- Data Representation:
    - The checkerboard will be comprised of a two dimensional ArrayList of type Checker, where Checker represents the Class of each individual piece.
    - The Checker Class
        - Fields:
            - King
                - Boolean, where true means that this piece is a king, giving it the ability to move backwards
            - Color
                - Assigned to a Color enum of BLACK and RED
                - Determines which direction the piece may move and also which pieces it may jump over
            - Position
                - Used to determine which piece is selected for moving.
                - Of generic type Pair, holding two Integers
    - Methods
        - Getters and setters for King, Color, and Position

- Determine what moves are Possible:
    - Regular Pieces
        - Simple Move
            - A regular piece may only move to the y+1 row, where y represents its row. Further, it can only move to a space with an even x, if y+1 is odd, or only to an odd x, if y+1 is even. Beyond that, a piece must always be moving to a space that is in either the x+1 or x-1 row from its current position.
        - Jump
            - If a piece encounters another piece of a different color in one of the legal spaces outlined above, it must then check to see if there is an open space in a legal position from the perspective of this new piece, which could be accomplished by simply calling the move method that was previously called on the original piece. If it returns a true for each of the two spaces, then the original piece must jump. Once landing, the check must be called again to determine if another jump is possible, as jumping is required.
    - Kings
        - All is the same as a regular piece with a single exception: the piece may move to either x+1, x-1, in regards to columns, and y+1, y-1 in regards to columns.
    - Methods
        - move(Checker checker,Pair<Integer,Integer>)
            - called when a move is attempted by a player. By passing the checker object, it has the information of which color the checker

is, if it's a king or not, and its position.The pair represents the position of the attempted move. This function will call another function to determine if the move is legal. If it is legal, it will update the boardArray.
        - Can be either a simple move or a jump. If the move is a jump, then if the jump is deemed legal, move is called again, with both all of the possible locations.
    - isLegal(Checker checker,Pair<Integer,Integer>)
        - Gets the position of the original checker, and using the Pair of Integers passed in, checks if the move is legal.
            - Checking if the move is legal consists of two parts:
                - Is there an obstruction
                    - In this case, simply rejects the move if the checker is of the same color.
                - Is there a jump elsewhere
                    - In this case, iterates through the array for checkers of the same color, and determines if any may move, by calling the move function with each checker and its possible moves.
- Determine if a move is illegal
    - A Jump is Possible Elsewhere
        - If a player attempts to make a move, the program should first check each piece of their color to determine if any of these pieces are able to jump
    -
    - Methods
        - IsLegal(Checker checker,Pair<Integer,Integer>) handles all cases and is outlined above
- Determine a winner, loser, or draw condition
    - Done by implementing a checkForEnd() function that is called at the end of every turn.
    - Conditions
        - Win/Lose
            - Parse through entire array, counting number of each color of pieces. If either color has 0, the other team wins.
        - Draw
            - Could be accomplished by checking every non-null checker to see if it has a possible move, if not, then it is a draw. This would be iterated via an iteration through the board array.
    - Methods
        - checkForEnd()
            - Does not require parameters, because boardArray will be global.
            - To check for Draw, it will call move() on each checker

- Implement the algorithms for the computer to determine moves
    - The computer, in its most basic form, must simply find a move and execute it. This can be done by iterating through all checkers of its color to find a move, and choosing one to execute. The trouble comes with the choosing. It should check the value of the move (Does it capture a piece/is it a jump), and also how safe the move is to determine a net worth. It will choose the highest worth.
    - Methods
        - The only new method required here is the isSafe(Color color,Pair<Integer,Integer>) method. It will check the space provided to see if it would be jumped by a checker of the opposing color.
- Implement a UI representation of the game and game play
    - To accomplish this, when the CheckerboardClass calls build(), it will check each position against the boardArray which holds the checker positions, and load an image in.
    - For moving, the player will select a checker first, then select a space it wants to move it to.