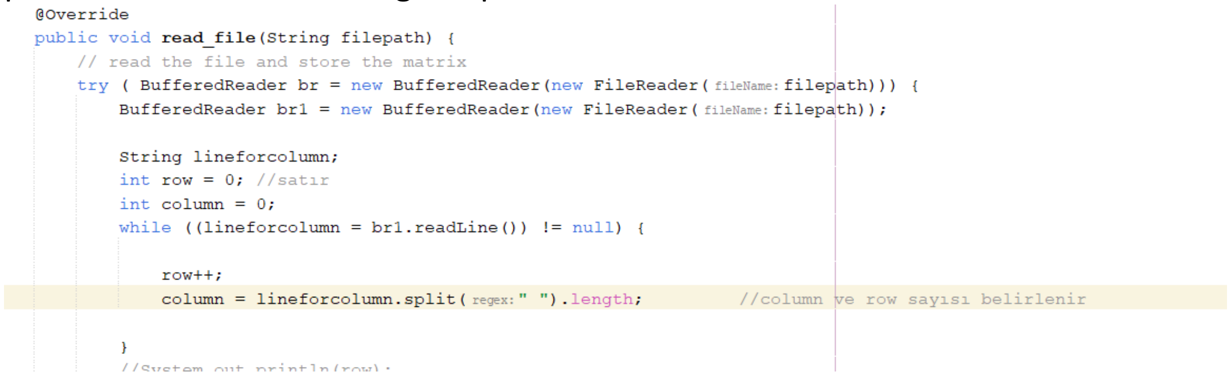
Introduction

First of all, I want to talk about my program. This program has 1 class and 1 interface. The concept of interface has an important place in Java. Methods defined on an interface are bodyless. The class that implements the interface must use the methods in the interface. In this program there is an interface named HW3Interface. There are methods that need to be written in this interface. There is a class called HW3. In this class the interface methods are overridden and used. In this class, the process of reading the shapes from the file, printing the number of shapes, extracting the shapes from the given matrix, and printing the shape with the number obtained to the console screen and to a new output file was realized.

Algorithm

public void read_file(String filepath):

I created a BufferedReader object called br to read the file. To store each line of the read file, I defined a String called line. Also, to keep track of the current row of the matrix, I defined an int called row and initialized it to 0. I created a new 2D array called matrix with 11 rows and 14 columns. It reads each line of the file using BufferedReader's readLine method. The loop continues until there are no more lines to read (readLine returns null). I created a string array called Values, which consists of the space character. (It splits the line into a string called values using the space character) It continues through the Values array and stores each value at the corresponding index of the matrix array. Then it increments the row variable. If an IOException is thrown while reading the file, this is caught and printed to the screen using the printStackTrace method.

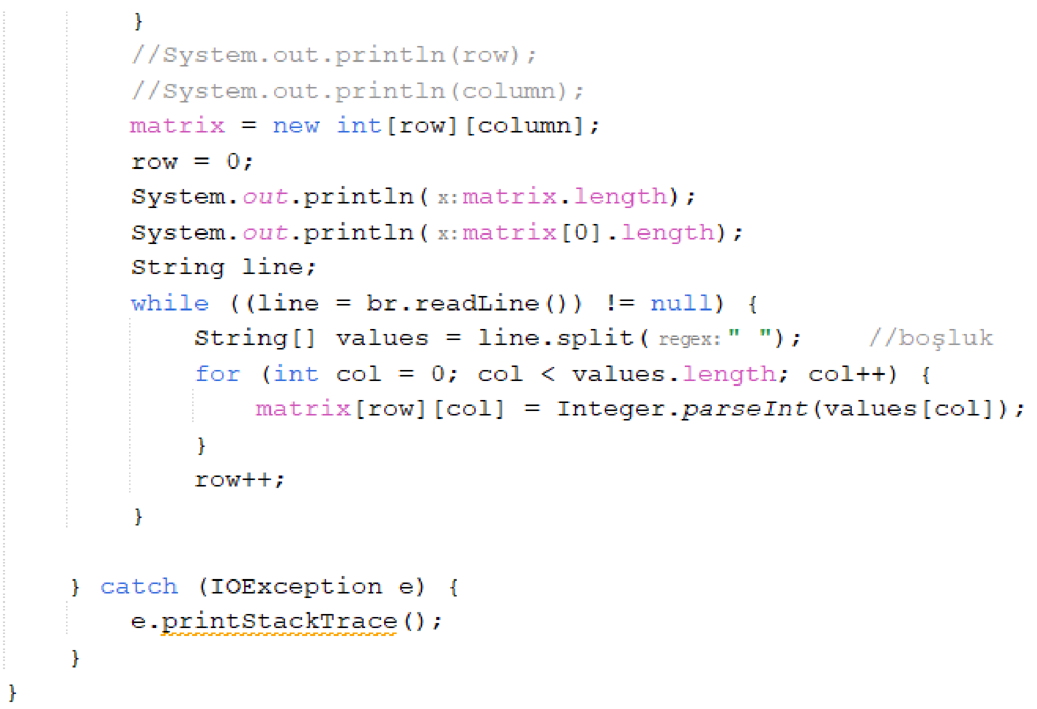


```
@Override
public void read_file(String filepath) {
    // read the file and store the matrix
    try ( BufferedReader br = new BufferedReader(new FileReader( fileName: filepath))) {
        BufferedReader br1 = new BufferedReader(new FileReader( fileName: filepath));

        String lineforcolumn;
        int row = 0; //satır
        int column = 0;
        while ((lineforcolumn = br1.readLine()) != null) {

            row++;
            column = lineforcolumn.split( regex: " ").length;         //column ve row sayısı belirlenir

        }
        //System.out.println(row);
```

