
    int linhaInicio;
    int linhaFim;
    // Verifica se a Origem ou o destino é maior para criar uma constante para andar
    // pelas casas no for;
    if (colunaOrigem < colunaDestino) {
        colunaInicio = colunaOrigem;
    } else
        colunaInicio = colunaDestino;
    if (linhaOrigem < linhaDestino) {
        linhaInicio = linhaOrigem;
    } else
        linhaInicio = linhaDestino;
    if (linhaOrigem > linhaDestino) {
        linhaFim = linhaOrigem;
    } else
        linhaFim = linhaDestino;

    // Checa quais das duas diagonais é o movimento;

    if (linhaDestino - linhaOrigem < 0 && colunaDestino - colunaOrigem < 0
            || linhaDestino - linhaOrigem > 0 && colunaDestino - colunaOrigem > 0) {
        // anda pela diagonal do movimento;
        for (int j = colunaInicio + 1, i = linhaInicio + 1; i < linhaFim; i++, j++) {
            // se alguma casa do caminho estiver ocupada retorna false;
            if (getTab(i, j).getOcupada()) {
                return false;
            }
        }
    } else if (linhaDestino - linhaOrigem < 0 && colunaDestino - colunaOrigem > 0
            || linhaDestino - linhaOrigem > 0 && colunaDestino - colunaOrigem < 0) {
        // anda pela diagonal do movimento;
        for (int j = colunaInicio + 1, i = linhaFim - 1; i > linhaInicio; i--, j++) {
            // se alguma casa do caminho estiver ocupada retorna false;
            if (getTab(i, j).getOcupada()) {
                return false;
            }
        }
    }
}
```

A checagem do movimento em diagonal é feita da mesma forma dos movimentos em vertical e horizontal, ele cria interadores para andar pelas casas do caminho com um for, se alguma das casas tiver ocupada retorna false.

E por último o método verifica se a posição final está ocupada:

```
    // Se nenhuma das situações a cima tiver acontecido que dizer que o caminho está;
    // livre;
    // Se a posição final estiver ocupada por uma cor diferente a mesma capitura ela;
    if (getTab(linhaDestino, colunaDestino).getOcupada()
            && getTab(linhaDestino, colunaDestino).getPeca().getCor() != p.getCor()) {
        // Capitua a peça da posição final
        return capiturarPeca(linhaDestino, colunaDestino, p);
        // Se a posição final estiver ocupada por uma peça da mesma cor da mesma retorna;
        // false;
    } else if (getTab(linhaDestino, colunaDestino).getOcupada()
            && getTab(linhaDestino, colunaDestino).getPeca().getCor() == p.getCor())
        return false;
}
// Se chegamos até aqui não tem nenhuma peça no caminho do movimento e retorna;
// true;
return true;
```

Se a posição final estiver ocupada por uma peça de cor diferente, captura a peça, se for de cor igual o movimento é inválido.

Se o programa não entrou em nenhum dos casos quer dizer que o caminho está livre e o método retorna true.

O método capiturarPeca captura a peça e checa se aquela peça é um rei, se for um rei o método encerra o jogo pois foi um Cheque-Mate, se não for o rei ele captura a peça normalmente e retorna true.

```
// função que capitura uma peça
private Boolean capiturarPeca(int linhaDestino, int colunaDestino, Peca p) {
    // Se a casa esta ocupada por uma peça de cor igual e quisermos capiturala não
    // conseguimos
    if (getTab(linhaDestino, colunaDestino).getOcupada()
            && getTab(linhaDestino, colunaDestino).getPeca().getCor() != p.getCor()) {
        if (getTab(linhaDestino, colunaDestino).getPeca() instanceof Rei) {
            System.out.println("\n");
            System.out.println("Xeque Mate!!!");
            System.out.println("O Jogo acabou");
            if (p.getCor().equals("BRANCA")) {
                System.out.println("O Jogador 1 ganhou!!!");
            } else
                System.out.println("O jogador 2 ganhou!!");
            System.exit(0);
        }
        // tira a peça do jogo
        getTab(linhaDestino, colunaDestino).getPeca().setEmJogo(false);
        // limpa a casa para receber a nova peça
        limpaCasa(linhaDestino, colunaDestino);
        return true;
    } else
        return false;
}
```

Primeiramente o método faz a checagem se tem uma peça de cor diferente no local informado, se tiver ele verifica se é um rei, se a peça for um rei ele avisa o cheque mate e encerra o programa, se não for um rei ele tira a peça capturada do jogo e limpa a posição.

O método imprime tabuleiro tem como função printar na tela o tabuleiro, suas posições a as peças que o ocupam o tabuleiro.

