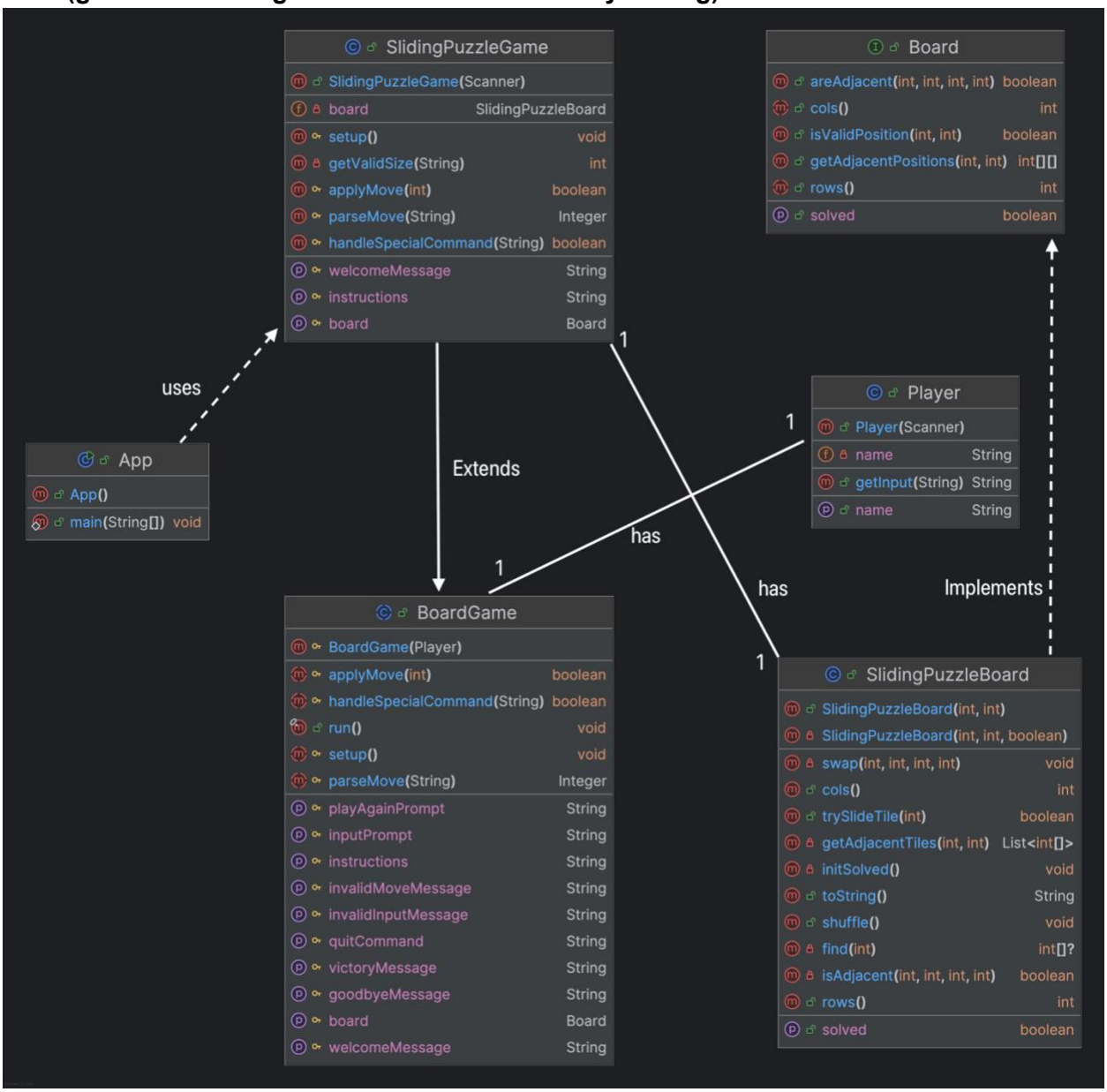
UML (generated through the use of IntelliJ and my editing)



How this allows for both scalability and extendibility:

1. Use of abstraction and inheritance

BoardGame is an abstract class which acts as a template by defining shared logic for any kind of board game like Tic Tac Toe. This includes things like setup, parsing moves, applying moves, handling any special commands (in this game it was the shuffle command), and replaying games. This implementation of the template can be seen through SlidingPuzzleGame which extends the BoardGame class. Also, because the game loop lives in BoardGame, you can add

more games later by simply extending BoardGame and overriding the abstract methods. This makes it an extensible design because you can apply this to many other games.

2. Using interfaces

Board is an interface that was used which defines what any board should provide, like rows, cols, isSolved, and adjacency helpers. SlidingPuzzleBoard implements Board, but nothing in BoardGame or SlidingPuzzleGame is tied to this specific board class. If you want a different kind of board, like a hexagonal grid, or a larger tile game, you can add a new class that implements Board without breaking the system. This makes the design scalable.

3. Seperation of concerns

Player handles player inputs and identity
BoardGame handles game logic
Board/SlidingPuzzleBoard hands the state of the puzzle
This separation of different concerns means I can modify one part without needing to rewrite the whole system, also achieving maintainability and scalability.

4. Encapsulation

Classes only expose whats necessary for example Board that exposes rows, cols, and status of if its solved or not, but internal details like swap or find in SlidingPuzzleBoard are hidden. Boardgame also interacts with Player only through getInput, so later on, a GUI can be implemented and easily be swapped from the console.