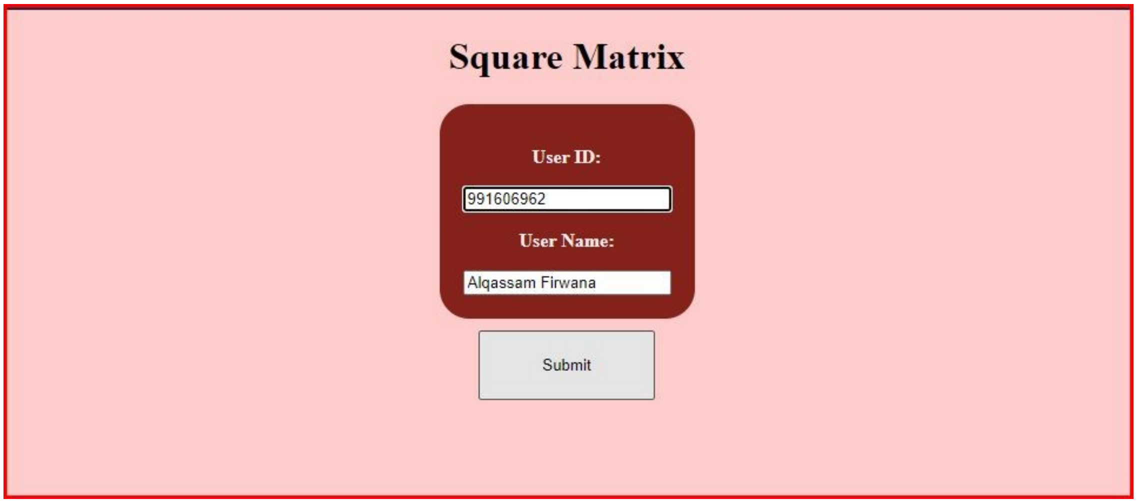
Table of Contents

The Web Page

1- The Landing Page



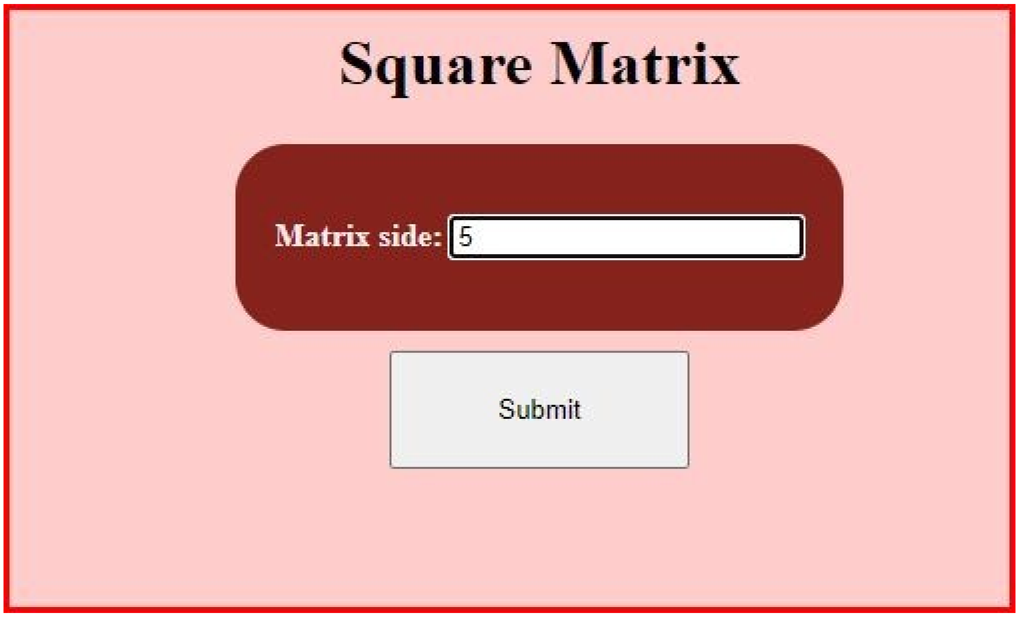
Square Matrix

User ID: 991606962

User Name: Alqassam Firwana

- ➔ Asking users for ID and Name, which will be added to the H2.DB.

2- Home Page

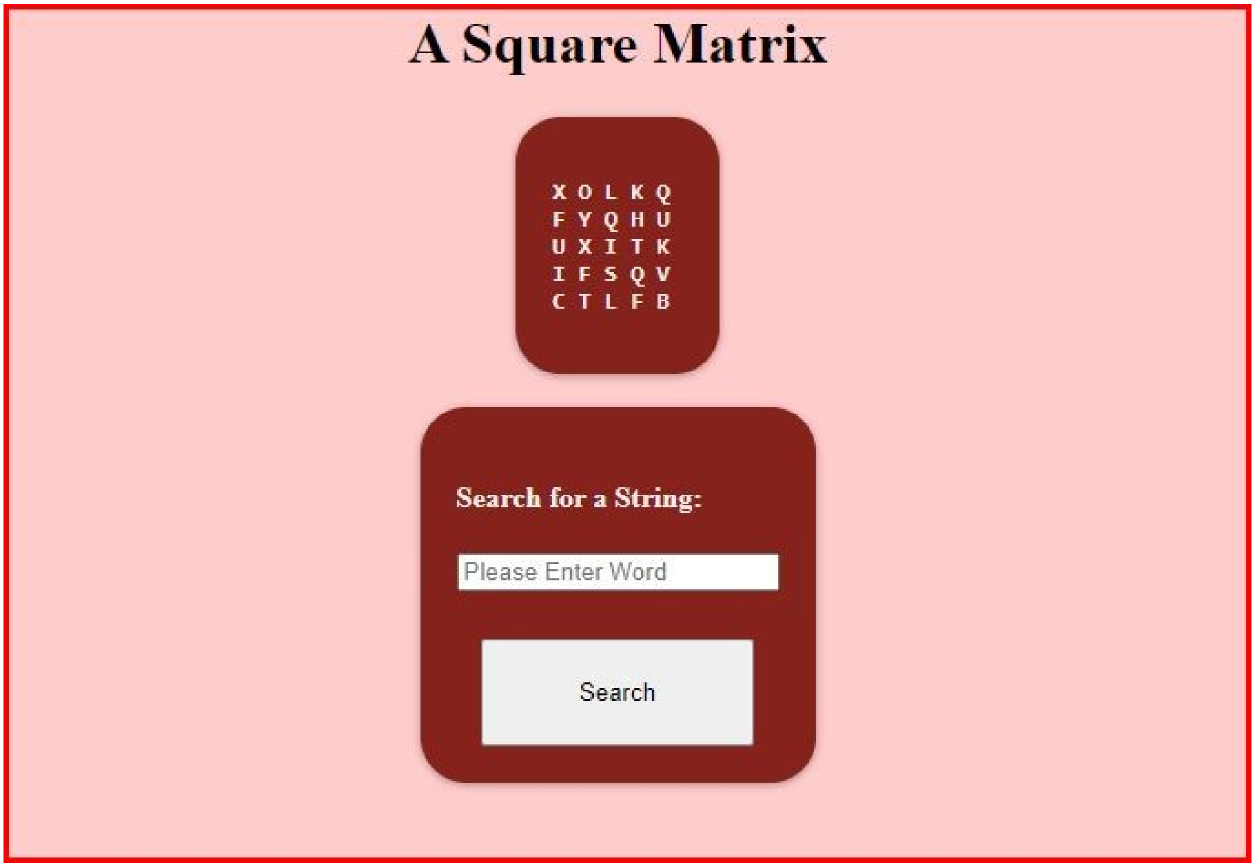


Square Matrix

Matrix side: 5

- ➔ Asking users for Matrix side length 5 or More.