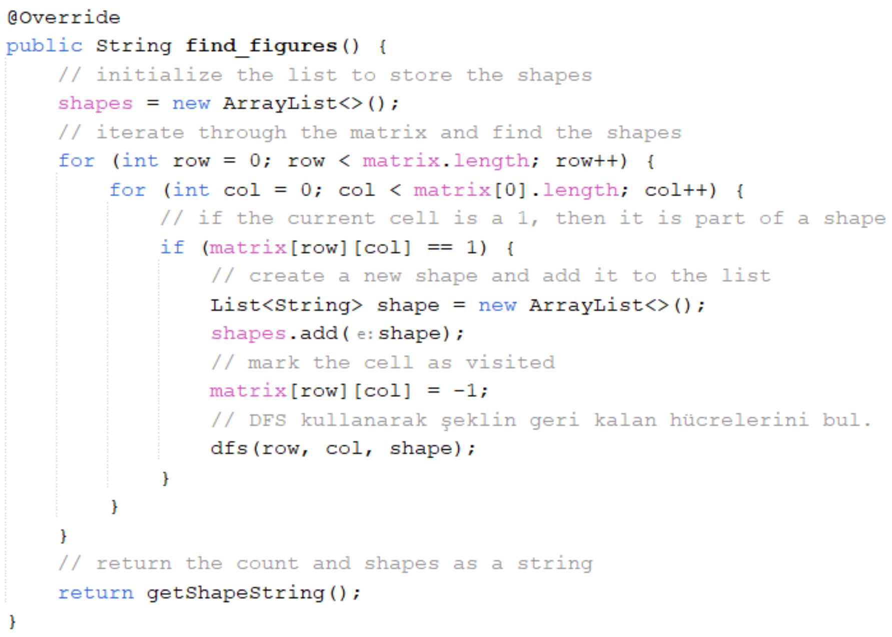


```
        }
        //System.out.println(row);
        //System.out.println(column);
        matrix = new int[row][column];
        row = 0;
        System.out.println( x: matrix.length);
        System.out.println( x: matrix[0].length);
        String line;
        while ((line = br.readLine()) != null) {
            String[] values = line.split( regex: " ");     //boşluk
            for (int col = 0; col < values.length; col++) {
                matrix[row][col] = Integer.parseInt(values[col]);
            }
            row++;
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

public String find_figures():

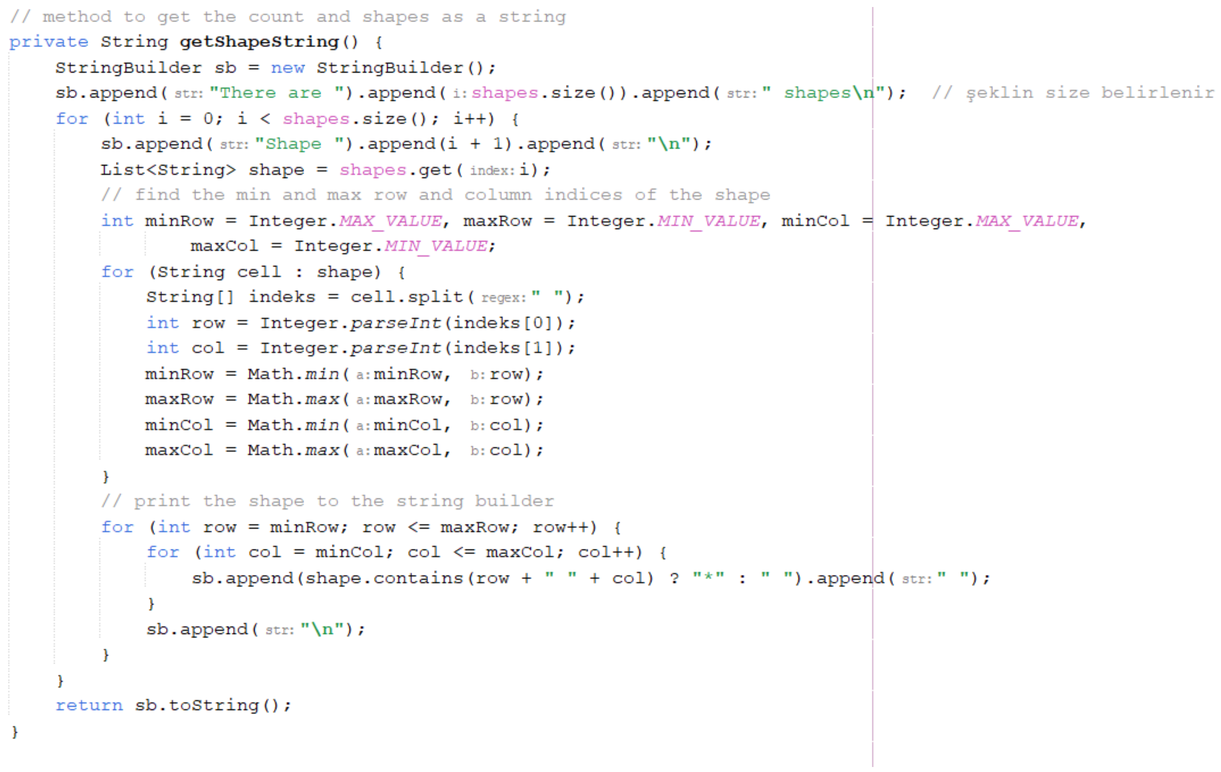
I created a list called shapes to store the shapes. I traversed through the matrix array using two nested loops. If the number in the index is 1, then it is part of a shape. To store the cells belonging to the current shape, I created a new list called shape, added it to the list of shapes, and marked the cells I traversed with -1 so that the program never visits them again. I called the dfs method to find the other cells in the shape using the DFS method. Once all the shapes were found, I called the getShapeString method to create a string representing the number and the shapes. This will return the array.



