Project 3 Documentation

Scott Rizzo

11/04/18

GitHub: smrizzo

1 Design Patterns

1.1 Singleton Pattern

When creating the pieces I used the Singleton pattern to create those pieces. I have a abstract class I named Piece which doesn't have much currently but was designed in a way to prepare for the future. It currently only has a character variable for each individual piece. The Piece class has many subclasses that extend it and all those classes are Singletons. So I currently have a blackPawn, blackKnight, blackRook, blackBishop, blackQueen, blackKing, whitePawn, whiteKnight, whiteRook, whiteBishop, whiteQueen, and whiteKing classes that are all singletons. This was designed one to only have one object at any given time of each piece as well as make it easy to add more pieces if ever wanted to change the game of chess.

Separating the pieces in this way allows me to also prepare for features like validMove where a specific piece has a specific move it can make. This will easily be incorporated by adding perhaps a moveBehavior interface that my Piece class will create using composition. Then each singleton class I have created will implement the appropriate move methods that are in the moveBehavior interface.

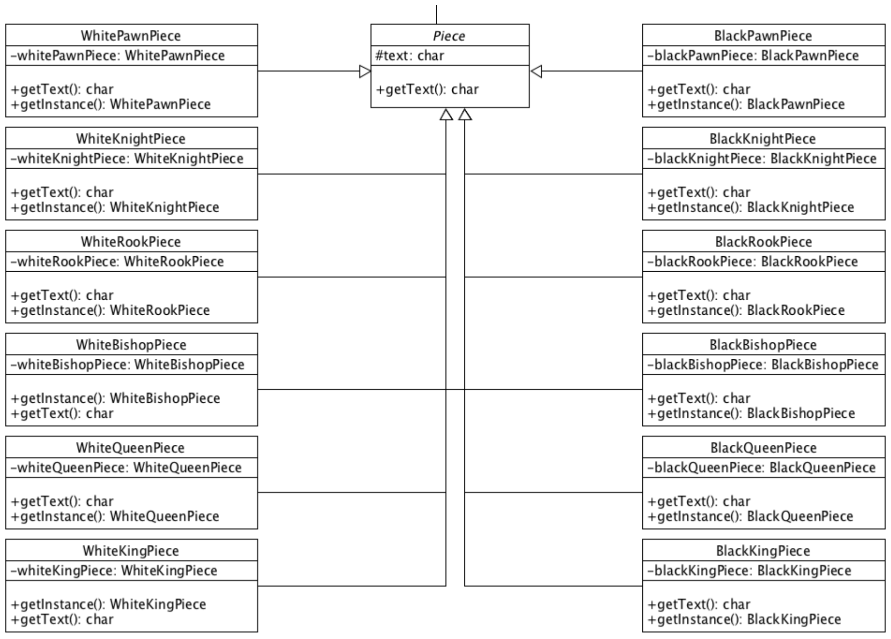
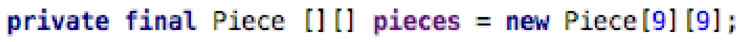

WhitePawnPiece
-whitePawnPiece: WhitePawnPiece
+getText(): char
+getInstance(): WhitePawnPiece
WhiteKnightPiece
-whiteKnightPiece: WhiteKnightPiece
+getText(): char
+getInstance(): WhiteKnightPiece
WhiteRookPiece
-whiteRookPiece: WhiteRookPiece
+getText(): char
+getInstance(): WhiteRookPiece
WhiteBishopPiece
-whiteBishopPiece: WhiteBishopPiece
+getInstance(): WhiteBishopPiece
+getText(): char
WhiteQueenPiece
-whiteQueenPiece: WhiteQueenPiece
+getText(): char
+getInstance(): WhiteQueenPiece
WhiteKingPiece
-whiteKingPiece: WhiteKingPiece
+getInstance(): WhiteKingPiece
+getText(): char
Piece
#text: char
+getText(): char
BlackPawnPiece
-blackPawnPiece: BlackPawnPiece
+getText(): char
+getInstance(): BlackPawnPiece
BlackKnightPiece
-blackKnightPiece: BlackKnightPiece
+getText(): char
+getInstance(): BlackKnightPiece
BlackRookPiece
-blackRookPiece: BlackRookPiece
+getText(): char
+getInstance(): BlackRookPiece
BlackBishopPiece
-blackBishopPiece: BlackBishopPiece
+getText(): char
+getInstance(): BlackBishopPiece
BlackQueenPiece
-blackQueenPiece: BlackQueenPiece
+getText(): char
+getInstance(): BlackQueenPiece
BlackKingPiece
-blackKingPiece: BlackKingPiece
+getText(): char
+getInstance(): BlackKingPiece

Figure 1: Singleton classes

1.2 Flyweight Pattern

When creating my 2d array of type Piece I had in mind the Flyweight pattern. It appears on the board visually I have many pieces when in fact I really only have one pawn piece objects but using the Flyweight pattern my 2d array is referencing the same object over and over again. There is not much to say for this pattern besides that the Flyweight and Singleton kind of work well together and it allows me to use less memory.



```
private final Piece [][] pieces = new Piece[9][9];
```

Figure 2: flyweight

1.3 Observer Pattern

The Observer pattern is one that I find really interesting. I designed it using the observer pattern so its very flexible and can add new Views very easily preparing for the future. I used this pattern so that I could decouple my View and Controller from my Model. I made each View and controller object an observer for the model which was the observable. The view never changed or updated the board when pertaining to pieces moving or getting promoted until the model call the update method inside using the observer pattern. A action happens in the view, the Controller decides that the action is something I can care about, then manipulates the model changing the board, then the model notifies all the observers that the board has changed. The views then re-render to display the new board.

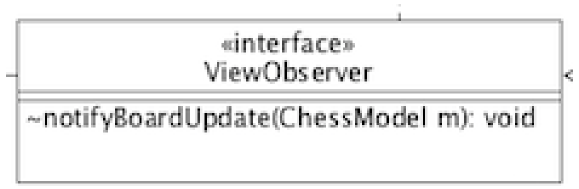


Figure 3: Observer Interface

1.4 Strategy Pattern

I used the Strategy pattern to really try and encapsulate each different task within its own class or objects. I refactored to better encapsulate specific task and through composition allowing other classes to gain access to other variables or methods that are not there own. I do feel I could have done better here but was struggling to do it. I currently have a single Controller for both local and network play and it might be better to separate those into two different controllers into a networkController and localController which are subclasses of a main Controller abstract class. This would allow easily being able to add new controllers if ever needed to for the future. But currently for the most part I improved p2 a lot and started to separate responsibilities using Strategy Pattern.

1.5 MVC Pattern

The way I designed my project with MVC is by creating a ChessModel class, and Chess Controller Class, and a View class. The view class is where all my animations are taking place for spritely and essentially everything that has to do with the GUI. This is where all the setClickHandlers reside and I when a click or key press happens it lets the Controller decide what to do with that action. If at some point there is an action that happens that is meaningful to the game of chess my Controller manipulates the ChessModel by invoking methods in the ChessModel class.

1.5.1 The Model

The model in my program is defined by the ChessModel class. The chess model class contains a 2d array that has references to Piece objects using the Flyweight and Singleton patter explained above. All the board logic as far as moving a piece from location to the next all happens in the model. It has methods like regularMove, promoteWhitePawn, promoteBlackPawn, promotingPawn etc.... It also the appropriate setter and getting methods so that the controller can set things up in the model when certain actions happen. This allows the model to not really know what's going in the View or the Controller it just provides methods to be invoked by whoever which in this case is the Controller.

1.5.2 The Controller

The controller in my program is defined by the ChessController class. The chess controller at this point has two responsibilities which I understand is not the best but considering the struggle to complete this assignment it will be something I have to fix later on. The controller as of right now is responsible for creating View objects and adding those to a List of threads. When the time is right we invoke the start() method on those threads which invoke the run() method in each View. The controller is really the middleware that was is responsible for manipulating the model and in some cases setting up booleans for the view to know when to highlight etc.

1.5.3 The View

My view is responsible for how the board will look on the screen. It does not communicate with the Model in anyway. The View has two subclasses which are WhiteView, and BlackView. The WhiteView creates the board for when a white player is on the bottom of the board, and the blackView creates a board for when its black player and the pieces need to be on the bottom. The view gets the updated board from the Model through the observer pattern and then re-renders the spritely window based on where the it finds certain piece objects in the 2d array given by the model. The model has no idea about how its being displayed it just says here you go View do what you will with this 2d array of pieces. The view is the only one that has access to the image files stored in the HashMap in a ChessPieces class which i will have to rename in the future.

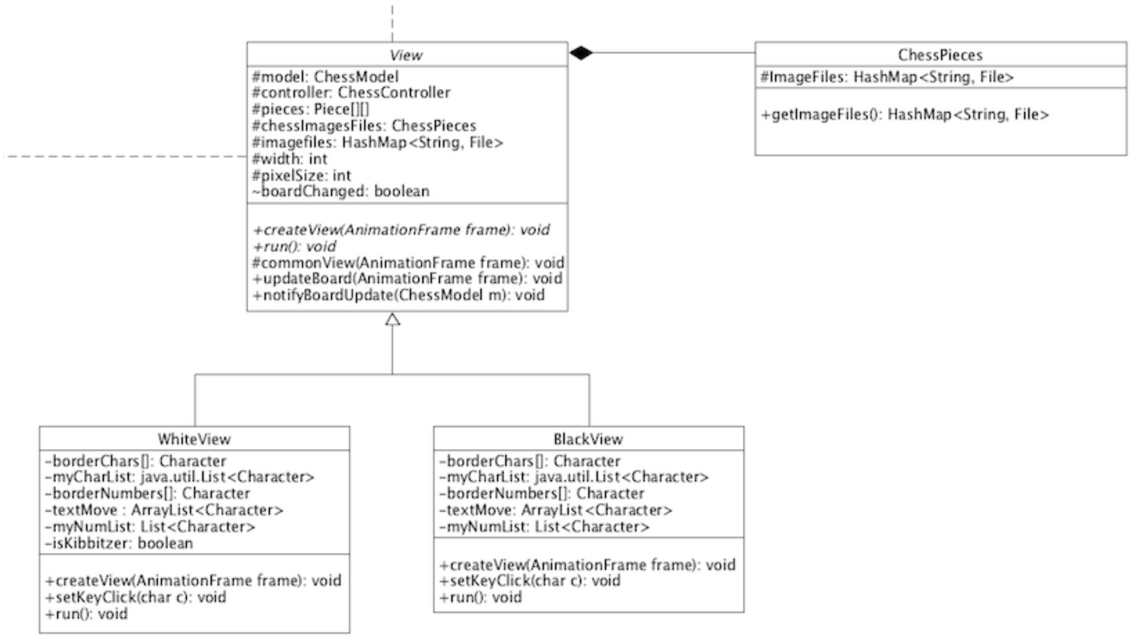

View
#model: ChessModel
#controller: ChessController
#pieces: Piece[][]
#chessImagesFiles: ChessPieces
#imagefiles: HashMap<String, File>
#width: int
#pixelSize: int
~boardChanged: boolean
+createView(AnimationFrame frame): void
+run(): void
#commonView(AnimationFrame frame): void
+updateBoard(AnimationFrame frame): void
+notifyBoardUpdate(ChessModel m): void
ChessPieces
#ImageFiles: HashMap<String, File>
+getImageFiles(): HashMap<String, File>
WhiteView
-borderChars[]: Character
-myCharList: java.util.List<Character>
-borderNumbers[]: Character
-textMove : ArrayList<Character>
-myNumList: List<Character>
-isKibbitzer: boolean
+createView(AnimationFrame frame): void
+setKeyClick(char c): void
+run(): void
BlackView
-borderChars[]: Character
-myCharList: java.util.List<Character>
-borderNumbers[]: Character
-textMove: ArrayList<Character>
-myNumList: List<Character>
+createView(AnimationFrame frame): void
+setKeyClick(char c): void
+run(): void

