Instituto Superior
de Engenharia
Politécnico de Coimbra

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Programação avançada

Relatório do Trabalho Pratico

Turmas:

Membros da Equipa:

- Sebastian Gonçalves (2023155905)– a2023155905@isec.pt
- Ivanilson Da Silva (2023158536) - a2023158536@isec.pt
- Frederico Quelhas (2022135081) – a2022135081@isec.pt

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

Conteúdo

1 Introdução

O presente relatório final descreve o desenvolvimento de uma aplicação em Java que simula o jogo de xadrez, no âmbito da unidade curricular de Programação Avançada. O principal objetivo deste trabalho consistiu na aplicação dos conceitos de Programação Orientada por Objetos, da biblioteca JavaFX para a interface gráfica, bem como de padrões arquiteturais e funcionais abordados ao longo do semestre.

O projeto foi desenvolvido de forma progressiva, ao longo de várias etapas práticas, cada uma centrada em componentes essenciais da aplicação: desde a modelação das peças e do tabuleiro, até à implementação da interface gráfica, gestão do estado do jogo e incorporação de funcionalidades de acessibilidade. Todas as decisões de implementação seguiram as orientações do enunciado, assegurando uma clara separação entre o modelo e a interface com o utilizador, bem como a utilização de uma fachada para mediar o acesso à lógica do jogo.

Adicionalmente, foi dada especial atenção à organização modular do projeto, à adoção de boas práticas de programação e à realização de testes unitários nas principais componentes do modelo. Este relatório apresenta as opções tomadas, os padrões utilizados, a estrutura das classes desenvolvidas e uma avaliação do grau de implementação das funcionalidades previstas.

2 Implementação

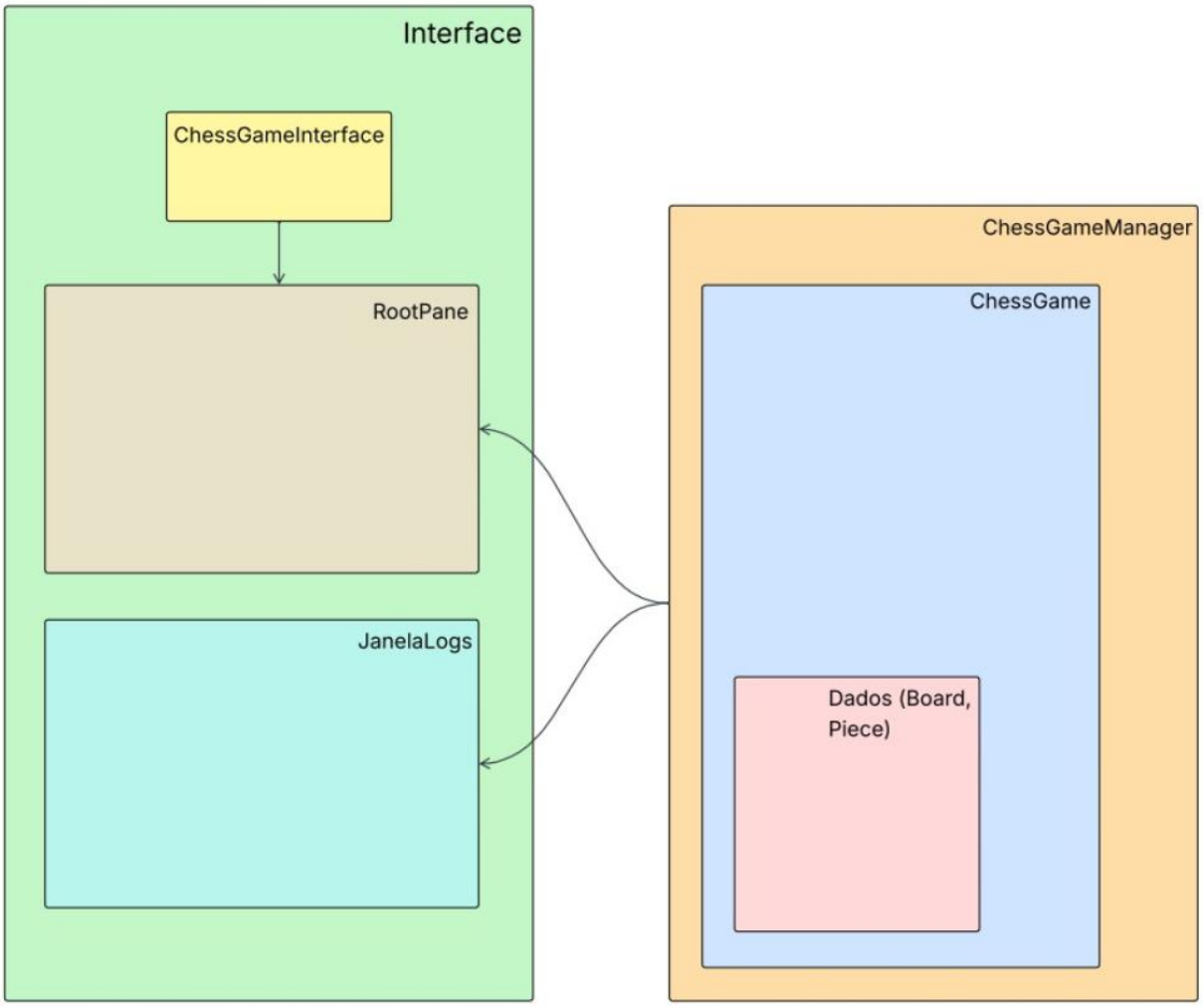
2.1 Arquitetura Geral

O principal padrão implementado neste trabalho foi o Model-View-Controller (MVC). Este padrão envolve as principais classes do projeto, sendo que, na parte do Modelo, se destacam as classes ChessGame.java, Board.java, Piece.java (e as suas classes derivadas) e ChessGameManager.

Relativamente à View, esta é suportada pelas classes ChessBoardInterface.java, RootPane.java e JanelaLogs.java.

Mais adiante será apresentada uma breve descrição das funcionalidades de cada uma destas classes. Para já, importa referir que estas constituem os principais componentes da implementação da arquitetura MVC.

A comunicação entre o Modelo de Dados (Model) e a Interface Gráfica (View) é realizada através da classe ChessGameManager, que funciona como o nosso Controller. Esta atua também como fachada, assegurando a separação entre os dados e a interface gráfica, com o objetivo de manter a persistência dos dados e garantir uma maior segurança e controlo sobre os mesmos.



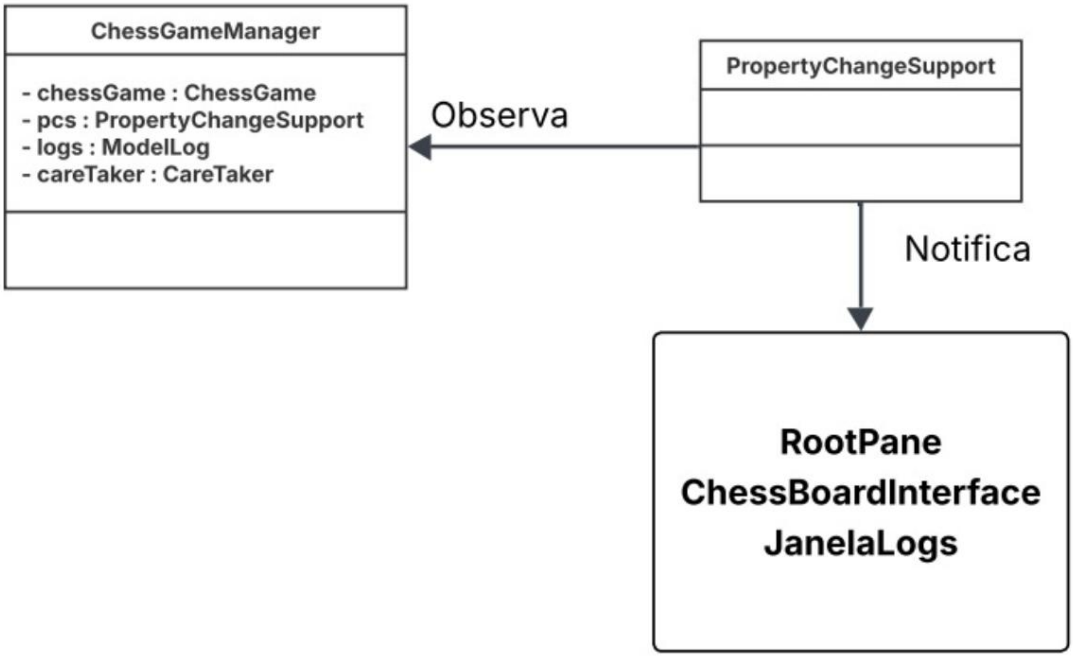
2.2 Padrões de Design de Software Implementados

Neste projeto foi implementado o MVC como arquitetura principal, porém também foram implementados outro tipo de padrões de design de software. Estes foram o Observer pattern, Memento pattern, Singleton Pattern, o Facade pattern e o Factory Pattern.

O padrão Observer está a ser implementado no projeto através do uso de PropertyChangeSupport e PropertyChangeListener da API Java, permitindo uma comunicação desacoplada entre o modelo e as interfaces gráficas.

Principais Componentes no padrão Observer:

- Observáveis:
  - ChessGameManager como observável principal:
    - Mantém um objeto PropertyChangeSupport pcs.
    - Define constantes para tipos de eventos (PROP_BOARD, PROP_CURRENT_PLAYER, etc.).
    - Fornece métodos para registar/remover listeners.
    - Notifica observadores com pcs.firePropertyChange().
  - ModelLog como observável secundário:
    - Implementa PropertyChangeSupport para notificar sobre novos logs.
    - Usa firePropertyChange("log", null, log) para notificar mudanças.
    - É um singleton que centraliza o registo de log.
- Observadores:
  - RootPane:
    - Registra-se como listener para mudanças no tabuleiro e jogador atual.
    - Implementa atualização da interface com update() e atualizarJogadorAtual().
    - Responde a mudanças de estado atualizando componentes visuais.
  - ChessBoardInterface:
    - Observa eventos de tabuleiro e jogador atual.
    - Redesenha o tabuleiro quando notificado com draw().
    - Atualiza informações visuais específicas do tabuleiro.
  - JanelaLogs:
    - Implementa PropertyChangeListener para receber eventos do ModelLog.
    - Atualiza a lista de logs em tempo real.
    - Responde a eventos do jogo para exibir histórico de ações.

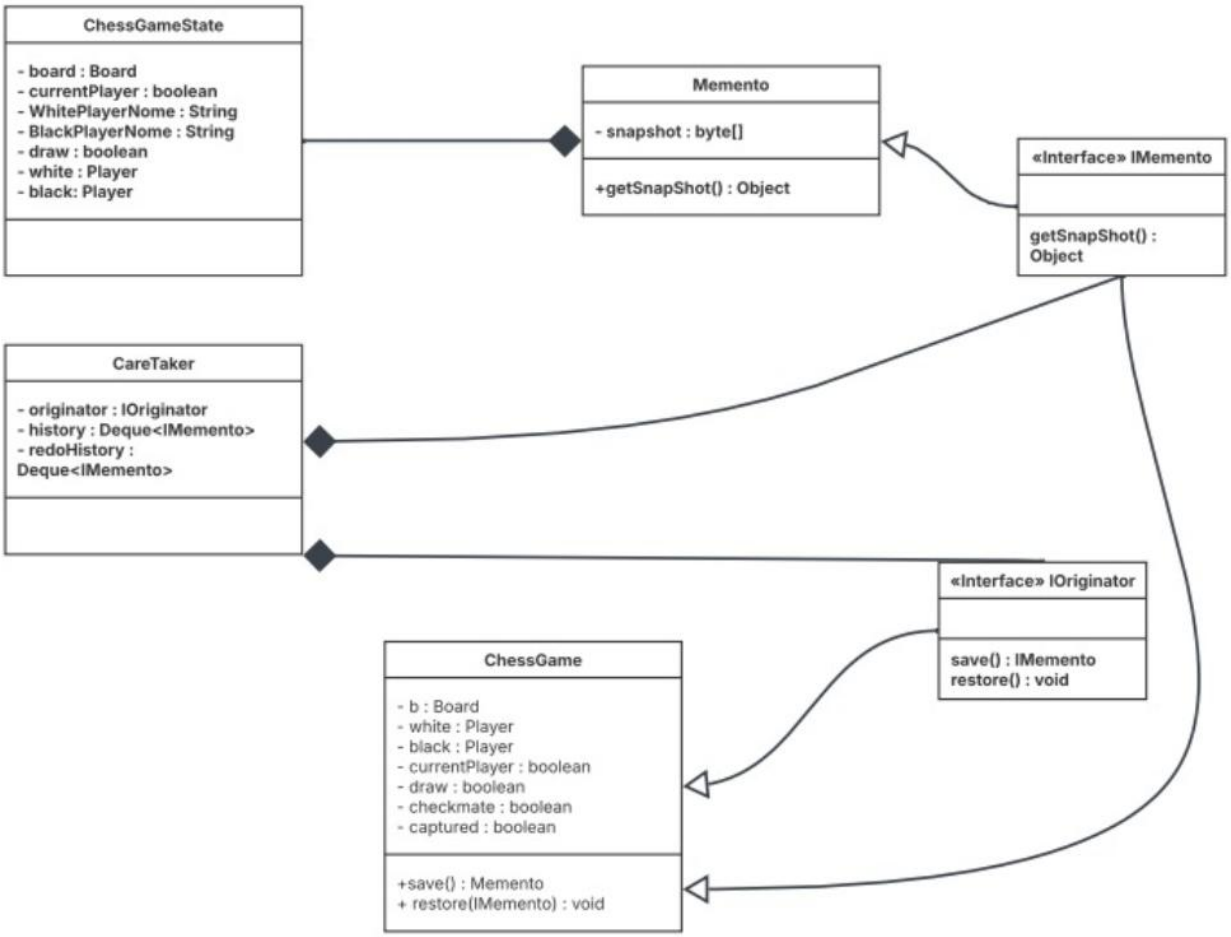


O padrão Memento está a ser implementado no projeto para permitir a funcionalidade de desfazer/refazer movimentos no jogo de xadrez, especialmente útil no modo de aprendizagem. No entanto não está a funcionar, mas a implementação vai ser detalhada abaixo.

Principais Componentes no padrão Memento:

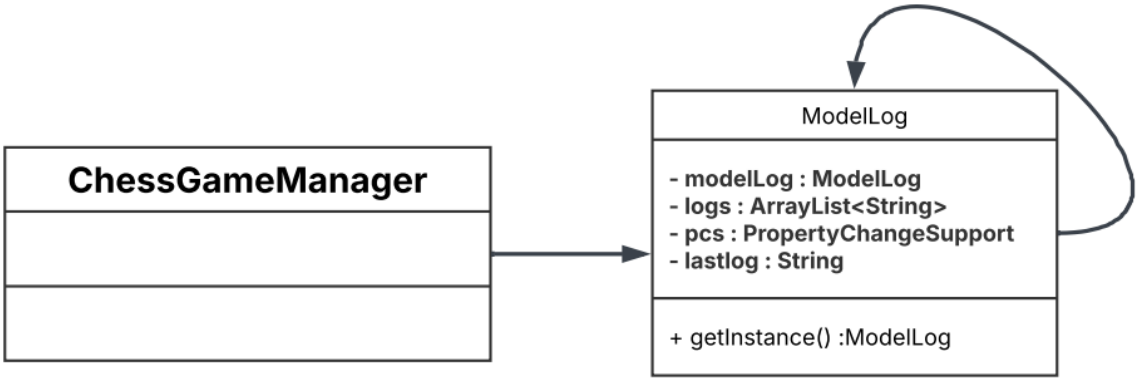
- Originator:
  - ChessGame:
    - Implementa a interface IOriginator.
    - Fornece método save() que cria um Memento contendo o estado atual do jogo.
    - Fornece método restore(IMemento memento) para restaurar um estado anterior.
- Memento:
  - Memento:
    - Implementa a interface IMemento.
    - Armazena um snapshot do estado do jogo via serialização binária.
    - Fornece método getSnapshot() para recuperar o estado salvo.
  - ChessGameState:
    - Encapsula os dados essenciais do estado do jogo.

- CareTaker:
  - Mantém duas pilhas (Deque<IMemento>): uma para histórico de ações e outra para redo.
  - Fornece métodos save(), undo() e redo().
  - Gerência a navegação pelo histórico de estados do jogo.
  - Mantém a integridade ao limpar o histórico de redo quando uma nova ação é realizada.



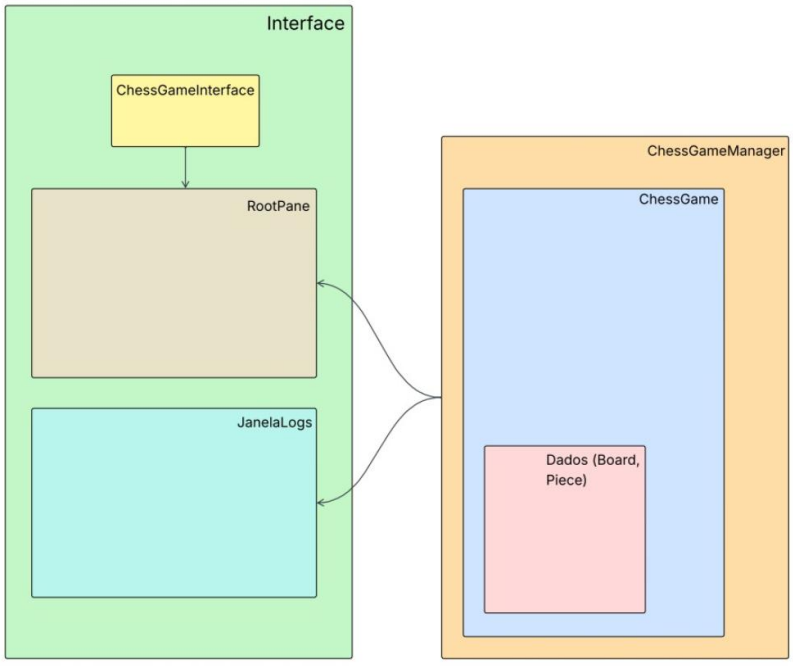
O padrão Singleton é implementado através da classe ModelLog. O ModelLog é um Singleton que mantém um registo centralizado de todas as ações e eventos do jogo de xadrez. As características implementadas deste padrão são as seguintes:

- Instância estática privada: Existe uma variável estática privada que mantém a única instância da classe.
- Construtor privado: O construtor da classe é privado, o que impede a criação de novas instâncias fora da classe.
- Método de acesso estático getInstance(): Este método fornece acesso à única instância da classe. Se a instância ainda não existir, ele cria-a e depois devolve-a; caso contrário, simplesmente devolve a instância já existente.
- Funcionalidades da classe: O ModelLog fornece métodos para adicionar logs, notificar ouvintes de eventos e gerenciar o histórico de logs do jogo.



O padrão Facade foi implementado principalmente através da classe ChessGameManager, que atua como uma fachada entre a interface de gráfica e a lógica do jogo de xadrez.

O ChessGameManager contém uma instância de ChessGame e coordena todas as interações com esta classe. Em vez de a interface gráfica interagir diretamente com ChessGame, ela usa os métodos do ChessGameManager.



O padrão de Factory está implementado para a criação de peças do jogo. Na classe Piece implementou – se dois métodos estáticos createPiece que funcionam como factory methods para criar os diferentes tipos de peças de xadrez. Estes métodos aceitam parâmetros como símbolos de caracteres ou um enum TYPE e retornam a peça apropriada.

3 Relação entre classes

3.1 Relação entre classes no Modelo de Dados (Model)

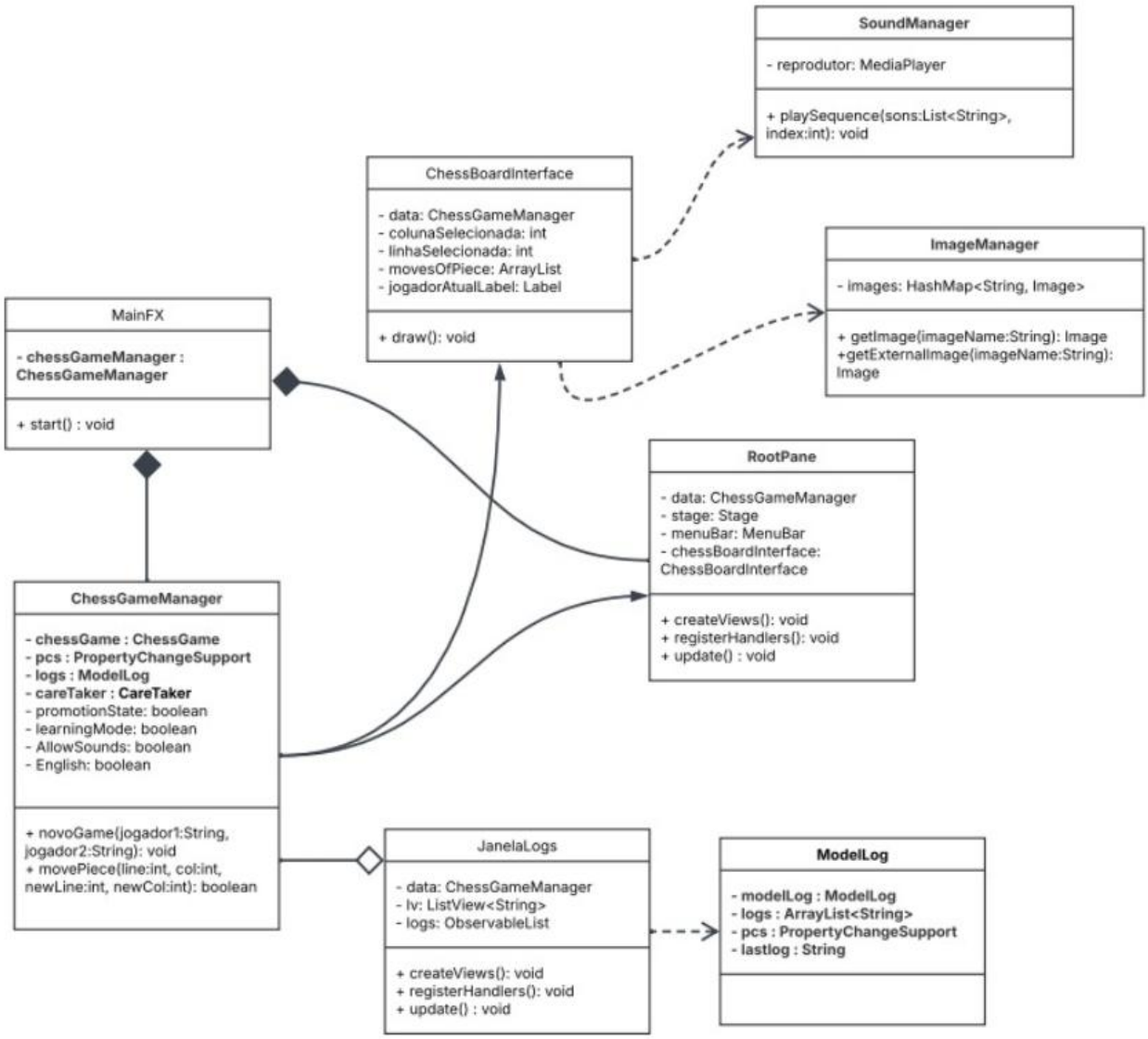
O sistema é construído em torno da classe ChessGame, que funciona como o núcleo do jogo, controlando o tabuleiro, os jogadores, o turno atual, verificações de xeque-mate e outras regras essenciais. Cada jogador é representado pela classe Player, que armazena seu nome, cor (branco ou preto) e a lista de peças sob seu controle.

O tabuleiro (classe Board) é modelado como uma matriz 8x8 de peças, onde cada posição pode conter uma peça ou ficar vazia. Todas as peças herdam da classe abstrata Piece, que define comportamentos comuns, enquanto classes especializadas como King, Queen, Rook, Bishop, Knight e Pawn implementam os movimentos específicos de cada tipo.

Para permitir recursos como desfazer e refazer jogadas, o sistema utiliza o padrão Memento, composto por três elementos principais: Memento (que armazena um estado do jogo), CareTaker (que gerencia o histórico de estados) e IOriginator (uma interface que permite salvar e restaurar o estado do jogo). Além disso, a classe ChessGameState é responsável por guardar um snapshot completo do jogo, facilitando a restauração em caso de necessidade.

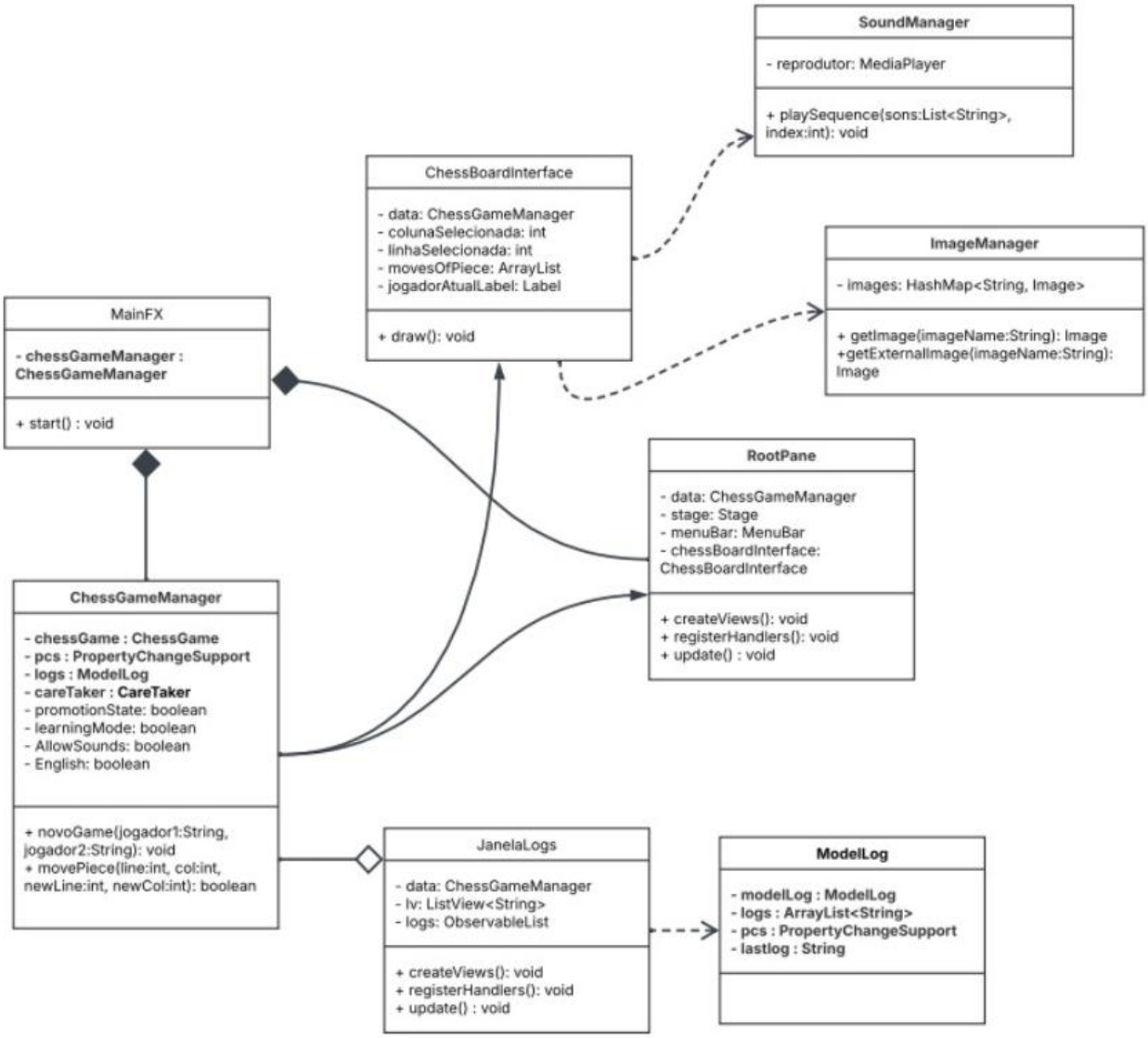
O ChessGameManager atua como o controlador principal, coordenando o fluxo do jogo, gerenciando o histórico de movimentos (através do CareTaker) e registrando eventos importantes em logs (via ModelLog). Movimentos especiais, como promoção de peão, xeque ou xeque-mate, são classificados pela enumeração AcontecimentoMovimento, que ajuda a identificar e tratar cada situação de forma adequada.

Por fim, o sistema oferece suporte a serialização por meio da classe ChessGameSerialization, permitindo que partidas sejam salvas em arquivo e carregadas posteriormente. Isso garante que os jogadores possam interromper e retomar o jogo sem perder progresso.



3.2 Relação entre as classes na Vista (View)

MainFX cria e detém uma instância de ChessGameManager, sendo o ponto de entrada da aplicação. MainFX também cria o RootPane, passando-lhe o ChessGameManager, estabelecendo a ligação com a lógica do jogo.
RootPane contém uma referência a ChessBoardInterface, o qual também recebe o ChessGameManager para refletir o estado do jogo.
Tanto RootPane como ChessBoardInterface comunicam com ChessGameManager para obter ou modificar o estado do jogo. ChessGameManager usa PropertyChangeSupport para notificar alterações, seguindo o padrão Observer. JanelaLogs é também observadora de ChessGameManager, escutando alterações nos logs de jogo.
ModelLog está associado a ChessGameManager e serve como fonte de dados para JanelaLogs.
SoundManager e ImageManager são utilizados por componentes gráficos (como ChessBoardInterface) para sons e imagens.
Todas as interfaces gráficas partilham uma dependência comum do ChessGameManager, mantendo a lógica centralizada.


SoundManager
- reprodutor: MediaPlayer
+ playSequence(sons:List<String>, index:int): void
ChessBoardInterface
- data: ChessGameManager
- colunaSelecionada: int
- linhaSelecionada: int
- movesOfPiece: ArrayList
- jogadorAtualLabel: Label
+ draw(): void
ImageManager
- images: HashMap<String, Image>
+ getImage(imageName:String): Image
+getExternalImage(imageName:String): Image
MainFX
- chessGameManager : ChessGameManager
+ start() : void
RootPane
- data: ChessGameManager
- stage: Stage
- menuBar: MenuBar
- chessBoardInterface: ChessBoardInterface
+ createViews(): void
+ registerHandlers(): void
+ update() : void
ChessGameManager
- chessGame : ChessGame
- pcs : PropertyChangeSupport
- logs : ModelLog
- careTaker : CareTaker
- promotionState: boolean
- learningMode: boolean
- AllowSounds: boolean
- English: boolean
+ novoGame(jogador1:String, jogador2:String): void
+ movePiece(line:int, col:int, newLine:int, newCol:int): boolean
JanelaLogs
- data: ChessGameManager
- lv: ListView<String>
- logs: ObservableList
+ createViews(): void
+ registerHandlers(): void
+ update() : void
ModelLog
- modelLog : ModelLog
- logs : ArrayList<String>
- pcs : PropertyChangeSupport
- lastlog : String

4 Breve Descrição de cada classe implementada

4.1 Classes das Peças

Nome da classe	Descrição
Piece	É uma classe abstrata que é usada para generalizar cada peça e ter uma base em comum
Pawn	A classe do peão encarrega-se de ter todas as regras com as quais o peão vai seguir
Rook	Igual que o peão, tem todas as regras para a torre, incluindo movimentos em linha reta horizontal e vertical
Bishop	Classe que implementa o movimento do bispo, que se desloca apenas na diagonal por quantas casas quiser
Knight	Classe responsável pelo cavalo, que se move em "L" (duas casas numa direção e uma casa perpendicular) e pode saltar sobre outras peças
Queen	Classe que combina os movimentos da torre e do bispo, podendo mover-se em qualquer direção (horizontal, vertical ou diagonal) por quantas casas quiser

4.2 Classes do Jogo

Nome da classe	Descrição
Board	Contem a matriz de peças que vai ser considerada como o tabuleiro do jogo, também tem alguns métodos referentes como por exemplo a obter a peça de uma determinada posição.
ChessGame	Encarrega-se de ter os métodos das regras do jogo assim como o nome dos jogadores e uma instância da classe **Board.**
Player	Nesta classe guardada qualquer tipo de informação referente a um dos jogadores, como as suas peças ou o seu nome.
ChessGameSerialization	O propósito desta classe é para controlar a serialização da classe **ChessGame.**

4.3 Classes da interface

Nome da classe	Descrição
ChessGameManager	Esta classe serve como uma façade para ligar a vista com o modelo.

ChessBoardInterface	Esta classe tem o propósito de desenhar o tabuleiro com as peças para depois inserir na stage definida no **RootPane.**
JanelaLogs	Esta classe é usada para para a posterior criação de uma segunda Stage para guardar os logs que neste caso seriam acontecimentos no jogo.
RootPane	Esta classe encarrega-se da criação das vistas e dos handlers que vão ser usados no **MainFX**
MainFX	Aqui usamos o **RootPane** criamos a Scene para o **RootPane** e também criamos a janela dos logs.

4.4 Memento

Nome da classe	**Descrição**
CareTaker	Guarda e gere os estados anteriores do jogo (Mementos). Responsável por: - Armazenar o histórico de estados - Recuperar estados anteriores (undo) - Limitar o número de estados guardados (para otimização de memória)
ChessGameState	Representa o estado completo do jogo num momento específico, incluindo: - Posição de todas as peças - Jogador atual - Estado de xeque/roque/etc. - Histórico de jogadas
Memento	Padrão de design que captura e externaliza o estado interno do tabuleiro sem violar o encapsulamento. Contém: - Snapshot do estado do tabuleiro - Métodos para restaurar estado

5 O que foi implementado

Feature	Desenvolvimento
Definição do modelo de dados	Completamente implementado
Gestão de jogo e persistência dos dados	Completamente implementado
Interface Gráfica java FX para a gestão da aplicação	Completamente implementado
Interface gráfica Java FX para o jogo	Completamente implementado
Incorporação de façade observável	Completamente implementado
Incorporação de modo de aprendizagem de xadrez e acessibilidade	Parcialmente implementado: foram completados a utilização de sons para indicar acontecimentos no jogo, mas faltou a completa implementação da funcionalidade de undo e redo.

6 Conclusão

A realização deste trabalho, surgiu como sendo uma oportunidade de aplicarmos os conceitos teóricos acerca dos paradigmas da Programação orientada a objetos adquiridos durante as aulas, no desenvolvimento de aplicação que simula um jogo de xadrez usando os recursos da biblioteca JavaFX.

A aplicação de design patterns como Observer, Memento, Singleton e Facade desempenharam um papel crucial na separação entre os elementos do modelo e da interface, garantindo o cumprimento das boas normas de programação.

Apesar da nossa notória dedicação em cumprir com todas as tarefas impostas por cada etapa, verifica-se o cumprimento parcila de alguns requisitos conforme descritos na tabela das funcionalidades no ponto 4. No entanto implementação do modo de aprendizagem de xadrez e acessibilidade verificou-se estas falhas, pois, embora a utilização de sons para indicar acontecimentos no jogo tenha sido concluída, a funcionalidade completa de desfazer (undo) e refazer (redo) não foi finalizada."