```
@Override
public String find_figures() {
    // initialize the list to store the shapes
    shapes = new ArrayList<>();
    // iterate through the matrix and find the shapes
    for (int row = 0; row < matrix.length; row++) {
        for (int col = 0; col < matrix[0].length; col++) {
            // if the current cell is a 1, then it is part of a shape
            if (matrix[row][col] == 1) {
                // create a new shape and add it to the list
                List<String> shape = new ArrayList<>();
                shapes.add( e: shape);
                // mark the cell as visited
                matrix[row][col] = -1;
                // DFS kullanarak şeklin geri kalan hücrelerini bul.
                dfs(row, col, shape);
            }
        }
    }
    // return the count and shapes as a string
    return getShapeString();
}
```

private String getShapeString():

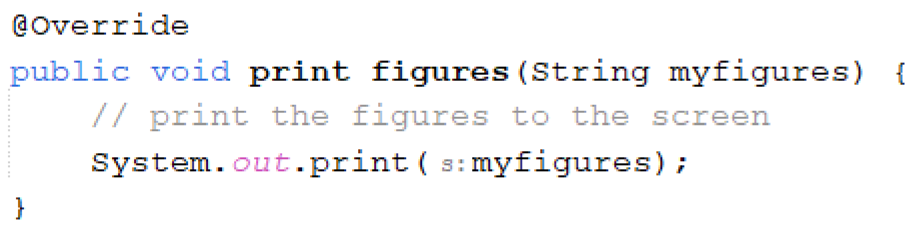
The getShapeString method is a helper function called by find_figures to create a string representing numbers and shapes. It iterates through the list of shapes and creates a string by adding the number and each shape to the string. First I created a new StringBuilder called sb to store the string. Then I added the number and shape of the shapes to the string sb. For each shape it adds the shape number and a newline character. It iterates through the cells in the shape and adds an asterisk character to the sb string. If the cell is the last cell in the row, it adds a newline character instead of a space. After all the shapes have been added to the string sb, it returns the string.
The append method is a method of the StringBuilder class in Java and allows us to add a string to the end of an existing string. The append method is useful when we need to build a string piece by piece, because it allows us to efficiently add new strings to the end of an existing string without creating a new string object each time.



