UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Task 1
Implementation of the software artifact field

Garrido Pozo, Enrique
Mora Herreros, Pablo
Ollero Jiménez, Adrián

Subject: Intelligent Systems

Group: 3ºA

Team: A2

Date: 10/10/17

INDEX

Previous considerations

We were thinking about the language that we were going to use, and we doubt about using Java, C or python. All in all, we are using Java, because is the language that we know better the use of classes and their relations.

In order to print the new distributions, we were thinking about write the new position of the tractor and their new distributions with their adjacent points. Finally, we change our idea, and we print the new distributions using North, East, West and South.

To solve the problem, we were thinking about using a recursive method, using 3 nested fors or an iterative problem. First, we used 3 fors, but it was so difficult to make distributions. For that reason, we make a recursive algorithm to make distributions.

To finish, if the input that you introduce doesn't have the same format as the problem says, a message of error will execute and the program will stop. We assume that k, max, columns and rows will have 1 as the minimum value.

Files

Action.java

This file is a class in which we store the coordinates of the next action. Apart of the constructor method there are other essential method called perform (). This class will be in charge of all operations related to the treatment of the new movement.

The method perform () is in charge of move the sand before change the tractor of the new position. After move all the arena and prove if it can move to the north, east, west or south, the coordinates of the tractor will change.
This is our method to do that:

```
public void perform(Field field) {

    if(sandMovement[1]!=0) {//Al norte
        field.setNumber(field.getXt()-1, field.getYt(), field.getNumber(field.getXt()-1, field.getYt())+sandMovement[1]);
        field.setNumber(field.getXt(), field.getYt(), field.getNumber(field.getXt(), field.getYt())-sandMovement[1]);
    }
    if(sandMovement[2]!=0) { //Al oeste
        field.setNumber(field.getXt(), field.getYt()-1, field.getNumber(field.getXt(), field.getYt()-1)+sandMovement[2]);
        field.setNumber(field.getXt(), field.getYt(), field.getNumber(field.getXt(), field.getYt())-sandMovement[2]);
    }
    if(sandMovement[3]!=0) { //Al este
        field.setNumber(field.getXt(), field.getYt()+1,field.getNumber(field.getXt(), field.getYt()+1)+ sandMovement[3]);
        field.setNumber(field.getXt(), field.getYt(), field.getNumber(field.getXt(), field.getYt())-sandMovement[3]);
    }
    if(sandMovement[4]!=0) { //Al sur
        field.setNumber(field.getXt()+1, field.getYt(), field.getNumber(field.getXt()+1, field.getYt())+sandMovement[4]);
        field.setNumber(field.getXt(), field.getYt(), field.getNumber(field.getXt(), field.getYt())-sandMovement[4]);
    }

    field.setXt(xt);
    field.setYt(yt);
}
```

In the method toString (), we print the new position of the tractor and after the sand that we will move. In our program, the positions of the array will be: Actual Position, North, West, East and South. An example:

```
-----ACTIONS-----
Tractor: (0, 1) Sand: [8, 0, 0, 0, 0]
Tractor: (0, 1) Sand: [7, 1, 0, 0, 0]
Tractor: (0, 1) Sand: [7, 0, 1, 0, 0]
Tractor: (0, 1) Sand: [7, 0, 0, 1, 0]
Tractor: (0, 1) Sand: [7, 0, 0, 0, 1]
Tractor: (0, 1) Sand: [6, 2, 0, 0, 0]
```

Field.java

In this class, we store all the variables related with the information that the file gives to us. For that reason, we have introduced the method to read the file and to generate the solution. We were thinking about to introduce another class to read and write, but finally we decide to do these methods in field. If we have to change these methods to another class, we will do in the next step.

First, in this class we will have the variables that we need to read the file:

```
private int [][] field;
private int xt, yt, k, max, sizec, sizer;
```

We have all the setters and getters to manage this data.

Method readField () is in charge of reading the file that you introduce as an input. If the format of the file is not the same as the example, the program gives you an error. Also, any error related with the maximum, k, number of rows, number of columns is controlled. This method calls another two methods called readFirstLine () and generateValuesField ().

```
public void readField() {
    File archivo = null;
    FileReader fr = null;
    BufferedReader br = null;
    try{
        archivo = new File("Setup.txt");
        fr = new FileReader(archivo);
        br = new BufferedReader(fr);
        String line=br.readLine();//first line
        readFirstLine(line);
        String matrix=br.readLine();//second line
        for(int s=1;s<sizer; s++) {
            matrix=matrix+br.readLine();
        }
        matrix= matrix.substring(1, matrix.length());
        generateValuesField(matrix);
    }catch (Exception e) {
        e.printStackTrace();
    }finally{
        try{
            if(null != fr){
                fr.close();
            }
        }catch(Exception e2){
            e2.printStackTrace();
        }
    }

}
```