```
// Imprime o tabuleiro mostrando se a casa é branca ou preta e mostrando as
// peças;
public void imprimeTabuleiro() {
    System.out.printf("  ");
    for (int i = 0; i < 8; i++) {
        System.out.printf("%d ", i + 1);
    }
    System.out.println("");
    for (int i = 0; i < 8; i++) {
        System.out.print(converter(i));
        System.out.print(" ");
        for (int j = 0; j < 8; j++) {
            Posicao casa = this.tab[i][j];
            if (casa.getOcupada()) {
                System.out.printf(casa.getPeca().getRepresentacao() + " ");
            } else if (casa.getCor().equals("BRANCA")) {
                System.out.printf("x ");
            } else
                System.out.printf("o ");
        }
        System.out.println("");
    }
}
```

O método imprime as linhas em letras e as colunas em números, as casas ocupadas ele printa a representação de cada peça q a ocupa, e as desocupadas printa x nas casas pretas e o nas casas brancas.

O método converter converte um número int para uma letra em char:

```
// comverte int para char
public char converter(int i) {
    return (char) (i + 'A');
}
```

Classe Jogo:

A classe Jogo é responsável por tudo que acontece no jogo, por criar ou carregar um novo jogo, por criar os jogadores, as peças e o tabuleiro, por chamar os métodos de fazer movimentação do tabuleiro, por fazer a interface do jogo, por capturar e checar as escolhas do jogador e por salvar o jogo, começar um novo jogo ou retomar um jogo salvo.

Atributos:

```
private Tabuleiro tab;
private Scanner scan = new Scanner(System.in);
private Jogador Jogador1;
private Jogador Jogador2;
private int turno;
```

- O atributo do tipo Tabuleiro, tab, é responsável por guardar o tabuleiro.
- O atributo de tipo Scanner, sacan, é responsável por guardar um sacanner para pegar as entradas do usuário.
- Os atributos do tipo Jogador, jogador1 e jogador2 é responsável por guardar os dois jogadores do jogo.
- O atributo de tipo int, turno, é responsável por guardar de quem é a vez de jogar.

A classe tem seus métodos getters e setters normais para o manuseamento dos seus atributos

```
public Jogador getJogador1() {
    return this.Jogador1;
}

private void setJogador1(Jogador jogador1) {
    this.Jogador1 = jogador1;
}

public Jogador getJogador2() {
    return this.Jogador2;
}

private void setJogador2(Jogador jogador2) {
    this.Jogador2 = jogador2;
}

public Tabuleiro getTab() {
    return this.tab;
}

private void setTab(Tabuleiro tab) {
    this.tab = tab;
}

private int getTurno() {
    return this.turno;
}

private void setTurno(int turno) {
    this.turno = turno;
}
```

Seus setters são privados pois seus atributos não precisam e não devem ser modificados por outra classe.

No seu construtor vemos uma sobre carga de construtores.

```
// cria o jogo setando os nomes do jogador e cria o tabuleiro
public Jogo(String nomeJogador1, String nomeJogador2) {

    novoJogo(nomeJogador1, nomeJogador2);
}

public Jogo() {
    carregarJogo();
}
```

Um construtor recebe o nome em string dos jogadores 1 e 2, e chama o método novoJogo que criará um jogo.

O outro construto não recebe nenhum parâmetro pois ele carregará um jogo salvo na memória

O método novo jogo funciona da seguinte maneira:

```
// inicia um novo jogo;
public void novoJogo(String jogador1, String jogador2) {

    // Cria o jogador1 com suas peças;
    Jogador player1 = new Jogador(jogador1, "BRANCA", fazerPecas("BRANCA"));
    setJogador1(player1);

    // Cria o Jogador2 com suas peças
    Jogador player2 = new Jogador(jogador2, "PRETA", fazerPecas("PRETA"));
    setJogador2(player2);

    // Cria um novo tabuleiro colocando as peças dos jogadores em sua devida
    // posição;
    Tabuleiro novoTabuleiro = new Tabuleiro(getJogador1().getPecas(), getJogador2().getPecas(), "NOVO");
    setTab(novoTabuleiro);

    // Seta o turno antes de começar o jogo
    setTurno(1);

    // inicia o jogo;
    jogar();
```

Primeiramente método cria os dois jogadores com os nomes passados pelo gerenciador, criando e passando ao jogador1 um vetor peças brancas e para o jogador 2 um vetor de peças pretas, esse vetor é criado com o método fazerPecas:

```java
// Faz a peça dos jogadores
private Peca[] fazerPecas(String cor) {

    Peca[] peca = new Peca[16];

    for (int i = 0; i < 16; i++) {
        if (i < 8) {
            peca[i] = new Peao(cor);
        } else if (i < 10) {
            peca[i] = new Torre(cor);
        } else if (i < 12) {
            peca[i] = new Cavalo(cor);
        } else if (i < 14) {
            peca[i] = new Bispo(cor);
        } else if (i == 14) {
            peca[i] = new Dama(cor);
        } else if (i == 15) {
            peca[i] = new Rei(cor);
        }
    }
    return peca;
}
```

O método fazerPecas retorna um vetor de pecas contendo oito peões, duas torres, dois cavalos, dois bispos, uma dama e um rei, ele chama o construtor de cada peça passando a cor que a peça deve ser, a atribuição das peças no vetor é feita da seguinte maneira:

Das posições de 0 a 7 ficam os 8 peões, nas posições 8 e 9 ficam as duas torres, nas posições 10 e 11 ficam as duas torres, nas posições 12 e 13 ficam os bispos, e nas posições 14 e 15 ficam o rei e a rainha.

Por fim ele retorna esse vetor com as peças da mesma cor passada para o método.

Depois de criar os jogadores o método novoJogo cria um tabuleiro, passando o vetor de peças do jogador 1 e 2 e fala para o tabuleiro que ele quer um novo tabuleiro.

Por fim ele seta o turno para começar com o jogador 1 que é o dono das peças brancas e começa o jogo.

O método carregarJogo funciona da seguinte maneira:

