Jami Biddle
HW5 Testing Document

Part 1: The individual chess pieces.

ChessPiece was tested and successful in the last project, so I'm not including those.

Individual Piece Tests:

1. **Xiangqi King Piece**
    a. isLegalNonCapture move should be tested for:
        i. a non legal move (any move more than one space away)
        ii. A non legal move (outside of the 3x3 grid)
        iii. A capture move (returns false)
        iv. The following legal moves
            1. Upwards
            2. Downwards
            3. To the left
            4. To the right
    b. Is LegalCaptureMove
        i. a non legal move (i.e. a knight move)
        ii. A non legal move (outside of the 3x3 grid)
        iii. A capture move (returns true)
            1. One Upwards
            2. One Downwards
            3. One To the left
            4. One To the right
        iv. A legal move to an empty space, so it returns false
        v. A 'legal' move that had the same side piece, so it should be false.
2. **Guard Piece**
    a. isLegalNonCapture move should be tested for:
        i. a non legal move (any move more than one space away)
        ii. A non legal move (outside of the 3x3 grid)
        iii. A capture move (returns false)
        iv. The following legal moves
            1. One to the top right
            2. One to the bottom right
            3. One to the bottom left
            4. One to the top left
    b. Is LegalCaptureMove
        i. a non legal move (i.e. a knight move)
        ii. A non legal move (outside of the 3x3 grid)
        iii. A capture move (returns true)
            1. One to the top right
            2. One to the bottom right

            3. One to the bottom left
            4. One to the top left
        iv. A legal move to an empty space, so it returns false
        v. A ‘legal’ move that had the same side piece, so it should be false.

3. **Elephant Piece**
    a. isLegalNonCapture move should be tested for:
        i. a non legal move (any move more than two spaces away)
        ii. A non legal move (across the ‘river’)
        iii. A capture move (returns false)
        iv. The following legal moves
            1. Two to the top right
            2. Two to the bottom right
            3. Two to the bottom left
            4. Two to the top left
        v. The following moves, but there is a piece between so returns false
            1. Two to the top right
            2. Two to the bottom right
            3. Two to the bottom left
            4. Two to the top left
    b. Is LegalCaptureMove
        i. a non legal move (i.e. a knight move)
        ii. A non legal move (outside of the 3x3 grid)
        iii. A capture move (returns true)
            1. Two to the top right
            2. Two to the bottom right
            3. Two to the bottom left
            4. Two to the top left
        iv. The following moves, but there is a piece between so returns false
            1. Two to the top right
            2. Two to the bottom right
            3. Two to the bottom left
            4. Two to the top left
        v. A legal move to an empty space, so it returns false
        vi. A ‘legal’ move that had the same side piece, so it should be false.

4. **Horse Piece**
    a. isLegalNonCapture move should be tested for:
        i. a non legal move (any non horse move)
        ii. A capture move (returns false)
        iii. A legal move to an empty space
            1. One for all 8 possible spots;
        iv. A move to those 8 spots, but there is a piece in the way.
    b. isLegalCapture move should be tested for:
        i. a non legal move (any non horse move)
        ii. A non capture move (returns false)

iii. A legal move to a filled space
    1. One for all 8 possible spots;
iv. A move each to those 8 spots, but there is a piece in the way.
v. A legal move to an empty space, so it returns false
vi. A ‘legal’ move that had the same side piece, so it should be false.

5. RookPiece:

a. isLegalNonCapture move should be tested for:
    i. a non legal move (against the rules of the piece)
    ii. A capture move (returns false)
    iii. The following, for the loop that checks to make sure there are no spaces in between:
        1. (none) A legal move to an empty space
        2. (one) A ‘legal’ move, but it has a piece in between
        3. (many) A ‘legal’ move, but is has a few pieces in between
        4. (first) A ‘legal’ move, but there is a piece in the way at the first space in between
        5. (Middle) A ‘legal’ move, but there is a piece in the way at the first space in between
        6. (Last) A ‘legal’ move, but there is a piece between at the space right before the intended space.
    iv. The following legal moves
        1. Upwards
        2. Downwards
        3. To the left
        4. To the right

b. Is LegalCaptureMove
    i. a non legal move (against the rules of the piece)
    ii. A legal capture move (returns true)
        1. Upwards
        2. Downwards
        3. To the left
        4. To the right
    iii. A legal move to an empty space, so it returns false
    iv. A ‘legal’ move that had the same side piece, so it should be false.
    v. The following, for the loop that checks to make sure there are no spaces in between:
        1. (none) A legal move to an empty space
        2. (one) A ‘legal’ move, but it has a piece in between
        3. (many) A ‘legal’ move, but is has a few pieces in between
        4. (first) A ‘legal’ move, but there is a piece in the way at the first space in between
        5. (Middle) A ‘legal’ move, but there is a piece in the way at the first space in between

6. (Last) A ‘legal’ move, but there is a piece between at the space right before the intended space.

6. Cannon Piece

a. isLegalNonCapture move should be tested for:
   i. a non legal move (against the rules of the piece)
   ii. A capture move (returns false)
   iii. The following, for the loop that checks to make sure there are no spaces in between:
      1. (none) A legal move to an empty space
      2. (one) A ‘legal’ move, but it has a piece in between
      3. (many) A ‘legal’ move, but is has a few pieces in between
      4. (first) A ‘legal’ move, but there is a piece in the way at the first space in between
      5. (Middle) A ‘legal’ move, but there is a piece in the way at the first space in between
      6. (Last) A ‘legal’ move, but there is a piece between at the space right before the intended space.
   iv. The following legal moves
      1. Upwards
      2. Downwards
      3. To the left
      4. To the right

b. Is LegalCaptureMove
   i. a non legal move (against the rules of the piece)
   ii. A legal capture move, w/ one piece in between (returns true)
      1. Upwards
      2. Downwards
      3. To the left
      4. To the right
   iii. A legal move to an empty space, so it returns false
   iv. A ‘legal’ move that had the same side piece, so it should be false.
   v. The following, for the loop that checks to make sure there is only one piece in between:
      1. (none) A legal move to an empty space
      2. (none) A ‘legal’ move, with no pieces between
      3. (many) A ‘legal’ move, but is has a few pieces in between
      4. (first) A legal move, with a piece in the way at the first space in between
      5. (Middle) A legal move, with a piece in the way at the first space in between
      6. (Last) A legal move with a piece between at the space right before the intended space.

7. SoldierPiece

a. isLegalNonCapture move should be tested for:

1. a non legal move (any move more than one space away (except castling, see the last section of KingPiece))
2. A capture move (returns false)
3. The following
    a. If North:
        i. move down by one
        ii. move left (false) (on the same side of the river)
        iii. move right (false) (on the same side of the river)
        iv. move left (true) (on the different side of the river)
        v. move right (true) (on the different side of the river)
    b. If South:
        i. move up by one
        ii. move left (false) (on the same side of the river)
        iii. move right (false) (on the same side of the river)
        iv. move left (true) (on the different side of the river)
        v. move right (true) (on the different side of the river)
    c. If East:
        i. move left by one
        ii. move up (false) (on the same side of the river)
        iii. move down (false) (on the same side of the river)
        iv. move up (true) (on the different side of the river)
        v. move down (true) (on the different side of the river)
    d. If west:
        i. move right by one
        ii. move up (false) (on the same side of the river)
        iii. move down (false) (on the same side of the river)
        iv. move up (true) (on the different side of the river)
        v. move down (true) (on the different side of the river))

b. Is LegalCaptureMove
    1. a non legal move (i.e. a knight move)
    2. A capture move (returns true)
    3. A legal move to an empty space, so it returns false
    4. A ‘legal’ move that had the same side piece, so it should be false.

Part 2: European Chess

1. **EuropeanChess**
    a. GetTurn()
        i. Set using SetTurn(), then get
    b. SetTurn()
        i. See the getter
    c. Main method
        i. If it is correct, the pieces i input should all pop up on the board, working.
            1. It does (It’s a miracle or something)

Part 3: Xiangqi

1. **Xiangqi**
    a. GetTurn()
        i. Set using SetTurn(), then get
    b. SetTurn()
        i. See the getter
    c. Main method
        i. If it is correct, the pieces i input should all pop up on the board, working.
            1. It does.

Part 4: JavaFX GUI

1. **JavaFXChessBoard**
    a. Main Method
        i. When called, it should launch the application and run the start method, so as long as it displays the board, the main method works.
            1. It does,
    b. start()
        i. Tested by launching the program. If the board displays and creates buttons that are able to be clicked, and the proper displays are shown, and the pieces are all present, the start method likely works
            1. It does and it works for the two corresponding boards for both xiangqi and chess
    c. addPiece
        i. This method is called by the start, so as long as the board displays with the necessary pieces on the board and they look correct, it's fine.
            1. It worked.
    d. removePiece
        i. When I moved a piece, the space it had been became empty. This is because the removePiece method worked correctly.
    e. ChessAction
        i. When I click a piece for the first time, it should highlight the piece. WHen I click a square it cannot move to, it should un-highlight the piece. When I again select the piece and click itself, it shouldn't do anything. WHen I click another square it can move to, it should move to that square.
            1. All of these did in fact work.

Because the chess board exists in the application, I wasn't able to figure out how to access the board beyond doing System.out.println within the methods called in the board, so I did a series of System.out.println's at the end of the start method to confirm that these methods didn't come back null.

```
> java Xiangqi javafx
Xiangqi@738b4531
true
false
RookPiece@26c7528d
false
true
10
9
```

f. getGameRules()
    i. Printed out correctly. There is never a possibility that there isn't a game made with the chessboard.
        1. Xiangqi@6b77629b
g. hasPiece()
    i. Tested for a piece space with a piece and without, they came back with:
        1. true
        2. false
h. getPiece()
    i. Tested on the same space at the first hasPiece, it came back with:
        1. RookPiece@6f015169
i. squareThreatened
    i. Tested on a piece with wasn't threatened, and on one that was (i messed with startGame to put a piece where there wouldn't be, to make one threatened. Returned
        1. false
        2. true
j. numRows()
    i. Tested on the board i made, which was a xiangqi board. Should return 10.
        1. It did
k. numColumns()
    i. Tested on the board I made, which was a xiangqi board. Should return 9.
        1. It did

2. JavaFXChessBoardDisplay

a. getSquareSize()
    i. This should make all the squared on either type of board be the screen width/40
        1. It seems vaguely correct, but they are all squares so I'll take it
b. This is an interface, so the most important thing is that the 2 implementing classes both work in any context which requires one or the other, which is the case.

3. JavaFXEuropeanChessDisplay

a. displayEmptySquare()
    i. This is proven when a board displays. If the squares are in a red and black alternating grid and the squares are all, well, square, the displays will work
        1. It displayed correctly
b. displayFilledSquare()
    i. When the board with pieces launches, the displayFIlledSquare() method will make the pieces show up on circles with the proper label and color on a square background of the correct color
        1. This is exactly what happened
c. highlightSquare()

i. When a piece is clicked on, it should go the highlight color (blue) but maintain the background square color and the square shape.
    1. It does.

4. **JavaFXXiangqiDisplay**
    a. displayEmptySquare()
        i. This is proven when a board displays. If the squares are in a light grey and dark grey alternating grid and the squares are all, well, square, the displays will work
            1. It displayed correctly
    b. displayFilledSquare()
        i. When the board with pieces launches, the displayFIlledSquare() method will make the pieces show up on circles with the proper label and color on a square background of the correct color
            1. This is exactly what happened
    c. highlightSquare()
        i. When a piece is clicked on, it should go the highlight color (blue) but maintain the background square color and the square shape.
            1. It does.

Part 4: Swing GUI

5. **SwingChessBoard**
    a. Main Method
        i. When called, it should create a new board and run the startGame() method, so as long as it displays the board, the main method works.
            1. It does,
    b. addPiece
        i. This method is called by the startGame() method, so as long as the board displays with the necessary pieces on the board and they look correct, it's fine.
            1. It worked.
    c. removePiece
        i. When I moved a piece, the space it had been became empty. This is because the removePiece method worked correctly.
    d. ChessAction
        i. When I click a piece for the first time, it should highlight the piece. WHen I click a square it cannot move to, it should un-highlight the piece. When I again select the piece and click itself, it shouldn't do anything. WHen I click another square it can move to, it should move to that square. When I try to click a piece whose turn it is not, it should do nothing.
            1. All of these did in fact work.
6. **SwingChessBoardDisplay**
    a. getSquareSize()
        i. This should make all the squared on either type of board be the screen width/40

            1. It seems vaguely correct, but they are all squares so I'll take it
    b. This is an interface, so the most important thing is that the 2 implementing classes both work in any context which requires one or the other, which is the case.
7. **SwingEuropeanChessDisplay**
    a. displayEmptySquare()
        i. This is proven when a board displays. If the squares are in a red and black alternating grid and the squares are all, well, square, the displays will work
            1. It displayed correctly
    b. displayFilledSquare()
        i. When the board with pieces launches, the displayFIlledSquare() method will make the pieces show up on squares with the proper label and color
            1. This is exactly what happened
    c. highlightSquare()
        i. When a piece is clicked on, it should go the highlight color (blue)
            1. It does.
8. **SwingXiangqiDisplay**
    a. displayEmptySquare()
        i. This is proven when a board displays. If the squares are in a light grey and dark grey alternating grid and the squares are all, well, square, the displays will work
            1. It displayed correctly
    b. displayFilledSquare()
        i. When the board with pieces launches, the displayFIlledSquare() method will make the pieces show up on squares with the proper label and color
            1. This is exactly what happened
    c. highlightSquare()
        i. When a piece is clicked on, it should go the highlight color (blue)
            1. It does.