The method readFirstLine () is in charge of save the variables of the first line.

```
public void readFirstLine(String line)throws Exception{
    String[] values=new String[6];
    int i=0;
    for (String retval: line.split(" ")) {
            values[i++]=retval;
    }
    xt=Integer.parseInt(values[0]);
    yt=Integer.parseInt(values[1]);
    k=Integer.parseInt(values[2]);
    max=Integer.parseInt(values[3]);
    sizec=Integer.parseInt(values[4]);
    sizer=Integer.parseInt(values[5]);
    try {
        if (max<=k) {
            throw new Exception();
        }
    }catch(Exception e) {
        System.out.println("Max must be greater than k");
        System.exit(0);
    }
}
```

The method generateValuesField () is to copy all the values in the matrix.

```
public void generateValuesField(String line) throws Exception{
    String[] values=new String[sizec*sizer];
    field=new int[sizer][sizec];
    try{
        int i=0;
        for (String retval: line.split(" ")) {
                values[i++]=retval;
                if(Integer.parseInt(retval)<0 || Integer.parseInt(retval)>max) throw new Exception();
        }
        i=0;
        for(int j=0;j<sizer;j++){
            for(int k=0;k<sizec;k++){
                field[j][k]=Integer.parseInt(values[i++]);
            }
        }
    }catch(Exception ex){ // handle your exception
        System.out.println("Error reading field. Check file. : " + ex.getMessage());
        System.exit(0);
    }

}
```

To finish, we do the method generateOuput () that we use it to write our solution in a file. We store in a variable called nextMov all the new variables that we store in the new distribution. We store in another variable called matrix to save the new distributions in the matrix.

```
public void generateOutput(String nextMov, String matrix) throws IOExcept
    String path= "Output.txt";
    File file=new File(path);
    BufferedWriter br;
    br= new BufferedWriter(new FileWriter(file));
    br.write(nextMov);
    br.newLine();
    br.write(matrix);
    br.close();
}
```

mainClass.java

This class is the most important in the program. First of all, it shows all the attributes of the file. After it looks for the adjacent points, and distribute to all positions the actual position. After that, we will applicate a new action in the coordinates, and we store in the new matrix. In this class, is where the majority of things we will do.

The most important method in this class is the successor (). This method calls other important methods as checkPositions (), printAdjacent (), moveSand (), removeMax (), createActions (), printActions (). The main functionality of this program is to find all successors of the position in which we are. This is the code:

```
private static ArrayList<Action> successor(Field field) {
    ArrayList <int[]> adjacentPositions, sandMovements;
    ArrayList <Action> actions =  new ArrayList<Action>();
    sandMovements= new ArrayList<int[]>();
    checkPositions(field);
    adjacentPositions=moveTractor(field);
    printAdjacent(adjacentPositions, field);
    moveSand(field, sandMovements);
    ////////////////////////////
    removeMax(sandMovements, field);
    ////////////////////////////
    createActions(adjacentPositions, sandMovements, actions);
    printActions(actions);
    return actions;
}
```

The method removeMax () is the method in charge of remove the distributions when the amount of money is bigger than the maximum. When is the maximum, it stops there.

```
private static void removeMax(ArrayList<int[]> sandMovements, Field field) {
    ArrayList<int[]> aux = new ArrayList<int[]>();
    Iterator<int[]> it = sandMovements.iterator();
    for(int i = 0; i< sandMovements.size(); i++) {
        int s [] = sandMovements.get(i);
        if( (s[1]!=0 && s[1]+field.getNumber(field.getXt()-1, field.getYt()) > field.getMax()) || //North
            (s[2]!=0 && s[2]+field.getNumber(field.getXt(), field.getYt()-1) > field.getMax()) || //West
            (s[3]!=0 && s[3]+field.getNumber(field.getXt(), field.getYt()+1) > field.getMax()) || //East
            (s[4]!=0 && s[4]+field.getNumber(field.getXt()+1, field.getYt()) > field.getMax())    //South V
        )
            aux.add(s);
    }
    sandMovements.removeAll(aux);
}
```

The method moveSand () is in charge of share the sand in all positions, and print0Adjacent () prints the positions in which you can move the sand in the coordinates.

The method printActions () is in charge of print all the possible actions in which the tractor and sand can move.

The method checkPositions () is in charge of prove if you can move to the north, the west, the east or the south.

```java
public static void checkPositions(Field field) {
    //NORTH
        if (field.getXt() > 0 && field.getXt() < field.getSizeR()) {
            positions[0]= true;
        }else {
            positions[0]=false;
        }
    //WEST
        if (field.getYt() > 0 && field.getYt() < field.getSizeC()) {
            positions[1]=true;
        }else {
            positions[1]=false;
        }
    //EAST
        if (field.getYt() < field.getSizeC()-1) {
            positions[2]=true;
        }else {
            positions[2]=false;
        }
    //SOUTH
        if (field.getXt() < field.getSizeR()-1) {
            positions[3]=true;
        }else {
            positions[3]=false;
        }
}
```

Examples

We will do with this example:

0 0 5 8 2 2
4 6
3 8

Graphic: