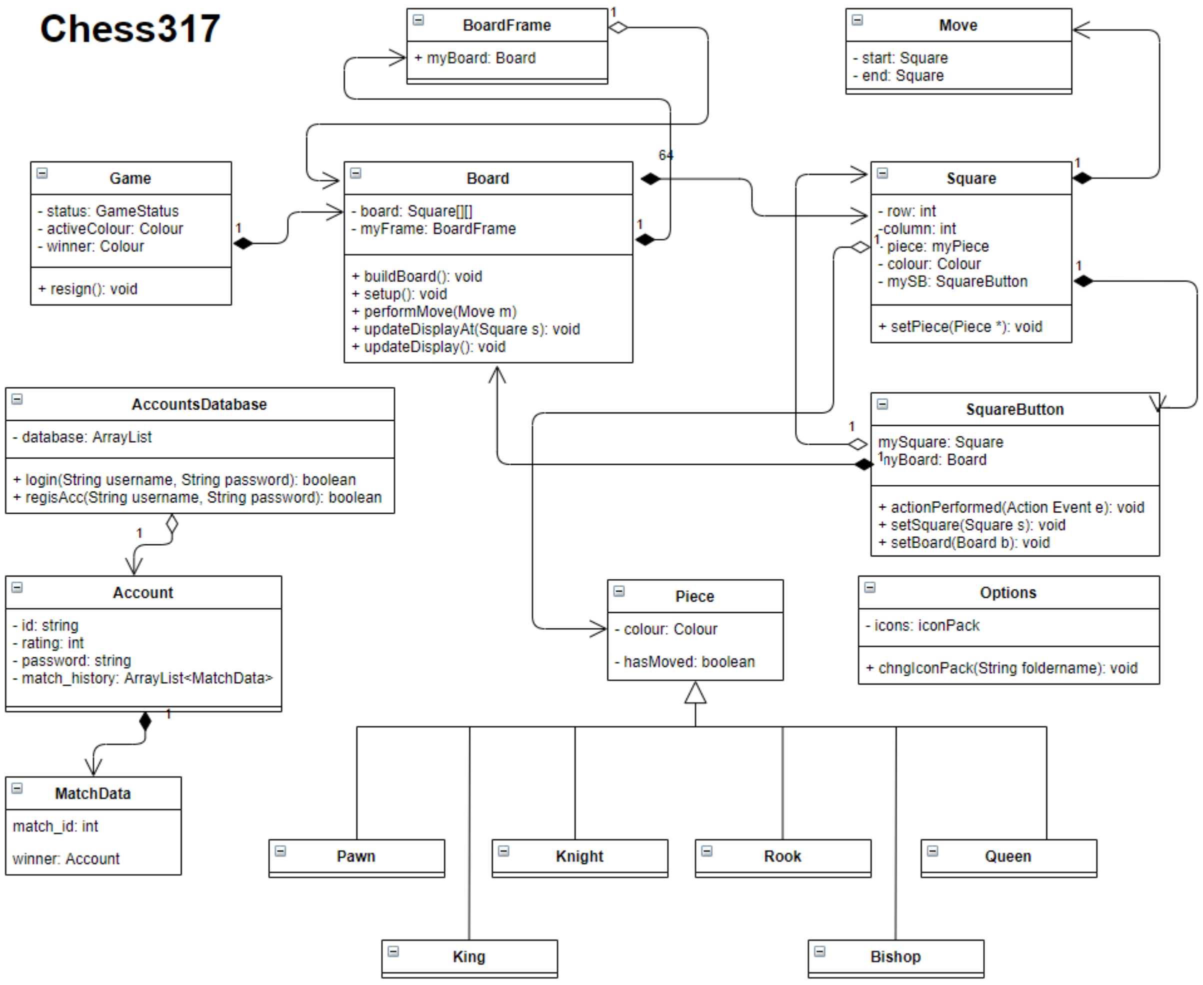
CP317 Assignment 3
Chess317

Project ID: CP317-TP19

Dennis Au - auxx1820
Andy Tang - tang8300

Table of Contents

1. Complete Class Diagram



2. Detailed Design (Pseudocode)

2.1 Account Class

```
boolean verifyLogin(String username, String password) {
    boolean verified;
    if (username in database and passwords match)
        verified = true;
    else verified = false;
    return verified;
}
```

