**package** RIM;

**import** java.io.\*;

**import** java.util.\*;

**import** javax.swing.\*;

**import** javax.swing.text.View;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** java.lang.\*;

**import** java.util.List;

**import** java.util.\*;

**class** position

{ **public** **int** row;

**public** **int** col;

position(**int** row, **int** col)

{**this**.row=row;

**this**.col=col;

}

position()

{row=0;

col=0;

}

**void** disp()

{System.***out***.print("\nRow:" + row);

System.***out***.print("\nCol:" + col);

}

}

**class** myexception **extends** Exception

{ String str;

myexception(String str)

{**this**.str=str;

}

**public** String toString()

{ **return**(str);

}

}

**class** ratinamaze **extends** JFrame

{ **public** **static** **int** *N*,*M*;

**public** **int**[][] maze;

Stack<position> path= **new** Stack<position>();

Stack<position>copy= **new** Stack<position>();

**private** **final** List<Integer> pathi=**new** ArrayList<Integer>();

**int** pathindex;

Scanner s=**new** Scanner(System.***in***);

**void** Rules()

{System.***out***.print("\t\t\t\t\t\t\tRULES\n\n");

System.***out***.print("1) 1 indicates blockages, 0 indicates free path.\n\n");

System.***out***.print("2) The first and last digit you enter HAVE to be 0 as they signify entrance and exit.\n");

System.***out***.print(" Else the maze will be invalid!\n\n\n\n");

System.***out***.print("ENJOY!");

}

**void** inputmaze()

{ System.***out***.print("\nEnter the number of Rows: ");

*N*=s.nextInt();

System.***out***.print("\nEnter the number of Columns: ");

*M*=s.nextInt();

maze=**new** **int**[*N*+2][*M*+2];

System.***out***.print("\n\n");

System.***out***.print("NOTE! 1 indicates blockages, 0 indicates free path.\n\n");

**try**

{**try**

{ System.***out***.print("Enter the Maze problem : \n\n");

**for**(**int** i=1;i<=*N*;i++)

{ **for**(**int** j=1;j<=*M*;j++)

{ maze[i][j]=s.nextInt();

}

}

**for**(**int** i=0;i<=*N*+1;i++) //Surrounding Maze by 1s

{ maze[i][0]=maze[i][*M*+1]=1;}

**for**(**int** i=0;i<=*M*+1;i++)

{ maze[0][i]=maze[*N*+1][i]=1;}

**if**(maze[1][1]==1||maze[*N*][*M*]==1)

**throw** **new** myexception("Please check your maze! Either the entrance or exit (or both) is blocked!");

}

**catch**(myexception e)

{ System.***out***.print(e);

System.***out***.print("Enter the Maze problem : \n\n");

**for**(**int** i=1;i<=*N*;i++)

{ **for**(**int** j=1;j<=*M*;j++)

{ maze[i][j]=s.nextInt();

}

}

**for**(**int** i=0;i<=*N*+1;i++) //Surrounding Maze by 1s

{ maze[i][0]=maze[i][*M*+1]=1;}

**for**(**int** i=0;i<=*M*+1;i++)

{ maze[0][i]=maze[*N*+1][i]=1;}

**if**(maze[1][1]==1||maze[*N*][*M*]==1)

**throw** **new** myexception("The entrance or exit (or both) has been blocked again!This program will be terminated.");

}

}

**catch**(myexception e)

{ System.***out***.print(e);

System.*exit*(0);

}

}

**public** **boolean** findpath()

{

position offset[]= **new** position[4];

offset[0]= **new** position(0,1); //To move right

offset[1]=**new** position(1,0);//To move down

offset[2]=**new** position(0,-1); //To move left

offset[3]=**new** position(-1,0);//To move up

position here=**new** position(1,1);

maze[1][1]=2;//Blocking the entrance

**int** option=0;

**int** lastop=3;

**while**(here.row!=*N* || here.col!=*M*)

{

**int** r=0;

**int** c=0;

**while**(option<=lastop)

{

r=here.row + (offset[option]).row;

//System.out.print(offset[option].row);

c=here.col + (offset[option]).col;

**if**(maze[r][c]==0)//next step of the path found

**break**;

option++;

}

**if**(option<=lastop)//neighbor found

{

position temp = **new** position(here.row, here.col);

path.push(temp);

//System.out.print(path.peek().row +"\t"+ path.peek().col);

//path.pop();

//System.out.print(path.peek().row +"\t"+ path.peek().col);

here.row=r;

here.col=c;

maze[r][c]=2;//Blocking the visited path

option=0;

}

**else** //neighbor not found

{ // System.out.println("\nentering else in findpath");

**if**(path.empty())

**return** **false**;

position next= **new** position();

