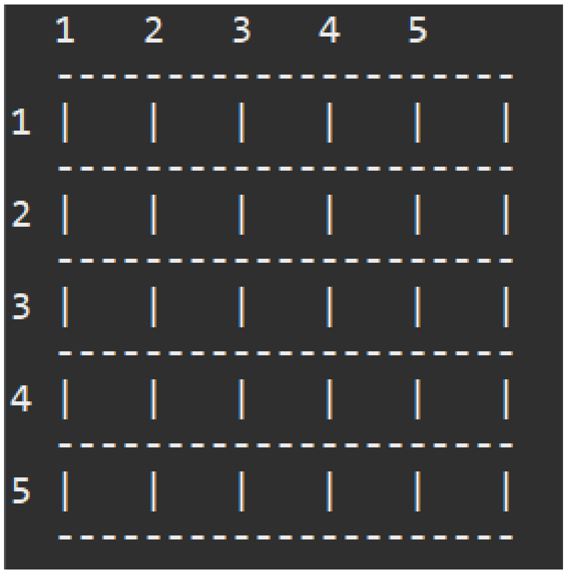
In my design I have four classes, my main class LinearDomination, is the start of my program. The other three classes, Game Cell, and Line contain the information about the current game.

- **LinearDomination (main class)**
  This is my main class where my program begins. This class has four methods: main, allConditionsMet, displayBoard, and takeTurn.
  1. Method main
     - When a new game begins, my program starts by instantiating a new Game called linearDomination, and an index called player is set to 1.
     - For as many turns as there are in the game, I increment player by 1, and grab 4 coordinates from the arraylist, the first two being the start point (sr,sc) and the second two the end point (er,ec). If player is odd, it is player 1's turn. If player is even, it is player 2's turn. If this is the beginning of the game or all conditions have been met, player takes their turn, and the board is displayed. If not, an error message is displayed and the turn advances to the next player.
  2. Method allConditionsMet
     - This method returns a Boolean value that is used as one of two conditions in main that make a player's turn valid (the other condition being it is the first move)
     - For K number of times, if this start-point equals last start-point, or this end-point equals last end-point, return false. If this midpoint equals last mid-point, return false. Finally, if this angle equals last angle, return false. If all three conditions return true, then the method returns true.
  3. Method displayBoard
     - This method is used to display the gameboard array in a nicely formatted way.
     - For every Cell in the array, I get that cell's text and display it. The below picture is an example of a displayed gameboard with all cells set as empty strings.



```
   1   2   3   4   5
  ---------------------
1 |   |   |   |   |   |
  ---------------------
2 |   |   |   |   |   |
  ---------------------
3 |   |   |   |   |   |
  ---------------------
4 |   |   |   |   |   |
  ---------------------
5 |   |   |   |   |   |
  ---------------------
```

4. Method takeTurn
   - This is my biggest method as there are 8 possible loops to go through. Depending on whether the components are greater than zero, less than zero or equal to zero, I will iterate through the gameboard in a different way.
     1. If vector[0] > 0 and vector[1] > 0
     2. If vector[0] < 0 and vector[1] > 0
     3. If vector[0] > 0 and vector[1] < 0
     4. If vector[0] < 0 and vector[1] < 0
     5. If vector[0] > 0 and vector[1] = 0
     6. If vector[0] < 0 and vector[1] = 0
     7. If vector[0] = 0 and vector[1] > 0
     8. If vector[0] = 0 and vector[1] < 0
   - Depending on which condition is met, I will loop through all the points in between the start and ends, and determine if they lie on the line.
   - If the point is between -0.5 and 0, or 0 and 0.5, then I include the cell on the line.

- **Game**
  The game class reads the game information from an input file. When a new game is created, the game class opens the file, stores the relevant game information in variables, and closes the file. After the game information is read, a game board is created using a 2-dimensional array of objects.

  1. First integer read from file is stored in **N**.
     a. Method getN
  2. Second integer read from file is stored in **K**.
     a. Method getK
  3. The remaining integers in file are stored in an array list of **coordinates**.
     a. Method getSize
     b. Method getCoordinate
     c. Method removeCoordinate
     d. Method setCoordinate
  4. The number of moves (lines in the file) are stored in a variable, **turns**. This is calculated by dividing the size of the coordinate list by 4.
  5. The game is displayed as a square using a 2-dimensional array of Cell objects called **gameBoard**. The gameBoard is of size N x N.
     a. Method createBoard
        (For each element in the gameBoard, a cell object is created, setting its current text as empty.)
     b. Method getGameBoard
  6. The lines from the last K turns are stored in an arraylist of Line objects, called **lines**.
     a. Method setLastLine

   b. Method getLastLine
   c. Method getLastLinesSize
      (Returns the number of lines currently stored.)
   d. Method lastLinesEmpty
      (Returns true if the lines list is empty)

- **Cell**
  The Cell class is used to keep track of which player has played which cell. When a Cell is created, it takes a string as an argument, which represents player's symbol, (X or O).
  1. This class has one private variable, **text**, which stores the cell's symbol.
     a. Method setText
     b. Method getText
- **Line**
  The Line class stores all relevant information about the current line in play. At the start of a turn, a new line object is instantiated, sending the given start and end points as arguments.

  1. The start point is stored in an array, **startPoint**.
     a. Method getStartPoint
  2. The end point is stored in an array, **endPoint**.
     a. Method getEndPoint
  3. The vector difference between the two points is stored in an array **difference**.
     a. Method setDifference
        (Subtracts both the start point components from the end points.)
     b. Method getDifference
  4. The line's angle is stored in a variable, **angle**.
     a. Method setAngle
        I compute a value to represent the line's angle by taking the arctan of its components
     b. Method getAngle
  5. The line's magnitude is stored in a variable, **magnitude**.
     a. Method setMagnitude
        (I compute the line's magnitude using $c = \sqrt{a^2 + b^2}$)
     b. Method getMagnitude
  6. I store the midpoint of the line in an array called **midpoint**
     a. Method setMidPoint
        (I compute the midpoint using $r = \frac{1}{2}p + \frac{1}{2}q$)
     b. Method getMidPoint

Pseudocode for main

```
Game linearDomination

Cell[][] gameBoard

player = 1

    FOR player TO linearDomination .numberOfTurns

        IF player is an odd number, THEN

            symbol = "O"

        ELSE THEN

            symbol = "X"
        ENDIF

         [] startPoint = {linearDomination.getCoordinate(0),
        linearDomination.getCoordinate(1)}

         [] endPoint = {linearDomination.getCoordinate(2),
        linearDomination.getCoordinate(3)}

        FOR j TO 4
            linearDomination.removeCoordinate(0)

        ENDFOR

        Line theLine = new Line(startPoint, endPoint)

        IF lastLines arraylist is empty OR all conditions have been met THEN
            takeTurn(theLine)
            displayBoard( )
            addThisLineToArrayList(theLine);
        ELSE THEN
            PRINT LINE(“invalid, advancing to next player”)
        ENDIF

    ENDFOR
```