Figure 4: ViewClasses

1.6 Class Diagram

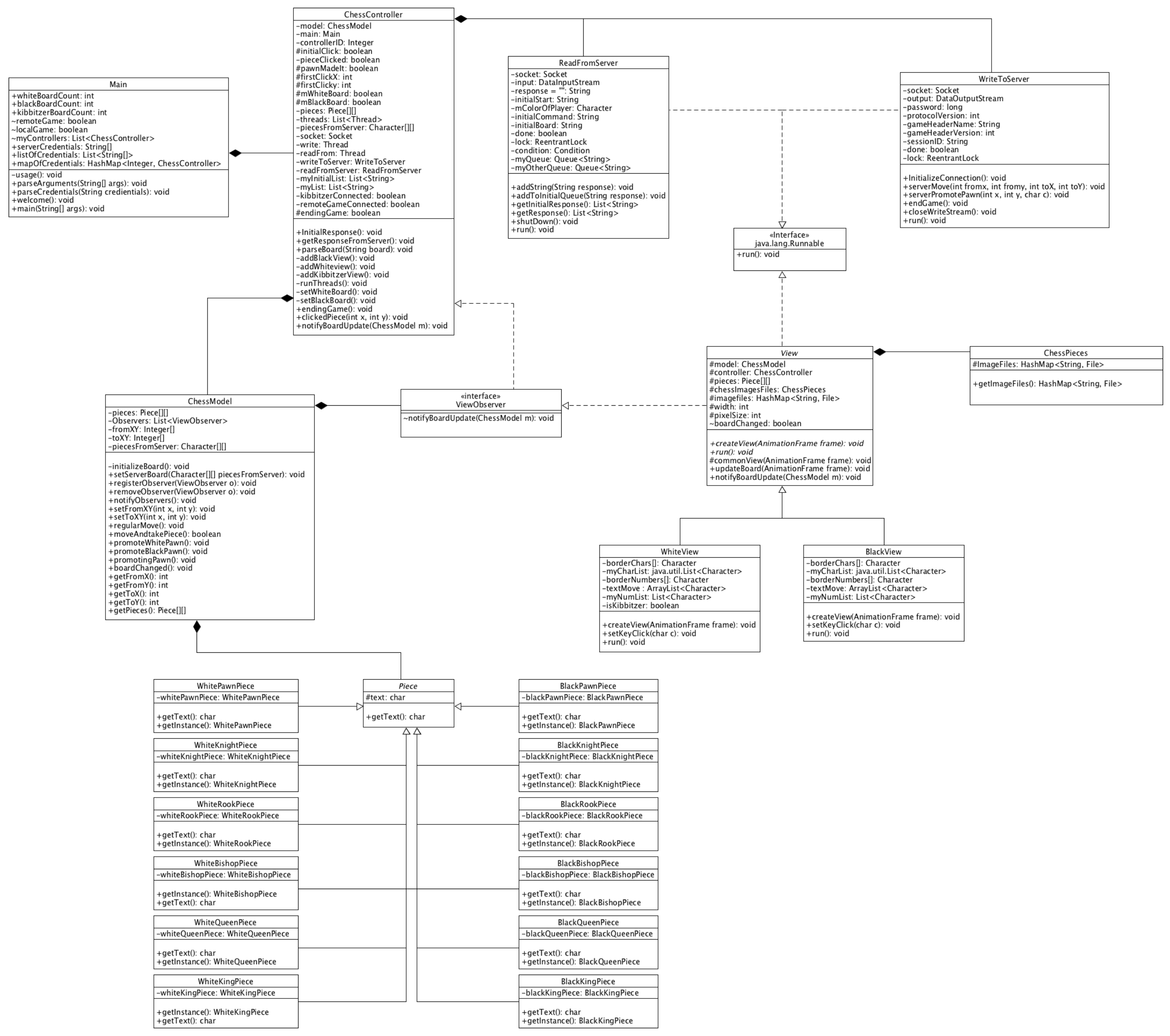

Main
+whiteBoardCount: int
+blackBoardCount: int
+kibbitzerBoardCount: int
~remoteGame: boolean
~localGame: boolean
~myControllers: List<ChessController>
+serverCredentials: String[]
+listOfCredentials: List<String[]>
+mapOfCredentials: HashMap<Integer, ChessController>
-usage(): void
+parseArguments(String[] args): void
+parseCredentials(String credientials): void
+welcome(): void
+main(String[] args): void
ChessController
-model: ChessModel
-main: Main
-controllerID: Integer
#initialClick: boolean
-pieceClicked: boolean
#pawnMadeIt: boolean
#firstClickX: int
#firstClicky: int
#mWhiteBoard: boolean
#mBlackBoard: boolean
-pieces: Piece[][]
-threads: List<Thread>
-piecesFromServer: Character[][]
-socket: Socket
-write: Thread
-readFrom: Thread
-writeToServer: WriteToServer
-readFromServer: ReadFromServer
-myInitialList: List<String>
-myList: List<String>
-kibbitzerConnected: boolean
-remoteGameConnected: boolean
#endingGame: boolean
+InitialResponse(): void
+getResponseFromServer(): void
+parseBoard(String board): void
-addBlackView(): void
-addWhiteview(): void
-addKibbitzerView(): void
-runThreads(): void
-setWhiteBoard(): void
-setBlackBoard(): void
+endingGame(): void
+clickedPiece(int x, int y): void
+notifyBoardUpdate(ChessModel m): void
ReadFromServer
-socket: Socket
-input: DataInputStream
-response = "": String
-initialStart: String
-mColorOfPlayer: Character
-initialCommand: String
-initialBoard: String
-done: boolean
-lock: ReentrantLock
-condition: Condition
-myQueue: Queue<String>
-myOtherQueue: Queue<String>
+addString(String response): void
+addToInitialQueue(String response): void
+getInitialResponse(): List<String>
+getResponse(): List<String>
+shutDown(): void
+run(): void
WriteToServer
-socket: Socket
-output: DataOutputStream
-password: long
-protocolVersion: int
-gameHeaderName: String
-gameHeaderVersion: int
-sessionID: String
-done: boolean
-lock: ReentrantLock
+InitializeConnection(): void
+serverMove(int fromx, int fromy, int toX, int toY): void
+serverPromotePawn(int x, int y, char c): void
+endGame(): void
+closeWriteStream(): void
+run(): void
«Interface»
java.lang.Runnable
+run(): void
View
#model: ChessModel
#controller: ChessController
#pieces: Piece[][]
#chessImagesFiles: ChessPieces
#imagefiles: HashMap<String, File>
#width: int
#pixelSize: int
~boardChanged: boolean
+createView(AnimationFrame frame): void
+run(): void
#commonView(AnimationFrame frame): void
+updateBoard(AnimationFrame frame): void