//System.out.print(path.peek().row + "\t" + path.peek().col);

next.row = path.peek().row;

next.col = path.peek().col;

path.pop();

//System.out.print(path);

//next.disp();

// path.pop();

**if**(next.row==here.row)//to set the option value

option=2+ next.col - here.col;

**else**

option=3+next.row-here.row;

here=next;//backtracking to previous position to look for other positions

//System.out.println(here.equals(next));

}

}

**return** **true**;

}

**void** outputmaze()

{ position res=**new** position();

Stack<position>opt= **new** Stack<position>();

**while**(!path.empty()) //The path is converted into 3s

{

res=path.peek();

**int** a = path.peek().row;

**int** b = path.peek().col;

//System.out.print(a+" "+b);

opt.push(res);

path.pop();

**for**(**int** i=1;i<=*N*;i++)

{

**for**(**int** j=1;j<=*M*;j++)

{

**if**(a==i&&b==j)

{ maze[i][j]=3;

**break**;

}

}

}

}

maze[*N*][*M*]=3;

System.***out***.print("\nThe PATH is : \n");

**while**(!opt.empty())

{

res=opt.peek();

opt.pop();

pathi.add(res.row);

pathi.add(res.col);

System.***out***.print("("+res.row+","+res.col+") --> ");

//System.out.print("("+copy.get(0).row+","+copy.get(0).col+") --> ");

//System.out.print("("+copy.peek().row+","+copy.peek().col+") --> ");

}

pathi.add(*N*);

pathi.add(*M*);

System.***out***.print("("+*N*+","+*M*+")");

System.***out***.print("\n\n\nThe PATH is highlighted by 3s!\n\n");

**for**(**int** i=1;i<=*N*;i++)

{

**for**(**int** j=1;j<=*M*;j++)

{

System.***out***.print(maze[i][j]+" ");

}

System.***out***.print("\n");

}

System.***out***.print("\n");

}

/\*void convert(){

//System.out.println(copy);

while(!copy.empty()){

pathi.add(copy.peek().row);

pathi.add(copy.peek().col);

copy.pop();

}

}\*/

**public** ratinamaze(){

Rules();

inputmaze();

setTitle("Rat in a Maze");

setSize(640, 480);

setLocationRelativeTo(**null**);

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

**if**(findpath())

{

outputmaze();

//convert();

pathindex=0;

}

**else**

System.***out***.print("\nNo path found!");

/\*int c;

do

{ System.out.print("\n1)Try another maze\n2)Terminate program\n");

c=s.nextInt();

switch(c)

{ case 1:

inputmaze();

if(findpath())

{outputmaze();

convert();

pathindex=0;

}

else

System.out.print("\nNo path found!");

break;

case 2:

System.exit(0);

}

}while(true); \*/

}

/\*System.out.print("\nPress enter to exit!");

String n= s.next();

if(n.equals(""))

System.exit(0);\*/

@Override

**public** **void** paint(Graphics g){

**super**.paint(g);

g.translate(100, 100);

**for**(**int** i=0;i<maze.length;i++){

**for**(**int** j=0;j<maze[0].length;j++){

Color color;

**switch**(maze[i][j]){

**case** 1:color=Color.***BLUE***;**break**;

**case** 3:color=Color.***green***;**break**;

**default**:color=Color.***WHITE***;**break**;

}

g.setColor(color);

g.fillRect(30\*j, 30\*i,30, 30);

g.setColor(Color.***BLACK***);

g.drawRect(30\*j, 30\*i,30, 30);

}

}

**int** pathx=pathi.get(pathindex);

**int** pathy=pathi.get(pathindex+1);

g.setColor(Color.***RED***);

g.fillOval(pathy\*30, pathx\*30, 30, 30);

System.***out***.println(pathx+" "+pathy);

//System.out.println(pathi);

}

@Override

**protected** **void** processKeyEvent(KeyEvent k){

**if**(k.getID() != KeyEvent.***KEY\_PRESSED***){

**return**;

}

**if**(k.getKeyCode() == KeyEvent.***VK\_RIGHT***){

pathindex=pathindex+2;

**if**(pathindex>pathi.size()-2){

pathindex=pathi.size()-2;

}

}

**else** **if**(k.getKeyCode()==KeyEvent.***VK\_LEFT***){

pathindex-=2;

**if**(pathindex<0){

pathindex=0;

}

}

repaint();

}

}

**public** **class** RIMmain1

{

**public** **static** **void** main(String[] args) {

SwingUtilities.*invokeLater*(**new** Runnable(){

**public** **void** run(){

ratinamaze R=**new** ratinamaze();

R.setVisible(**true**);

}

});

}