Gebze Technical University
Computer Engineering

CSE 222 - 2019 Spring

HOMEWORK 3 REPORT

MUHAMMET BURAK ÖZÇELİK
151044050

Course Assistant: Özgü GÖKSU

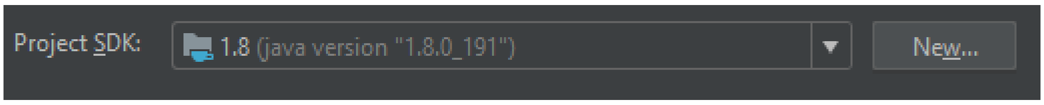
1 INTRODUCTION

1.1 Problem Definition

In the 1st part of the given assignment, we were asked to find the components in the text file given to us and to find out how many of them were. In this file, the components are specified by 1, and the components in the neighboring points are named with the same name.

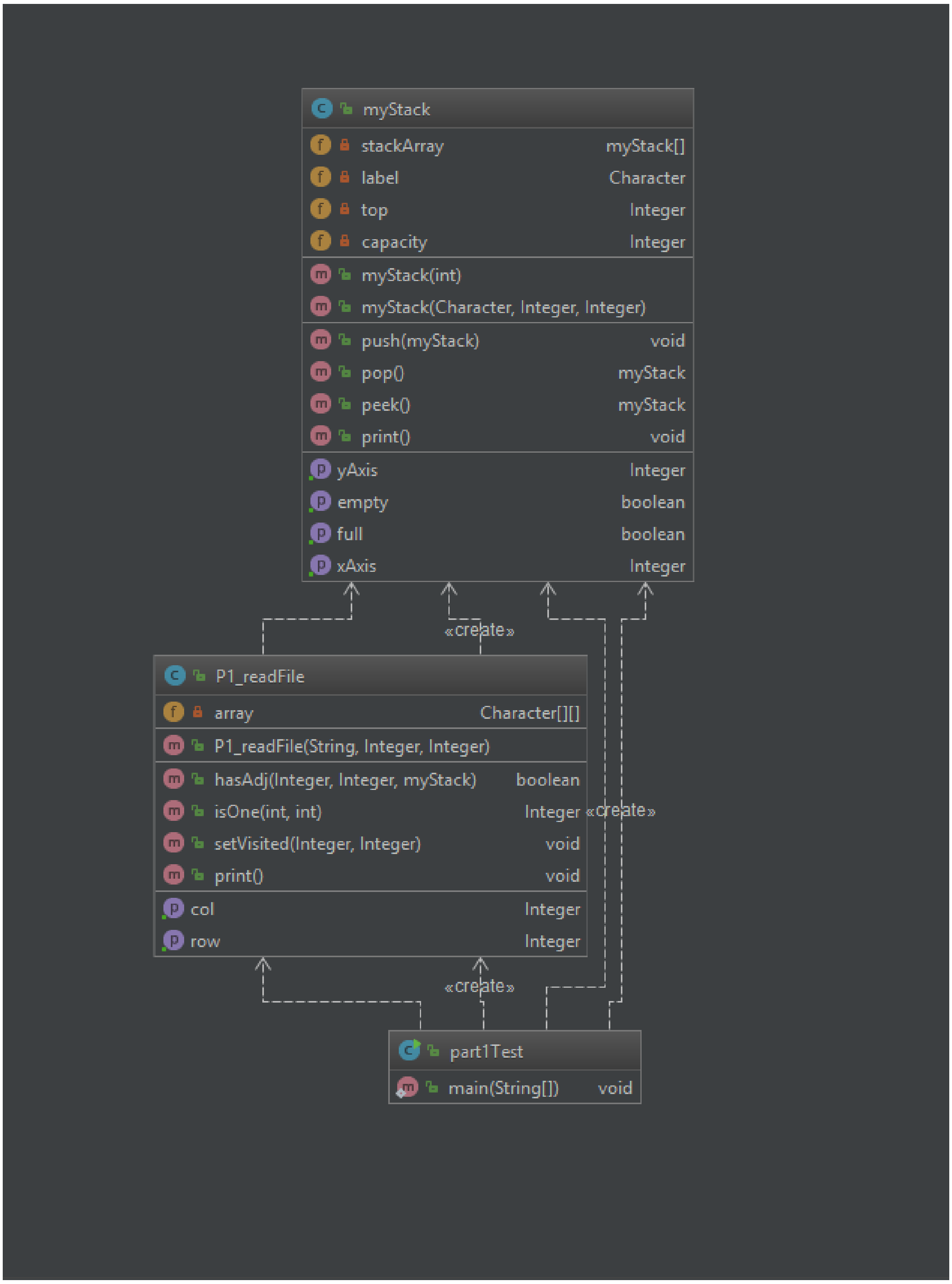
1.2 System Requirements

As a system requirement, we will use the stack structure when doing this. We will access and check our components through reading from the file. Components are shown with ‘1’ and empty cells are shown with ‘0’ and all compenets are seperated with whitespace.
In general, we have 2 classes written for the solution. One of these is the P1_readFile class, which we will solve using the myStack class and the other using the myStack class.
Program will run on PC with IntelliJ and on Java 8 version.

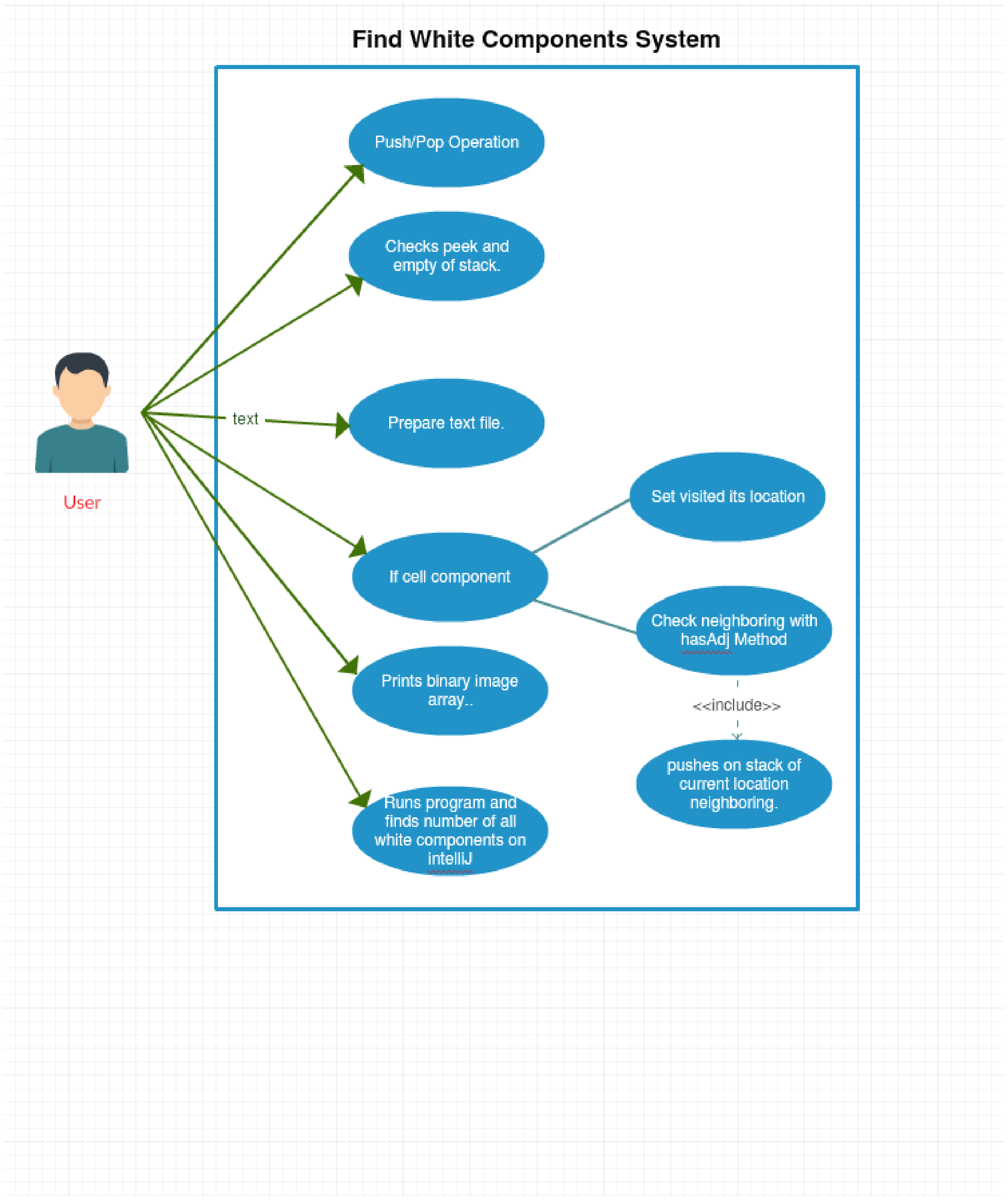

Project SDK: 1.8 (java version "1.8.0_191")

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagrams


Find White Components System
Push/Pop Operation
Checks peek and empty of stack.
text
Prepare text file.
Set visited its location
If cell component
Check neighboring with hasAdj Method
<<include>>
pushes on stack of current location neighboring.
Prints binary image array..
Runs program and finds number of all white components on intelliJ
User

Since it is a single user as user, it will be able to access all public methods. In simple terms, the use - case diagram is shown above. The user will be able to prepare the text file and obtain the number of components in the file.

