PROBLEM DESCRIPTION

I need to write a program to play a board game with the following properties:

- ➔ There are **16** characters on the board, **4** characters at each line (a total of 4 lines). There is **only 1 star (*)** and **15 other characters**.
- ➔ The aim is to move the * to the other characters in order to get points. The * can move only to the **adjacent** characters and in **4 possible directions (⇐ , ⇒ , ⇑ , ⇓)**. It swaps with the character in order to move.
- ➔ The user moves the * by giving directions to the computer: **L** for left, **R** for right, **U** for up, **D** for down.
- ➔ The user gets **5** points if the * swaps with **G**, **1** point for swapping with **P** and **0** point for swapping with other characters.
- ➔ The characters **P** and **G** are changed to **I** after swapping with the * whereas the other characters remain the same.
- ➔ At the beginning of the game, the user chooses to play with the default board configuration or to set up his/her own board configuration. In either case, the user specify how many moves he/she wants to have for the game.
- ➔ The game prints the current board configuration before asking the user to enter the move number.
- ➔ The game prints the current board configuration and the current score the user has gained after every move.

PROBLEM SOLUTION

We can write such a game with the help of the methods of the String class, IF STATEMENTs and a FOR LOOP. I begin with importing the **Scanner** class. In my main() method,

- I declare a Scanner named **keyboard** to be able to get input from the user.
- I declare a String named **line** which is the combination of 4 rows of the board in a single line. From the beginning of the `line`, the first 4 characters form the first row of the board, next 4 characters form the second row of the board and so on. I assign an initial value, which is the default board configuration, to the `line`. I store the whole board configuration as a single line (*in one String*) instead of a String for each row, because it would be difficult to access an index (*or a character*) at a String from another String.
- I use a `System.out.print` to welcome the user and ask him/her whether he/she wants to play with the default board configuraiton or not.
- I declare a String named **choice** to store the user's answer to the question above.

- I define an **IF STATEMENT** which executes if the user's answer is "**NO**". In the test of IF STATEMENT, I used **equalsIgnoreCase** so that it will not matter how the user has typed the word "NO". (*no, nO, No, NO*). In the IF STATEMENT,
  - I use a `System.out.print` to give the user some *explanation* about typing the characters for the rows and ask the user to type 4 characters of the first row of the board.
  - I declare a String named **line1** to store the sequence of 4 charachers the user types for the first row of the board.
  - I repeat the last 2 steps (except for *explanation*) for the **second**, **third** and **fourth** row of the board.
  - I update `line` with the user's board configuration set-up and use **toUpperCase** to convert all the letters in the String into uppercase. (*This process will be helpful later in my code.*)

- I use `boardDrawing()` method in a `System.out.print` to draw the current board

configuration and I ask the user how many moves she/he wants to make in the game.

➔ I declare an int named **totalMove** which is the number of moves the user will have in the game.

➔ I use a `System.out.println` to give the user some explanation about the game.

➔ I declare an int named **curScore**, which is the current score the user has gained, and give it an initial value of `0`.

➔ I define a **FOR LOOP** which limits the game to the number of moves the user has. In the test, I declare an int named **move**, which is the current move the user in, that goes from `1` to `totalMove` increasing by **1** at every step. In the FOR LOOP,

- ➔ I use a `System.out.print` to print the current move number and possible inputs for the direction.
- ➔ I declare a String named **direction** which is the direction in which the user wants to move the *.

- ➔ I define an **IF STATEMENT** (Let's call it IFS1) which executes and moves the * to the **left** if the direction is left and it is possible for the * to move to the left. The first condition in the test (`equalsIgnoreCase("L")`) checks whether the direction is left (*L or* l) and the second condition in the test (`line.indexOf("*") % 4 != 0`) checks whether it is possible for the * to move to the left. In IFS1,
  - ➔ I declare a char named **c** and assign it the character which stays at the left side of the *.
  - ➔ I update `curScore` with `score()` method.
  - ➔ I update `c` with `pG()` method.
  - ➔ I update `line`. I write the characters from the beginning to the character which stays at the left side of `c`; then, the `*`; then, `c`; then, the characters from the one which stays at the right side of the `*` to the end of the line. As a result, the `*` and the character that stays at the left side of the `*` swap with each other.

- ➔ I define an **(ELSE) IF STATEMENT** (Let's call it IFS2) which executes and moves the *

to the **right** if the direction is right and it is possible for the * to move to the right. The first condition in the test (equalsIgnoreCase("R")) checks whether the direction is right (*R or r*). The second condition in the test (line.indexOf("*") % 4 != 3) checks whether it is possible for the * to move to the right. In IFS2,

- ➔ I declare a char named **c** and assign it the character which stays at the right side of the *.
- ➔ I update curScore with score() method.
- ➔ I update c with pG() method.
- ➔ I update line. I write the characters from the beginning to the character which stays at the left side of the *; then, c; then, the *; then, the characters from c to the end of the line. As a result, the * and the character that stays at the right side of the * swap with each other.

