

# Projeto 1

Mariane Lamas Malheiros

726569

## 1. Introdução

Neste documento está registrada a documentação externa da primeira fase do trabalho prático da disciplina de Programação Orientada a Objetos, que consiste na implementação e teste das classes necessárias para criar um jogo de xadrez para 2 jogadores em Java.

Todas as classes implementadas e testes serão descritos detalhadamente.

## 2. Classes

As classes cuja implementação foi exigida para esta fase do projeto são descritas a seguir.

### a. Jogo

A classe Jogo possui três atributos:

- “**Tabuleiro** *tabuleiro*”, objeto onde é definida a matriz de ponteiros para posições (que será alocada futuramente)
- “**StatusJogo** *status*”, que indica os 3 possíveis estados do jogo (START, CHECK, CHECK\_MATE).
- “**Vez** *vez*”, que identifica de qual jogador é a vez (JOGADOR1, JOGADOR2).

Os métodos da classe são:

- construtor “*Jogo()*”, que atribui START ao atributo status, indicando o início do jogo, JOGADOR1 ao atributo vez;
- “*getStatus()*”
- “*getVez()*”
- *setStatus(StatusJogo status)*
- *setVez(Vez vez)*



### b. Tabuleiro

A classe Tabuleiro possui apenas um atributo:

- “**Posicao** *posicoes[8][8]*” que é uma matriz de objetos de tipo Posicao, formando o tabuleiro.

Os métodos da classe são: .

- *Tabuleiro()*: Aloca espaço da matriz de posições e inicializa cada uma delas.
- *desenharTabuleiro()*: Imprime na primeira linha e primeira coluna os índices da matriz. Logo depois imprime em matriz a cor respectiva de cada posicao, indicando que o tabuleiro está vazio.



- **Rei(*Teams team*)** → construtor com parâmetro “*team*” instancia um novo objeto *Rei* com atributo *team* inicializado de acordo com o parâmetro fornecido e *status* inicializado como *INGAME*
- ***getStatus()*** → recupera o time da peça
- ***setStatus( status)*** → atualiza o status da peça de acordo com o parâmetro dado(INGAME ou CAPTURED)
- ***getTeam()*** → recupera o time da peça
- ***desenho()*** → método verifica a qual time a peça pertence e retorna o símbolo correspondente, sendo ‘’, caso o time seja preto, e ‘’, caso seja branco
- ***checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)*** → valida o movimento da peça dado como entrada pelo usuário, caso o movimento seja para qualquer direção em apenas 1 casa, o movimento é dado como válido.

## e. Rainha

A classe Rainha possui dois atributos:

- “**Teams team**”, que define em qual equipe a peça está(branca(WHITE) ou preta(BLACK)), e
- “**StatusPiece status**”, que representa o estado atual da peça, ou seja, se ela ainda está no jogo ou se já foi eliminada.

Os métodos da classe são: um construtor, “*desenho()*” e “*checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)*”.



- *Rainha(Teams team)* -> construtor com parâmetro “*team*” instancia um novo objeto *Rainha* com atributo *team* inicializado de acordo com o parâmetro fornecido e *status* inicializado como INGAME
- *getStatus()* -> recupera o time da peça
- *setStatus( status)* -> atualiza o status da peça de acordo com o parâmetro dado(INGAME ou CAPTURED)
- *getTeam()* -> recupera o time da peça
- *desenho()* -> método verifica a qual time a peça pertence e retorna o símbolo correspondente, sendo ‘’ caso o time seja preto, e ‘’ caso seja branco
- *checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)* -> valida o movimento da peça dado como entrada pelo usuário, caso o movimento seja na vertical, horizontal ou em qualquer diagonal, não importando o número de posições percorridas, o método retorna verdadeiro, caso não, retorna falso.

## f. Cavalo

A classe Cavalo possui dois atributos:

- “**Teams team**”, que define em qual equipe a peça está(branca(WHITE) ou preta(BLACK)), e
- “**StatusPiece status**”, que representa o estado atual da peça, ou seja, se ela ainda está no jogo ou se já foi eliminada.

