PROBLEM DESCRIPTION

There is a **16×16** board. Each cell of the board has a color of **white** or **gray**. In every cell, the initial letter of its color is written. It is possible to change the color of the cells.

We start at the **uppermost and leftmost** cell of the board. If possible, we change its color and jump to the **right** cell. We do so until we reach the end of the row. After that, we jump to the cell that lies in the **next row and leftmost column**. We do so until we reach the **bottommost and rightmost** cell. After that, we jump to again **uppermost and leftmost** cell and repeat same things until it is not possible to change the color of any cells.

Rules:

1. Everything outside the board is **Orange**.
2. Two cells are connected if they are neighbors in **up-down** or **left-right** direction.
3. There are 4 Dark Colors (DC): Gray (G), Black (B), Purple (P) and Chocolate (C)
4. There are 4 Light Colors (LC): White (W), Orange (O), Yellow (Y) and Light Blue (L)
5. If a cell is DC and is connected to **4 DC**, it becomes B.
6. If a cell is DC and is connected to **3 LC** or **2 LC and at least 1 P** or **1 LC and at least 2 P**, it becomes P.
7. If a cell is DC and is **neither B nor P**, it becomes C.
8. If a cell is W and is connected to **at least 1 O**, it becomes O.
9. If a cell is O and is connected to **at least 2 Y and at most 1 O** or **1 Y and at least 2 DC** or **at least 2 DC and at least 1 O**, it becomes Y.
10. If a cell is W and **it is not possible to change for any cells** of the board, it becomes L.

The order in the list of rules is important. For example, think of a C that is connected to 4 other C. According to **rule#5**, it should become B and according to **rule#7** it should become C (*stay unchanged*). In this case, the rule which has the smaller number (*in this case, 3*) holds and C becomes B.

I need to write a program which gets a board configuration from a file (*called input.txt*). The board is written in the file in **16×16** format. My program should print the **initial** board configuration on the screen with the statistics of the colors of the cells and then process it until it is not possible to change for any cells of the board. And finally it should print on the screen **final** board configuration with **final** statistics.

PROBLEM SOLUTION

We can write such a program with the help of **Two-Dimensional Arrays**, **IF STATEMENT**s and **FOR LOOP**s.

- ➔ I import **java.io** package for **File** class.
- ➔ I import **java.util** package for **Scanner** class.

In my `main()` method,

- ➔ I declare a **Two-Dimensional Array** of char named **cell**. It represents an **18×18** board and every index of **cell[][]** represents a cell. Actually, the board in the problem is **16×16** but since everything outside the board is orange, I create an **18×18** board and make every cell that lies in the **first** and **last** rows and in the **first** and **last** columns of that board **O**. As a result, there is a **16×16** board in an orange frame. By doing so, it gets easier to process the cells that lies in the edges of the **16×16** board.
- ➔ I define a **FOR LOOP**. It makes every cell that lies in the **first** and **last** rows and in the **first** and **last** columns of **18×18** board **O**.
- ➔ I declare a **Scanner** named **file** in order to read the input file (*input.txt*).
- ➔ I define nested **FOR LOOP** which copies the characters in the `file` to to `cell[][]`. In each loop, it gets a **line** from the file (*a row of the board*), make it **uppercase** and assign every character in that line to an element of `cell[][]`.
- ➔ I use `printBoard()` method to print the **initial** board configuration and **initial** statistics on the screen. I pass '**i**' as a parameter to tell the method that it is the **initial** board.
- ➔ I declare a **boolean** named **cont** (*continuous*) with an **initial** value of **true**. It is the **test** of the **WHILE LOOP** below.

- ➔ I define a **WHILE LOOP** which runs and processes every cell of the board until it is not possible for any cells to change. In the **WHILE LOOP**,
  - ➔ I change the value of `cont` to **false**.
  - ➔ I declare a **Two-Dimensional Array** named **temp**. I first copy `cell[][]` into `temp[][]` and after `cell[][]` gets processed, I compare them to see whether the board has

changed or not.

- ➔ I define nested **FOR LOOP** which copies `cell[][]` into `temp[][]` **index-to-index**. (I *mean, cell[3][5] = temp[3][5], cell[12][13] = temp[12][13] and so on.*)
- ➔ I define another nested **FOR LOOP**. It specifies every cell of the board and with the help of some **IF STATEMENT**s and **methods**, it learns the color of the cells and sends them to the right method for being processed.
- ➔ I define another nested **FOR LOOP**. With the help of an **IF STATEMENT**, it compares every elements of `temp[][]` with the elements of updated `cell[][]` index-to-index to understand whether the board has changed or not. It the board has changed, it makes the value of `cont` true, so that the WHILE LOOP can work again.

➔ In `main()` method, I define nested **FOR LOOP** with an **IF STATEMENT.** They change the color of **White (W)** cells to **Light Blue (L)** after all the processes are completed at the board.

➔ I use `printBoard()` method to print the **final** board configuration and **final** statistics on the screen. I pass '**f**' as a parameter to tell the method that it is the **final** board.

➔ After `main()` method, I define a method named **darkColor** which processes the **Dark Color** cells. It takes a **Two-Dimensional Array** and 2 **int** as parameters. The parameters are **cell[][]**, **row** and **clmn** (*column*). I declare 3 **int** at the top of the method, they represent the number of **Dark Color** / **Light Color** / **Purple** cells among 4 neighbor cells of **cell[row][clmn]**. This method examines the upper neighbor of **cell[row][clmn]** and if it is **Dark Color** / **Light Color** / **Purple**, the method increases **numDC** / **numLC** / **numP** by **1**. It does the **same** things for 3 other neighbors of **cell[row][clmn]**. Finally, according to some rules, the method changes the color of **cell[row][clmn]** or leaves it unchanged.

➔ After `darkColor()` method, I define a method named **white** which processes the **White** cells. It takes a **Two-Dimensional Array** and 2 **int** as parameters. The parameters are **cell[][]**, **row** and **clmn** (*column*). The methods examines 4 neighbors of **cell[row][clmn]** to see whether they are **Orange** or not. If **at least one** of the 4 neighbors is **Orange**, the method changes the color of **cell[row][clmn]** from **White** to **Orange**.

➔ After white() method, I define a method named **orange** which processes the **Orange** cells. It takes a **Two-Dimensional Array** and 2 **int** as parameters. The parameters are **cell[][]**, **row** and **clmn** (*column*). I declare 3 **int** at the top of the method, they represent the number of **Yellow** / **Orange** / **Dark Color** cells among 4 neighbor cells of **cell[row][clmn]**. This method examines the upper neighbor of **cell[row][clmn]** and if it is **Dark Color** / **Yellow** / **Orange**, the method increases **numDC** / **numY** / **numO** by **1**. It does the **same** things for 3 other neighbors of **cell[row][clmn]**. Finally, according to some rules, the method changes the color of **cell[row][clmn]** or leaves it unchanged.

➔ After orange() method, I define a method named **isDarkColor** which tells us whether a cell is **Dark Color** or not.. It takes a **Two-Dimensional Array** and 2 **int** as parameters. The parameters are **cell[][]**, **row** and **clmn** (*column*). If **cell[row][clmn]** is **Dark Color** (*Gray, Black, Purple or Chocolate*), the method returns **true**; if not returns **false.**

➔ After isDarkColor() method, I define a method named **printBoard** which prints on the screen the board configuration and the statistics of the colors of the cells. It takes a **Two-Dimensional Array** and a **char** as parameters. The parameters are **cell[][]** and **board**. The parameter **board** helps the method to understand if it prints the **initial** board statistics or the **final** board statistics.

- ➔ I define nested **FOR LOOP** which prints on the screen the board configuration in **16×16** format.
- ➔ When the **board** is '**i**', the method prints the **initial** statistics. Since initially there are **only White and Gray** cells, I declare two **int** which represent the number of **White** cells and the number of **Gray** cells. I define nested **FOR LOOP** and **IF-ELSE STATEMENT**s to look at every cell of the board and **increase w by 1 if the cell is White and increase g by 1 if the cell is Gray**. Finally, I print on the screen the number of **White** cells, the number of **Gray** cells and the **total** number of the cells.
- ➔ When the **board** is '**f**' (*or any other character different than 'i'*), the method prints the **final** statistics. Since there can be **only Purple, Chocolate, Black, Orange, Yellow and Light Blue** cells at the end, I declare 6 **int** which represent the number of such-colored-cells. I define nested **FOR LOOP** and **IF-ELSE STATEMENT**s to look at every cell of the board and **increase p / c / b / o / y / l by 1 if the cell is such color**. Finally, I print on the screen the number of **Purple** / **Chocolate** / **Black** / **Orange** / **Yellow** / **Light Blue** cells and the **total** number of the cells.

