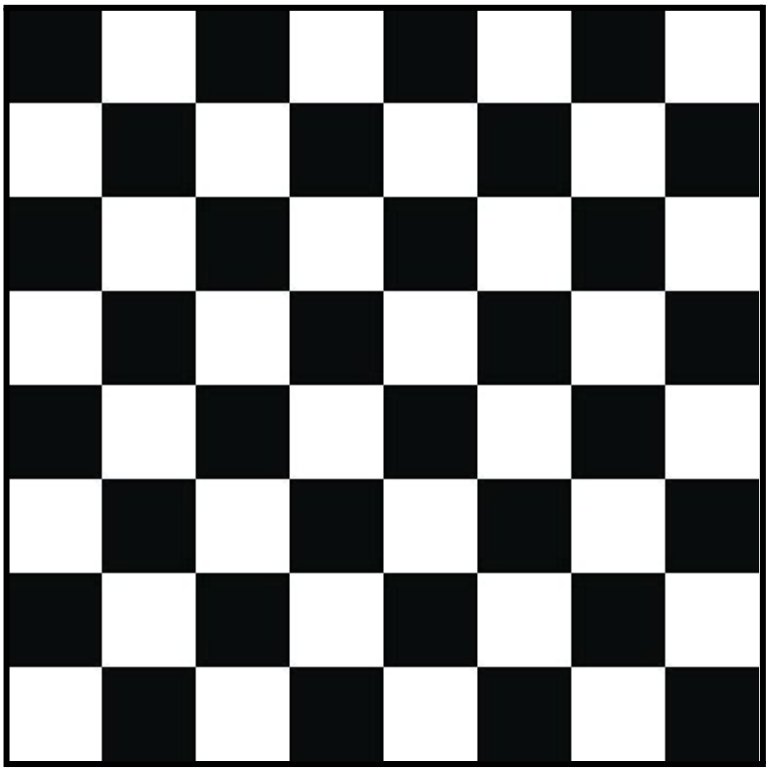
CHECKERS

Object Oriented Project in Java



June 2024

Index

Introduction

The objective of this project was to develop a "Checkers" game using Java classes. This report provides a detailed overview of the class structures and offers a comprehensive explanation of how the Checkers game operates behind the scenes.

In the following sections, we will explore the design and implementation of the various classes that form the backbone of the game. This includes classes representing the game board, the individual pieces, and the user interface. Each class has been designed to encapsulate specific functionalities, making the overall system modular and easy to manage.

The game includes features such as forced captures, piece promotion to queens, and alternating turns between two players. These rules are implemented through a series of method calls and conditional checks, ensuring a seamless and accurate gameplay experience.

Moreover, the report highlights the use of XML for saving the outcome of the game (recording which pieces were taken and by what player). This involves creating XML files using Java's built-in XML processing libraries.

The game offers both a console and graphical user interface.

By the end of this report, readers will gain a thorough understanding of both the structural and functional aspects of the Checkers game developed in Java, appreciating the complexity and thoughtfulness required to bring a classic board game to the digital realm.

Key Features

1. **Inheritance**: The use of inheritance allows for a clean separation of shared and unique behaviors between different types of pieces. The ‘Piece’ class provides common functionality, while the Pawn and Queen classes override methods to implement their specific movement rules.

2. **Board Management**: The Board class encapsulates the game board's state and provides methods to manipulate it. This includes moving pieces, checking availability, and handling captures.

3. **User Interface**: The UI_console class provides a text-based interface for interacting with the game. It handles user input, displays the board, and manages the game loop. Alternatively a GUI can also be used. The GUI offers a more immersive and enjoyable experience, however, for the sake of simplicity, the console UI will be the one represented in the UML diagrams.

4. **XML Update**: The updateXML method in UI_console demonstrates how the game can persist data, such as captured pieces, in an XML format. This method creates an XML document and writes the captured pieces' information to it.

UML Diagram

In this section are presented the Unified Modeling Language (UML) diagrams that illustrate the structural and behavioral aspects of the "Checkers" game developed in Java. UML is a standardized modeling language that provides a comprehensive way to visualize the design and architecture of software systems.