Os métodos da classe são: um construtor, “*desenho()*” e “*checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)*”.

- *Cavalo(Teams team)* -> construtor com parâmetro “*team*” instancia um novo objeto *Cavalo* com atributo *team* inicializado de acordo com o parâmetro fornecido e *status* inicializado como INGAME
- *getStatus()* -> recupera o time da peça
- *setStatus( status)* -> atualiza o status da peça de acordo com o parâmetro dado(INGAME ou CAPTURED)
- *getTeam()* -> recupera o time da peça
- *desenho()* -> método verifica a qual time a peça pertence e retorna o símbolo correspondente, sendo ‘’ caso o time seja preto, e ‘’ caso seja branco



- *checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)* -> valida o movimento da peça dado como entrada pelo usuário, caso este seja de duas posições em linha reta (horizontal ou vertical), e depois uma posição em uma direção perpendicular à primeira, o método retorna verdadeiro, caso não, retorna falso.

## g. Bispo

A classe Bispo possui dois atributos:

- “**Teams team**”, que define em qual equipe a peça está(branca(WHITE) ou preta(BLACK)), e
- “**StatusPiece status**”, que representa o estado atual da peça, ou seja, se ela ainda está no jogo ou se já foi eliminada.

Os métodos da classe são: um construtor, “*desenho()*” e “*checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)*”.

- *Bispo(Teams team)* -> construtor com parâmetro “*team*” instancia um novo objeto *Bispo* com atributo *team* inicializado de acordo com o parâmetro fornecido e *status* inicializado como INGAME
- *getStatus()* -> recupera o time da peça
- *setStatus(status)* -> atualiza o status da peça de acordo com o parâmetro dado(INGAME ou CAPTURED)
- *getTeam()* -> recupera o time da peça
- *desenho()* -> método verifica a qual time a peça pertence e retorna o símbolo correspondente, sendo ‘’ caso o time seja preto, e ‘’ caso seja branco
- *checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)* -> valida o movimento da peça dado como entrada pelo usuário, caso este seja na diagonal, não importando o número de posições percorridas, o método retorna verdadeiro, caso não, retorna falso.

## h. Torre

A classe Torre possui dois atributos:

- “**Teams team**”, que define em qual equipe a peça está(branca(WHITE) ou preta(BLACK)), e
- “**StatusPiece status**”, que representa o estado atual da peça, ou seja, se ela ainda está no jogo ou se já foi eliminada.

Os métodos da classe são: um construtor, “*desenho()*” e “*checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)*”.

- *Torre(Teams team)* -> construtor com parâmetro “*team*” instancia um novo objeto *Bispo* com atributo *team* inicializado de acordo com o parâmetro fornecido e *status* inicializado como INGAME
- *getStatus()* -> recupera o time da peça

- *setStatus( status)* -> atualiza o status da peça de acordo com o parâmetro dado(INGAME ou CAPTURED)
- *getTeam()* -> recupera o time da peça
- *desenho()* -> método verifica a qual time a peça pertence e retorna o símbolo correspondente, sendo '♖' caso o time seja preto, e '♜' caso seja branco
- *checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)* -> valida o movimento da peça dado como entrada pelo usuário, caso este seja na vertical ou horizontal, não importando o número de posições percorridas, o método retorna verdadeiro, caso não, retorna falso.

## i. Peão

A classe Peao possui dois atributos:

- “**Teams team**”, que define em qual equipe a peça está(branca(WHITE) ou preta(BLACK)), e
- “**StatusPiece status**”, que representa o estado atual da peça, ou seja, se ela ainda está no jogo ou se já foi eliminada.

Os métodos da classe são: um construtor, “*desenho()*” e “*checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)*”.