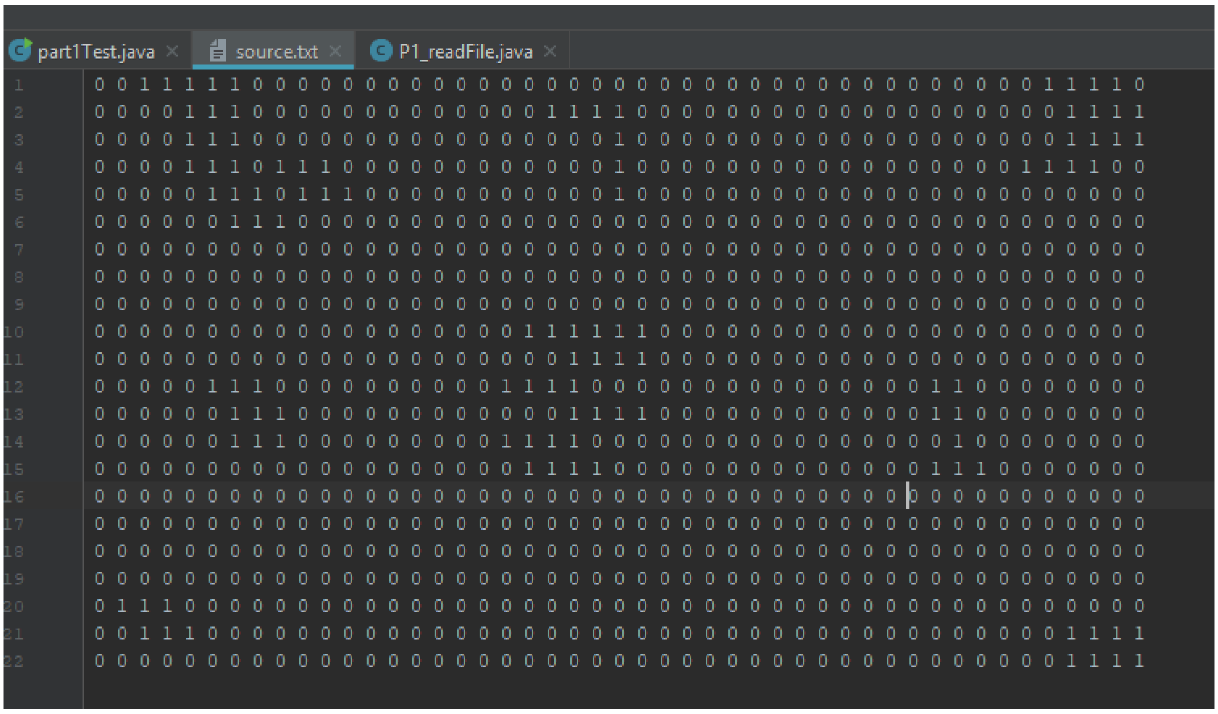
2.3 Problem Solution Approach

First of all, I learned the number of lines and columns of the file I read with the file reading process and I sent it to the P1_readFile constructor and created the object that I will do the operation. There is a two-dimensional character array in this object. I've made this array out of the Whitespace we read from the file. Then I created a stack object by giving the rows and columns of the array we have edited. I visited all the elements on this array, when the component saw its neighbor, if any, push all the neighbors to the stack, by editing the current position with the SetVisited method I reached the number of all the components. Finally I printed edited array and number of White Components. My program complexity is worstcase is O(n^m), avarage case is O(n*m) and best case is O(1).

3 RESULT

3.1 Test Cases

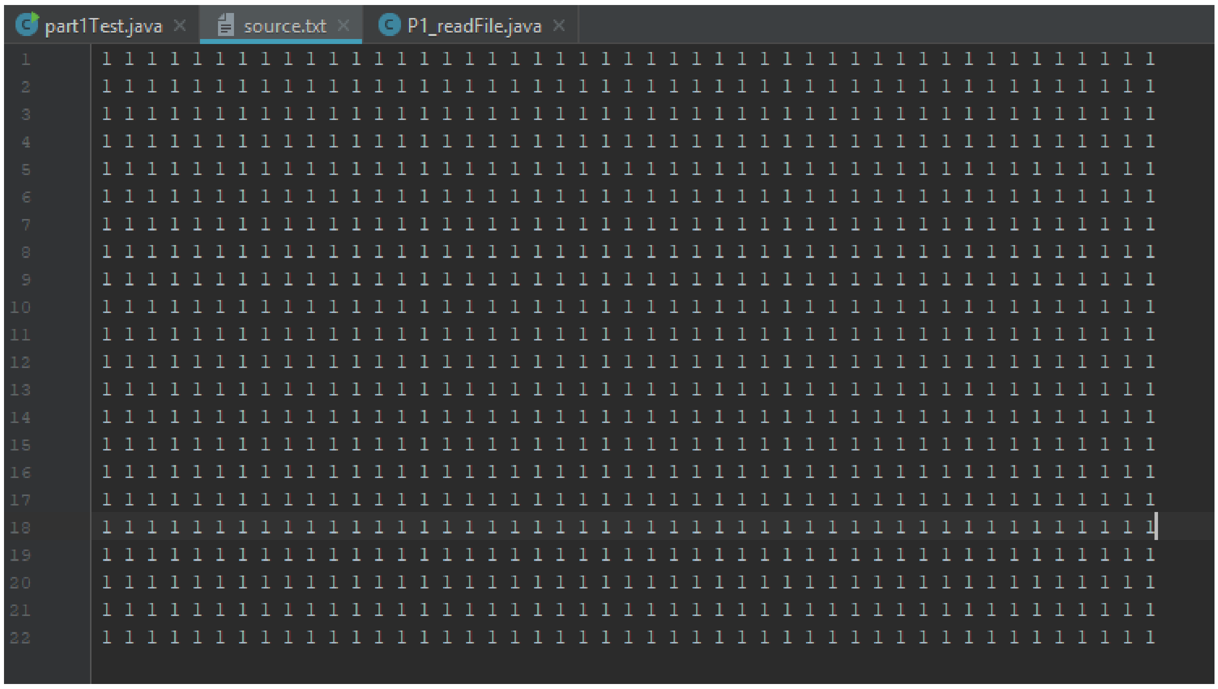
- Main Test
- First I tested it with test file on moodle.



```
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0
0 0 0 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
```

Figure 1.