IMPLEMENTATION

```
/*
CmpE 150 Introduction to Computing, Fall 2015
Project 3
CEMAL BURAK AYGÜN, 2014400072
*/

import java.io.*;
import java.util.*;

public class CBA2014400072 {


   public static void main(String[] args) throws FileNotFoundException{

      char[][] cell = new char[18][18];   // 18x18 board and every cell[][] is a cell of that board

      /* The FOR LOOP below writes O in the cells in the first and last rows and the first and last columns of the 18x18 board */
      for(int i = 0 ; i < 18 ; i ++){    // i is the row and column number of the board
         cell[0][i] = 'O';   // first row
         cell[17][i] = 'O';   //last row
         cell[i][0] = 'O';   // first column
         cell[i][17] = 'O';   // last column
      }

      Scanner file = new Scanner(new File("input.txt"));

      /* The nested FOR LOOPs below copy every single character of the input.txt to the elements of cell[][] */
      for(int row = 0 ; row < 16 ; row ++){   // row is the row number of the board
         String line = file.next().toUpperCase();   // line is a row of the board
         for(int clmn = 0 ; clmn < 16 ; clmn++){   // clmn is the column number of the board
            cell[row+1][clmn+1] = line.charAt(clmn);
         }
      }

      printBoard(cell, 'i');   // i for initial
      boolean cont = true;   // cont is continue, it determines whether the WHILE LOOP runs or stops

      while( cont ){

        cont = false;
         char[][] temp = new char[18][18];   // temp is temporary, it is an Array to check whether any of the elements of cell[][] has
changed

         /* The nested FOR LOOPs below copy cel[][] into temp[][] */
         for(int row = 1 ; row <= 16 ; row++){   // row is the row number
            for(int clmn = 1 ; clmn <= 16 ; clmn++){   // clmn is the column number
               temp[row][clmn] = cell[row][clmn];
            }
         }

          /* The nested FOR LOOPs below specify the cell */
          for(int row = 1 ; row <= 16 ; row++){   // row is the row number
            for(int clmn = 1 ; clmn <= 16 ; clmn++){   // clmn is the column number

               /* The IF STATEMENT below runs darkColor method if the cell is Dark Color */
               if( isDarkColor(cell, row, clmn) ){
                  darkColor(cell, row, clmn);
               /* The IF STATEMENT below runs white method if the cell is White */
               }else if( cell[row][clmn] == 'W' ){
```

```
                white(cell, row, clmn);
            /* The IF STATEMENT below runs orange method if the cell is Orange */
            }else if( cell[row][clmn] == 'O' ){
                orange(cell, row, clmn);
            }

        }
    }

    /* The nested FOR LOOPs below specify the cell */
    for(int row = 1 ; row <= 16 ; row++){   // row is the row number
        for(int clmn = 1 ; clmn <= 16 ; clmn++){   // clmn is the column number

            /* The IF STATEMENT below makes cont true if a cell's color has changed */
            if( temp[row][clmn] != cell[row][clmn] ){
                cont = true;
            }
        }
    }

}

System.out.println();
System.out.println();

/* The nested FOR LOOPs below specify the cell */
for(int row = 1 ; row <= 16 ; row++){   // row is the row number
    for(int clmn = 1 ; clmn <= 16 ; clmn++){   // clmn is the column number

        /* The IF STATEMENT below changes the color of the cell to (L)ight Blue if it is (W)hite */
        if( cell[row][clmn] == 'W' ){
            cell[row][clmn] = 'L';
        }

    }
}

printBoard(cell, 'f');   // f for final

}

/*
The method below processes the Dark Color cells.
It examines the 4 neighbour cells (up, down, left and right) of the Dark Color cell and changes it according to some rules.
*/
public static void darkColor(char[][] cell, int row, int clmn){

    int numDC = 0, numLC = 0, numP = 0;   // number of (D)ark (C)olor, (L)ight (C)olor and (P)urple cells

    if( isDarkColor(cell, row-1, clmn) ){
        numDC++;
    }else{
        numLC++;
    }
    if( cell[row-1][clmn] == 'P' ){
        numP++;
    }

    if( isDarkColor(cell, row+1, clmn) ){
        numDC++;
    }else{
        numLC++;
    }
```

```
    if( cell[row+1][clmn] == 'P' ){
        numP++;
    }

    if( isDarkColor(cell, row, clmn-1) ){
        numDC++;
    }else{
        numLC++;
    }
    if( cell[row][clmn-1] == 'P' ){

        numP++;

    }

    if( isDarkColor(cell, row, clmn+1) ){
        numDC++;
    }else{
        numLC++;
    }
    if( cell[row][clmn+1] == 'P' ){
        numP++;
    }

    if( numDC == 4 ){
        cell[row][clmn] = 'B';
    }else if( numLC == 3 || (numLC == 2 && numP >= 1) || (numLC == 1 && numP >= 2) ){
        cell[row][clmn] = 'P';
    }else if( cell[row][clmn] != 'B' && cell[row][clmn] != 'P' ){
        cell[row][clmn] = 'C';
    }

}

/*
The method below processes the White cells.
It examines the 4 neighbour cells (up, down, left and right) of the White cell and changes it according to some rules.
*/
public static void white(char[][] cell, int row, int clmn){

    if( cell[row-1][clmn] == 'O' || cell[row+1][clmn] == 'O' || cell[row][clmn-1] == 'O' || cell[row][clmn+1] == 'O' ){
        cell[row][clmn] = 'O';
    }

}

/*
The method below processes the Orange cells.
It examines the 4 neighbour cells (up, down, left and right) of the Orange cell and changes it according to some rules.
*/
public static void orange(char[][] cell, int row, int clmn){

    int numY = 0, numO = 0, numDC = 0;   // number of (Y)ellow, (O)range and (D)ark (C)olor cells

    if( isDarkColor(cell, row-1, clmn) ){
        numDC++;
    }else if( cell[row-1][clmn] == 'Y' ){
        numY++;
    }else if( cell[row-1][clmn] == 'O' ){
        numO++;
    }

    if( isDarkColor(cell, row+1, clmn) ){
```

```
        numDC++;
    }else if( cell[row+1][clmn] == 'Y' ){
        numY++;
    }else if( cell[row+1][clmn] == 'O' ){
        numO++;
    }

    if( isDarkColor(cell, row, clmn-1) ){
        numDC++;
    }else if( cell[row][clmn-1] == 'Y' ){
        numY++;
    }else if( cell[row][clmn-1] == 'O' ){
        numO++;
    }

    if( isDarkColor(cell, row, clmn+1) ){

        numDC++;

    }else if( cell[row][clmn+1] == 'Y' ){
        numY++;
    }else if( cell[row][clmn+1] == 'O' ){
        numO++;
    }

    if( (numY >= 2 && numO <= 1) || (numY == 1 && numDC >= 2) || (numDC >= 2 && numO >= 1) ){
        cell[row][clmn] = 'Y';
    }

}

/* The method below checks whether a cell is Dark Color and returns true if so */
public static boolean isDarkColor(char[][] cell, int row, int clmn){

    return cell[row][clmn] == 'G' || cell[row][clmn] == 'B' || cell[row][clmn] == 'P' || cell[row][clmn] == 'C';

}

/*
The method below prints the board configuration and the statistics of the cells about their colors.
To print the INITIAL statistics, the parameter board should be i.
*/
public static void printBoard(char[][] cell, char board){

    /* The nested FOR LOOPs below print the 16x16 board configuration */
    for(int row = 1 ; row <=16 ; row++){   // row is the row number
        for(int clmn = 1 ; clmn <=16 ; clmn++){   // clmn is the column number
            System.out.print(cell[row][clmn]);
        }
        System.out.println();
    }

    /* The IF STATEMENT below executes for the INITIAL statistics of the colors */
    if( board == 'i' ){

        int w = 0, g = 0;   // number of (w)hite and (g)ray cells

        /* The nested FOR LOOPs below specify the cell */
        for(int row = 1 ; row <= 16 ; row++){   // row is the row number
            for(int clmn = 1 ; clmn <= 16 ; clmn++){   // clmn is the column number

                /* The IF STATEMENT below increases w by 1 if the cell is White */
                if( cell[row][clmn] == 'W' ){
```

```
                w++;
            /* The ELSE STATEMENT below increases g by 1 if the cell is Gray */
            }else if( cell[row][clmn] == 'G' ){
                g++;
            }

        }
    }

    System.out.print("INITIAL: G=" + g + " W=" + w + " ALL=" + (w+g) );

/* The ELSE STATEMENT below executes for the FINAL statistics of the colors */
}else{
    int p = 0, c = 0, b = 0, o = 0, y = 0, l = 0;   // number of (p)urple, (c)hocolate, (b)lack, (o)range, (y)ellow and (l)ight blue cells

    /* The nested FOR LOOPs below specify the cell */
    for(int row = 1 ; row <= 16 ; row++){   // row is the row number
        for(int clmn = 1 ; clmn <= 16 ; clmn++){   // clmn is the column number

            /* The IF STATEMENT below increases p by 1 if the cell is Purple */
            if( cell[row][clmn] == 'P' ){
                p++;
            /* The IF STATEMENT below increases c by 1 if the cell is Chocolate */
            }else if( cell[row][clmn] == 'C' ){
                c++;
            /* The IF STATEMENT below increases b by 1 if the cell is Black */
            }else if( cell[row][clmn] == 'B' ){
                b++;
            /* The IF STATEMENT below increases o by 1 if the cell is Orange */
            }else if( cell[row][clmn] == 'O' ){
                o++;
            /* The IF STATEMENT below increases y by 1 if the cell is Yellow */
            }else if( cell[row][clmn] == 'Y' ){
                y++;
            /* The ELSE STATEMENT below increases l by 1 if the cell is Light Blue */
            }else if( cell[row][clmn] == 'L' ){
                l++;
            }

        }
    }

    System.out.print("EXPLORED: P=" + p + " C=" + c + " B=" + b + " O=" + o + " Y=" + y + " L=" + l + " ALL=" + (p+c+b+o+y+l) );

}

}

}
```