CYC = 2

```
boolean regisAcc(String username, String password) {
    boolean success
    if username in database;
        success = false;
    else
        store new account info in database;
        success = true;
    return success;
}
```

CYC = 2

2.2 Game Class

```
void resign() {
    if current player = white
        gameStatus = blackWin
    else gameStatus = whiteWin
    return void
}
```

CYC = 2

2.3 Options Class

```
void chngIconPack(String folderName) {
    iconFolder = folderName
    return void
}
```

CYC = 1

2.4 Square Class

```java
public class Square {
    int row, col;
    Colour colour;
    Piece myPiece;
    SquareButton mySB;

    public Square(int row, int col, Colour colour, SquareButton sb) {
        this.row = row;
        this.col = col;
        this.colour = colour;
        myPiece = null;
        mySB = sb;
    }

    public void setPiece(Piece p) {
        myPiece = p;
    }
}
```

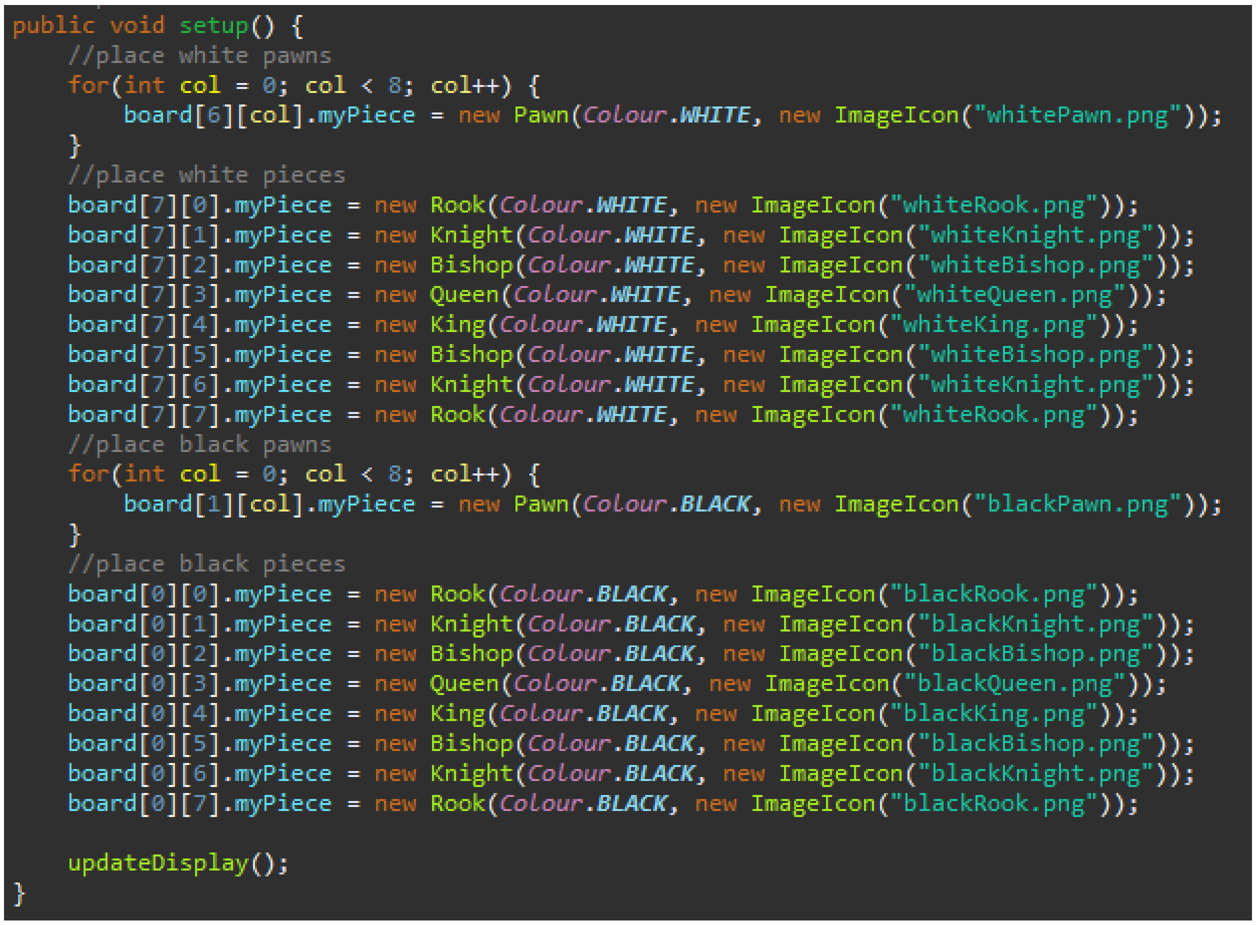
CYC = 1

2.5 Board Class

```java
public void buildBoard(BoardFrame frame) {
    for(int row = 0; row < 8; row++) {
        for(int col = 0; col < 8; col++) {
            int sum = row+col;
            SquareButton sbutton = new SquareButton();
            if (sum%2==0) {
                frame.add(sbutton);
                sbutton.setBackground(Color.LIGHT_GRAY);
                //if the sum of row and col is even, then it's a white square
                board[row][col] = new Square(row, col, Colour.WHITE, sbutton);

            } else {
                frame.add(sbutton);
                sbutton.setBackground(Color.DARK_GRAY);
                //if the sum of row and col is odd, then it's a black square
                board[row][col] = new Square(row, col, Colour.BLACK, sbutton);
            }
            sbutton.setSquare(board[row][col]);
            sbutton.setBoard(this);
        }
    }
    frame.setTitle("Chess317");
    frame.setSize(600,600);
    frame.setVisible(true);
}
```

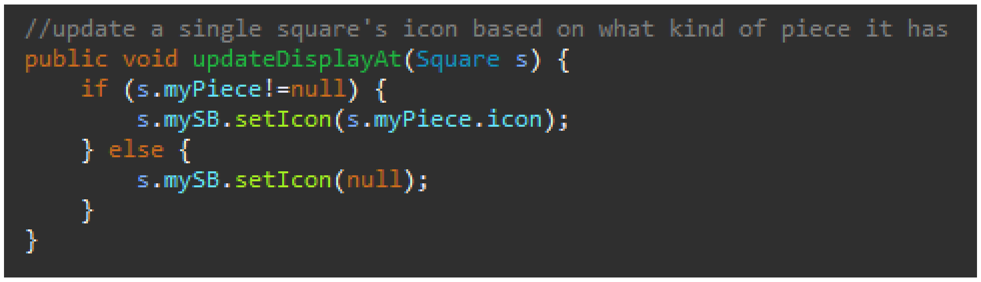
CYC = 2



```java
public void setup() {
    //place white pawns
    for(int col = 0; col < 8; col++) {
        board[6][col].myPiece = new Pawn(Colour.WHITE, new ImageIcon("whitePawn.png"));
    }
    //place white pieces
    board[7][0].myPiece = new Rook(Colour.WHITE, new ImageIcon("whiteRook.png"));
    board[7][1].myPiece = new Knight(Colour.WHITE, new ImageIcon("whiteKnight.png"));
    board[7][2].myPiece = new Bishop(Colour.WHITE, new ImageIcon("whiteBishop.png"));
    board[7][3].myPiece = new Queen(Colour.WHITE, new ImageIcon("whiteQueen.png"));
    board[7][4].myPiece = new King(Colour.WHITE, new ImageIcon("whiteKing.png"));
    board[7][5].myPiece = new Bishop(Colour.WHITE, new ImageIcon("whiteBishop.png"));
    board[7][6].myPiece = new Knight(Colour.WHITE, new ImageIcon("whiteKnight.png"));
    board[7][7].myPiece = new Rook(Colour.WHITE, new ImageIcon("whiteRook.png"));
    //place black pawns
    for(int col = 0; col < 8; col++) {
        board[1][col].myPiece = new Pawn(Colour.BLACK, new ImageIcon("blackPawn.png"));
    }
    //place black pieces
    board[0][0].myPiece = new Rook(Colour.BLACK, new ImageIcon("blackRook.png"));
    board[0][1].myPiece = new Knight(Colour.BLACK, new ImageIcon("blackKnight.png"));
    board[0][2].myPiece = new Bishop(Colour.BLACK, new ImageIcon("blackBishop.png"));
    board[0][3].myPiece = new Queen(Colour.BLACK, new ImageIcon("blackQueen.png"));
    board[0][4].myPiece = new King(Colour.BLACK, new ImageIcon("blackKing.png"));
    board[0][5].myPiece = new Bishop(Colour.BLACK, new ImageIcon("blackBishop.png"));
    board[0][6].myPiece = new Knight(Colour.BLACK, new ImageIcon("blackKnight.png"));
    board[0][7].myPiece = new Rook(Colour.BLACK, new ImageIcon("blackRook.png"));

    updateDisplay();
}
```