```java
// method to get the count and shapes as a string
private String getShapeString() {
    StringBuilder sb = new StringBuilder();
    sb.append( str: "There are ").append( i: shapes.size()).append( str: " shapes\n");  // şeklin size belirlenir
    for (int i = 0; i < shapes.size(); i++) {
        sb.append( str: "Shape ").append(i + 1).append( str: "\n");
        List<String> shape = shapes.get( index: i);
        // find the min and max row and column indices of the shape
        int minRow = Integer.MAX_VALUE, maxRow = Integer.MIN_VALUE, minCol = Integer.MAX_VALUE,
                maxCol = Integer.MIN_VALUE;
        for (String cell : shape) {
            String[] indeks = cell.split( regex: " ");
            int row = Integer.parseInt(indeks[0]);
            int col = Integer.parseInt(indeks[1]);
            minRow = Math.min( a: minRow,  b: row);
            maxRow = Math.max( a: maxRow,  b: row);
            minCol = Math.min( a: minCol,  b: col);
            maxCol = Math.max( a: maxCol,  b: col);
        }
        // print the shape to the string builder
        for (int row = minRow; row <= maxRow; row++) {
            for (int col = minCol; col <= maxCol; col++) {
                sb.append(shape.contains(row + " " + col) ? "*" : " ").append( str: " ");
            }
            sb.append( str: "\n");
        }
    }
    return sb.toString();
}
```

public void print_figures(String myfigures):

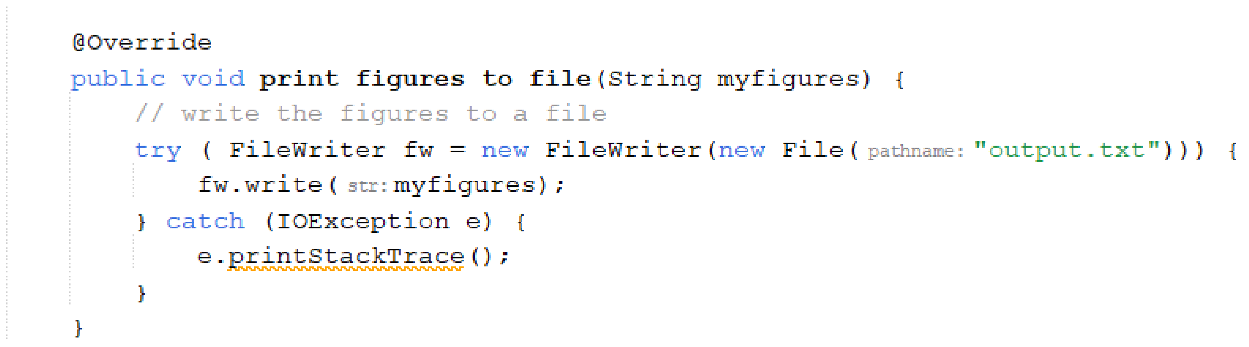
Prints our shape to console.



```java
@Override
public void print_figures(String myfigures) {
    // print the figures to the screen
    System.out.print( s: myfigures);
}
```

public void print_figures_to_file(String myfigures):

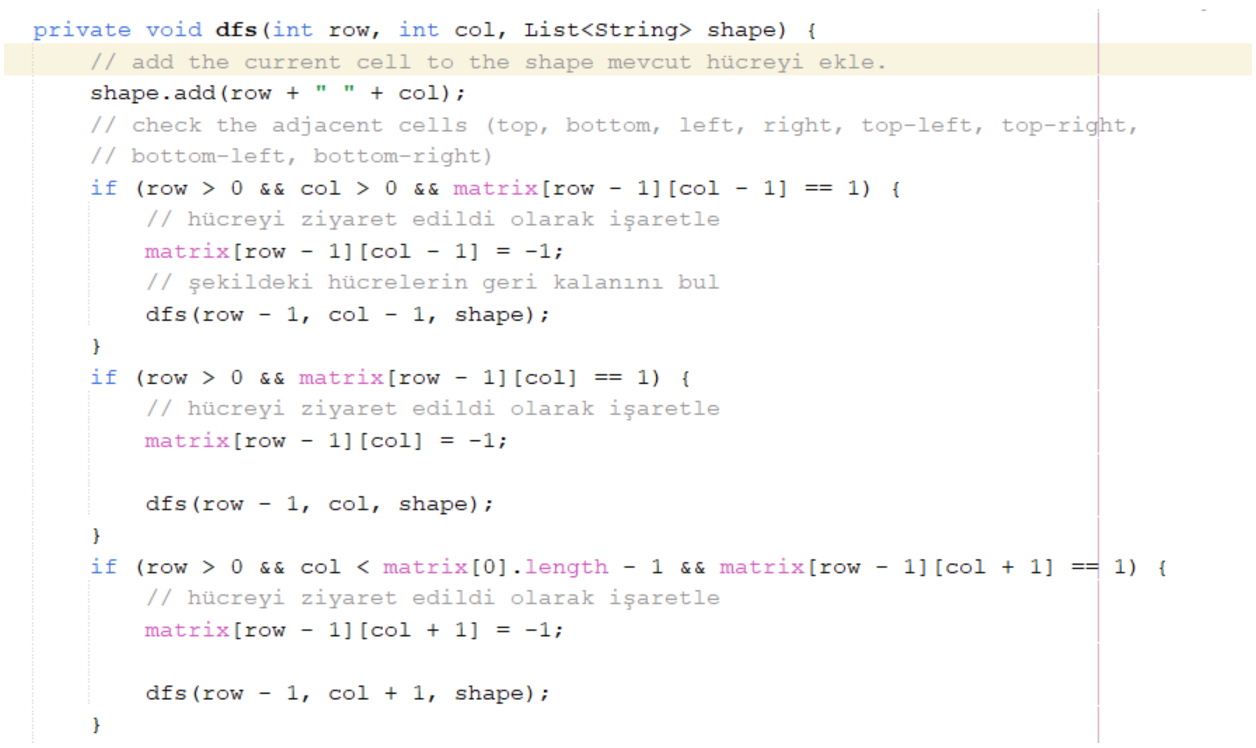
The print_figures_to_file method, which takes a string as an argument and writes it to a file. In this method, I created a FileWriter object named fw to write to a file named "output.txt". The write method of the fw object is then called to write the string passed as an argument to the file.



```java
@Override
public void print_figures_to_file(String myfigures) {
    // write the figures to a file
    try ( FileWriter fw = new FileWriter(new File( pathname: "output.txt"))) {
        fw.write( str: myfigures);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

private void dfs(int row, int col, List<String> shape) :