+notifyBoardUpdate(ChessModel m): void
ChessPieces
#ImageFiles: HashMap<String, File>
+getImageFiles(): HashMap<String, File>
ChessModel
-pieces: Piece[][]
-Observers: List<ViewObserver>
-fromXY: Integer[]
-toXY: Integer[]
-piecesFromServer: Character[][]
-initializeBoard(): void
+setServerBoard(Character[][] piecesFromServer): void
+registerObserver(ViewObserver o): void
+removeObserver(ViewObserver o): void
+notifyObservers(): void
+setFromXY(int x, int y): void
+setToXY(int x, int y): void
+regularMove(): void
+moveAndtakePiece(): boolean
+promoteWhitePawn(): void
+promoteBlackPawn(): void
+promotingPawn(): void
+boardChanged(): void
+getFromX(): int
+getFromY(): int
+getToX(): int
+getToY(): int
+getPieces(): Piece[][]
«interface»
ViewObserver
~notifyBoardUpdate(ChessModel m): void
WhiteView
-borderChars[]: Character
-myCharList: java.util.List<Character>
-borderNumbers[]: Character
-textMove : ArrayList<Character>
-myNumList: List<Character>
-isKibbitzer: boolean
+createView(AnimationFrame frame): void
+setKeyClick(char c): void
+run(): void
BlackView
-borderChars[]: Character
-myCharList: java.util.List<Character>
-borderNumbers[]: Character
-textMove: ArrayList<Character>
-myNumList: List<Character>
+createView(AnimationFrame frame): void
+setKeyClick(char c): void
+run(): void
Piece
#text: char
+getText(): char
WhitePawnPiece
-whitePawnPiece: WhitePawnPiece
+getText(): char
+getInstance(): WhitePawnPiece
BlackPawnPiece
-blackPawnPiece: BlackPawnPiece
+getText(): char
+getInstance(): BlackPawnPiece
WhiteKnightPiece
-whiteKnightPiece: WhiteKnightPiece
+getText(): char
+getInstance(): WhiteKnightPiece
BlackKnightPiece
-blackKnightPiece: BlackKnightPiece
+getText(): char
+getInstance(): BlackKnightPiece
WhiteRookPiece
-whiteRookPiece: WhiteRookPiece
+getText(): char
+getInstance(): WhiteRookPiece
BlackRookPiece
-blackRookPiece: BlackRookPiece
+getText(): char
+getInstance(): BlackRookPiece
WhiteBishopPiece
-whiteBishopPiece: WhiteBishopPiece
+getInstance(): WhiteBishopPiece
+getText(): char
BlackBishopPiece
-blackBishopPiece: BlackBishopPiece
+getText(): char
+getInstance(): BlackBishopPiece
WhiteQueenPiece
-whiteQueenPiece: WhiteQueenPiece
+getText(): char
+getInstance(): WhiteQueenPiece
BlackQueenPiece
-blackQueenPiece: BlackQueenPiece
+getText(): char
+getInstance(): BlackQueenPiece
WhiteKingPiece
-whiteKingPiece: WhiteKingPiece
+getInstance(): WhiteKingPiece
+getText(): char
BlackKingPiece
-blackKingPiece: BlackKingPiece
+getText(): char
+getInstance(): BlackKingPiece

Figure 5: Class Diagram