To solve the Eight Queens problem, I first wanted to make sure I could store and print a chessboard like structure. To do this, I have used a 2D array as this stores x and y coordinates. I played around with the printing of this until I was happy with how this looked.

Once I was happy with how my chessboard looked, I started to work on checking whether a given coordinate was a valid place to insert a new queen. I did this by adding a few queens manually, and then running the isSafe() method that I was working on to see if it would return the correct result. I then tweaked the loops based on the results of these tests.

My next task was to get the user input and use this to check if there is a queen at this position. To do this, I took a string from the user which contained 2 characters – a letter and a number. These translated to the coordinates on the board, with the letters being the x coordinate and the number being the y coordinate. I then ran the toUpperCase() method built into java so that I didn’t have to deal with checking uppercase and lowercase characters, and then used a while loop to check if the given coordinates were within the range of coordinates on the chessboard.

When inserting the user’s queen onto the board, I had to use a nested for loop to check through each value within the 2D array and convert the column value to a character using the char cast. I had to add a value of 65 to the I value in order to get the correct ASCII value for upper case letters. When checking the y value, which is the number in the users inputted coordinate, I used the .charAt() method to get the second character in this string, and then had to say –‘0’ to convert this from a character to an integer.

Finally, I had the algorithm to actually solve the solution to write. For this, I started in the leftmost column worked my way right, putting a queen in the first available safe position found. I then used backtracking to say if there was no safe position in a column then go back a column and move that queen to the next safe position in that column. Once this had found a solution then it would print this and try to find another. I implemented this by using recursion within my solveNQUtil() method. For the backtracking part of this algorithm, I used a short circuit evaluation as this is more efficient than writing an if statement that does the same thing. This says that if there is a queen in the next column, then set the result to true. Otherwise, keep this variable as false and get rid of the queen in this coordinate. Then keep checking the rest of the rows in this column to find another position to place this queen.

As I already had a queen placed on the board that I didn’t want the program to overwrite, I added an if else statement to check to see if the current column being looked at is the one that the first queen had been placed in. If it was, then the algorithm skips to the next column, otherwise runs the algorithm described above on that column.

As a final touch, I added a variable that tracks how many solutions have been printed and gives that number before each solution. This makes it easy to see how many solutions you can get for each of the starting queen locations.