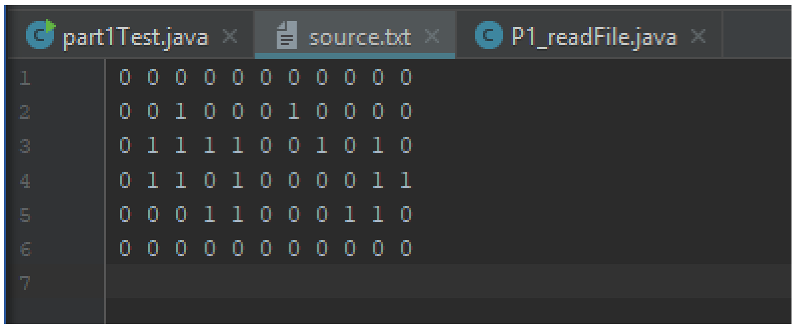
- I tested it with test file on moodle but I edited all elements by 1.



```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Figure 2

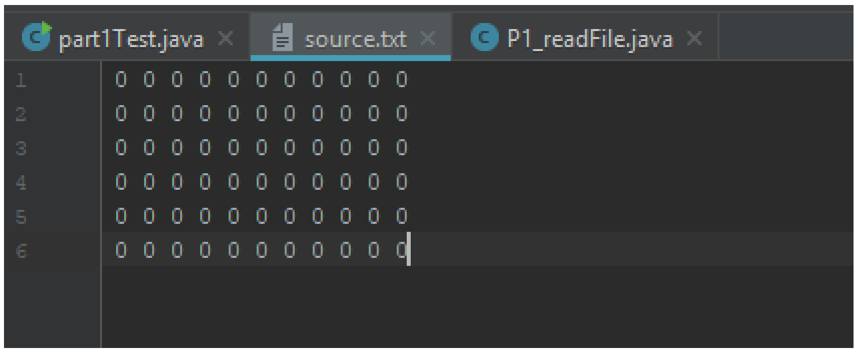
- I tested it with on pdf example.