```
// Caarrega o ultimo jogo salvo
public void carregarJogo() {
    try {
        // Carrega o arquivo a onde o Jogo foi salvo;
        FileInputStream arquivo = new FileInputStream("Jogo.txt");
        InputStreamReader input = new InputStreamReader(arquivo);
        BufferedReader br = new BufferedReader(input);

        String nome1, nome2;

        // Copia o Nome do Jogador 1
        nome1 = br.readLine();

        Jogador player1 = new Jogador(nome1, "BRANCA", fazerPecas("BRANCA"));
        setJogador1(player1);

        // Copia o nome do Jogador 2
        nome2 = br.readLine();

        // Cria o Jogador2 com suas peças
        Jogador player2 = new Jogador(nome2, "PRETA", fazerPecas("PRETA"));
        setJogador2(player2);

        // Carrega o tabuleiro
        Tabuleiro novoTabuleiro = new Tabuleiro(getJogador1().getPecas(), getJogador2().getPecas(), "CARREGAR");
        setTab(novoTabuleiro);

        // Copia o turno e o transforma em int
        int turno = Integer.parseInt(br.readLine());

        // Seta o turno antes de começar o jogo
        setTurno(turno);

        br.close();

        // inicia o jogo;
        jogar();

    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("Erro ao Carregar Jogo");
    }
}
```

Primeiramente o método carrega o arquivo jogo.txt da memória, nesse arquivo salvo pelo método salvarJogo, encontramos o estado do último jogo salvo, encontramos os nomes dos jogadores 1 e 2 que estavam jogando o último jogo, e encontramos na vez de quem o jogo tinha parado.

Depois de carregar o arquivo da memória, o método coleta os nomes salvos dos jogadores 1 e 2 e cria eles com seus respectivos nomes, logo depois ele cria um tabuleiro mandando-o carregar o último tabuleiro salvo, e por último ele pega o turno salvo do arquivo, seta o turno, facha o arquivo e começa o jogo.

- No método tratamos a exceção o caso de não conseguir abrir o arquivo, ou de ser acessada uma posição inválida do arquivo.

O método salvarJogo funciona da seguinte maneira:

```
private Boolean salvarJogo() {

    // Abre o arquivo Jogo;
    File arquivo = new File("jogo.txt");
    tab.salvarTabuleiro();

    try {
        if (!arquivo.exists()) {
            arquivo.createNewFile();
        }
        FileWriter fw = new FileWriter(arquivo, false);
        fw.write(getJogador1().getNome() + "\n" + getJogador2().getNome() + "\n" + getTurno());
        fw.close();
        return true;
    } catch (IOException ex) {
        return false;
    }
}
```

Ele abre o arquivo jogo.txt, se o arquivo não existir ele cria um, e depois grava no arquivo o nome do jogador1, o nome do jogardor2 e o turno que marca na vez de quem o jogo parou.

- O método cuida da exceção de não poder carregar ou criar o arquivo, e também da exceção de um acesso inválido ao arquivo.

O método jogar é responsável pela interface e jogabilidade de usuário, veremos em partes como ele funciona:

```
private void jogar() {
    String linhaOrigem;
    int colunaOrigem;
    String linhaDestino;
    int colunaDestino;
    System.out.println("");
    System.out.println("Dica: Todas as jogadas são feitas com letra maiúscula, deixe seu capslock ligado!!");
    System.out.println("Dica: Digite X nas capituras de linha quando quiser sair do jogo");
    System.out.println("");
    System.out.println("Digite C para continuar ou X para sair");
    linhaOrigem = scan.next();
    // imprime o Tabuleiro inicial
    tab.imprimeTabuleiro();
```

Primeiramente ele passa algumas informações sobre a jogabilidade para o usuário, e imprime o tabuleiro inicial.

Logo após isso ele pega a movimentação que o usuário quer fazer e manda a movimentação para o método fazerJogada:

```
while (!linhaOrigem.equals("X")) {
    // Checa de quem é a vez
    switch (getTurno()) {
        case 1:
            do {

                // Capitura a jogada do jogador 1
                System.out.println("É a vez do jogardor(a): " + getJogador1().getNome());
                System.out.print("Digite a Linha da peça que voce quer mover: ");
                linhaOrigem = scan.next();
                if (linhaOrigem.equals("X"))
                    sair();
                System.out.println("");
                System.out.print("Digite a coluna da peça que voce quer mover: ");
                colunaOrigem = scan.nextInt();
                System.out.println("");
                System.out.print("Digite a Linha da Casa da sua jogada: ");
                linhaDestino = scan.next();
                if (linhaDestino.equals("X"))
                    sair();
                System.out.println("");
                System.out.print("Digite a Coluna Da Casa da sua jogada: ");
                colunaDestino = scan.nextInt();
                // Faz a jogada do Jogador1
                fazerJogada(converteParaInt(linhaOrigem), colunaOrigem - 1, converteParaInt(linhaDestino),
                        colunaDestino - 1, getJogador1());
                // Se não mudou o turno, a jogada foi inválida, e o vez continua com o mesmo
                // jogador;
            } while (getTurno() == 1);
            break;
        case 2:
            do {