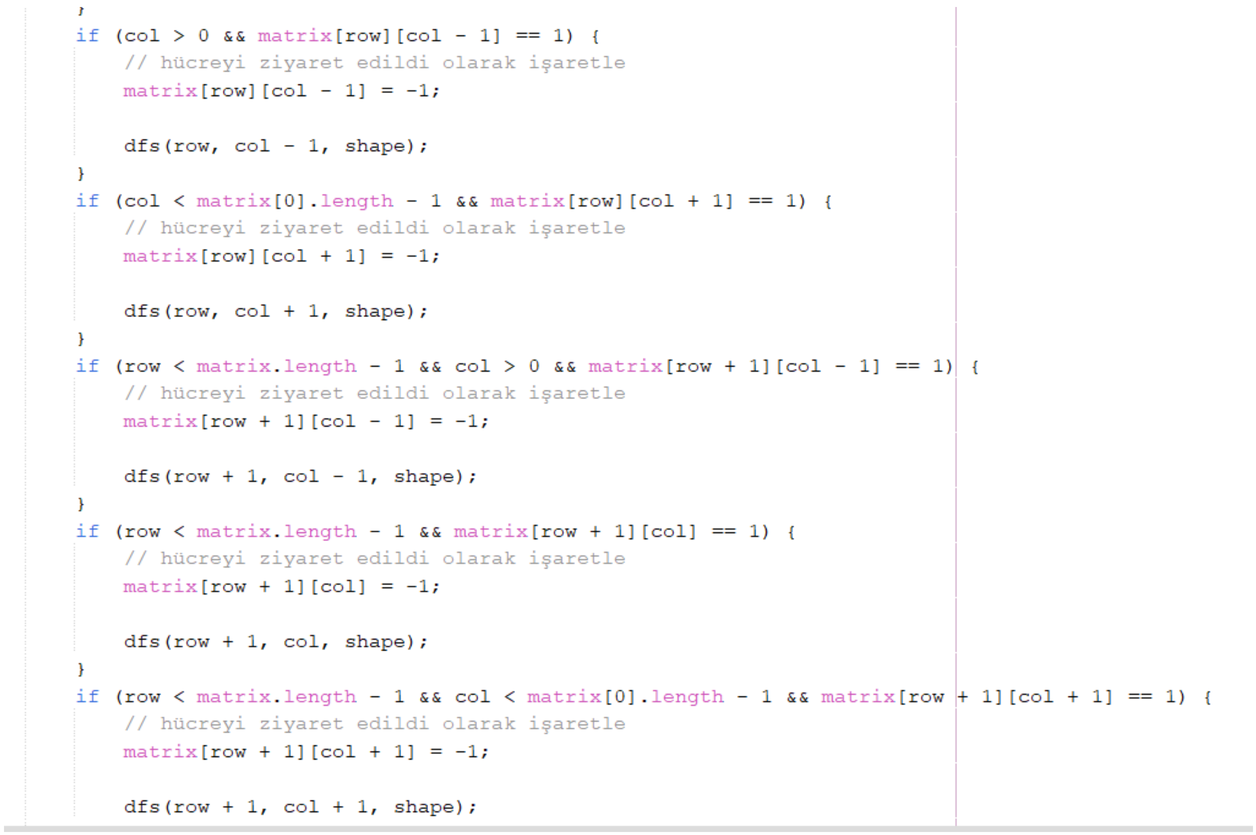
The method takes three arguments. The row and column indices of the starting cell and a list of strings to store the cells that are part of the shape.It adds the current cell to the shape by adding its row and column indices to the list of strings. Checks the neighboring cells of the current cell to see if they are part of the shape. If one of the adjacent cells is 1, it means it is part of the shape, so the method marks it as visited by setting its value to -1, and then calls itself recursively to find the other cells in the shape. This process continues until all cells in the shape have been found.



```
private void dfs(int row, int col, List<String> shape) {
    // add the current cell to the shape mevcut hücreyi ekle.
    shape.add(row + " " + col);
    // check the adjacent cells (top, bottom, left, right, top-left, top-right,
    // bottom-left, bottom-right)
    if (row > 0 && col > 0 && matrix[row - 1][col - 1] == 1) {
        // hücreyi ziyaret edildi olarak işaretle
        matrix[row - 1][col - 1] = -1;
        // şekildeki hücrelerin geri kalanını bul
        dfs(row - 1, col - 1, shape);
    }
    if (row > 0 && matrix[row - 1][col] == 1) {
        // hücreyi ziyaret edildi olarak işaretle
        matrix[row - 1][col] = -1;

        dfs(row - 1, col, shape);
    }
    if (row > 0 && col < matrix[0].length - 1 && matrix[row - 1][col + 1] == 1) {
        // hücreyi ziyaret edildi olarak işaretle
        matrix[row - 1][col + 1] = -1;

        dfs(row - 1, col + 1, shape);
    }
```



```
    if (col > 0 && matrix[row][col - 1] == 1) {
        // hücreyi ziyaret edildi olarak işaretle
        matrix[row][col - 1] = -1;

        dfs(row, col - 1, shape);
    }
    if (col < matrix[0].length - 1 && matrix[row][col + 1] == 1) {
        // hücreyi ziyaret edildi olarak işaretle
        matrix[row][col + 1] = -1;

        dfs(row, col + 1, shape);
    }
    if (row < matrix.length - 1 && col > 0 && matrix[row + 1][col - 1] == 1) {
        // hücreyi ziyaret edildi olarak işaretle
        matrix[row + 1][col - 1] = -1;

        dfs(row + 1, col - 1, shape);
    }
    if (row < matrix.length - 1 && matrix[row + 1][col] == 1) {
        // hücreyi ziyaret edildi olarak işaretle
        matrix[row + 1][col] = -1;

        dfs(row + 1, col, shape);
    }
    if (row < matrix.length - 1 && col < matrix[0].length - 1 && matrix[row + 1][col + 1] == 1) {
        // hücreyi ziyaret edildi olarak işaretle
        matrix[row + 1][col + 1] = -1;

        dfs(row + 1, col + 1, shape);
```