CYC = 1



```java
//update a single square's icon based on what kind of piece it has
public void updateDisplayAt(Square s) {
    if (s.myPiece!=null) {
        s.mySB.setIcon(s.myPiece.icon);
    } else {
        s.mySB.setIcon(null);
    }
}
```

CYC = 2

```
//update every square's icon based on what kind of piece it has
public void updateDisplay() {
    for(int row = 0; row < 8; row++) {
        for(int col = 0; col < 8; col++) {
            if (board[row][col].myPiece!=null) {
                board[row][col].myPiece.icon.getImage().flush();
                board[row][col].my_sb.setIcon(board[row][col].myPiece.icon);
            } else {
                board[row][col].my_sb.setIcon(null);
            }
        }
    }
}
```

CYC = 2

```
//moves a piece from starting square to endings square
public void performMove(Move m) {
    //set the destination square to contain the moving piece
    board[m.end.x][m.end.y].myPiece = board[m.start.x][m.start.y].myPiece;
    //set the starting square to contain nothing
    board[m.start.x][m.start.y].myPiece = null;
    updateDisplay();
}
```

CYC = 1

2.6 Piece Class

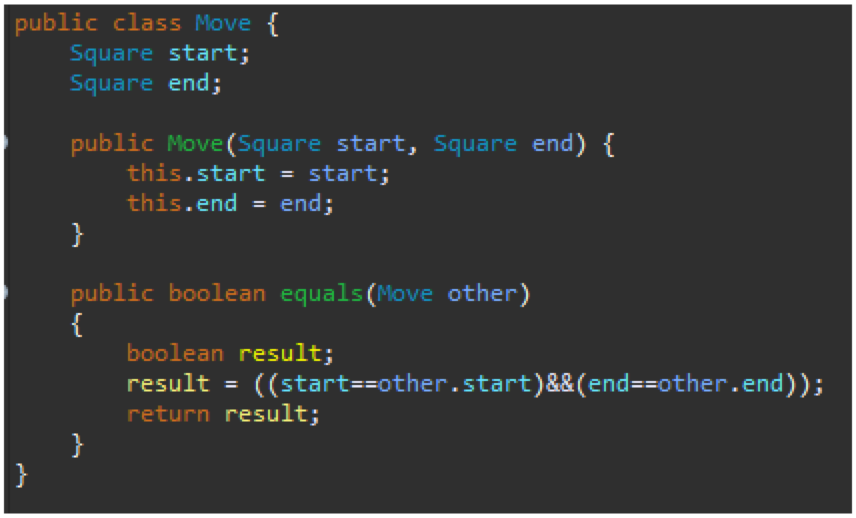
```
public abstract class Piece {

    public Piece(Colour colour, ImageIcon icon) {
        this.colour = colour;
        this.icon = icon;
    }

    Colour colour;
    boolean hasMoved = false;
    ImageIcon icon;
    public abstract char mySymbol(); //returns the symbol for each piece
}
```

CYC = 0 (no functions)

2.7 Move Class



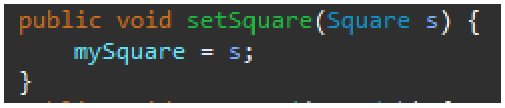
```java
public class Move {
    Square start;
    Square end;

    public Move(Square start, Square end) {
        this.start = start;
        this.end = end;
    }

    public boolean equals(Move other)
    {
        boolean result;
        result = ((start==other.start)&&(end==other.end));
        return result;
    }
}
```

