Kriegspiel Chess Documentation

The project is implemented in 2 languages: Java and C++. As the main concentration of this project is using design patterns, the language of implementation is not so important. That is why I first implemented in Java and then converted (by hand, without using any tools) to c++ by trying to use c++ shared pointers. As a result both implementations are available in the repository.

Architecture

I referred to the following source for the design of the main components of the game: https://www.geeksforgeeks.org/design-a-chess-game/ . However my implementation almost totally differs from that one, as there were no design patterns used in that implementation. The architecture design of my application is based on responsibility-driven design, which focuses on the objects as behavioral abstractions which are characterized by their responsibilities. This means that each object should implement the functionalities for which it is responsible.

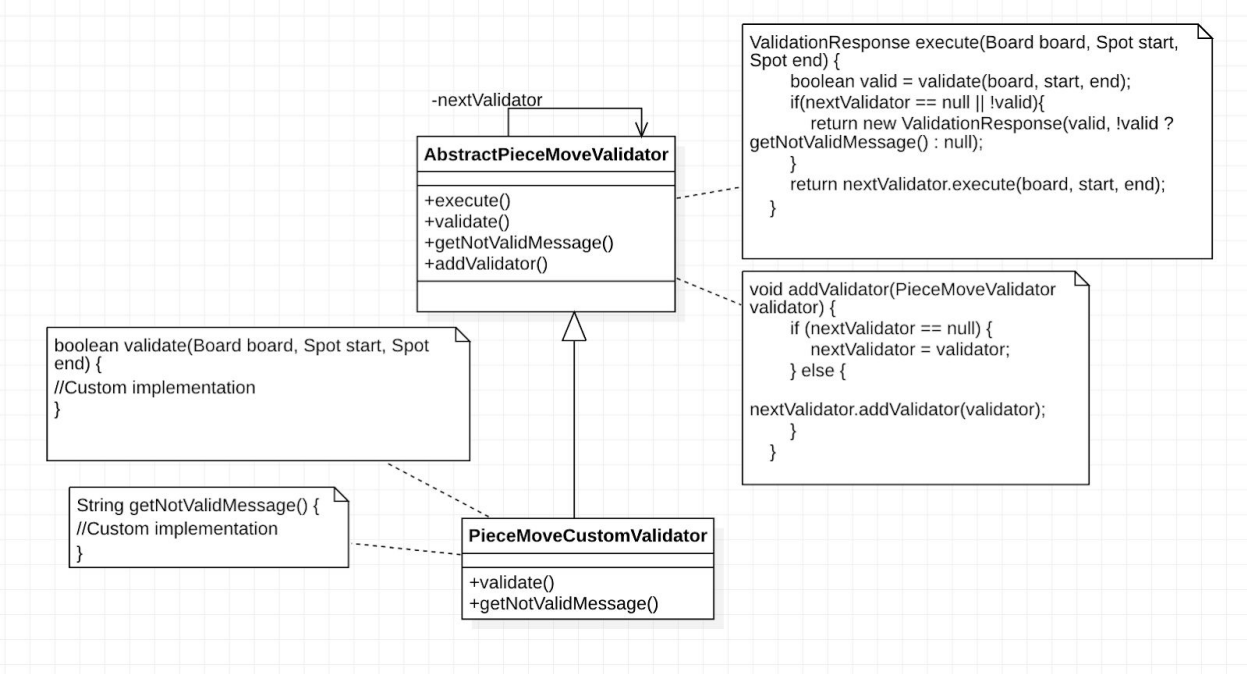
For example, the validation of the move should be asked from the Piece, because each piece should be aware of its own validation rules and should be responsible for validating its moves based on the rules. To that end Piece class encapsulates PieceMoveValidator (which is a chain of validations) and checks move's validity based on it.

In the design of the pieces the Template pattern was used, in which each piece extend from the "Piece" template class and overrides its custom functionality. The PieceMoveValidator object should be set on each piece during its initialization, after which when making a move, we just need to call the "canMove()" method (which is virtual) from the inherited "Piece" class and the validation will be done.

From the requirements of the task I understood that it mainly consists of two important functionality, which should be designed in a way that it will be easy to customize and add new functionalities. These are:

- Validating the move: we should be able to easily add and remove rules
- Emitting messages about state changes after the move, such as "piece gone", "pawn gone", "check on horizontal" and etc.

For that purpose the Chain of Responsibility design pattern was used to chain those rules. The following is the class diagram of my validator chain implementation:



As we see the validators are connected through a chain, and in order to add a new rule validator we just need to extend from the abstract validator, implement its two methods validate() and getNotValidMessage(), and add the validator to the chain. As we see the calls of these functions are made in the parent class, however as they are virtual, the implementations of their true object types will be called.

In this way we will have also order of validations, which is very applicable to our project, because if some rule does not hold, there is no sense of continuing the validation chain.

For the chain initialization I used the Factory design pattern. There is a factory class which is responsible for initializing the chains for each piece type.

Java code example for the ValidatorFactory:

```
public class ValidatorFactory {

    public PieceMoveValidator getPieceMoveValidator(String piece){

        PieceMoveValidator validatorChain = getBaseValidatorChain();
        if(piece == "bishop"){
            validatorChain.addValidator(new BishopMoveValidator());
        } else if (piece == "king") {
            validatorChain.addValidator(new KingMoveValidator());
        } else if (piece == "knight") {
            validatorChain.addValidator(new KnightMoveValidator());
        } else if (piece == "pawn") {
            validatorChain.addValidator(new PawnMoveValidator());
        } else if (piece == "queen") {
            validatorChain.addValidator(new QueenMoveValidator());
        } else if (piece == "rook") {
            validatorChain.addValidator(new RookMoveValidator());
        }

        return validatorChain;
    }

    public PieceMoveValidator getBaseValidatorChain(){
        PieceMoveValidator validatorChain = new EmptySlotValidator();
        validatorChain.addValidator(new EndSlotWithinBorderValidator());
        validatorChain.addValidator(new DestinationValidator());
        validatorChain.addValidator(new KingUnderAttackValidator());
        validatorChain.addValidator(new PathAvailableValidator());
        return validatorChain;
    }
}
```

As we see there is a base validator chain, to which custom validators are added.

The second functionality, the stateChangeChecker, is implemented in the same design of a chain and with its factory class for the initialization. The execution of stateChangeChecker chain is called in the Piece.move() method, which is virtual and can be overridden by the derived classes for custom implementation. However the implementations of those custom moves are not done in this project as its goal was to implement the validation logic.

I also used state like pattern for some pieces such as Pawn, King and Rook. This is not the well known State pattern, but it is similar to it, as during the moves for example a Pawn must change its state to enPassant in some cases or should set that it has been moved and cannot make two step move any more, or King should set that castling is done. And their custom validators should take into account those states.

For the High Cohesion and Loose Coupling I also separated some utils classes such as MessageHandler and PathCalculator. For instance Validators are not coupled with message texts, and any time when we need to change some message contents we will need to make changes just in on place - MessageHandler, and we will not need to go through all validators to make the update.

```
public class MessageHandler {

    private static final String NO = "No";
    private static final String HELL_NO = "Hell No";

    public static String getNoMessage() { return NO; }

    public static String getHellNoMessage() { return HELL_NO; }
}
```

References

https://codereview.stackexchange.com/questions/71790/design-a-chess-game-using-object-oriented-principles

https://massivetechinterview.blogspot.com/2015/07/design-chess-game-using-oo-principles.html

https://www.geeksforgeeks.org/design-a-chess-game/

https://www.geeksforgeeks.org/design-a-chess-game/