- *Peao(Teams team)* -> construtor com parâmetro “team” instancia um novo objeto *Peao* com atributo *team* inicializado de acordo com o parâmetro fornecido e *emJogo* inicializado como verdadeiro;
- *getStatus()* -> recupera o time da peça
- *setStatus( status)* -> atualiza o status da peça de acordo com o parâmetro dado(INGAME ou CAPTURED)
- *getTeam()* -> recupera o time da peça
- *desenho()* -> método verifica a qual time a peça pertence e retorna o símbolo correspondente, sendo '♟' caso o time seja preto, e '♞' caso seja branco
- *checaMovimento(int linhaOrigem, int colunaOrigem, int linhaDestino, int colunaDestino)* -> valida o movimento da peça dado como entrada pelo usuário. Caso este seja de uma posição ou duas para frente na primeira jogada e apenas uma posição para frente nas jogadas subsequentes, o método retorna verdadeiro, caso não, retorna falso.

### 3. Testes

- Teste 1

**Objetivo do teste:** Criar um objeto de tipo Jogo, testa seu método getEstado(), getVez() e verifica se seu construtor funciona corretamente

**Objeto:** Jogo

**Método testado:**

- Classe -> Jogo
- Método -> Jogo()
- Método -> getEstado()
- Método -> getVez()

**Valores dos parâmetros:** getEstado() retorna 1, getVez retorna 1.

**Valor de retorno:** não há valor de retorno

**Tela:**

```
Status do Jogo: START
Vez de qual jogador: JOGADOR1
```

- Teste 2

**Objetivo do teste:** Verificar se o método desenharTabuleiro() imprime corretamente as posições do tabuleiro com suas respectivas cores em posições livres ou um símbolo de peça em posições ocupadas.

**Objeto:** tabuleiro

**Método testado:**

- Classe -> Tabuleiro
- Método -> desenharTabuleiro()

**Valores dos parâmetros:** não há valores de parâmetros

**Valor de retorno:** não há valor de retorno

**Tela:**

	A	B	C	D	E	F	G	H	
1	♣	♣	♣	♣	♣	♣	♣	♣	1
2	♣	♣	♣	♣	♣	♣	♣	♣	2
3	-	+	-	+	-	+	-	+	3
4	+	-	+	-	+	-	+	-	4
5	-	+	-	+	-	+	-	+	5
6	+	-	+	-	+	-	+	-	6
7	♣	♣	♣	♣	♣	♣	♣	♣	7
8	♣	♣	♣	♣	♣	♣	♣	♣	8
	A	B	C	D	E	F	G	H	

- Teste 3

**Objetivo do teste:** Para cada time(peças pretas ou peças brancas), retorna o caractere que representa seu objeto.


**Objeto:** rei1, rei0


**Método testado:**

- Classe -> Rei
- Método -> char desenho()

**Valores dos parâmetros:** O método *desenho()* não tem parâmetros, usa apenas o parâmetro do construtor.

**Valor de retorno:** o método *desenho()* retorna diferentes caracteres. *whiteKing(WHITE)*

cria uma peça branca, e o método *desenho()* retorna . *blackKing(BLACK)* cria uma

peça preta, e o método *desenho()* retorna .

**Tela:**



- Teste 4

**Objetivo do teste:** Verificar se o movimento dado é válido.

**Objeto:** whiteKing, blackKing

**Método testado:**

- Classe -> Rei
- Método -> checaMovimento()

**Valores dos parâmetros:**

6,4,5,4  
6,4,5,5  
6,4,6,5  
6,4,2,2  
6,4,6,9  
8,1,9,1

**Valor de retorno:** retorna um booleano True caso o movimento seja válido, e um booleano False caso o movimento seja inválido.

True  
True  
True  
False  
False  
False

**Tela:**

true  
true  
true  
false  
false  
false

- Teste 5

**Objetivo do teste:** Para cada time(peças pretas ou peças brancas), retorna o caractere que representa seu objeto.

**Objeto:** rainha1, rainha0

**Método testado:**

- Classe -> Rainha
- Método -> desenho()

**Valores dos parâmetros:** O método `desenho()` não tem parâmetros, usa apenas o parâmetro do construtor.

**Valor de retorno:** o método `desenho()` retorna diferentes caracteres.

`whiteQueen(WHITE)` cria uma peça branca, e o método `desenho()` retorna 🤴.