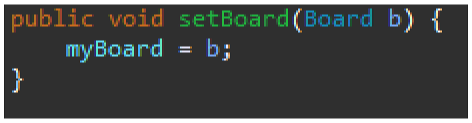
CYC = 2

2.8 SquareButton Class



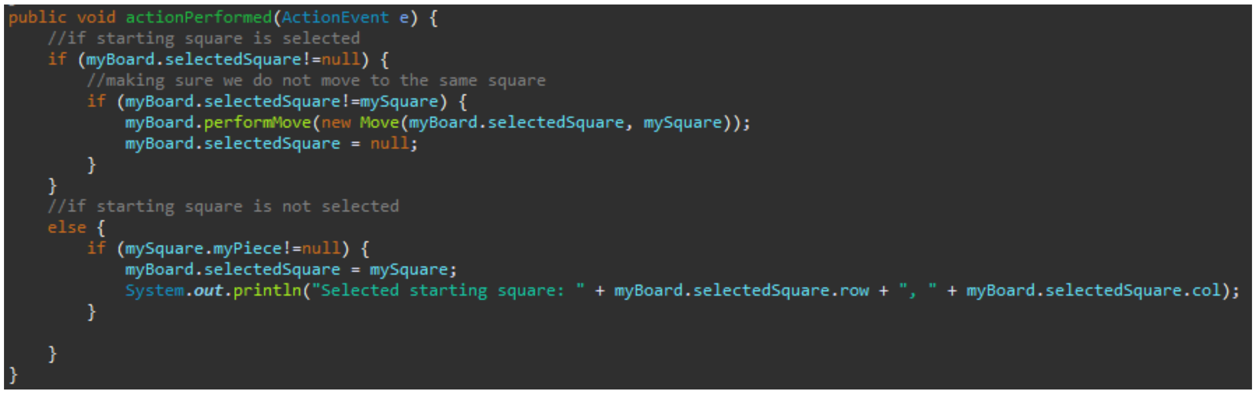
```java
public void setSquare(Square s) {
    mySquare = s;
}
```

CYC = 1



```java
public void setBoard(Board b) {
    myBoard = b;
}
```

CYC = 1



```java
public void actionPerformed(ActionEvent e) {
    //if starting square is selected
    if (myBoard.selectedSquare!=null) {
        //making sure we do not move to the same square
        if (myBoard.selectedSquare!=mySquare) {
            myBoard.performMove(new Move(myBoard.selectedSquare, mySquare));
            myBoard.selectedSquare = null;
        }
    }
    //if starting square is not selected
    else {
        if (mySquare.myPiece!=null) {
            myBoard.selectedSquare = mySquare;
            System.out.println("Selected starting square: " + myBoard.selectedSquare.row + ", " + myBoard.selectedSquare.col);
        }

    }
}
```

CYC = 4

3. Design Metrics

3.1 Design Quality

3.1.1 Coupling

Chess317 aims to have a low-coupling code structure by minimizing interdependencies across classes. Most of the classes in Chess317 do not rely on others to function properly, and information is passed through via Data Coupling. For example, the performMove function in the Board Class takes in a Move object as a parameter (Defined in the Move Class) but does not otherwise deal with the Move Class itself. The most notable instance of coupling in our program is between the Board Class and the SquareButton Class. The chessboard needs to be able to identify which SquareButton (and subsequently, which Square) was clicked on, and so the two objects have to be somewhat coupled together. Otherwise, Chess317 still maintains a relatively low level of coupling, allowing for good readability and maintainability.

3.1.2 Cohesion

Chess317 is built around a highly cohesive model, with each class having responsibilities that are highly specific to its role. For example, we have made piece movement into its own class, and (Move) for general move validation. In the Move class, all the contained attributes and functions are strictly related to changing positions of pieces. Having high cohesion ensures that our code is more readable, maintainable, and reusable. It also enables a smoother debugging process, since problems can be more easily traced.

3.2 Design Size

There are 17 Classes, and 13 Functions so far.

Class	*# of Functions*
Game	1
Board	5
BoardFrame	0
Square	1
SquareButton	3
Move	0
Piece	0
Pawn	0
Rook	0
Knight	0
Bishop	0
King	0
Queen	0
Account	0
AccountsDatabase	2
MatchData	0
Options	1