- In this project the classes are defined as followed:

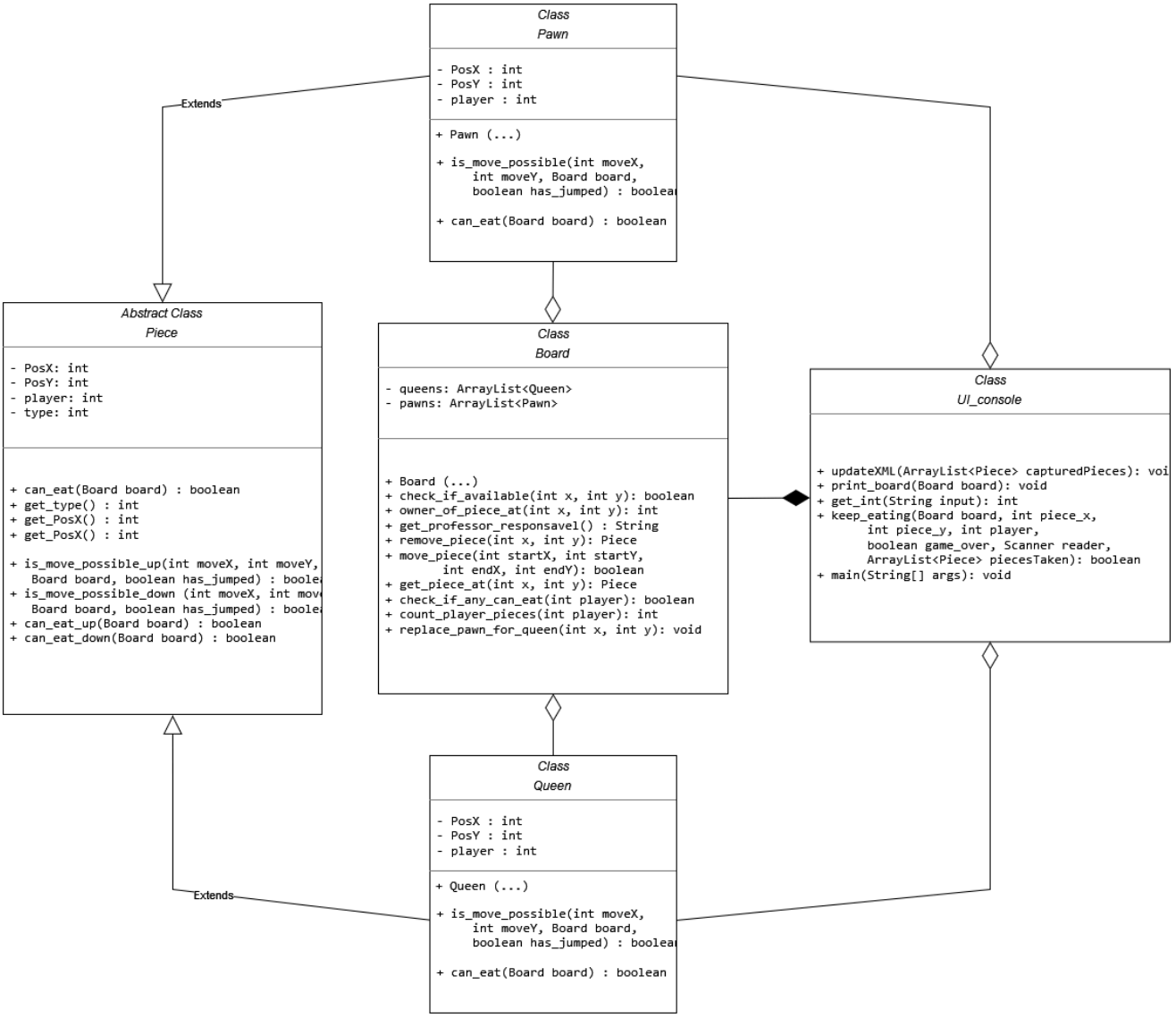


Figure 1 - UML Diagram

The UML diagram presented previously illustrates the main classes involved in the game, such as Board, Piece, Pawn, Queen, and Player, along with their attributes and methods.

A key aspect of the "Checkers" game design is the use of inheritance to manage the different types of pieces on the board. In this implementation, the ‘Piece’ class serves as a base class, with the ‘Pawn’ and ‘Queen’ classes inheriting from it. This inheritance structure allows for a clean and efficient representation of the shared and unique characteristics of different game pieces.

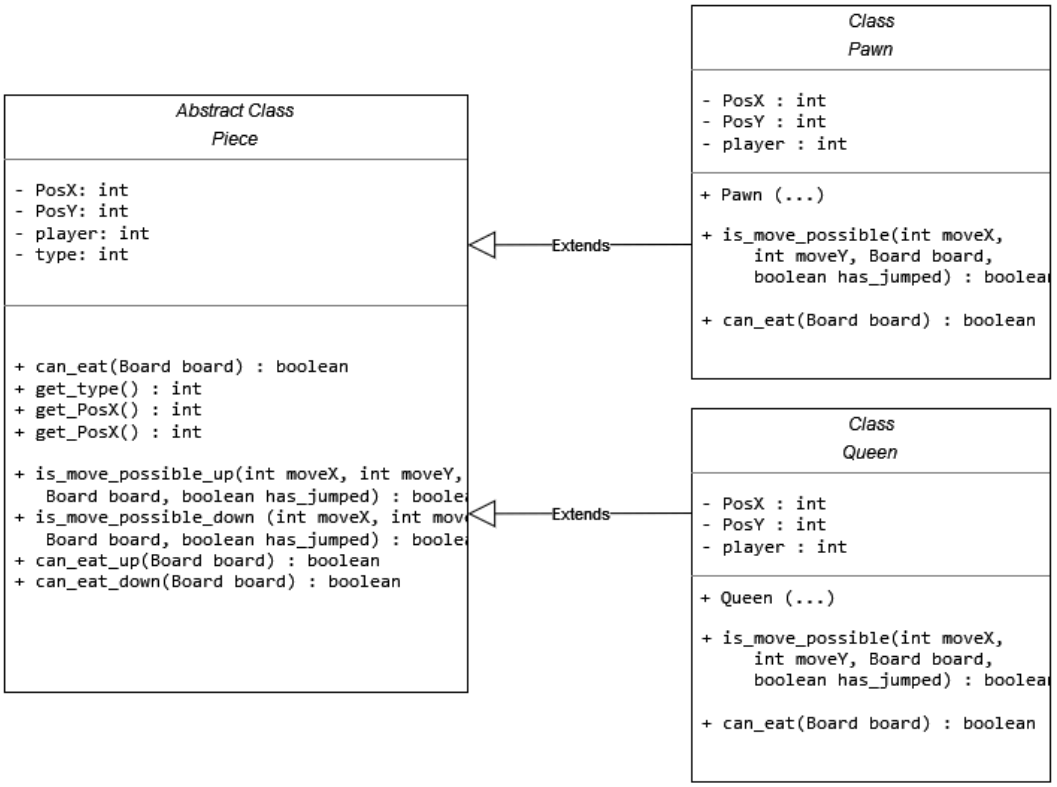


Figure 2 - Demonstration of the inheritance of class ‘Piece'

Classes

Piece

The Piece class serves as the abstract base class for all game pieces. It encapsulates common attributes and methods that are shared among different types of pieces. Key attributes include the piece's position (posX, posY), type (pawn or queen), and the owning player. It also defines abstract methods like is_move_possible and can_eat, which are implemented by derived classes.

Pawn

The Pawn class extends the Piece class, representing a basic pawn in the game. It implements the abstract methods defined in Piece, providing specific logic for pawn movements and captures. Pawns can only move diagonally forward and have unique capturing mechanics.

Queen

The Queen class also extends the Piece class, representing a queen in the game. It overrides the movement and capturing methods to account for the enhanced capabilities of a queen, which can move and capture diagonally in any direction.

Board

The Board class represents the game board, managing the state of all pieces and providing methods to interact with the board. It includes methods to check the availability of positions, move pieces, remove pieces, and count pieces for each player.

UI_console

The UI_console class handles the user interface and interaction logic for the console-based version of the game. It includes methods to print the board, process player inputs, and manage the game flow. It also contains the updateXML method for saving captured pieces to an XML file.

User Interface

The UI_console class provides a console-based user interface for a simple checkers game. This UI allows users to play the game by inputting coordinates and provides a visual representation of the game board. The key components and functionality of the user interface are described below.

1. **User Prompts and Input**: The console UI prompts the user for input at various stages:
    - **Choosing Pieces**: Players are asked to select pieces they wish to move by inputting coordinates (e.g., 'A3', 'G6').
    - **Moving Pieces**: After selecting a piece, players input the coordinates where they want to move the piece.
    - **Forced Moves**: The UI handles scenarios where a player is forced to make a capturing move, guiding the player to comply with the rules.

2. **Game Status Updates**: The UI provides real-time feedback on game status:
    - **Turn Indication**: Indicates which player's turn it is.
    - **Move Validity**: Checks and notifies players if their moves are invalid, prompting for correct input.
    - **Piece Upgrades**: Notifies players when a pawn is upgraded to a queen.

3. **Game Over Announcement**: At the end of the game, the UI announces the winner based on the remaining pieces on the board.

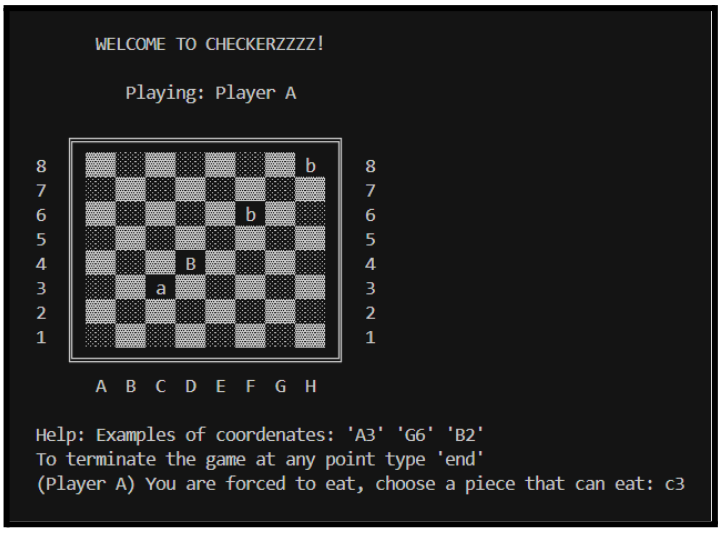


Figure 3 - User interface

XML

At the end of each game, the pieces taken (and their positions) are recorded into an XML file. The XML file used to store captured pieces follows a structured format. Here's an example of what the XML content might look like:

```
2   <CapturedPieces>
3       <PlayerA>
4           <Queen coords="D4"/>
5           <Pawn coords="F6"/>
6       </PlayerA>
7       <PlayerB>
8           <Pawn coords="G7"/>
9       </PlayerB>
10  </CapturedPieces>
```

Figure 4 - Example of XML output

Each <Player> element contains information about captured pieces, including its types (Pawn or Queen) and its positions (posX and posY).

Graphical User Interface

In addition to the console UI, the code also offers the possibility to use a graphical interface. This allows the user to have a more immersive and enjoyable experience.
This GUI replaces the old UI_console, maintaining the same relations to the other classes.

The following diagram represents the class relations for the new GUI that were not present for the console UI.

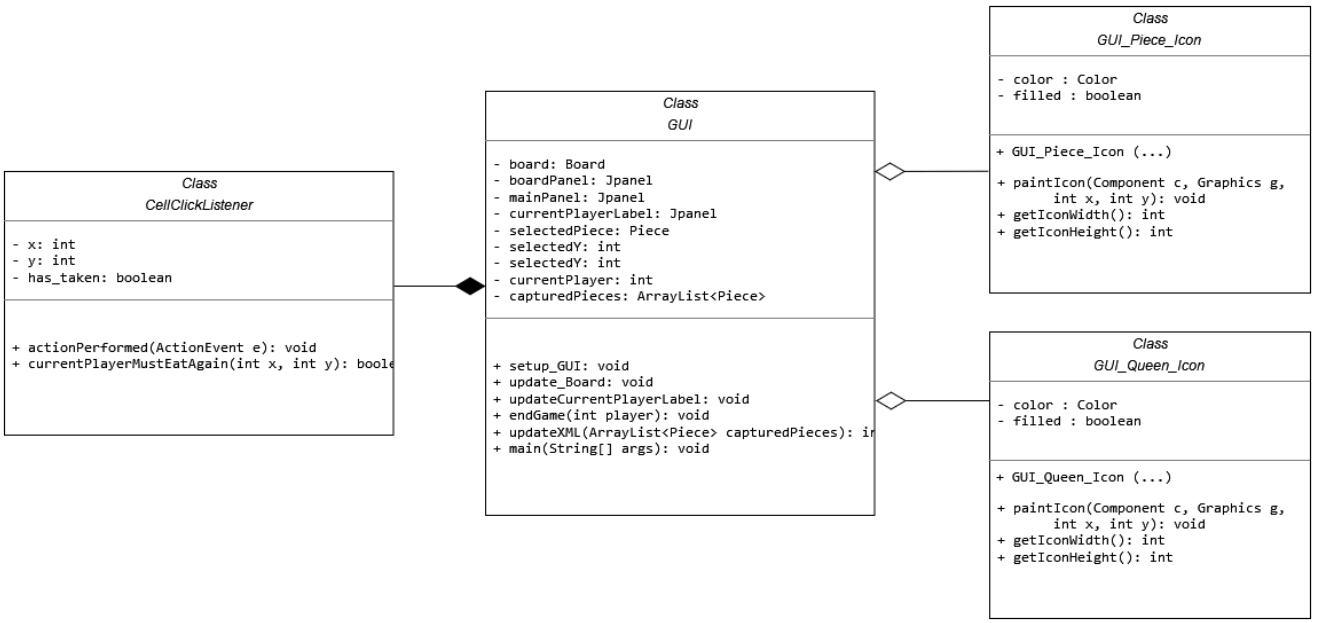


Figure 5 - UML class diagram for the new classes

These new classes allow the user to interact with the GUI, being GUI_Piece_Icon and GUI_Queen_Icon the classes that allow us to draw the pieces and CellClickListener the class that allows us to react to the user input.

In addition to these classes, the GUI has the same relation to the Pawn, Queen and Board classes as the UI_console, as represented in page 4 of this report.

Graphical Interface Features

User Prompts and Input: The console UI prompts the user for input at various stages:

- **Choosing Pieces**: Players are asked to select a piece. Selected piece is highlighted.
- **Moving Pieces**: After selecting a piece, players choose the position where they want to move the piece to.
- **Forced Moves**: The GUI also handles scenarios where a player is forced to make a capturing move, guiding the player to comply with the rules.

Game Status Updates: The GUI provides real-time feedback on game status:

- **Turn Indication**: Indicates which player's turn it is.
- **Move Validity**: Highlights the piece chosen by the player if it is a valid movable piece.

Game Over Announcement: At the end of the game, the UI announces the winner based on the remaining pieces on the board. After that, the pieces taken during the game are recorded to an xml file.

'End Game' Button: In case a player wants to resign or end the game he can do so by clicking the 'End Game' button.

Figure 6 - Graphical user interface

Conclusion

This project successfully demonstrates the development of a Checkers game using Java, focusing on object-oriented programming principles and effective data management techniques. By employing classes and inheritance, the project encapsulates the core functionalities and behaviors of the game pieces and the game board, ensuring a clean and maintainable codebase.

Key Accomplishments:

1. **Object-Oriented Design**:
    - The game leverages object-oriented principles, creating a robust hierarchy with Piece, Pawn, and Queen classes. This design promotes code reuse and extensibility.

2. **Graphical Representation**:
    - The console-based user interface provides a clear and user-friendly way to interact with the game, displaying the board state and guiding players through their moves.

3. **Data Persistence with XML**:
    - The implementation of XML for captured pieces adds an extra layer of functionality, allowing the game to record the state of captured pieces efficiently. This can be extended to save and load entire game states in future enhancements.

4. **Comprehensive Game Logic**:
    - The game includes all essential rules of Checkers, including forced captures and piece promotion, ensuring a challenging and realistic gameplay experience.

5. **Development of a graphical user interface**:
    - The game includes a graphic interface. This is created using JavaSwing, allowing the game to be more interactive and entertaining for the user.

Overall, this project not only achieves its objective of creating a functional Checkers game but also serves as an educational tool for understanding and applying key concepts in object-oriented programming and data management. The structured approach and clear separation of concerns make the codebase easy to understand, maintain, and extend. This project exemplifies the power of combining robust design with practical implementation, resulting in a well-rounded and enjoyable software application.