                // Capitura a jogada do Jogador2
                System.out.println("É a vez do jogardor(a): " + getJogador2().getNome());
                System.out.print("Digite a Linha da peça que voce quer mover: ");
                linhaOrigem = scan.next();
                if (linhaOrigem.equals("X"))
                    sair();
                System.out.println("");
                System.out.print("Digite a coluna da peça que voce quer mover: ");
                colunaOrigem = scan.nextInt();
                System.out.println("");
                System.out.print("Digite a Linha da Casa da sua jogada: ");
                linhaDestino = scan.next();
                if (linhaDestino.equals("X"))
                    sair();
                System.out.println("");
                System.out.print("Digite a Coluna Da Casa da sua jogada: ");
                colunaDestino = scan.nextInt();

                // Faz a jogada do jogador2
                fazerJogada(converteParaInt(linhaOrigem), colunaOrigem - 1, converteParaInt(linhaDestino),
                        colunaDestino - 1, getJogador2());
                // Se não mudou o turno, a jogada foi inválida, e o vez continua com o mesmo
                // jogador;
            } while (getTurno() == 2);
            break;
    }
}
```

Antes de fazer a Jogada o método verifica se o jogador não digitou x em algumas das entradas de linha, se ele digitou x quer dizer que o usuário quer sair do jogo, assim chamando o método sair

O método sair funciona da seguinte maneira;

```
// função para sair do jogo;
private void sair() {

    String escolha;

    System.out.println("");
    System.out.println("");
    System.out.println("Voce deseja Salvar o jogo para continuar depois?");
    System.out.println("Digite SIM ou NAO");

    escolha = scan.next();

    // Salva o jogo se o Jogador quiser continuar depois
    if (escolha.equals("SIM")) {
        if (salvarJogo()) {
            System.out.println("O Jogo foi Salvo");
            System.exit(0);
        }
    } else {
        System.exit(0);
    }
}
```

Ele pergunta para o usuário se ele deseja salvar o jogo ou são, se ele selecionar salvar jogo o método chama o método salvarJogo e encerra o programa, se ele não quiser salvar ele somente encerra o programa.

Método fazerJogada:

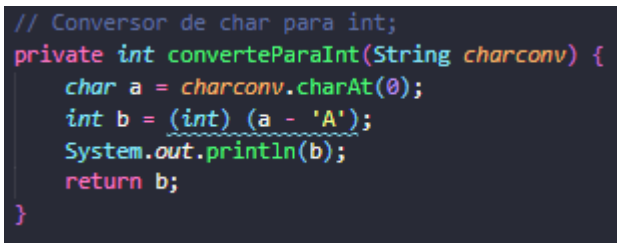
```
// Faz a Jogada;
private void fazerJogada(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino, Jogador jogador) {
    // Checa se o movimento está dentro do tabuleiro;
    try {
        // Move a peça;
        if (tab.moverPeca(linhaOrigem, colunaOrigem, linhaDestino, colunaDestino, jogador)) {
            // Se ocorreu o movimento troca de turno;
            if (getTurno() == 1) {
                setTurno(2);
            } else
                setTurno(1);
            tab.imprimeTabuleiro();
        }
    } catch (NullPointerException e5) {
        System.out.println("A casa selecionada para a jogada não está no tabuleiro");
    }
}
```

O método chama o moverPeca do tabuleiro, se ele conseguir mover a peça ele troca de turno, se ele não conseguiu ele não troca de turno e o continua na vez do jogar até ele fazer um movimento correto.

- O método tem um tratamento de exceção que trata se o movimento mandado não está fora do tabuleiro.

Método converteParaInt:

Esse método tem a função de converter a string passada pelo jogador para int, pois todos os algoritmos de funcionamento recebem como parâmetro um int, e na interface o jogador passa uma string.



Classe Gerenciador:

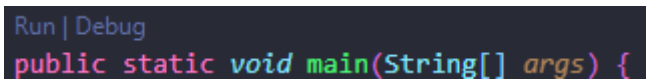
A classe gerenciador tem como objetivo pegar a seleção do usuário se ele quer começar um novo jogo ou se ele quer carregar o último jogo salvo, essa é a classe mais simples de todo o projeto, ele faz uma mine interface inicial, logo depois de verificar a seleção do usuário, se ele quiser começar um novo jogo ele captura o nome do jogador 1 e do jogador 2 e inicia um novo jogo falando para a classe Jogo iniciar um, se o jogador quiser carregar um jogo, ele chama o construtor do Jogo que carrega o ultimo Jogo salvo:

```
public class Gerenciador {

    Run | Debug
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);


        System.out.println("Bem Vindo a este LINDO jogo De Xadrez!!\n");
        System.out.println("Se voce quizer começar um novo jogo digite Novo");
        System.out.println("Se voce quiser carregar seu ultimo jogo digite Carregar");


        String selecao;
        selecao = scan.nextLine();

        if (selecao.equals("Novo")) {

            String jogador1, jogador2;
                System.out.println("");
                System.out.println("Digite o nome do jogador 1:");
                jogador1 = scan.nextLine();
                System.out.println("");
                System.out.println("Digite o nome do jogador 2:");
                jogador2 = scan.nextLine();

            Jogo xadrez = new Jogo(jogador1, jogador2);

        } else if (selecao.equals("Carregar")) {
            Jogo xadrez = new Jogo();
        }


    }
}
```

Ele também tem a função de main do programa.

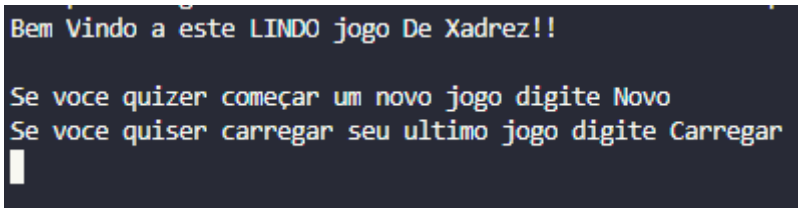
Compilação e uso do programa:

Para a compilação e uso do programa você deve ter o Java Development Kit baixado e abrir o código em uma IDE e executar o programa, por exemplo no VS Code basta apertar em rum no código da main:



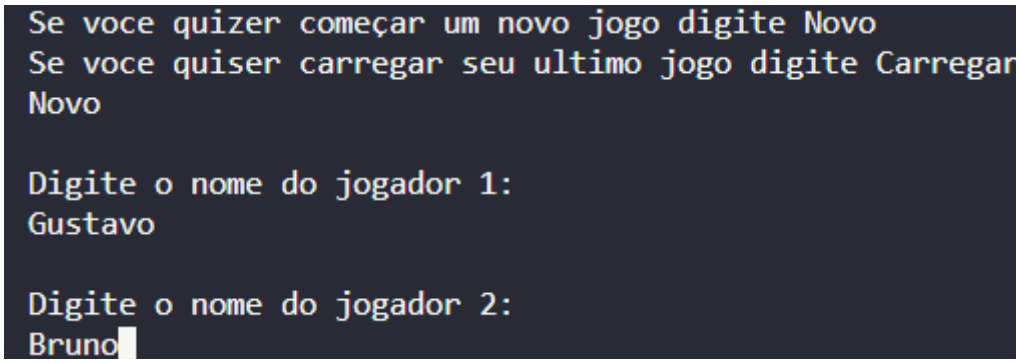
Interface e Jogabilidade:

A interface do Jogo é feita por terminal, primeiramente o jogo inicia perguntando se queremos começar um novo jogo ou se queremos carregar o ultimo jogo salvo:



Para iniciar um novo jogo temos que digitar Novo, e se quisermos carregar o último jogo salvo digitamos Carregar.

Se entrarmos com novo a interface irá pedir para digitar o nome do jogador 1 e do jogador 2:

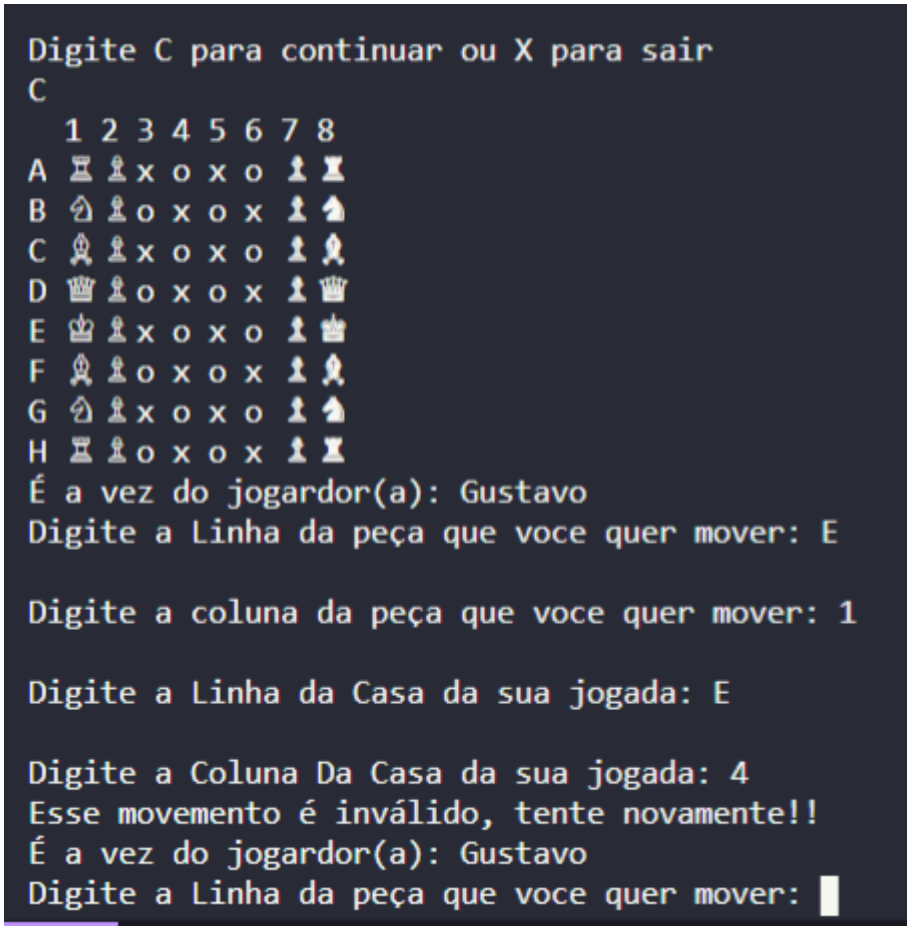


Lembrando que sempre o Jogador 1 ficara com as peças brancas e o jogador 2 com as pretas.

Depois de pegar o nome dos jogadores a interface irá passar algumas dicas para a jogabilidade e perguntará se o jogador quer Sair ou continuar:

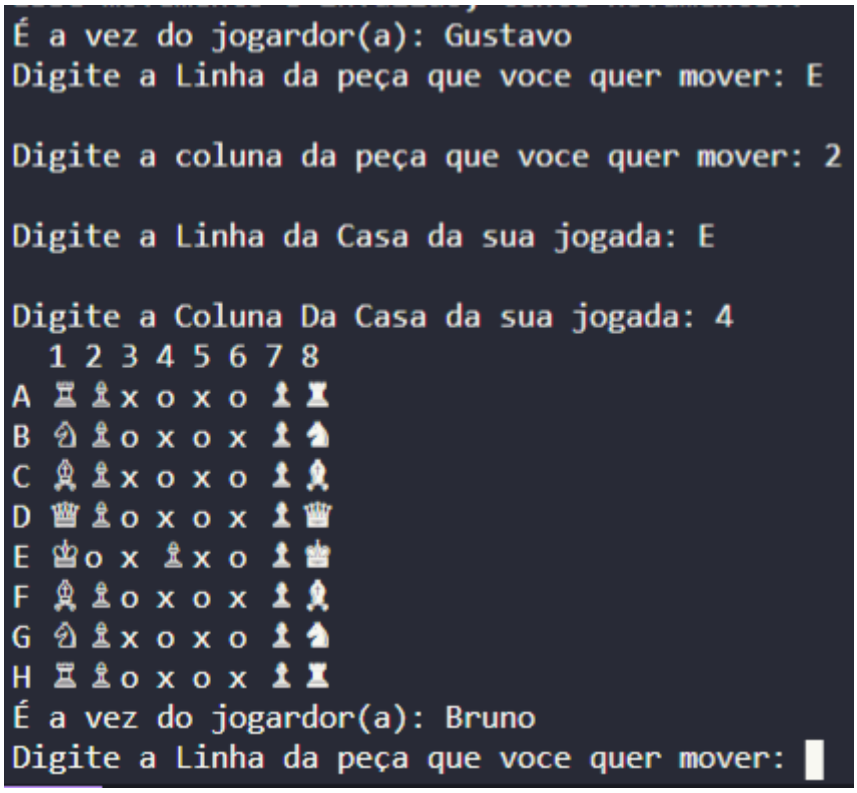
```
Dica: Todas as jogadas são feitas com letra maiúscula, deixe seu capslock ligado!!
Dica: Digite X nas capituras de linha quando quiser sair do jogo

Digite C para continuar ou X para sair
```

Depois de dermos a entrada como C, o jogo começara mostrando o tabuleiro, informando de quem é a vez e pedindo a posição de linha e coluna de onde está a peça que queremos movimentar e a linha e coluna da casa para onde queremos movimentar a peça:



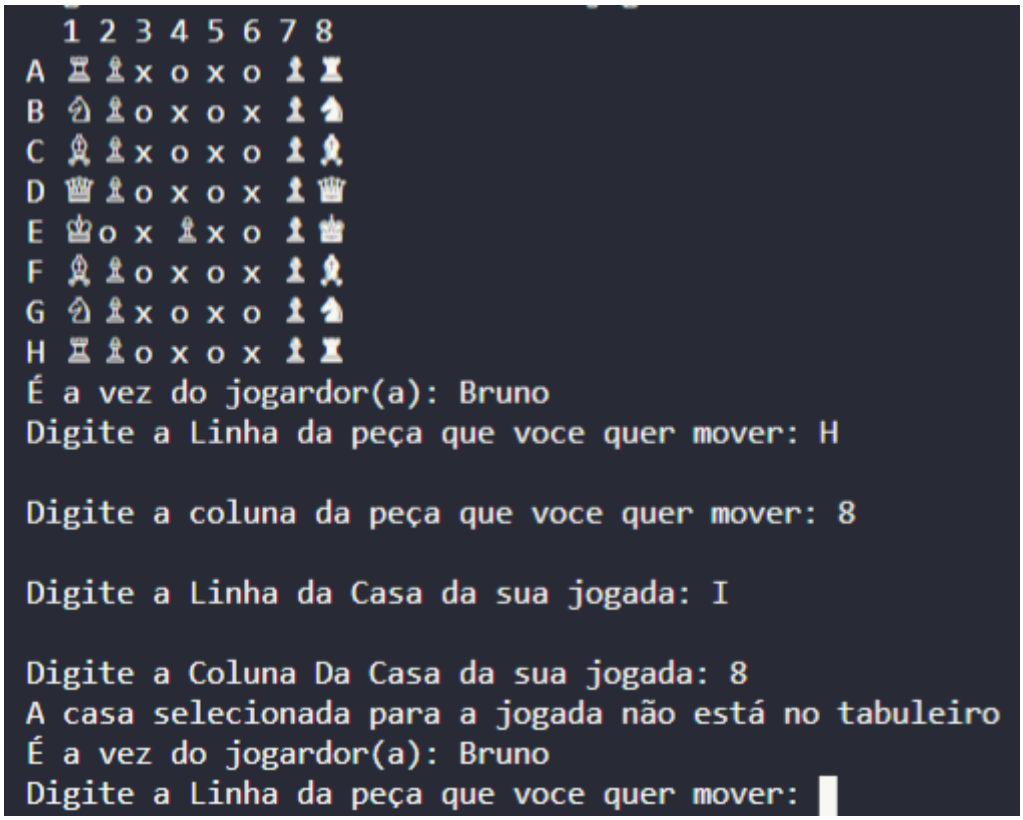
Se digitarmos um movimento que é inválido, como no caso acima que eu passo um movimento do rei de forma erronia, a interface fala que o movimento é inválido e não passa a vez para o outro jogador, fazendo o jogador que fazer o movimento inválido refazer o movimento.

Caso fizermos um movimento válido:



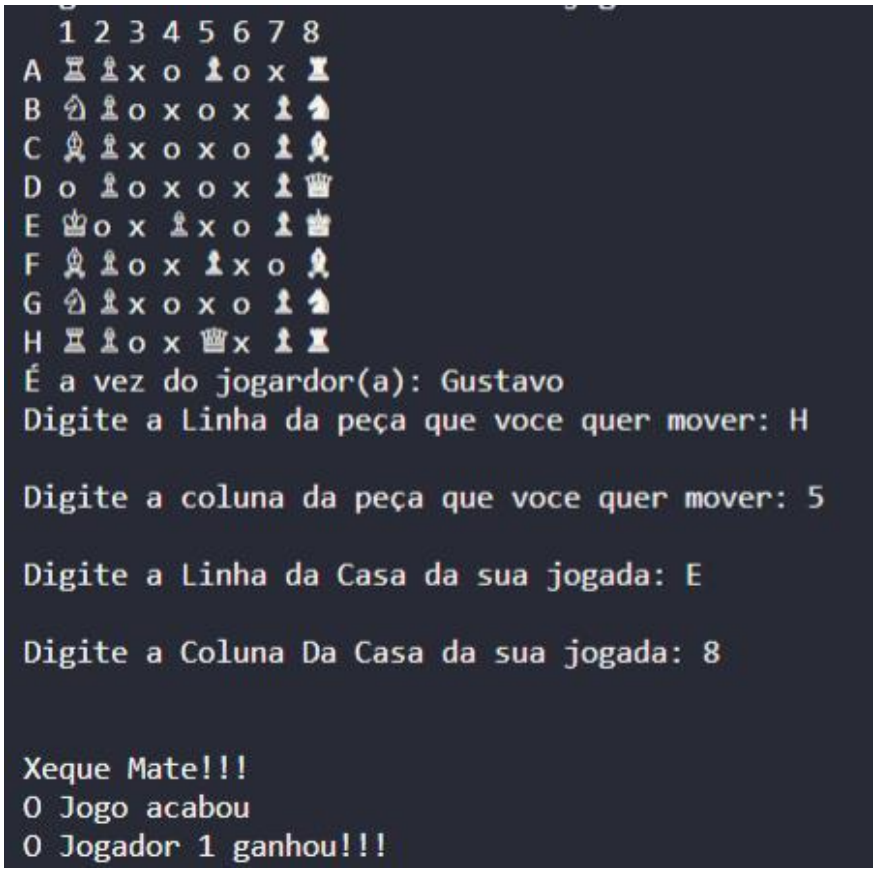
A interface imprimira o tabuleiro já com a peça movimentada e passará a vez para o próximo jogador.

Caso o Jogador tente movimentar uma peça para fora do tabuleiro:

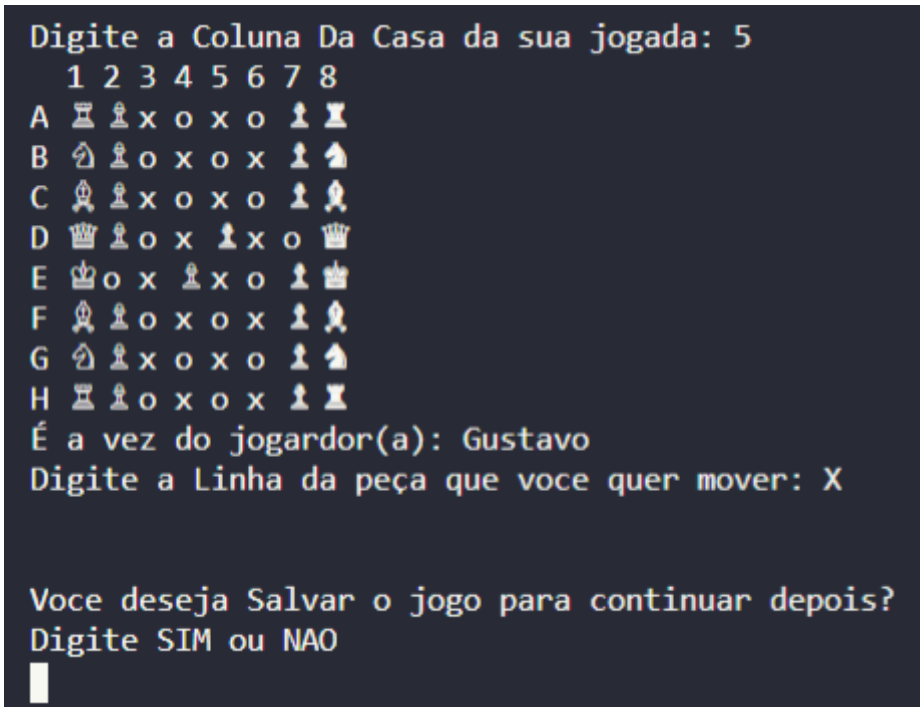


A interface retorna que a casa selecionada não está no tabuleiro.

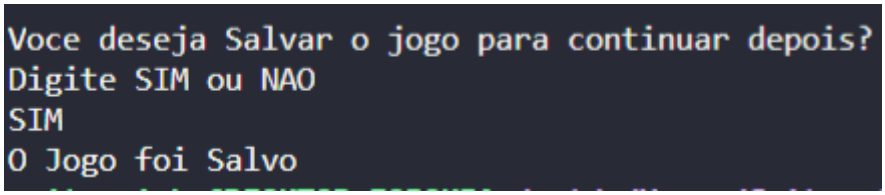
Para o Jogo acabar basta capturar o rei inimigo:



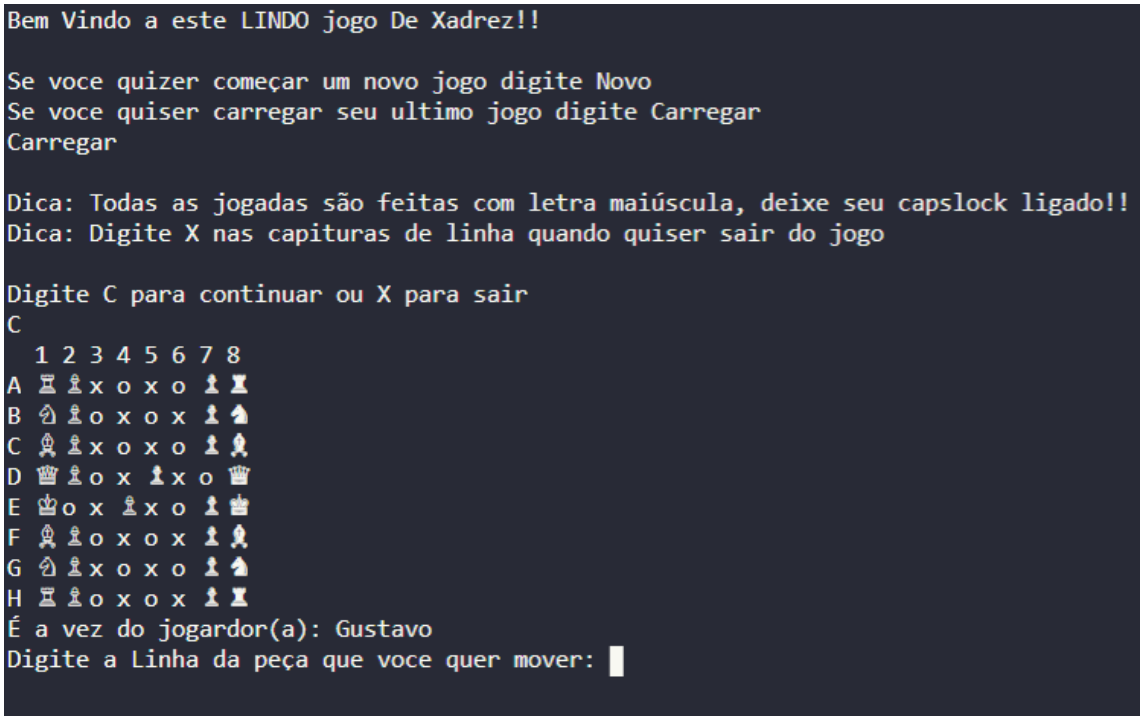
Caso você queira Sair do jogo antes dele acabar é só digitar X em uma das entradas de linha



O jogo perguntará se você quer salvar o jogo para continuar depois.



Para carregar o último jogo para voltar a Joga-lo basta escolher a opção Carregar no início do jogo:



Assim você pode continuar o jogo.

Fora isso o jogo funciona como um jogo de xadrez normal, más sem movimentos especiais, somente com movimentos simples.

Limitações e problemas:

A maior limitação do projeto é ela não ter uma interface gráfica, assim a jogabilidade fica mais cansativa e para tentar amenizar um pouco do problema, como o jogo é multiplayer, eu imprimo o tabuleiro deitado para ficar mais fácil para ambos os jogadores poderem ver o jogo.

Outra limitação é o projeto não ter movimentos especiais e nem o xeque, isso implica de o jogo não poder ser utilizado numa partida séria com regulamento oficial.

A última limitação que vi é de o jogador não conseguir salvar mais de uma partida.