```
0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0 0 0 0
0 1 1 1 1 0 0 1 0 1 0
0 1 1 0 1 0 0 0 0 1 1
0 0 0 1 1 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0 0 0
```

Figure 3

- I tested it with on pdf example but I edited all components with ‘0’.



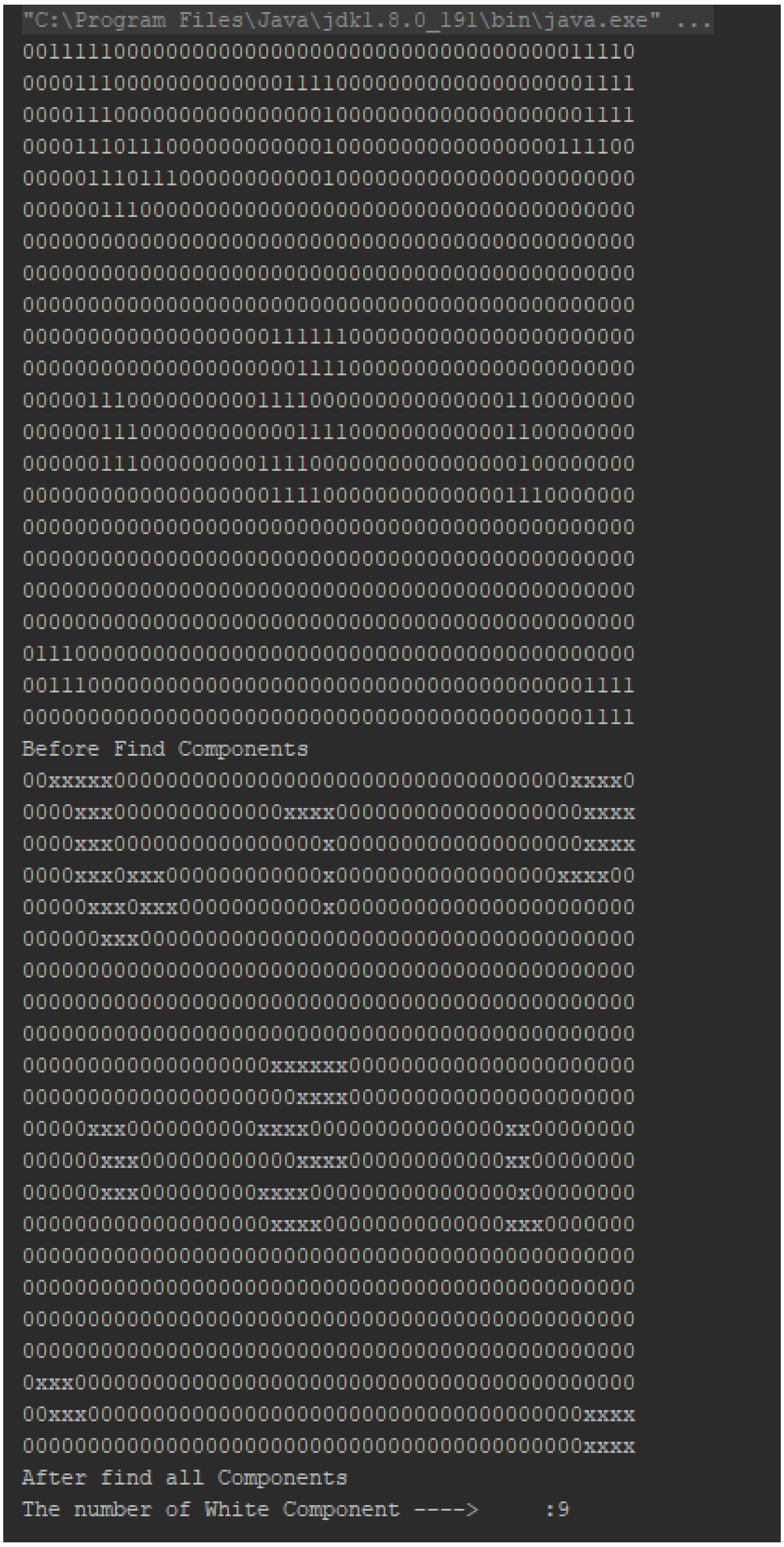
```
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
```

Figure 4

All these cases have been tested in MainTest.

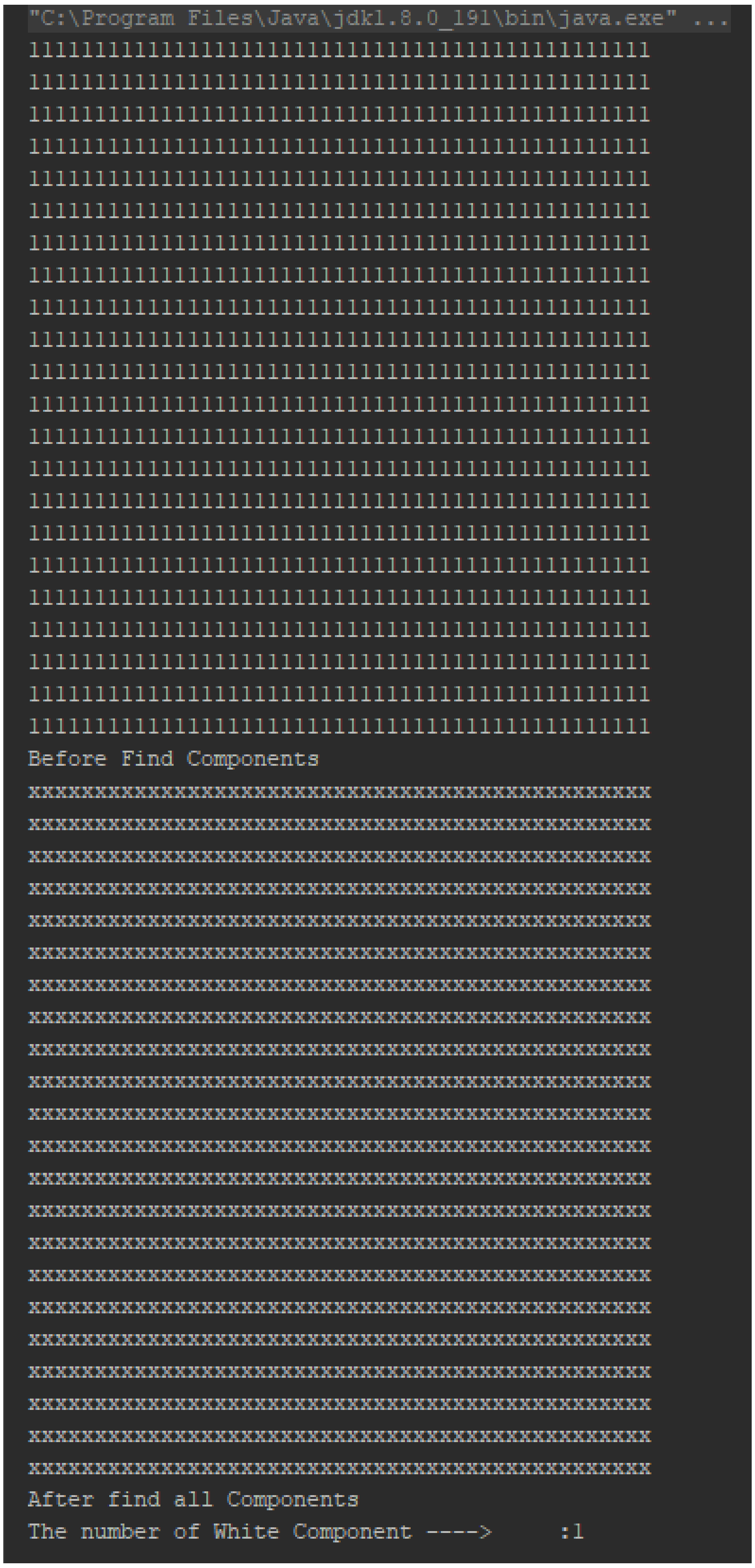
3.2 Running Results

- **Figure 1 Results**



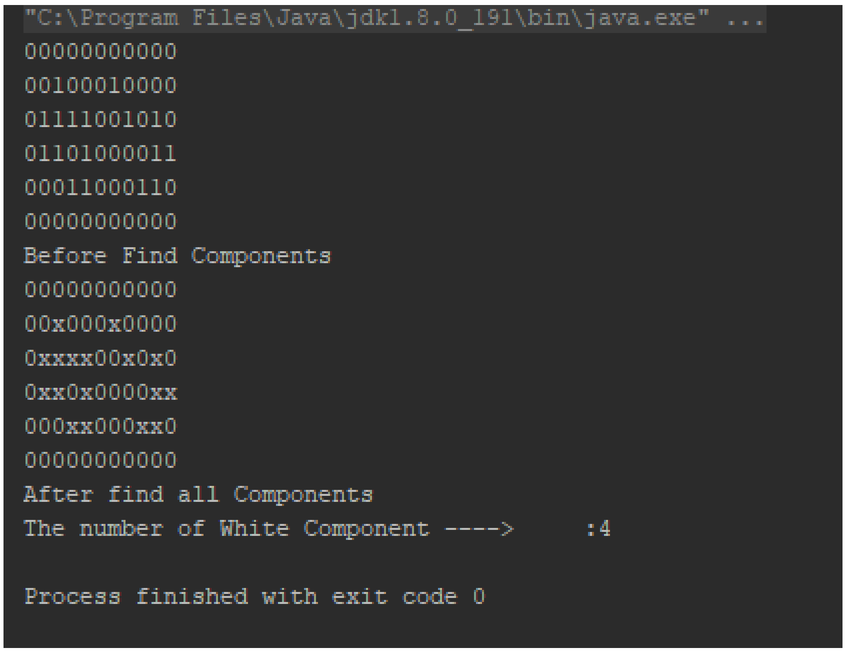
```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
00111111000000000000000000000000000000000011110
00001110000000000000111100000000000000000001111
00001110000000000000000100000000000000000001111
00001110111000000000000100000000000000000111100
00000111011100000000000100000000000000000000000
00000011100000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
00000000000000000001111110000000000000000000000
00000000000000000000011110000000000000000000000
00000111000000000011110000000000000110000000000
00000011100000000000011110000000000110000000000
00000011100000000011110000000000000010000000000
00000000000000000001111000000000000011100000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
01110000000000000000000000000000000000000000000
00111000000000000000000000000000000000000001111
00000000000000000000000000000000000000000001111
Before Find Components
00xxxxxx0000000000000000000000000000000000xxxx0
0000xxx0000000000000xxxx0000000000000000000xxxx
0000xxx0000000000000000x0000000000000000000xxxx
0000xxx0xxx000000000000x00000000000000000xxxx00
00000xxx0xxx00000000000x00000000000000000000000
000000xxx00000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
0000000000000000000xxxxxx0000000000000000000000
000000000000000000000xxxx0000000000000000000000
00000xxx0000000000xxxx000000000000000xx00000000
000000xxx000000000000xxxx000000000000xx00000000
000000xxx000000000xxxx0000000000000000x00000000
0000000000000000000xxxx00000000000000xxx0000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
0xxx0000000000000000000000000000000000000000000
00xxx00000000000000000000000000000000000000xxxx
0000000000000000000000000000000000000000000xxxx
After find all Components
The number of White Component ---->     :9
```

- **Figure 2 Results**



```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
Before Find Components
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
After find all Components
The number of White Component ---->     :1
```

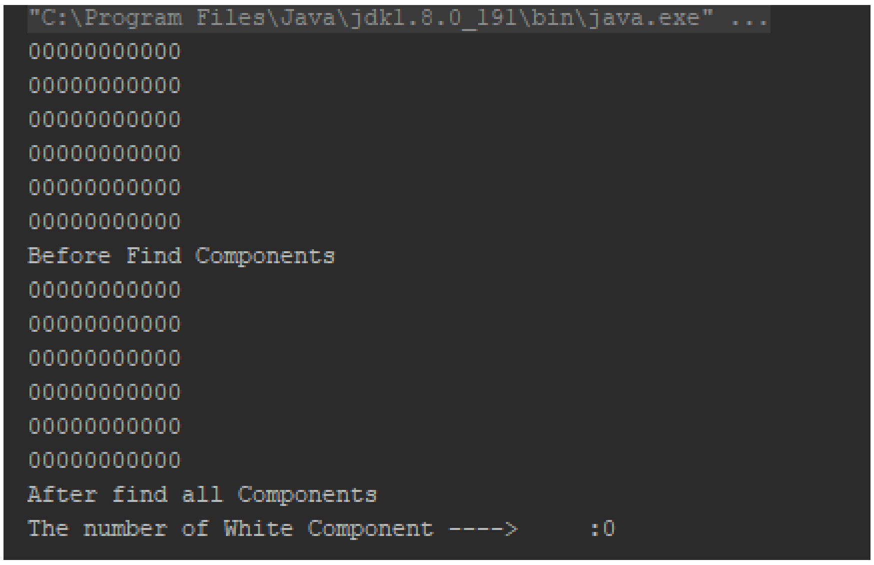
- **Figure 3 Results**



```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
00000000000
00100010000
01111001010
01101000011
00011000110
00000000000
Before Find Components
00000000000
00x000x0000
0xxxx00x0x0
0xx0x0000xx
000xx000xx0
00000000000
After find all Components
The number of White Component ---->     :4

Process finished with exit code 0
```

- **Figure 4 Results**

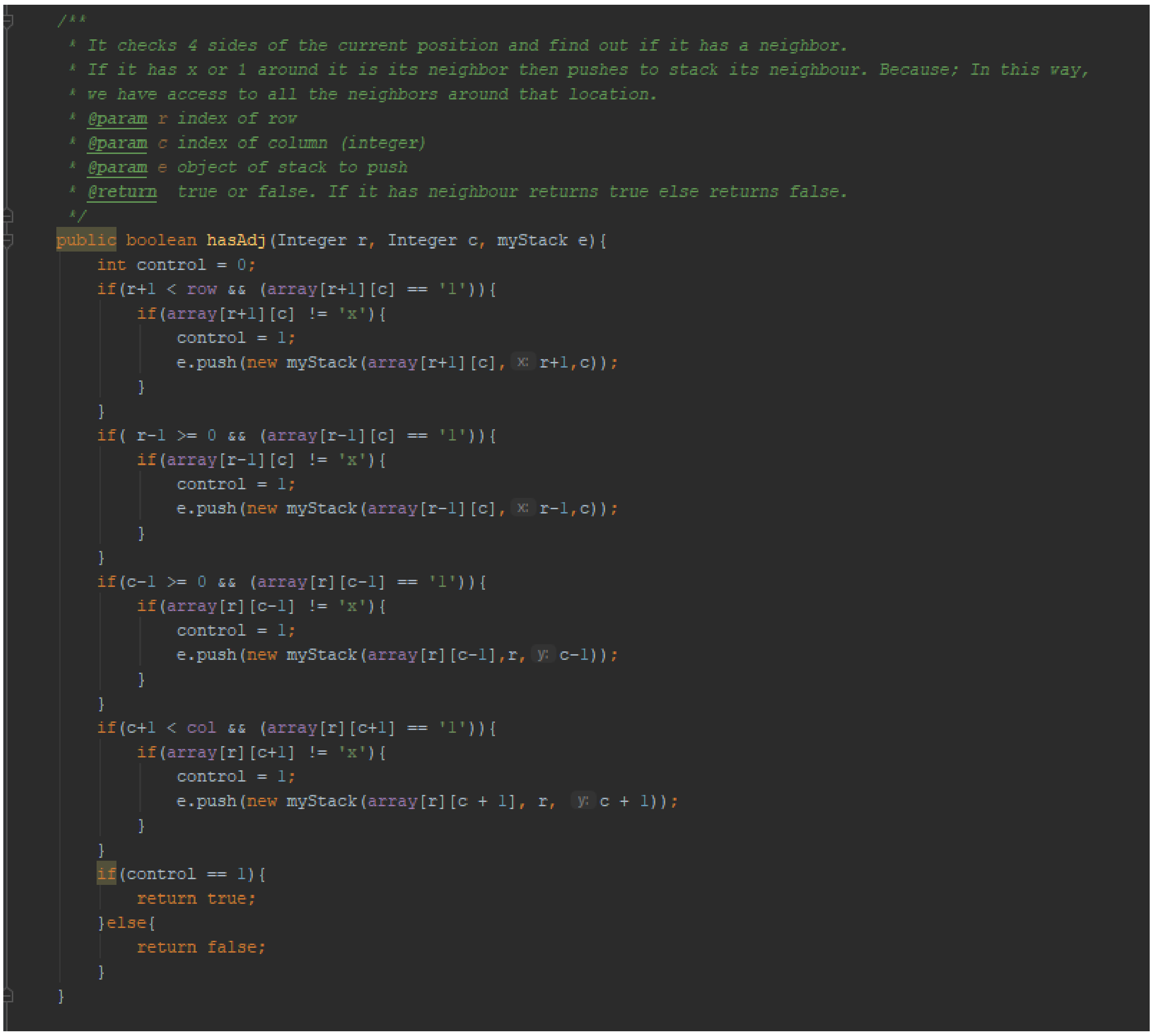


```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
00000000000
00000000000
00000000000
00000000000
00000000000
00000000000
Before Find Components
00000000000
00000000000
00000000000
00000000000
00000000000
00000000000
After find all Components
The number of White Component ---->     :0
```

3.3 Algorithm Analysis

All operations are performed and shown in the Main test scenario. However, the sample outputs and algorithm analysis of the methods are given below.

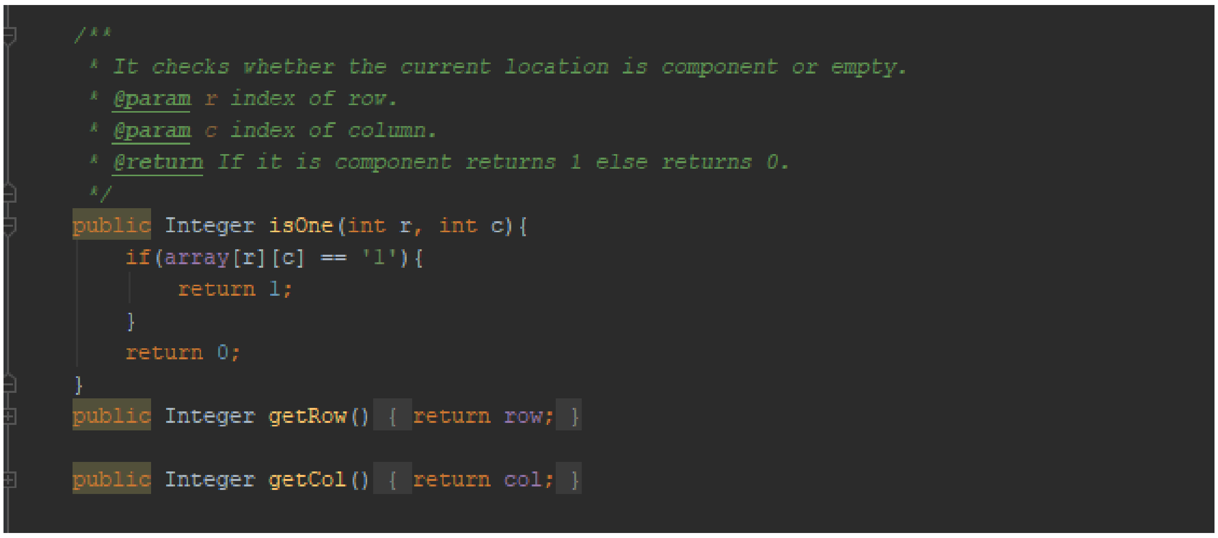
- **hasAdj() Method**



```
/**
 * It checks 4 sides of the current position and find out if it has a neighbor.
 * If it has x or 1 around it is its neighbor then pushes to stack its neighbour. Because; In this way,
 * we have access to all the neighbors around that location.
 * @param r index of row
 * @param c index of column (integer)
 * @param e object of stack to push
 * @return  true or false. If it has neighbour returns true else returns false.
 */
public boolean hasAdj(Integer r, Integer c, myStack e){
    int control = 0;
    if(r+1 < row && (array[r+1][c] == '1')){
        if(array[r+1][c] != 'x'){
            control = 1;
            e.push(new myStack(array[r+1][c], x: r+1,c));
        }
    }
    if( r-1 >= 0 && (array[r-1][c] == '1')){
        if(array[r-1][c] != 'x'){
            control = 1;
            e.push(new myStack(array[r-1][c], x: r-1,c));
        }
    }
    if(c-1 >= 0 && (array[r][c-1] == '1')){
        if(array[r][c-1] != 'x'){
            control = 1;
            e.push(new myStack(array[r][c-1],r, y: c-1));
        }
    }
    if(c+1 < col && (array[r][c+1] == '1')){
        if(array[r][c+1] != 'x'){
            control = 1;
            e.push(new myStack(array[r][c + 1], r,  y: c + 1));
        }
    }
    if(control == 1){
        return true;
    }else{
        return false;
    }
}
```

Algorithm Analysis: It is a method for pushing them into the stack only if there is a component by looking at the right, left, bottom and top positions. Since it has direct access to its neighbors, it has a constant complexity since it has direct access to the constant and push method. That is to say the complexity of the hasAdj method is Θ(1).

- **getRow(), isOne() and getCol() Methods**

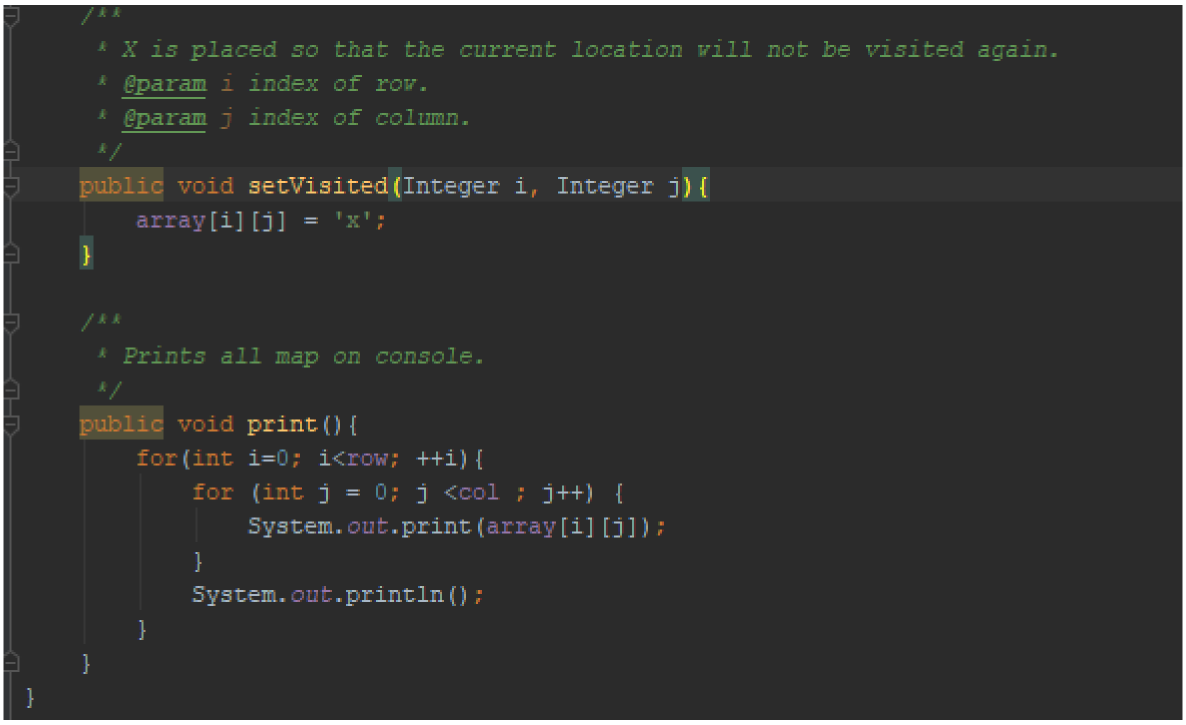


```
/**
 * It checks whether the current location is component or empty.
 * @param r index of row.
 * @param c index of column.
 * @return If it is component returns 1 else returns 0.
 */
public Integer isOne(int r, int c){
    if(array[r][c] == '1'){
        return 1;
    }
    return 0;
}
public Integer getRow() { return row; }

public Integer getCol() { return col; }
```

Algorithm Analysis: All these method's complexity is Θ(1).

- **setVisited() and print() Methods**



```
    /**
     * X is placed so that the current location will not be visited again.
     * @param i index of row.
     * @param j index of column.
     */
    public void setVisited(Integer i, Integer j){
        array[i][j] = 'x';
    }

    /**
     * Prints all map on console.
     */
    public void print(){
        for(int i=0; i<row; ++i){
            for (int j = 0; j <col ; j++) {
                System.out.print(array[i][j]);
            }
            System.out.println();
        }
    }
}
```

Algorithm Analysis: print() method's complexity Θ(n) because it visits all elements and prints on the console. setVisited() method's complexity is Θ(1) because it set direct index of array and replace 'x' that current location.