`blackQueen(BLACK)` cria uma peça preta, e o método `desenho()` retorna 🤴.

**Tela:**





- Teste 6

**Objetivo do teste:** Verificar se o movimento dado é válido.

**Objeto:** whiteQueen, blackQueen

**Método testado:**

- Classe -> Rainha
- Método -> checaMovimento()

**Valores dos parâmetros:**

6,4,5,4

6,4,5,5

6,4,6,5

6,4,2,2

6,4,6,9

8,1,9,1

**Valor de retorno:** retorna um booleano True caso o movimento seja válido, e um booleano False caso o movimento seja inválido.

True

True

True

False

False

False

**Tela:**

true

true

true

false

false

false

- Teste 7

**Objetivo do teste:** para cada time(peças pretas ou peças brancas), retorna o caractere que representa seu objeto.

**Objeto:** whiteHorse, blackHorse

**Método testado:**

- Classe -> Cavalo
- Método -> char desenho()

**Valores dos parâmetros:** O método *desenho()* não tem parâmetros, usa apenas o parâmetro do construtor

**Valor de retorno:** o método *desenho()* retorna diferentes caracteres.

*whiteHorse(WHITE)* cria uma peça branca, e o método *desenho()* retorna ♖.

*BlackHorse(BLACK)* cria uma peça preta, e o método *desenho()* retorna ♜.

**Tela:**



- Teste 8

**Objetivo do teste:** Verificar se o movimento dado é valido

**Objeto:** cavalo1

**Método testado:**

- Classe -> Cavalo
- Método -> checaMovimento()

**Valores dos parâmetros:**

5,5,4,3

5,5,3,4

5,5,3,6

5,5,4,7

5,5,6,7

5,5,7,6

5,5,7,4

5,5,6,3

5,5,3,5

5,5,5,9

5,5,5,4

5,5,2,2

**Valor de retorno:** retorna um booleano True caso o movimento seja válido, e um booleano False caso o movimento seja inválido.

True

True

True

True

True

True  
True  
True  
False  
False  
False  
False

Tela:

true  
true  
true  
true  
true  
true  
true  
true  
false  
false  
false  
false

- Teste 9

**Objetivo do teste:** para cada time(peças pretas ou peças brancas), retorna o caractere que representa seu objeto.

**Objeto:** whiteBishop, blackBishop

**Método testado:**

- Classe -> Bispo
- Método -> char desenho()

**Valores dos parâmetros:** O método *desenho()* não tem parâmetros, usa apenas o parâmetro do construtor

**Valor de retorno:** o método *desenho()* retorna diferentes caracteres.

*whiteBishop(WHITE)* cria uma peça branca, e o método *desenho()* retorna ♖.

*blackBishop(BLACK)* cria uma peça preta, e o método *desenho()* retorna ♜.

Tela:



- Teste 10

**Objetivo do teste:** Verificar se o movimento dado é válido

**Objeto:** bispo1

**Método testado:**

- Classe -> Bispo
- Método -> checaMovimento()

**Valores dos parâmetros:**

5,5,2,2

5,5,3,7

5,5,7,7

5,5,7,3

5,5,2,5

5,5,4,8

5,5,5,4

5,5,2,9

5,5,5,9

**Valor de retorno:** retorna um booleano True caso o movimento seja válido, e um booleano False caso o movimento seja inválido.

True

True

True

True

False

False

False

False

False

**Tela:**

true  
true  
true  
true  
false  
false  
false  
false  
false

- Teste 11

**Objetivo do teste:** para cada time(peças pretas ou peças brancas), retorna o caractere que representa seu objeto.

**Objeto:** whiteTower, blackTower

**Método testado:**

- Classe -> Torre
- Método -> char desenho()

**Valores dos parâmetros:** O método *desenho()* não tem parâmetros, usa apenas o parâmetro do construtor

**Valor de retorno:** o método *desenho()* retorna diferentes caracteres.

*whiteTower(WHITE)* cria uma peça branca, e o método *desenho()* retorna ♖.

*blackTower(BLACK)* cria uma peça preta, e o método *desenho()* retorna ♜.

**Tela:**



- Teste 12

**Objetivo do teste:** Verificar se o movimento dado é válido

**Objeto:** torre1

**Método testado:**

- Classe -> Torre
- Método -> checaMovimento()

**Valores dos parâmetros:**

5,5,7,5

5,5,5,1

5,5,4,4

8,1,8,10

8,1,9,9

**Valor de retorno:** retorna um booleano True caso o movimento seja válido, e um booleano False caso o movimento seja inválido.

True

True

False

False

False

**Tela:**

true  
true  
false  
false  
false

- Teste 13

**Objetivo do teste:** para cada time(peças pretas ou peças brancas), retorna o caractere que representa seu objeto.

**Objeto:** whitePawn, blackPawn

**Método testado:**

- Classe -> Peao
- Método -> char desenho()

**Valores dos parâmetros:** O método *desenho()* não tem parâmetros, usa apenas o parâmetro do construtor

**Valor de retorno:** o método *desenho()* retorna diferentes caracteres.

*whitePawn(WHITE)* cria uma peça branca, e o método *desenho()* retorna ♔.

*blackPawn(BLACK)* cria uma peça preta, e o método *desenho()* retorna ♚.

**Tela:**



- Teste 14

**Objetivo do teste:** Verificar se o movimento dado é válido para o time branco(true)

**Objeto:** whitePawn

**Método testado:**

- Classe -> Peao
- Método -> checaMovimento()

**Valores dos parâmetros:**

6,3,5,3  
6,3,4,3  
6,3,6,4  
6,3,7,4  
6,3,8,3  
6,3,9,3

**Valor de retorno:** retorna um booleano True caso o movimento seja válido, e um booleano False caso o movimento seja inválido.

True  
True  
False  
False  
False  
False

**Tela:**

true  
true  
false  
false  
false  
false

- **Teste 15**

**Objetivo do teste:** Verificar se o movimento dado é válido para o time preto(false)

**Objeto:** blackPawn

**Método testado:**

- Classe -> Peao
- Método -> checaMovimento()

**Valores dos parâmetros:**

1,4,3,4  
1,4,2,4  
1,4,3,5  
1,4,2,5  
1,4,1,4  
1,4,2,9

**Valor de retorno:** retorna um booleano True caso o movimento seja válido, e um booleano False caso o movimento seja inválido.

True

True  
False  
False  
False  
False

Tela:

true  
true  
false  
false  
false  
false

## 4. Enums

Como é possível ver, foram utilizados enums para definir constantes de cores, para as posições, status, para o jogo, peças, e posições, vez e times. Isso foi feito para facilitar o entendimento do código.

## 5. Dificuldades encontradas

As dificuldades foram principalmente por conta do ensino remoto e da situação atual do mundo, o que afetou minha capacidade de concentração e levou a erros que normalmente não aconteceriam. Outra dificuldade foi encontrar um jeito melhor de representar as peças.

## 6. Possíveis extensões e melhorias no programa

Uma possível extensão e melhoria no projeto seria melhorar o tamanho dos caracteres utilizados para representar as peças. É possível que algum erro no movimento das peças seja encontrado, pois testes feitos pelo desenvolvedor do código tendem a ser tendenciosos e nem sempre encontram todos os erros na implementação. Além disso, também é possível que haja erro nas entradas fornecidas.

## 7. Diagrama de Classes

Devido ao tamanho da imagem, o diagrama de classes será anexado no formato de uma imagem PNG nomeada "Diagrama de Classes".