3- View Page:



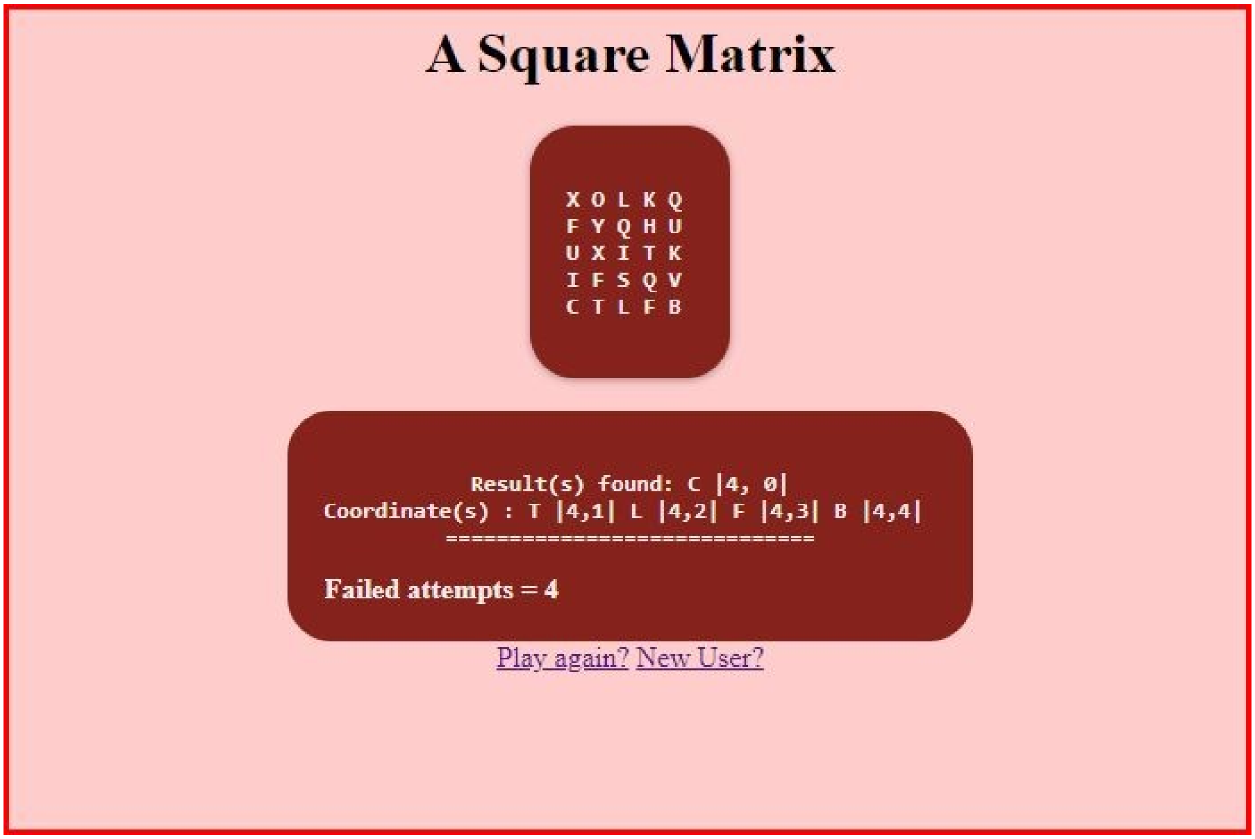
A Square Matrix

```
X O L K Q
F Y Q H U
U X I T K
I F S Q V
C T L F B
```

Search for a String:

➔ This page shows the generated matrix and asks for a Search word.

4- Search Results Page:



A Square Matrix

```
X O L K Q
F Y Q H U
U X I T K
I F S Q V
C T L F B
```

```
           Result(s) found: C |4, 0|
Coordinate(s) : T |4,1| L |4,2| F |4,3| B |4,4|
          ==============================
```

Failed attempts = 4

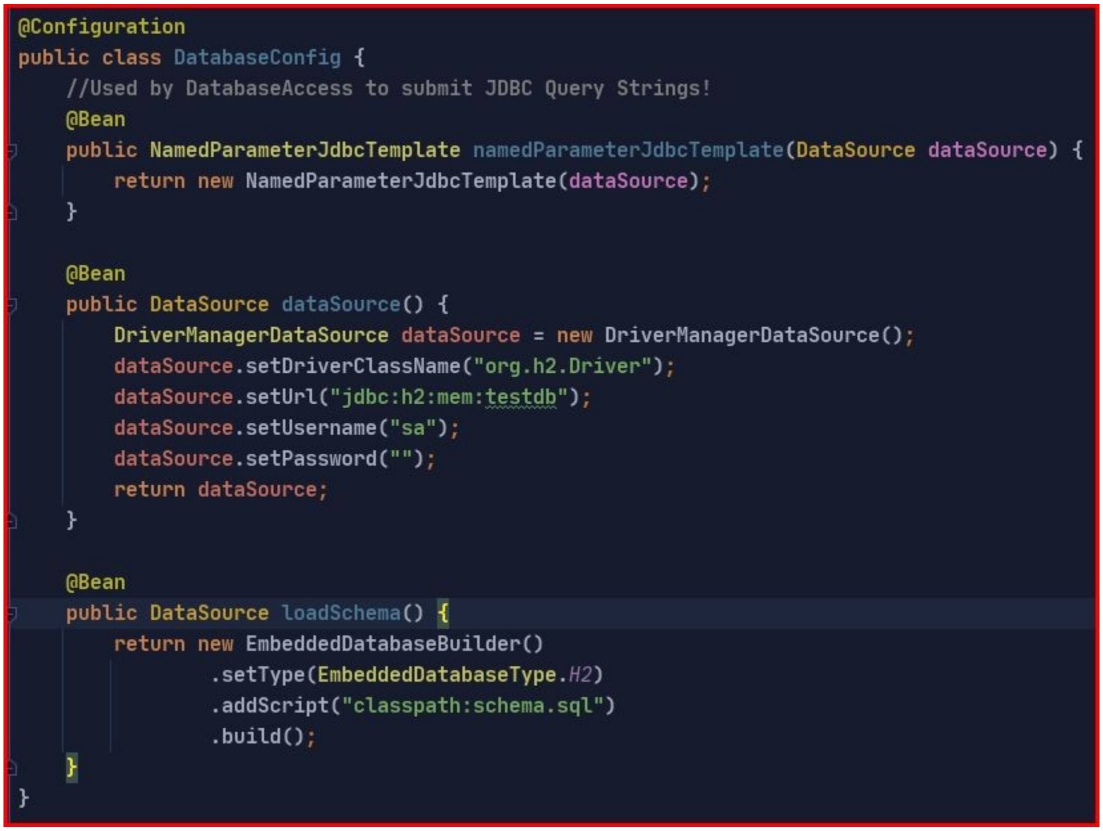
- ➔ This page shows the search results and location of first occurrence and the rest of the coordinates of the word.
- ➔ And shows the number of failed attempts after the first letter matched.
- ➔ Give the user to play again or enter as a new user.

The Code

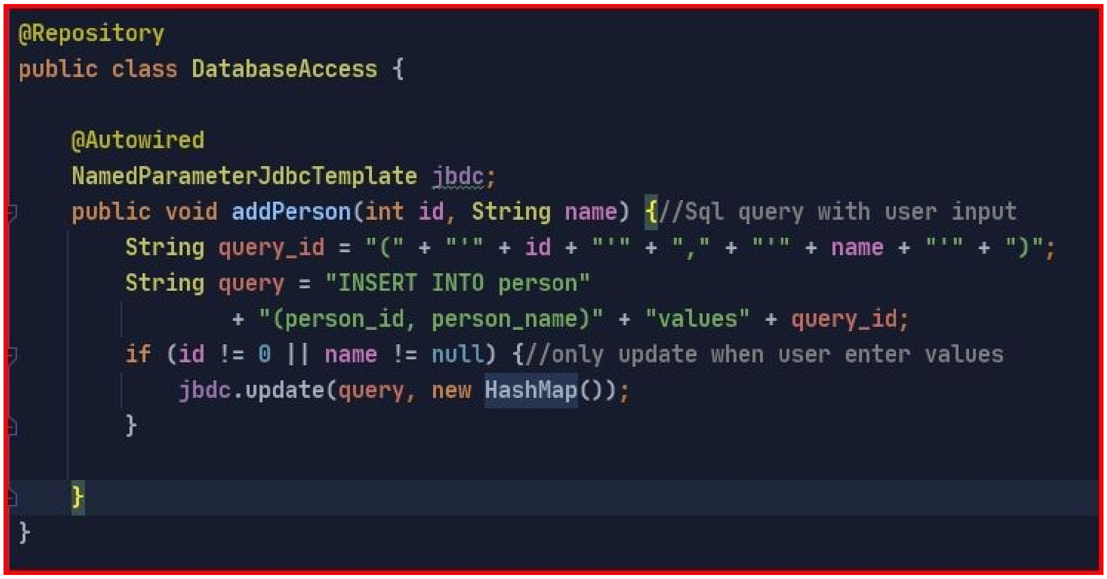
Part "A"

DataBase: UserInfo

-Database Configuration



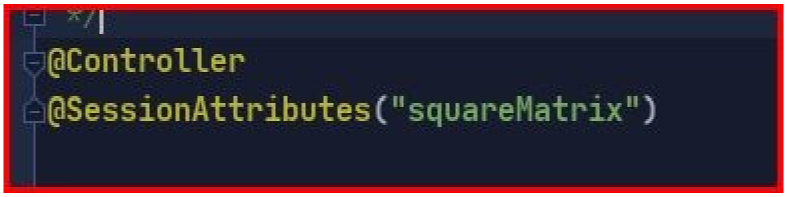
```java
@Configuration
public class DatabaseConfig {
    //Used by DatabaseAccess to submit JDBC Query Strings!
    @Bean
    public NamedParameterJdbcTemplate namedParameterJdbcTemplate(DataSource dataSource) {
        return new NamedParameterJdbcTemplate(dataSource);
    }

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("org.h2.Driver");
        dataSource.setUrl("jdbc:h2:mem:testdb");
        dataSource.setUsername("sa");
        dataSource.setPassword("");
        return dataSource;
    }

    @Bean
    public DataSource loadSchema() {
        return new EmbeddedDatabaseBuilder()
                .setType(EmbeddedDatabaseType.H2)
                .addScript("classpath:schema.sql")
                .build();
    }
}
```



```java
@Repository
public class DatabaseAccess {

    @Autowired
    NamedParameterJdbcTemplate jbdc;
    public void addPerson(int id, String name) {//Sql query with user input
        String query_id = "(" + "'" + id + "'" + "," + "'" + name + "'" + ")";
        String query = "INSERT INTO person"
                + "(person_id, person_name)" + "values" + query_id;
        if (id != 0 || name != null) {//only update when user enter values
            jbdc.update(query, new HashMap());
        }

    }
}
```

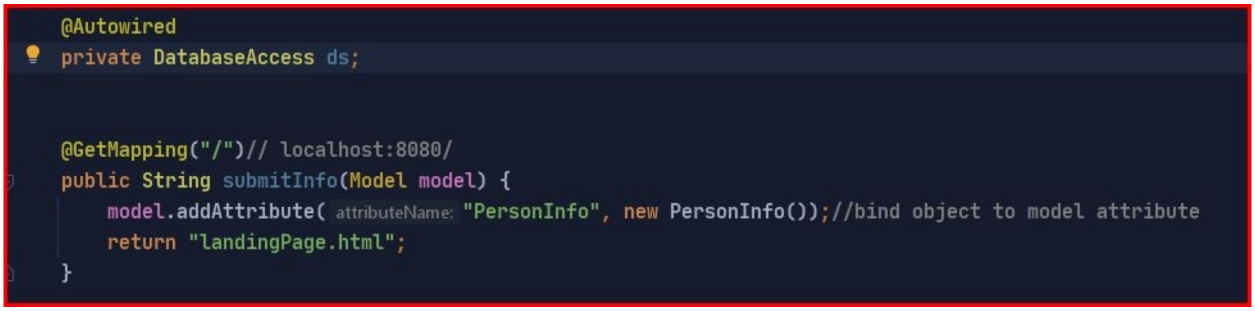
- → H2 Database configurations, that will be used by DatabaseAccess which will take a query of user “from PersonInfo Class” input and store it to inpresistant database.
- → Creating schema that contain user info tables

Controller



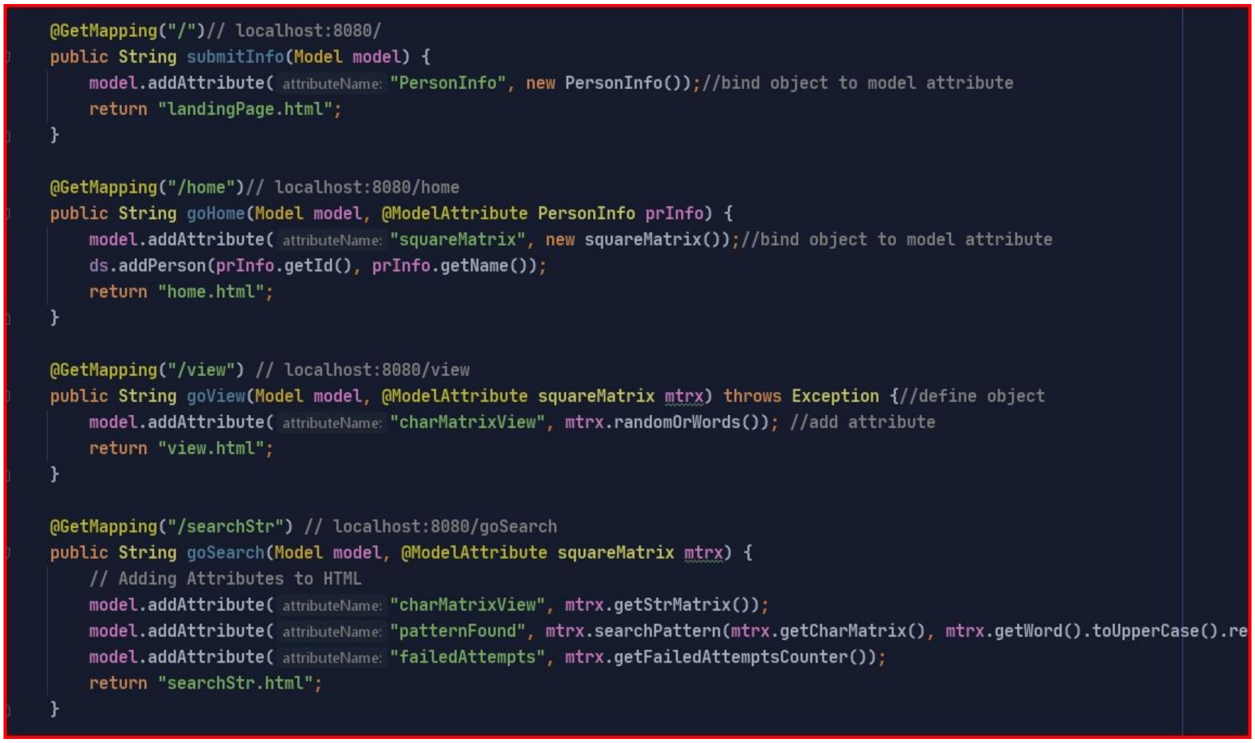
```java
*/
@Controller
@SessionAttributes("squareMatrix")
```

- ➔ ***@SessionAttribute*** to make define session scope and make the program remember this attribute.[4]



```
@Autowired
private DatabaseAccess ds;


@GetMapping("/")// localhost:8080/
public String submitInfo(Model model) {
    model.addAttribute( attributeName: "PersonInfo", new PersonInfo());//bind object to model attribute
    return "landingPage.html";
}
```

- ***@Autowired*** annotation to enable Spring boot framework to perform dependency injection.[3]
- Binding certain method parameter or return to a named model which will be viewed on HTML page.[1]
- Mapping to “landingpage.html”



```
@GetMapping("/")// localhost:8080/
public String submitInfo(Model model) {
    model.addAttribute( attributeName: "PersonInfo", new PersonInfo());//bind object to model attribute
    return "landingPage.html";
}

@GetMapping("/home")// localhost:8080/home
public String goHome(Model model, @ModelAttribute PersonInfo prInfo) {
    model.addAttribute( attributeName: "squareMatrix", new squareMatrix());//bind object to model attribute
    ds.addPerson(prInfo.getId(), prInfo.getName());
    return "home.html";
}

@GetMapping("/view") // localhost:8080/view
public String goView(Model model, @ModelAttribute squareMatrix mtrx) throws Exception {//define object
    model.addAttribute( attributeName: "charMatrixView", mtrx.randomOrWords()); //add attribute
    return "view.html";
}

@GetMapping("/searchStr") // localhost:8080/goSearch
public String goSearch(Model model, @ModelAttribute squareMatrix mtrx) {
    // Adding Attributes to HTML
    model.addAttribute( attributeName: "charMatrixView", mtrx.getStrMatrix());
    model.addAttribute( attributeName: "patternFound", mtrx.searchPattern(mtrx.getCharMatrix(), mtrx.getWord().toUpperCase().re
    model.addAttribute( attributeName: "failedAttempts", mtrx.getFailedAttemptsCounter());
    return "searchStr.html";
}
```

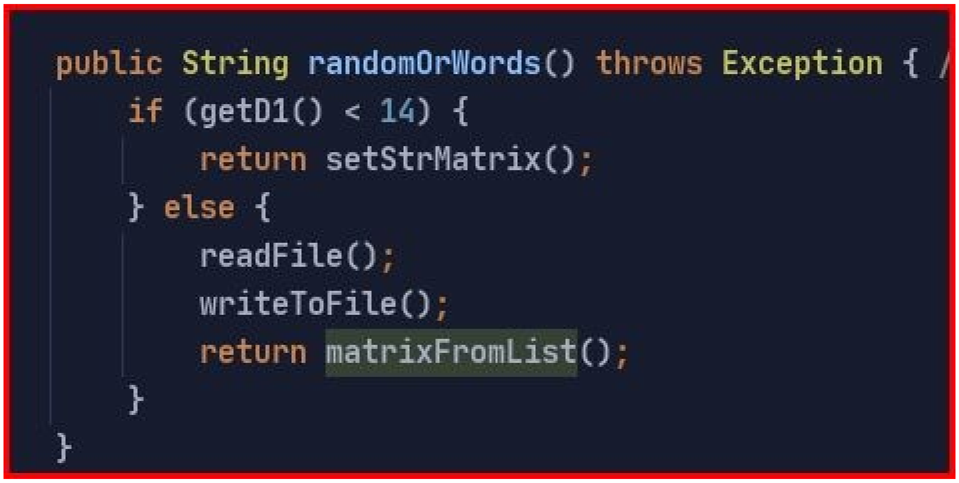
- Mapping to “home.html”, “view.html” and “searchStr.html”.
- Binding certain method parameter or return to a named model which will be viewed on HTML page.[1]

Square Matrix Class

- Implementing serializable
- All properties are private
- Lombok Getters/Setters and no-arg constructor

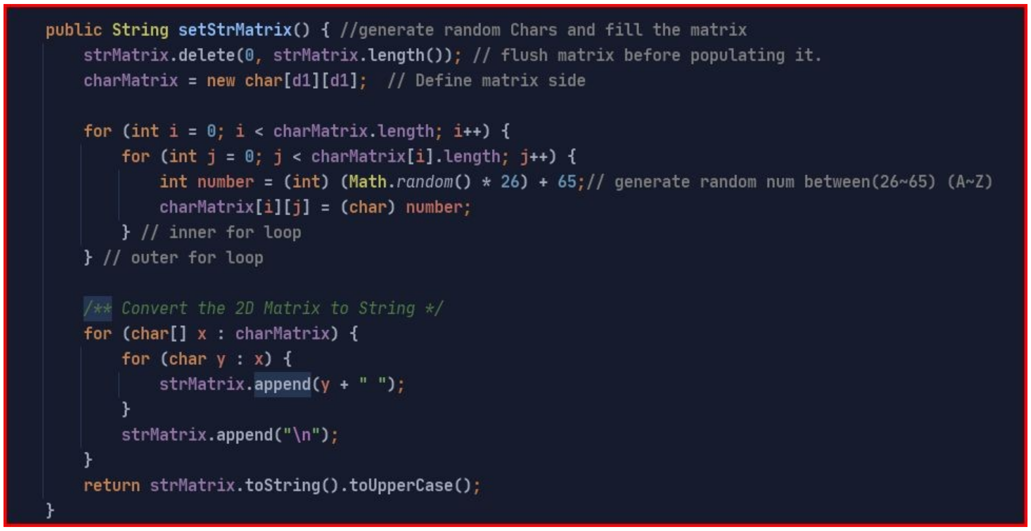
matrixGenerator

- A method to check user input if less than 14 invokes a method that generates a random char matrix, if else generates a randomly selected char matrix from words that are predefined in an array list from a file.



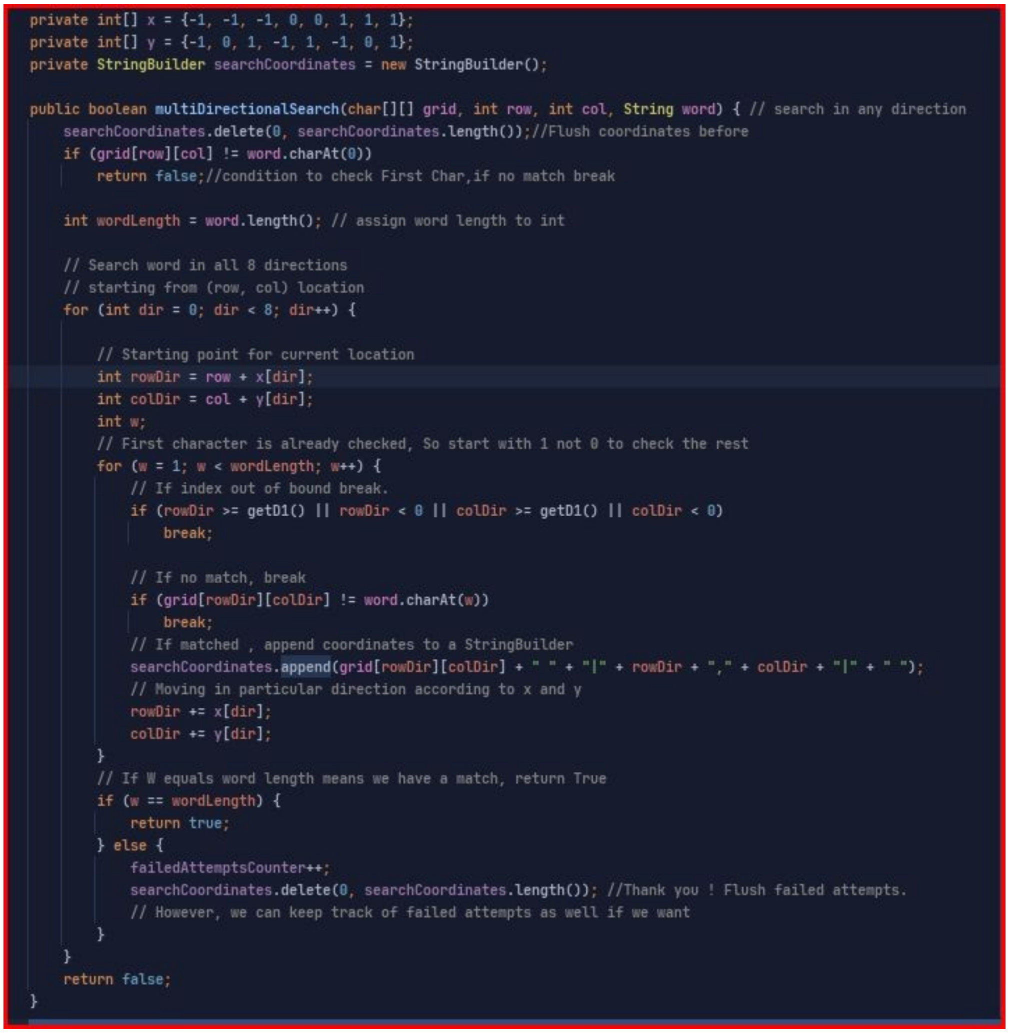
```
public String randomOrWords() throws Exception { /
    if (getD1() < 14) {
        return setStrMatrix();
    } else {
        readFile();
        writeToFile();
        return matrixFromList();
    }
}
```

➔ Generate random char matrix



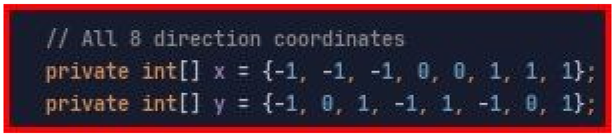
```java
public String setStrMatrix() { //generate random Chars and fill the matrix
    strMatrix.delete(0, strMatrix.length()); // flush matrix before populating it.
    charMatrix = new char[d1][d1];  // Define matrix side

    for (int i = 0; i < charMatrix.length; i++) {
        for (int j = 0; j < charMatrix[i].length; j++) {
            int number = (int) (Math.random() * 26) + 65;// generate random num between(26~65) (A~Z)
            charMatrix[i][j] = (char) number;
        } // inner for loop
    } // outer for loop

    /** Convert the 2D Matrix to String */
    for (char[] x : charMatrix) {
        for (char y : x) {
            strMatrix.append(y + " ");
        }
        strMatrix.append("\n");
    }
    return strMatrix.toString().toUpperCase();
}
```

➔ Search Algorithm



```java
private int[] x = {-1, -1, -1, 0, 0, 1, 1, 1};
private int[] y = {-1, 0, 1, -1, 1, -1, 0, 1};
private StringBuilder searchCoordinates = new StringBuilder();

public boolean multiDirectionalSearch(char[][] grid, int row, int col, String word) { // search in any direction
    searchCoordinates.delete(0, searchCoordinates.length());//Flush coordinates before
    if (grid[row][col] != word.charAt(0))
        return false;//condition to check First Char,if no match break

    int wordLength = word.length(); // assign word length to int

    // Search word in all 8 directions
    // starting from (row, col) location
    for (int dir = 0; dir < 8; dir++) {

        // Starting point for current location
        int rowDir = row + x[dir];
        int colDir = col + y[dir];
        int w;
        // First character is already checked, So start with 1 not 0 to check the rest
        for (w = 1; w < wordLength; w++) {
            // If index out of bound break.
            if (rowDir >= getD1() || rowDir < 0 || colDir >= getD1() || colDir < 0)
                break;

            // If no match, break
            if (grid[rowDir][colDir] != word.charAt(w))
                break;
            // If matched , append coordinates to a StringBuilder
            searchCoordinates.append(grid[rowDir][colDir] + " " + "|" + rowDir + "," + colDir + "|" + " ");
            // Moving in particular direction according to x and y
            rowDir += x[dir];
            colDir += y[dir];
        }
        // If W equals word length means we have a match, return True
        if (w == wordLength) {
            return true;
        } else {
            failedAttemptsCounter++;
            searchCoordinates.delete(0, searchCoordinates.length()); //Thank you ! Flush failed attempts.
            // However, we can keep track of failed attempts as well if we want
        }
    }
    return false;
}
```

This search algorithm use a set of coordinates (x, y) to to move in all 8 direction from a given point:[5] will explain all 8 scenarios



```
// All 8 direction coordinates
private int[] x = {-1, -1, -1, 0, 0, 1, 1, 1};
private int[] y = {-1, 0, 1, -1, 1, -1, 0, 1};
```

- **Scenario A**: moving diagonal “**LR**”

[0,0]	[0,1]	[0,2]
[1,0]	[1,1]	[1,2]
[2,0]	[2,1]	[2,2]

Will notice that we add [1,1] each step, so **x,y** will be = [1,1]

Starting point underlined.

- **Scenario B**: moving diagonal “Backward **LR**”

[0,0]	[0,1]	[0,2]
[1,0]	[1,1]	[1,2]
[2,0]	[2,1]	[2,2]

Will notice that we add [-1,-1] each step, so **x,y** will be = [-1,-1]

- **Scenario C**: moving horizontal “left - right”

[0,0]	[0,1]	[0,2]
[1,0]	[1,1]	[1,2]
[2,0]	[2,1]	[2,2]

Will notice that we add [0,1] each step, so **x,y** will be = [0,1]

- **Scenario D**: moving horizontal “right - left”

[0,0]	[0,1]	[0,2]
[1,0]	[1,1]	[1,2]
[2,0]	[2,1]	[2,2]

Will notice that we add [0,-1] each step, so **x,y** will be = [0,-1]

- **Scenario E**: moving vertical “Bottom - Top”

[0,0]	[0,1]	[0,2]
[1,0]	[1,1]	[1,2]
[2,0]	[2,1]	[2,2]

Will notice that we add [-1,0] each step so **x,y** will be = [-1,0]

- **Scenario F**: moving vertical “Top - Bottom”

[0,0]	[0,1]	[0,2]
[1,0]	[1,1]	[1,2]
[2,0]	[2,1]	[2,2]

Will notice that we add [1,0] each step so **x,y** will be = [1,0]

- **Scenario J**: moving diagonal “**RL**”

[0,0]	[0,1]	[0,2]
[1,0]	[1,1]	[1,2]
[2,0]	[2,1]	[2,2]

Will notice that we add [1,-1] each step so **x,y** will be = [1,-1]

- **Scenario H**: moving diagonal “Backward **RL**”

[0,0]	[0,1]	[0,2]
[1,0]	[1,1]	[1,2]
[2,0]	[2,1]	[2,2]

Will notice that we add [-1,1] each step so **x,y** will be = [1,-1]

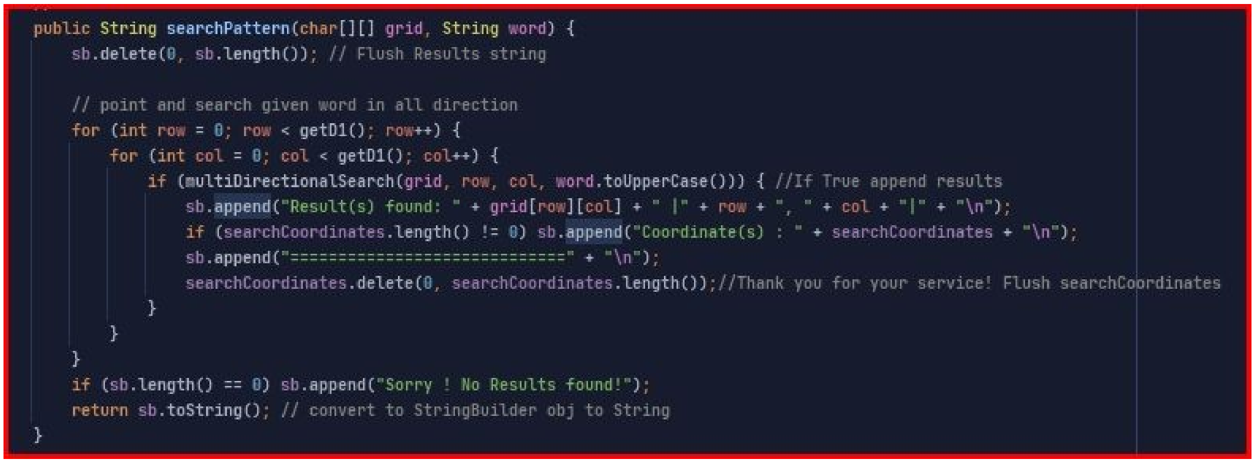
After the algorithm finds a match from a specific start point and then confirms that the length of the given “word” equals the matched characters will return true. Moreover, the algorithm keeps track of all successful matched coordinates.

Additionally the algorithm also counts the unsuccessful attempts.

The results of this algorithm will be passed to another method that returns the starting point value and appends the rest of the matched coordinates to a string builder object.

Note: the following line ensures there is no indexOutofBoundException , so the program can check all locations “8 directions” around a given point and stays within the matrix boundaries.

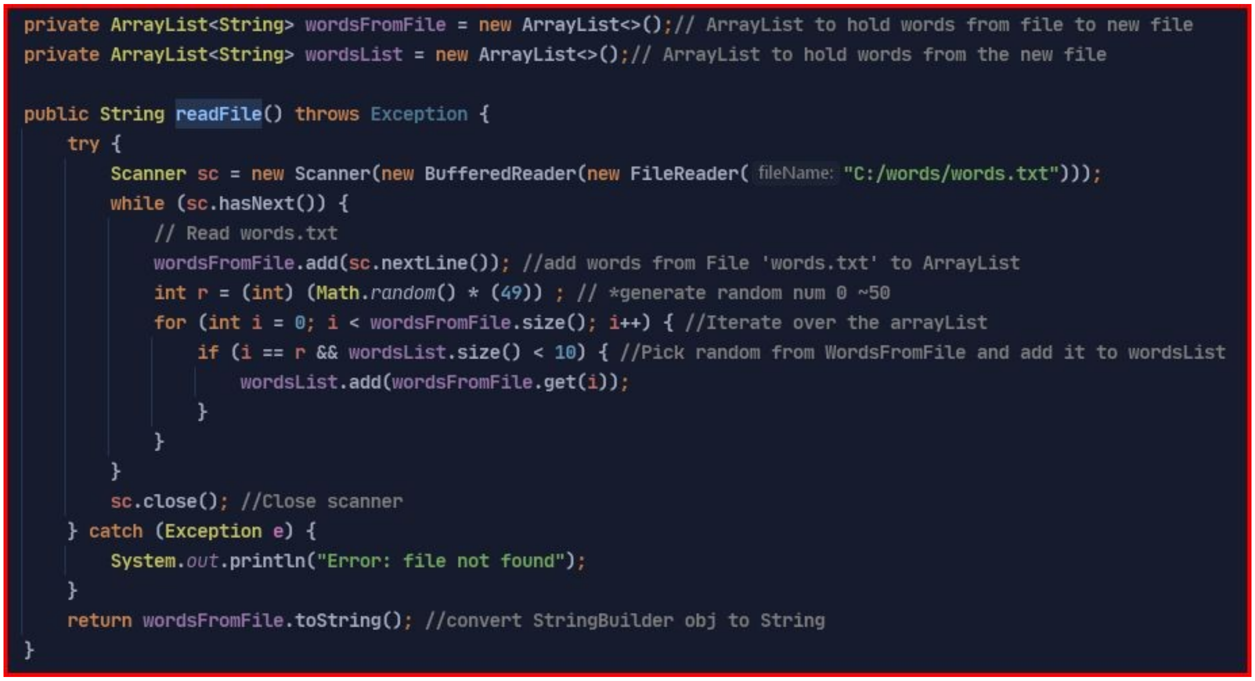
This method will invoke the previous method and report the results described above.



```java
public String searchPattern(char[][] grid, String word) {
    sb.delete(0, sb.length()); // Flush Results string

    // point and search given word in all direction
    for (int row = 0; row < getD1(); row++) {
        for (int col = 0; col < getD1(); col++) {
            if (multiDirectionalSearch(grid, row, col, word.toUpperCase())) { //If True append results
                sb.append("Result(s) found: " + grid[row][col] + " |" + row + ", " + col + "|" + "\n");
                if (searchCoordinates.length() != 0) sb.append("Coordinate(s) : " + searchCoordinates + "\n");
                sb.append("=============================" + "\n");
                searchCoordinates.delete(0, searchCoordinates.length());//Thank you for your service! Flush searchCoordinates
            }
        }
    }
    if (sb.length() == 0) sb.append("Sorry ! No Results found!");
    return sb.toString(); // convert to StringBuilder obj to String
}
```

Part "B"

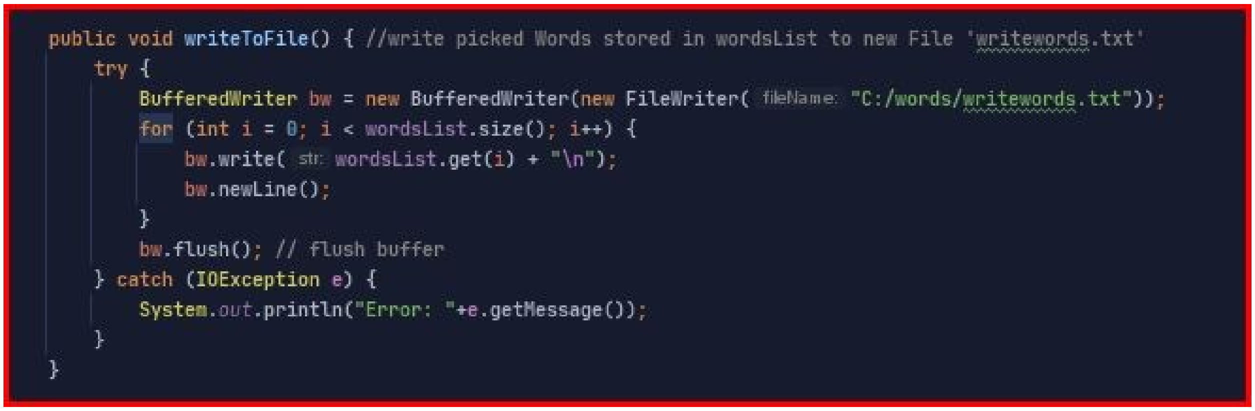
readFile()



```
private ArrayList<String> wordsFromFile = new ArrayList<>();// ArrayList to hold words from file to new file
private ArrayList<String> wordsList = new ArrayList<>();// ArrayList to hold words from the new file

public String readFile() throws Exception {
    try {
        Scanner sc = new Scanner(new BufferedReader(new FileReader( fileName: "C:/words/words.txt")));
        while (sc.hasNext()) {
            // Read words.txt
            wordsFromFile.add(sc.nextLine()); //add words from File 'words.txt' to ArrayList
            int r = (int) (Math.random() * (49)) ; // *generate random num 0 ~50
            for (int i = 0; i < wordsFromFile.size(); i++) { //Iterate over the arrayList
                if (i == r && wordsList.size() < 10) { //Pick random from WordsFromFile and add it to wordsList
                    wordsList.add(wordsFromFile.get(i));
                }
            }
        }
        sc.close(); //Close scanner
    } catch (Exception e) {
        System.out.println("Error: file not found");
    }
    return wordsFromFile.toString(); //convert StringBuilder obj to String
}
```

This method reads file that contain 50 words, adds them to an array list and randomly selects 10 words and adds them to another array list.

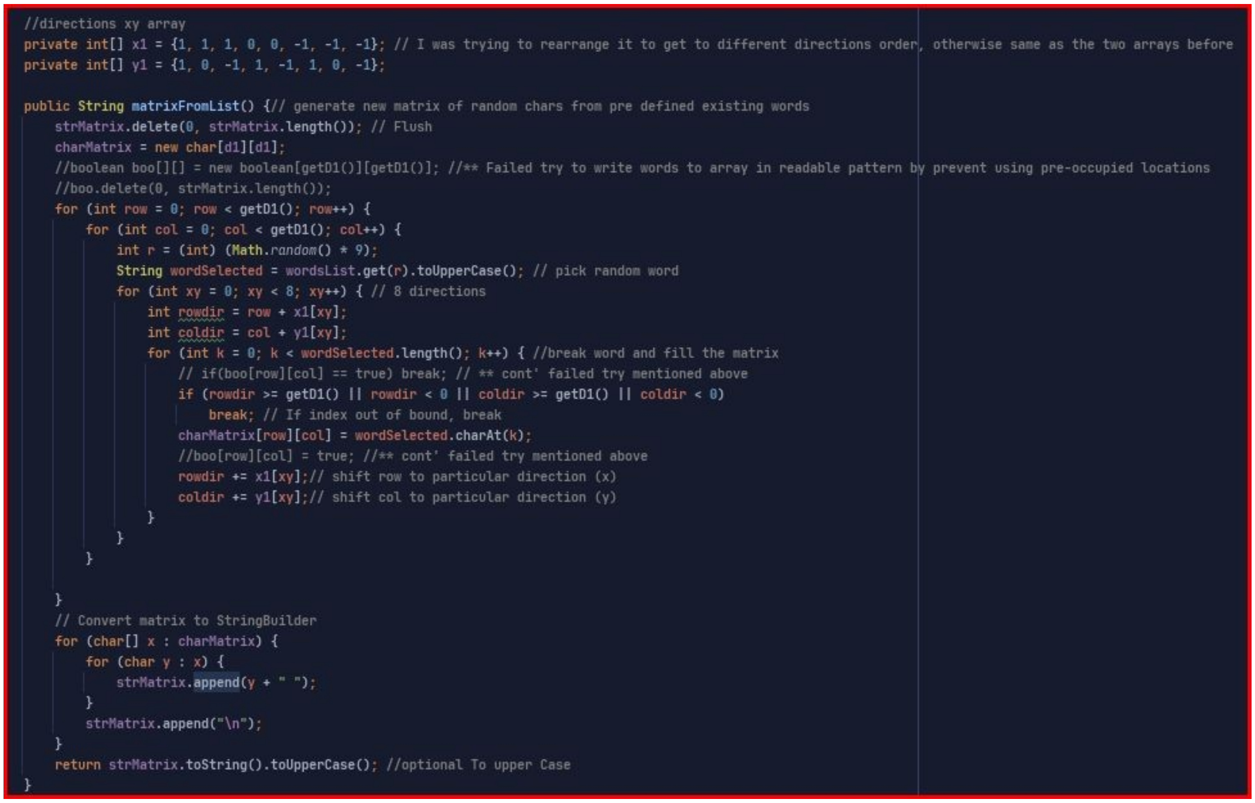
writeToFile()



```
public void writeToFile() { //write picked Words stored in wordsList to new File 'writewords.txt'
    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter( fileName: "C:/words/writewords.txt"));
        for (int i = 0; i < wordsList.size(); i++) {
            bw.write( str: wordsList.get(i) + "\n");
            bw.newLine();
        }
        bw.flush(); // flush buffer
    } catch (IOException e) {
        System.out.println("Error: "+e.getMessage());
    }
}
```

This method writes the randomly selected words to a txt file.

matrixFromList()



```java
//directions xy array
private int[] x1 = {1, 1, 1, 0, 0, -1, -1, -1}; // I was trying to rearrange it to get to different directions order, otherwise same as the two arrays before
private int[] y1 = {1, 0, -1, 1, -1, 1, 0, -1};

public String matrixFromList() {// generate new matrix of random chars from pre defined existing words
    strMatrix.delete(0, strMatrix.length()); // Flush
    charMatrix = new char[d1][d1];
    //boolean boo[][] = new boolean[getD1()][getD1()]; //** Failed try to write words to array in readable pattern by prevent using pre-occupied locations
    //boo.delete(0, strMatrix.length());
    for (int row = 0; row < getD1(); row++) {
        for (int col = 0; col < getD1(); col++) {
            int r = (int) (Math.random() * 9);
            String wordSelected = wordsList.get(r).toUpperCase(); // pick random word
            for (int xy = 0; xy < 8; xy++) { // 8 directions
                int rowdir = row + x1[xy];
                int coldir = col + y1[xy];
                for (int k = 0; k < wordSelected.length(); k++) { //break word and fill the matrix
                    // if(boo[row][col] == true) break; // ** cont' failed try mentioned above
                    if (rowdir >= getD1() || rowdir < 0 || coldir >= getD1() || coldir < 0)
                        break; // If index out of bound, break
                    charMatrix[row][col] = wordSelected.charAt(k);
                    //boo[row][col] = true; //** cont' failed try mentioned above
                    rowdir += x1[xy];// shift row to particular direction (x)
                    coldir += y1[xy];// shift col to particular direction (y)
                }
            }
        }

    }
    // Convert matrix to StringBuilder
    for (char[] x : charMatrix) {
        for (char y : x) {
            strMatrix.append(y + " ");
        }
        strMatrix.append("\n");
    }
    return strMatrix.toString().toUpperCase(); //optional To upper Case
}
```

This method generate a random char matrix in all 8 directions from randomly selected words and generate a matrix, following the same logic of the previously mentioned algorithm with a failed attempt to create a new pattern of readable words by creating a boolean[][] 2d matrix with the same size as char matrix that hold “true” for pre-occupied locations.

OOP principles:

- Encapsulation: where all values are hidden and can be access only by getters and setters.
- Single responsibility: where each class responsible for one object operations.

- ➔ Decoupling: where controller class dose not contain any logic beside mapping and binging objects and attributes

Classes Interelation:

- ➔ **matrixGenerator class**: contain the methods that generate matrix and search for words in the matrix
- ➔ **DatabaseAcess**: takes user info from PersonInfo class and store it to database
- ➔ **Controller**: map to HTML pages and bindes objects to attributes, and call required methods from other classes.

References:

1- Baeldung. (2020, January 13). Spring MVC and the @ModelAttribute Annotation. Retrieved October 28, 2020, from https://www.baeldung.com/spring-mvc-and-the-modelattribute-annotation

2- Baeldung. (2020, August 13). What is a POJO Class? Retrieved October 28, 2020, from https://www.baeldung.com/java-pojo-class

3 - Baeldung. (2020, October 21). Guide to Spring @Autowired. Retrieved October 28, 2020, from https://www.baeldung.com/spring-autowire

4-Oberle, C. (2020, March 21). Session Attributes in Spring MVC. Retrieved October 28, 2020, from https://www.baeldung.com/spring-mvc-session-attributes

5-Depth First Search or DFS for a Graph. (2020, September 30). Retrieved October 28, 2020, from https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

6- Search a Word in a 2D Grid of characters. (2020, June 25). Retrieved October 28, 2020, from https://www.geeksforgeeks.org/search-a-word-in-a-2d-grid-of-characters/

7- Terapalli, N. (2019, August 9). Check if a word exists in a grid or not | GeeksforGeeks. YouTube. https://www.youtube.com/watch?v=UDQ_FgNbArA&t=375s&ab_channel=GeeksforGeeks