➔ I define an **(ELSE) IF STATEMENT** (Let's call it IFS3) which executes and moves the * **up** if the direction is up and it is possible for the * to move up. The first condition in the test (equalsIgnoreCase("U")) checks whether the direction is up (*U or u*). The second condition in the test (line.indexOf("*") < 4) checks whether it is possible for the * to move up. In IFS3,

- ➔ I declare a char named **c** and assign it the character which stays at the top side of the *.
- ➔ I update curScore with score() method.
- ➔ I update c with pG() method.
- ➔ I update line. I write the characters from the beginning to the character which stays at the left side of c; then, the *; then, the characters from the one that stays at the right side of c to the one that stays at the left side of the *; then, c; then, the characters from the one that stays at the right side of the * to the end of the line. As a result, the * and the character that stays at the top side of the * swap with each other.

➔ I define an **(ELSE) IF STATEMENT** (Let's call it IFS4) which executes and moves the * **down** if the direction is down and it is possible for the * to move down. The first condition in the test (equalsIgnoreCase("D")) checks whether the direction is down (*D or d*). The second condition in the test (line.indexOf("*") > 11) checks whether it is possible for the * to move down. In IFS4,

- ➔ I declare a char named **c** and assign it the character which stays at the bottom side of the *.
- ➔ I update curScore with score() method.
- ➔ I update c with pG() method.
- ➔ I update line. I write the characters from the beginning to the character which stays at the left side the * ; then, c ; then, the characters from the one that stays at the right side of the * to the one that stays at the left side of c ; then, the * ; then, the characters from the one that stays at the right side of c to the end of the line. As a result, the * and the character that stays at the bottom side of the * swap with each other.

More Explanation for IFS1, IFS2, IFS3 and IFS4

When we write the board configuration in a single line (in a String), the characters get the indexes as shown below.

A	B	G	G
R	T	F	P
P	K	V	I
G	V	J	*

Default Board Configuration

^{0}A	^{1}B	^{2}G	^{3}G
^{4}R	^{5}T	^{6}F	^{7}P
^{8}P	^{9}K	^{10}V	^{11}I
^{12}G	^{13}V	^{14}J	^{15}*

Default Board Configuration with Indexes

CHAR.	A	B	G	G	R	T	F	P	P	K	V	I	G	V	J	*
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Default Board Configuration in a Single Line and the Indexes of the Characters in the String

In the board, we see that the characters that stay at the **first column** have the indexes **0**, **4**, **8** and **12**. These numbers have a propertry in common: They all are an integer multiple of **4**. In other words, they all equal **0 (mod 4)**. So, having an index which is equal 0 (mod 4) means being at the first column of the board and it is not possible to move to the **left** in this case. This is the logic the second condition in the test of **IFS1** uses.

Similarly, having an index which is equal to **3 (mod 4)** means being at the **fourth column** of the board and being unable to move to the **right**. This is the logic of the second

condition in the test of **IFS2**.

We see from the board that the characters which stay at the **first** row have indexes **0**, **1**, **2** and **3**. These all numbers are **less than 4**. So, having an index which is less than 4 means being at the **first** row of the board and in this case it is not possible to move **up**. This is the logic the second condition in the test of **IFS3** uses.

Similarly, having an index which is **greater than 11** means being at the **fourth** row of the board and being unable to move **down**. This is the logic of the second condition in the test of **IFS4**.

- ➔ I use `boardDrawing()` method in a `System.out.print` to draw the current board configuration and I write the current score.

➔ I use a `System.out.print` to write a message at the end of the game.

➔ After `main()` method, I define a method named **pG**, which replaces **P** and **G** with **I**, that takes a **char** as a parameter and returns a **char**. In `pG()` method,

- ➔ I define an **IF-ELSE STATEMENT** which returns **I** when the parameter of `pG()` is **P** or **G** and returns **the parameter itself** when it is **neither P nor G**.

➔ After `pG()` method , I define a method named **score**, which gives the user some points for the last move, that takes a **char** as a parameter and returns an **int**. In `score()` method,

- ➔ I define an **IF-ELSE IF-ELSE STATEMENT** which return **1** (point) if the parameter of `score()` is **P**, **5** (points) for **G** and **0** (point) for **other characters**. Since I convert the user's board configuration set-up into uppercase by using **toUpperCase** before, I can use ***c == 'P'*** and ***c == 'G'*** and I don't have to take into account ***c == 'p'*** and ***c == 'g'***

➔ After `score()` method, I define a method named **boardDrawing**, which draws the current board configuration, that takes a **String** as a parameter and returns a **String**. In `boardDrawing()` method,

- ➔ I divide the current board configuration (16 *characters in a single line*) into 4 lines and return the configuration as a 4-row and 4-column table.

IMPLEMENTATION

```
/*
CmpE 150 Introduction to Computing, Fall 2015
Project 2
CEMAL BURAK AYGÜN, 2014400072
*/

import java.util.Scanner;
public class CBA2014400072 {



   public static void main(String[] args){

      Scanner keyboard = new Scanner(System.in);
      String line = "ABGGRTFPPKVIGVJ*";   // line is the combination of 4 rows of the board in a single line and it's initial value is the default board configuration
      System.out.print("WELCOME TO THIS WEIRD GAME OF P*G \nDo you want to use the default board configuration? (YES/NO) ");
      String choice = keyboard.next();   // choice is the user's choice about using the default board configuration or not

      /* The IF STATEMENT below executes if the user's choice above is "No". It takes 4 rows from the user and updates the String line. */
      if(choice.equalsIgnoreCase("NO")){

         System.out.print("\n>> You are going to type 4 characters for each row. (16 characters in total.)\n>> Do NOT type any spaces between any two characters and make sure that you write exactly 1 * besides 15 other characters.\n\nWrite four characters for the 1st row:   ");
         String line1 = keyboard.next();   // line1 is the first row of the board

         System.out.print("Write four characters for the 2nd row:   ");
         String line2 = keyboard.next();   // line2 is the second row of the board

         System.out.print("Write four characters for the 3rd row:   ");
         String line3 = keyboard.next();   // line3 is the third row of the board

         System.out.print("Write four characters for the 4th row:   ");
         String line4 = keyboard.next();   // line4 is the fourth row of the board

         line = (line1 + line2 + line3 + line4).toUpperCase();   // line gets updated

      }

     System.out.print(boardDrawing(line) + "How many moves do you want to make? ");
      int totalMove = keyboard.nextInt();   // totalMove is the number of moves the user has in the game

      System.out.println("\n>> Make a move and press ENTER.\n>> After each move, the board configuration and your total points will be printed.\n>> Type L for Left, R for Right, U for Up, and D for Down.");
      int curScore = 0;   // curScore is the current score the user has gained

      /* The FOR LOOP below limits the game to the number of moves the user has */
      for(int move = 1 ; move <= totalMove ; move++){   // move is the current move the user is in

         System.out.print("\nMove#" + move + " (L/R/U/D) : ");
```

```
        String direction = keyboard.next();   // direction is the direction in which the * moves

        /* The IF STATEMENT below moves the * to the left if the * is not at the leftmost side of the row */
        if(direction.equalsIgnoreCase("L") && line.indexOf("*") % 4 != 0){

            char c = line.charAt(line.indexOf("*") - 1);   // c is the character on the left side of the *
            curScore += score(c);   // scoreSoFar gets updated
            c = pG(c);   // c gets updated
            line = line.substring(0,line.indexOf("*")-1) + "*" + c + line.substring(line.indexOf("*") + 1);   // line gets updated

        /* The ELSE IF STATEMENT below moves the * to the right if the * is not at the rightmost side of the row */
        }else if(direction.equalsIgnoreCase("R") && line.indexOf("*") % 4 != 3){

            char c = line.charAt(line.indexOf("*") + 1);   // c is the character on the right side of the *
            curScore += score(c);   // scoreSoFar gets updated
            c = pG(c);   // c gets updated
            line = line.substring(0,line.indexOf("*")) + c + "*" + line.substring(line.indexOf("*") + 2);   // line gets updated

        /* The ELSE IF STATEMENT below moves the * up if the * is not at the topmost side of the column */
        }else if(direction.equalsIgnoreCase("U") && !(line.indexOf("*") < 4)){

            char c = line.charAt(line.indexOf("*") - 4);   // c is the character on the top side of the *
            curScore += score(c);   // scoreSoFar gets updated
            c = pG(c);   // c gets updated
            line = line.substring(0,line.indexOf("*") - 4) + "*" + line.substring(line.indexOf("*") - 3, line.indexOf("*")) + c +
line.substring(line.indexOf("*") + 1);   // line gets updated

        /* The ELSE IF STATEMENT below moves the * down if the * is not at the bottommost side of the column */
        }else if(direction.equalsIgnoreCase("D") && !(line.indexOf("*") > 11)){

            char c = line.charAt(line.indexOf("*") + 4);   // c is the character on the bottom side of the *
            curScore += score(c);   // scoreSoFar gets updated
            c = pG(c);   // c gets updated
            line = line.substring(0,line.indexOf("*")) + c + line.substring(line.indexOf("*") + 1, line.indexOf("*") + 4) + "*" +
line.substring(line.indexOf("*") + 5);   // line gets updated

        }

        System.out.print(boardDrawing(line) + "Current Score: " + curScore + "\n");

    }

    System.out.print("\nEND OF THE GAME\nThank you for playing this game.");

  }



  /*
  The method below updates the character that swaps with the *.
  It changes the character to I if it is P or G.
  It leaves the character unchanged if it is neither P nor G.
  */
  public static char pG(char c){

    if(c == 'P' || c == 'G'){
```

```
            return 'I';

        }else{

            return c;

            }

    }



    /*
    The method below calculates the score that the user gains from the last move.
    It returns 1 (points) if * swaps with P, 5 (points) for swapping with G and 0 (points) for swapping with other characters.
    */
    public static int score(char c){

        if(c == 'P'){

            return 1;

        }else if(c == 'G'){

            return 5;

            }else{

                return 0;

                }

    }



    /*
    The method below draws the current board configuration.
    It gets the current configuration as a single line and divides it after every four characters from the beginning of the line.
    At the end, it returns the configuration as a 4-line table.
    */
    public static String boardDrawing(String line){

        return "\nThis is the board configuration now: \n\n" + line.substring(0,4) + "\n" + line.substring(4,8) + "\n" +
line.substring(8,12) + "\n" + line.substring(12) + "\n\n";

    }



}
```

OUTPUT OF THE PROGRAM

```
 ----jGRASP exec: java CBA2014400072
WELCOME TO THIS WEIRD GAME OF P*G
Do you want to use the default board configuration? (YES/NO) No

>> You are going to type 4 characters for each row. (16 characters in total.)
>> Do NOT type any spaces between any two characters and make sure that you write exactly 1 * besides 15 other characters.

Write four characters for the 1st row:   aBcD
Write four characters for the 2nd row:   PGgp
Write four characters for the 3rd row:   x*uŞ
Write four characters for the 4th row:   PagY

This is the board configuration now:

ABCD
PGGP
X*UŞ
PAGY

How many moves do you want to make? 4

>> Make a move and press ENTER.
>> After each move, the board configuration and your total points will be printed.
>> Type L for Left, R for Right, U for Up, and D for Down.

Move#1 (L/R/U/D) : u

This is the board configuration now:

ABCD
P*GP
XIUŞ
PAGY

Current Score: 5

Move#2 (L/R/U/D) : R

This is the board configuration now:

ABCD
PI*P
XIUŞ
PAGY

Current Score: 10

Move#3 (L/R/U/D) : r

This is the board configuration now:

ABCD
PII*
XIUŞ
PAGY

Current Score: 11

Move#4 (L/R/U/D) : D

This is the board configuration now:

ABCD
PIIŞ
XIU*
PAGY

Current Score: 11

END OF THE GAME
Thank you for playing this game.
 ----jGRASP: operation complete.
```

```
 ----jGRASP exec: java CBA2014400072
WELCOME TO THIS WEIRD GAME OF P*G
Do you want to use the default board configuration? (YES/NO) yEs

This is the board configuration now:

ABGG
RTFP
PKVI
GVJ*

How many moves do you want to make? 5

>> Make a move and press ENTER.
>> After each move, the board configuration and your total points will be printed.
>> Type L for Left, R for Right, U for Up, and D for Down.

Move#1 (L/R/U/D) : l

This is the board configuration now:

ABGG
RTFP
PKVI
GV*J

Current Score: 0

Move#2 (L/R/U/D) : d

This is the board configuration now:

ABGG
RTFP
PKVI
GV*J

Current Score: 0

Move#3 (L/R/U/D) : u

This is the board configuration now:

ABGG
RTFP
PK*I
GVVJ

Current Score: 0

Move#4 (L/R/U/D) : U

This is the board configuration now:

ABGG
RT*P
PKFI
GVVJ

Current Score: 0

Move#5 (L/R/U/D) : r

This is the board configuration now:

ABGG
RTI*
PKFI
GVVJ

Current Score: 1

END OF THE GAME
Thank you for playing this game.
 ----jGRASP: operation complete.
```

CONCLUSION

I have solved the problem correctly with the following assumptions:

- ➔ The user writes only “YES” or “NO” (case-insensitive) as an answer to the question “*Do you want to use the default board configuration? (YES/NO)*”.
- ➔ When the user chooses to set up his/her own board configuration, she/he writes exactly 4 characters without any space for each of the 4 rows of the board and writes exactly 1 star (*) and 15 other characters in total.
- ➔ The user writes only L, R, U or D (uppercase or lowercase) as the direction in which the * moves.

My program may work incorrectly or crash when any of the assumptions above is not valid. We can use use some **IF** or/and **ELSE IF** statements to control whether the user's inputs are valid or not bu this requires putting a large part of the code inside an IF statement and I think such a mechanism is not a good way to control the inputs of the user.