CIS 163
Project 3 – Chess Game
Group project

Due Date

- At the beginning of the lab; see the schedule, last page of the syllabus.

Before Starting the Project

- Review Inheritance, Polymorphism, and Interfaces (Chapters 8 and 9 in the textbook)
- Read this entire project description before starting

Learning Objectives

After completing this project you should be able to:

- design, implement, and test a small class hierarchy
- use two-dimensional arrays and enum types, and
- implement a GUI-based game
- Working with a group on a project.

Project Description

Your assignment is to implement a simple GUI program that allows two humans to play chess game. Your design must organize the different pieces into a class hierarchy that utilizes polymorphism.

For information on objective of the chess game, board setup, and how the chess pieces move, see http://www.thechessstore.com/category/rulesofchess/. If you want to learn to play chess, see http://www.gamesgames.com/game/easy-chess.html.

Steps 1 – 7 should be completed as a group (the ordering is only a suggestion). Steps 1 – 7 are worth 60 points.

Steps 8 – 10 must be completed in sequence. Do not start on Step 8 until steps 1 – 7 are completed.

Step 1: Using your favorite IDE, create a project called "ChessPrj"

- Create a package named "`chess`"
- Include the following classes/interfaces in the `chess` package. These classes/interfaces are supplied to you. You **must** use these as provided, i.e., you are not allowed to make any changes to these classes/interfaces.
  - `IChessPiece`
  - `IChessModel`
  - `Player`
  - `Move`

Step 2: Implement the `ChessPiece` class

- The `ChessPiece` class implements the `IChessPiece` interface
- A player (black or white) owns a chess piece.
- `type()` method is `abstract`.
- `isValidMove()` method should
  - Verify that the starting and ending locations are different.
  - Verify that `this` piece is located at [move.fromRow, move.fromColumn] on the board.
  - Verify that [move.toRow, move.toColumn] does not contain a piece belonging to the same player.

```
public abstract class ChessPiece implements IChessPiece {

   private Player owner;

   protected ChessPiece(Player player) {
       this.owner = player;
   }

   public abstract String type();

   public Player player() {
      // complete this
   }

   public boolean isValidMove(Move move, IChessPiece[][] board) {
     // complete this
   }
}
```

Step 3: Implement the `Pawn` and Rook classes

- `Pawn` and `Rook` classes extend the `ChessPiece` class
- Implement `type()` method
- Implement `isValidMove()` method. Make sure to utilize the `isValidMove()` method from the base `ChessPiece` class and add functionality specific to the piece.

Step 4: Implement the `King`, `Queen`, `Knight`, and `Bishop` classes

- `King, Queen, Knight` and `Bishop` classes also extend the `ChessPiece` class.
- Implement `type()` method.
- For now, make `isValidMove()` method return `false`. Full implementation of this method is not part of the base functionality (see Step 8).

Step 5: Implement the `ChessModel` class

- The `ChessModel` class implements the `IChessModel` interface.
- This class is responsible for storing the chessboard and implementing the game logic.
- Implement the methods from the `IChessModel` interface.
- For now, make `inCheck()` method return `false`. Full implementation of this method is part of the additional functionality (see Step 9).
- For now, make `isComplete()` method return `false`. Full implementation of this method is part of the extra/bonus functionality (see Step 10).

```
public class ChessModel implements IChessModel {
   private IChessPiece[][] board;
   private Player player;
   // declare other instance variables as needed

   public ChessModel() {
       // complete this
   }

   public boolean isComplete() {
      return false;
   }

   public boolean isValidMove(Move moce) {
      // complete this
   }

   public void move(Move move) {
     // complete this
   }

   public boolean inCheck(Player p) {
     return false;
   }

   public Player currentPlayer() {
     // complete this
   }

   public int numRows() {
     // complete this
   }

   public int numColumns() {
     // complete this
   }

   public IChessPiece pieceAt(int row, int column) {
     // complete this
   }
```

```
    // add other public or helper methods as needed
}
```