OUTPUT OF THE PROGRAM

```
 ----jGRASP exec: java CBA2014400072
WWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWW
GGGGGGWWWWWWWWWW
GGWWGGWWWWWWWWWW
GGGWGGWWWWWWWWWW
GGWWGGWWWGGGWGWW
WGGGGGGGGGGGGGWW
WWWWWWWGGGWWWGGW
WWWWWWWGGGWWWGGW
WWWWWWWGGGGWWGGW
WWWWWWWWGGWWWGGW
WWWWWWWWWGWWWWWW
WWWWWWWWWGWWWWWW
WWWWWWWWGGGWWWWW
WWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWW
INITIAL: G=61 W=195 ALL=256

OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
CCCCCCOOOOOOOOOO
CCLLCCOOOOOOOOOO
CBPLCCOOOOOOOOOO
CCLLCCYYYCCCYPOO
YCCCCCCCCBCCCCYO
OOOOOOYCBCYYYCCO
OOOOOOOCBCYOOCCO
OOOOOOOCBBPOOCCO
OOOOOOOYCCYOOCCO
OOOOOOOOYPYOOOOO
OOOOOOOOYPYOOOOO
OOOOOOOOPPPOOOOO
OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
EXPLORED: P=8 C=47 B=6 O=173 Y=17 L=5 ALL=256
 ----jGRASP: operation complete.
```

```
 ----jGRASP exec: java CBA2014400072
GGGGGGWWGWGGGWWW
WWWWGGWWGGWWWGGW
GWWWWWGWGWGWGGGW
WGWWWGWWGWGWWWWG
GGGGWGWGWGWWGWWG
WGWWWGGWWWWWGWGW
GGWGWGGWWGGWWWWG
GWGGGWWGWGGWGWWW
GWGGGWWGWGWGGGWW
WWWGWGWWGWWGWGWW
GWWGWWWGWGGGWWGG
WGGGWWWGGWGGGWGG
GWGGGWWGGGGWGGWW
WGGGGGGWGGWGGWGW
WWWWWWWGGWGGWWGW
WGGWWGGWWGGGGGGW
INITIAL: G=126 W=130 ALL=256

PPPPCCOOPYPPPYOO
YYYYCCYYPPYYYCCO
CYYOOYCYPLPYPCCY
YPYOOPYYPLPYYYYP
PBPPYPYCYCYYPYYP
YPYYYCCYYYYYPYCY
PPYPYCCYYCCYYYYC
PYCBCLLPYCCYPYOO
PYCBCLLPYPLPPPOO
YYYCLCLLCLLPLPYO
CYYCLLLPLPCCLLCC
YPCCLLLCCLCCCLCC
CLCBCLLCBCCLCPYO
YPCCCPPLCCLCCLPO
OYYOOYYPPLCCLLPO
OPPOOPPYYPCCPPPO
EXPLORED: P=59 C=62 B=5 O=24 Y=72 L=34 ALL=256
 ----jGRASP: operation complete.
```

CONCLUSION

I have solved the problem correctly assuming that the board will be stabilized at some point.

I wrote the same nested FOR LOOP below a lot of times. There might be some improvements about that.

```
for(int row = 1 ; row <= 16 ; row ++){        // row is the row number of the board
        for(int clmn = 1 ; clmn <= 16 ; clmn++){    // clmn is the column number of the board
                STATEMENT();
        }
}
```