Step 6: Implement the `ChessPanel` class

- The `ChessPanel` class extends the `JPanel` class.
- This class is responsible for presenting the graphical user interface, responding to user actions, and updating the view.
- The game should implement a standard form of chess; white moves then black moves.
- Only allow valid moves.

```
public class ChessPanel extends JPanel {

   private JButton[][] board;
   private ChessModel model;

   // declare other instance variables as needed

   private ButtonListener buttonListener = new ButtonListener();

   public ChessPanel() {
     // complete this
   }

   // method that updates the board
   private void displayBoard() {
     // complete this
   }

   // add other helper methods as needed

   // inner class that represents action listener for buttons
   private class ButtonListener implements ActionListener {

      public void actionPerformed(ActionEvent event) {
         // complete this
      }
   }
}
```

Step 7: Implement the `ChessGUI` class

- The `ChessGUI` class contains the `main` method that creates and displays the chess game GUI.

```
public class ChessGUI {

    public sttic void main(String[] args) {
        JFrame frame = new JFRame("Chess Game");
        frame.setDefaultCloseOperation(JFrame.EXIT_OC_CLOSE);
```

```
        ChessPanel panel = new ChessPanel();
        frame.getContentPane().add(panel);

        frame.pack();
        frame.setVisible(true);
    }
}
```

Step 8: Complete the `King`, `Queen`, `Knight`, and `Bishop` classes

- Fully implement `isValidMove()` method of `King` class.
- Fully implement `isValidMove()` method of `Queen` class.
- Fully implement `isValidMove()` method of `Knight` class.
- Fully implement `isValidMove()` method of `Bishop` class.
- Should only be able to move if it is a valid move.

Step 9: Implement the `inCheck()` method of `ChessModel` class

- Fully implement the `inCheck()` method of `ChessModel` class.
- Your program must display a message when the current player is in check using `JOptionPane.showMessageDialog()`.
- Solid error checking.

Step 10: Full Functionality

- Fully implement the `isComplete()` method of `ChessModel` class. For example: Check to see if the King is checkmated or can move out the way (i.e., uncheck itself) or another player can block the check. See the instructor for more details.
- Your program must display a message when the game is complete using `JOptionPane.showMessageDialog()`
- FULL Error checking! YES the means some JUnit testing. Please see the instructor for this step.

Step 11: Functionality

- ADD on one cool feature… like, undo, redo, use your imagination!

Javadoc Commenting and Coding Style/Technique [10 points]

- Use http://www.cis.gvsu.edu/studentsupport/javaguide as a guide to document the source code in your project and observe good coding style practices.

What/How to Turn in?

- Sign up for project demo (sign-up sheet will be available to you later).
- Print out a copy of source code and have it ready for demonstration.
- Staple rubric below to front of print out.

CIS 163 – Computer Science II
Project 3: Chess Game

Student Name	
Due Date	

Graded Item	Pts	Points Awarded
Javadoc comments and coding style/technique (http://www.cis.gvsu.edu/studentsupport/javaguide) • Code Indentation (auto format source code in IDE) • Naming Conventions (see Java style guide) • Proper access modifiers for fields and methods • Use of helper (private) methods	10	
Steps 1 – 7: Base Functionality ▪ Model/View separation ▪ Functioning GUI ▪ Initial chess board is set up correctly ▪ `Pawn` and `Rook` pieces move correctly ▪ `King`, `Queen`, `Knight`, and `Bishop` pieces should NOT move/respond to user actions	50	
Step 8: Additional functionality ▪ King piece moves correctly ▪ Queen piece moves correctly ▪ Knight piece moves correctly ▪ Bishop piece moves correctly	10	
Step 9: Additional functionality ▪ inCheck() of ChessModel class	10	
Step 10: Full Functionality ▪ isComplete() of ChessModel class ▪ Some JUnit testing	15	
Step 11: New Feature	5	
Total	**100**	

Comments: