**Assignment 2: Compiler**

**Part 1**

--------------------------------------------------------------------------------------------------------------------

Name: Avishek Rauniyar

Roll No.: AM.EN.U4AIE22062

--------------------------------------------------------------------------------------------------------------------

Objective of this Assignment

In this Assignment we build the Jack Tokenizer that takes the Jack programs and generates a list of Jack tokens

Guidelines:

1. Submit source code files and Submit the tokenized output XML files

Question 1

• First, write and test the Tokenizer module

**Test Programs [tokenizer]**

ExpressionLessSquare

Square

Array Test

OUTPUT : (*While Running the default set file Running directly will run Main.jack*)

Now the code is running with a default file

please enter the name correctly next time

Operation Successful

OUTPUT *: (While running the file that we need to convert through java command)*

PS A:\Avishek\IMP\S2\EOC\LAB\Tokenizer> javac Tokenizer.java

PS A:\Avishek\IMP\S2\EOC\LAB\Tokenizer> java Tokenizer SquareGame.jack

Operation Successful

*Running the Program*:

For running the default set file ie Main.jack, we can simply run the program .

For running any other file use terminal and type

* + - *javac Tokenizer.java* (Replace Tokenizer.java with your filename and a class file will appear)
    - *java Tokenizer <filename.jack>* (Replace Tokenizer with your compiler name without .java extension and <filename.jack> with the filename)

***FILE : Tokenizer.java***

import java.util.\*;

import java.io.\*;

public class Tokenizer {

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

        try{

            String filename;

            if (args.length == 0) {

                System.out.println("Now the code is running with a default file \n please enter the name correctly next time");

                filename="Main.jack";

            }

            else{

                filename = args[0];

            }

            File read = new File(filename);

            Scanner in = new Scanner(read);

            FileWriter writer = new FileWriter(filename.substring(0,filename.indexOf("."))+".xml");

            List<String> keywords = List.of("class", "constructor", "function","field","method","static","char","var","int","boolean","void","true","false","null","this","let","do","if","else","while","return");

            List<String> symbols = List.of("{","}","|","(",")","[","]",".",",",";","+","-","\*","/","&","&lt;","&gt","=","~");

            writer.write("<tokens>\n");

            String s=null;

            while(in.hasNextLine()){

            String data=in.nextLine();

                data = data.replaceAll("\n", "");

                data = data.replaceAll("\t", "");

                if (data.length() > 1 && data.substring(0, 2).equals("//")) {

                    continue;

                } else if (data.length() == 0)

                    continue;

                if (data.isEmpty()) {

                    continue;

                }

                String a=data.replaceAll(" ","");

                if(a.startsWith("\*"))

                {

                    continue;

                }

                else if(a.startsWith("/"))

                {

                    continue;

                }

                data=data.replaceAll("\\?","ques");

                if(data.contains("\"")){

                    int startIndex = data.indexOf("\"");

                    int endIndex = data.indexOf("\"", startIndex + 1);

                    s=data.substring(startIndex + 1, endIndex);

                    data=data.replaceAll("\""+s+"\"", " \" ");

                }

                data=data.replaceAll("\\["," \\[ ");

                data=data.replaceAll("\\]"," \\] ");

                data=data.replaceAll("\\{"," \\{ ");

                data=data.replaceAll("\\}"," \\} ");

                data=data.replaceAll("\\("," \\( ");

                data=data.replaceAll("\\)"," \\) ");

                data=data.replaceAll("\\;"," \\; ");

                data=data.replaceAll("\\<"," \\&lt; ");

                data=data.replaceAll("\\>"," \\&gt; ");

                data=data.replaceAll("\\."," \\. ");

                data=data.replaceAll("\\,"," \\, ");

                data=data.replaceAll("\\-"," \\- ");

                String[] tokens = data.split("\\s+");

                for (String token: tokens){

                    if (token != null && !token.isEmpty()) {

                        if (token.contains("//"))

                        {

                            break;

                        }

                        char firstChar = token.charAt(0);

                        if (firstChar >= '0' && firstChar <= '9') {

                            writer.write("<integerConstant> ");

                            writer.write(token);

                            writer.write(" </integerConstant>\n");

                        }

                        else if(symbols.contains(token)){

                            writer.write("<symbol> ");

                            writer.write(token);

                            writer.write(" </symbol>\n");

                        }

                        else if(keywords.contains(token)){

                            writer.write("<keyword> ");

                            writer.write(token);

                            writer.write(" </keyword>\n");

                        }

                        else if (token.contains("\"")){

                            writer.write("<stringConstant> ");

                            s=s.replaceAll("ques","\\?");

                            writer.write(s);

                            writer.write(" </stringConstant>\n");

                        }

                        else{

                            writer.write("<identifier> ");

                            writer.write(token);

                            writer.write(" </identifier>\n");

                        }

                    }

                }

            }

            writer.write("</tokens>\n");

            in.close();

            writer.close();

System.out.println("Operation Successful ");

        }catch (FileNotFoundException e) {

            System.out.println("An error occured.");

            e.printStackTrace();

        }

    }

}

INPUT FILE: (Main.jack,Arraytest)

// This file is part of www.nand2tetris.org

// and the book "The Elements of Computing Systems"

// by Nisan and Schocken, MIT Press.

// File name: projects/10/ArrayTest/Main.jack

// (identical to projects/09/Average/Main.jack)

/\*\* Computes the average of a sequence of integers. \*/

class Main {

    function void main() {

        var Array a;

        var int length;

        var int i, sum;

    let length = Keyboard.readInt("HOW MANY NUMBERS? ");

    let a = Array.new(length);

    let i = 0;

    while (i < length) {

        let a[i] = Keyboard.readInt("ENTER THE NEXT NUMBER: ");

        let i = i + 1;

    }

    let i = 0;

    let sum = 0;

    while (i < length) {

        let sum = sum + a[i];

        let i = i + 1;

    }

    do Output.printString("THE AVERAGE IS: ");

    do Output.printInt(sum / length);

    do Output.println();

    return;

    }

}

*OUTPUT FILE : (Main.xml, Arraytest)*

<tokens>

<keyword> class </keyword>

<identifier> Main </identifier>

<symbol> { </symbol>

<keyword> function </keyword>

<keyword> void </keyword>

<identifier> main </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> var </keyword>

<identifier> Array </identifier>

<identifier> a </identifier>

<symbol> ; </symbol>

<keyword> var </keyword>

<keyword> int </keyword>

<identifier> length </identifier>

<symbol> ; </symbol>

<keyword> var </keyword>

<keyword> int </keyword>

<identifier> i </identifier>

<symbol> , </symbol>

<identifier> sum </identifier>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> length </identifier>

<symbol> = </symbol>

<identifier> Keyboard </identifier>

<symbol> . </symbol>

<identifier> readInt </identifier>

<symbol> ( </symbol>

<stringConstant> HOW MANY NUMBERS?  </stringConstant>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> a </identifier>

<symbol> = </symbol>

<identifier> Array </identifier>

<symbol> . </symbol>

<identifier> new </identifier>

<symbol> ( </symbol>

<identifier> length </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> i </identifier>

<symbol> = </symbol>

<integerConstant> 0 </integerConstant>

<symbol> ; </symbol>

<keyword> while </keyword>

<symbol> ( </symbol>

<identifier> i </identifier>

<symbol> &lt; </symbol>

<identifier> length </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> a </identifier>

<symbol> [ </symbol>

<identifier> i </identifier>

<symbol> ] </symbol>

<symbol> = </symbol>

<identifier> Keyboard </identifier>

<symbol> . </symbol>

<identifier> readInt </identifier>

<symbol> ( </symbol>

<stringConstant> ENTER THE NEXT NUMBER:  </stringConstant>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> i </identifier>

<symbol> = </symbol>

<identifier> i </identifier>

<symbol> + </symbol>

<integerConstant> 1 </integerConstant>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> let </keyword>

<identifier> i </identifier>

<symbol> = </symbol>

<integerConstant> 0 </integerConstant>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> sum </identifier>

<symbol> = </symbol>

<integerConstant> 0 </integerConstant>

<symbol> ; </symbol>

<keyword> while </keyword>

<symbol> ( </symbol>

<identifier> i </identifier>

<symbol> &lt; </symbol>

<identifier> length </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> sum </identifier>

<symbol> = </symbol>

<identifier> sum </identifier>

<symbol> + </symbol>

<identifier> a </identifier>

<symbol> [ </symbol>

<identifier> i </identifier>

<symbol> ] </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> i </identifier>

<symbol> = </symbol>

<identifier> i </identifier>

<symbol> + </symbol>

<integerConstant> 1 </integerConstant>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> do </keyword>

<identifier> Output </identifier>

<symbol> . </symbol>

<identifier> printString </identifier>

<symbol> ( </symbol>

<stringConstant> THE AVERAGE IS:  </stringConstant>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Output </identifier>

<symbol> . </symbol>

<identifier> printInt </identifier>

<symbol> ( </symbol>

<identifier> sum </identifier>

<symbol> / </symbol>

<identifier> length </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Output </identifier>

<symbol> . </symbol>

<identifier> println </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<symbol> } </symbol>

</tokens>

*INPUT FILE : (Main.jack, ExpressionLessSquare)*

// This file is part of www.nand2tetris.org

// and the book "The Elements of Computing Systems"

// by Nisan and Schocken, MIT Press.

// File name: projects/10/ExpressionLessSquare/Main.jack

/\*\* Expressionless version of projects/10/Square/Main.jack. \*/

class Main {

    static boolean test;    // Added for testing -- there is no static keyword

                            // in the Square files.

    function void main() {

        var SquareGame game;

        let game = game;

        do game.run();

        do game.dispose();

        return;

    }

    function void more() {  // Added to test Jack syntax that is not used in

        var boolean b;      // the Square files.

        if (b) {

        }

        else {              // There is no else keyword in the Square files.

        }

        return;

    }

}

*OUTPUT FILE : (Main.xml, ExpressionLessSquare)*

<tokens>

<keyword> class </keyword>

<identifier> Main </identifier>

<symbol> { </symbol>

<keyword> static </keyword>

<keyword> boolean </keyword>

<identifier> test </identifier>

<symbol> ; </symbol>

<keyword> function </keyword>

<keyword> void </keyword>

<identifier> main </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> var </keyword>

<identifier> SquareGame </identifier>

<identifier> game </identifier>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> game </identifier>

<symbol> = </symbol>

<identifier> game </identifier>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> game </identifier>

<symbol> . </symbol>

<identifier> run </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> game </identifier>

<symbol> . </symbol>

<identifier> dispose </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> function </keyword>

<keyword> void </keyword>

<identifier> more </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> var </keyword>

<keyword> boolean </keyword>

<identifier> b </identifier>

<symbol> ; </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> b </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<symbol> } </symbol>

<keyword> else </keyword>

<symbol> { </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<symbol> } </symbol>

</tokens>

INPUT FILE: (Square.jack, ExpressionLessSquare)

// This file is part of www.nand2tetris.org

// and the book "The Elements of Computing Systems"

// by Nisan and Schocken, MIT Press.

/// File name: projects/10/ExpressionLessSquare/Square.jack

/\*\* Expressionless version of projects/10/Square/Square.jack. \*/

class Square {

   field int x, y;

   field int size;

   constructor Square new(int Ax, int Ay, int Asize) {

      let x = Ax;

      let y = Ay;

      let size = Asize;

      do draw();

      return x;

   }

   method void dispose() {

      do Memory.deAlloc(this);

      return;

   }

   method void draw() {

      do Screen.setColor(x);

      do Screen.drawRectangle(x, y, x, y);

      return;

   }

   method void erase() {

      do Screen.setColor(x);

      do Screen.drawRectangle(x, y, x, y);

      return;

   }

   method void incSize() {

      if (x) {

         do erase();

         let size = size;

         do draw();

      }

      return;

   }

   method void decSize() {

      if (size) {

         do erase();

         let size = size;

         do draw();

      }

      return;

   }

   method void moveUp() {

      if (y) {

         do Screen.setColor(x);

         do Screen.drawRectangle(x, y, x, y);

         let y = y;

         do Screen.setColor(x);

         do Screen.drawRectangle(x, y, x, y);

      }

      return;

   }

   method void moveDown() {

      if (y) {

         do Screen.setColor(x);

         do Screen.drawRectangle(x, y, x, y);

         let y = y;

         do Screen.setColor(x);

         do Screen.drawRectangle(x, y, x, y);

      }

      return;

   }

   method void moveLeft() {

      if (x) {

         do Screen.setColor(x);

         do Screen.drawRectangle(x, y, x, y);

         let x = x;

         do Screen.setColor(x);

         do Screen.drawRectangle(x, y, x, y);

      }

      return;

   }

   method void moveRight() {

      if (x) {

         do Screen.setColor(x);

         do Screen.drawRectangle(x, y, x, y);

         let x = x;

         do Screen.setColor(x);

         do Screen.drawRectangle(x, y, x, y);

      }

      return;

   }

}

*OUTPUT FILE : (Square.xml, ExpressionLessSquare)*

<tokens>

<keyword> class </keyword>

<identifier> Square </identifier>

<symbol> { </symbol>

<keyword> field </keyword>

<keyword> int </keyword>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> ; </symbol>

<keyword> field </keyword>

<keyword> int </keyword>

<identifier> size </identifier>

<symbol> ; </symbol>

<keyword> constructor </keyword>

<identifier> Square </identifier>

<identifier> new </identifier>

<symbol> ( </symbol>

<keyword> int </keyword>

<identifier> Ax </identifier>

<symbol> , </symbol>

<keyword> int </keyword>

<identifier> Ay </identifier>

<symbol> , </symbol>

<keyword> int </keyword>

<identifier> Asize </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> x </identifier>

<symbol> = </symbol>

<identifier> Ax </identifier>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> y </identifier>

<symbol> = </symbol>

<identifier> Ay </identifier>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> size </identifier>

<symbol> = </symbol>

<identifier> Asize </identifier>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> draw </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<identifier> x </identifier>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> dispose </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Memory </identifier>

<symbol> . </symbol>

<identifier> deAlloc </identifier>

<symbol> ( </symbol>

<keyword> this </keyword>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> draw </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> erase </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> incSize </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> erase </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> size </identifier>

<symbol> = </symbol>

<identifier> size </identifier>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> draw </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> decSize </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> erase </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> size </identifier>

<symbol> = </symbol>

<identifier> size </identifier>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> draw </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> moveUp </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> y </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> y </identifier>

<symbol> = </symbol>

<identifier> y </identifier>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> moveDown </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> y </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> y </identifier>

<symbol> = </symbol>

<identifier> y </identifier>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> moveLeft </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> x </identifier>

<symbol> = </symbol>

<identifier> x </identifier>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> moveRight </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> x </identifier>

<symbol> = </symbol>

<identifier> x </identifier>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<symbol> } </symbol>

</tokens>

INPUT FILE: (SquareGame.jack, ExpressionLessSquare)

// This file is part of www.nand2tetris.org

// and the book "The Elements of Computing Systems"

// by Nisan and Schocken, MIT Press.

// File name: projects/10/ExpressionLessSquare/SquareGame.jack

/\*\* Expressionless version of projects/10/Square/SquareGame.jack. \*/

class SquareGame {

   field Square square;

   field int direction;

   constructor SquareGame new() {

      let square = square;

      let direction = direction;

      return square;

   }

   method void dispose() {

      do square.dispose();

      do Memory.deAlloc(square);

      return;

   }

   method void moveSquare() {

      if (direction) { do square.moveUp(); }

      if (direction) { do square.moveDown(); }

      if (direction) { do square.moveLeft(); }

      if (direction) { do square.moveRight(); }

      do Sys.wait(direction);

      return;

   }

   method void run() {

      var char key;

      var boolean exit;

      let exit = key;

      while (exit) {

         while (key) {

            let key = key;

            do moveSquare();

         }

         if (key) { let exit = exit; }

         if (key) { do square.decSize(); }

         if (key) { do square.incSize(); }

         if (key) { let direction = exit; }

         if (key) { let direction = key; }

         if (key) { let direction = square; }

         if (key) { let direction = direction; }

         while (key) {

            let key = key;

            do moveSquare();

         }

      }

      return;

    }

}

*OUTPUT FILE : (SquareGame.xml, ExpressionLessSquare)*

<tokens>

<keyword> class </keyword>

<identifier> SquareGame </identifier>

<symbol> { </symbol>

<keyword> field </keyword>

<identifier> Square </identifier>

<identifier> square </identifier>

<symbol> ; </symbol>

<keyword> field </keyword>

<keyword> int </keyword>

<identifier> direction </identifier>

<symbol> ; </symbol>

<keyword> constructor </keyword>

<identifier> SquareGame </identifier>

<identifier> new </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> square </identifier>

<symbol> = </symbol>

<identifier> square </identifier>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> direction </identifier>

<symbol> = </symbol>

<identifier> direction </identifier>

<symbol> ; </symbol>

<keyword> return </keyword>

<identifier> square </identifier>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> dispose </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> dispose </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Memory </identifier>

<symbol> . </symbol>

<identifier> deAlloc </identifier>

<symbol> ( </symbol>

<identifier> square </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> moveSquare </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> direction </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> moveUp </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> direction </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> moveDown </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> direction </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> moveLeft </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> direction </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> moveRight </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> do </keyword>

<identifier> Sys </identifier>

<symbol> . </symbol>

<identifier> wait </identifier>

<symbol> ( </symbol>

<identifier> direction </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> run </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> var </keyword>

<keyword> char </keyword>

<identifier> key </identifier>

<symbol> ; </symbol>

<keyword> var </keyword>

<keyword> boolean </keyword>

<identifier> exit </identifier>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> exit </identifier>

<symbol> = </symbol>

<identifier> key </identifier>

<symbol> ; </symbol>

<keyword> while </keyword>

<symbol> ( </symbol>

<identifier> exit </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> while </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> key </identifier>

<symbol> = </symbol>

<identifier> key </identifier>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> moveSquare </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> exit </identifier>

<symbol> = </symbol>

<identifier> exit </identifier>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> decSize </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> incSize </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> direction </identifier>

<symbol> = </symbol>

<identifier> exit </identifier>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> direction </identifier>

<symbol> = </symbol>

<identifier> key </identifier>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> direction </identifier>

<symbol> = </symbol>

<identifier> square </identifier>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> direction </identifier>

<symbol> = </symbol>

<identifier> direction </identifier>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> while </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> key </identifier>

<symbol> = </symbol>

<identifier> key </identifier>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> moveSquare </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<symbol> } </symbol>

</tokens>

INPUT FILE: (Main.jack, Square)

// This file is part of www.nand2tetris.org

// and the book "The Elements of Computing Systems"

// by Nisan and Schocken, MIT Press.

// File name: projects/10/Square/Main.jack

// (derived from projects/09/Square/Main.jack, with testing additions)

/\*\* Initializes a new Square Dance game and starts running it. \*/

class Main {

    static boolean test;    // Added for testing -- there is no static keyword

                            // in the Square files.

    function void main() {

      var SquareGame game;

      let game = SquareGame.new();

      do game.run();

      do game.dispose();

      return;

    }

    function void more() {  // Added to test Jack syntax that is not used in

        var int i, j;       // the Square files.

        var String s;

        var Array a;

        if (**false**) {

            let s = "string constant";

            let s = **null**;

            let a[1] = a[2];

        }

        else {              // There is no else keyword in the Square files.

            let i = i \* (-j);

            let j = j / (-2);   // note: unary negate constant 2

            let i = i | j;

        }

        return;

    }

}

*OUTPUT FILE : (Main.xml, Square)*

<tokens>

<keyword> class </keyword>

<identifier> Main </identifier>

<symbol> { </symbol>

<keyword> static </keyword>

<keyword> boolean </keyword>

<identifier> test </identifier>

<symbol> ; </symbol>

<keyword> function </keyword>

<keyword> void </keyword>

<identifier> main </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> var </keyword>

<identifier> SquareGame </identifier>

<identifier> game </identifier>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> game </identifier>

<symbol> = </symbol>

<identifier> SquareGame </identifier>

<symbol> . </symbol>

<identifier> new </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> game </identifier>

<symbol> . </symbol>

<identifier> run </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> game </identifier>

<symbol> . </symbol>

<identifier> dispose </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> function </keyword>

<keyword> void </keyword>

<identifier> more </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> var </keyword>

<keyword> int </keyword>

<identifier> i </identifier>

<symbol> , </symbol>

<identifier> j </identifier>

<symbol> ; </symbol>

<keyword> var </keyword>

<identifier> String </identifier>

<identifier> s </identifier>

<symbol> ; </symbol>

<keyword> var </keyword>

<identifier> Array </identifier>

<identifier> a </identifier>

<symbol> ; </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<keyword> false </keyword>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> s </identifier>

<symbol> = </symbol>

<stringConstant> string constant </stringConstant>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> s </identifier>

<symbol> = </symbol>

<keyword> null </keyword>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> a </identifier>

<symbol> [ </symbol>

<integerConstant> 1 </integerConstant>

<symbol> ] </symbol>

<symbol> = </symbol>

<identifier> a </identifier>

<symbol> [ </symbol>

<integerConstant> 2 </integerConstant>

<symbol> ] </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> else </keyword>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> i </identifier>

<symbol> = </symbol>

<identifier> i </identifier>

<symbol> \* </symbol>

<symbol> ( </symbol>

<symbol> - </symbol>

<identifier> j </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> j </identifier>

<symbol> = </symbol>

<identifier> j </identifier>

<symbol> / </symbol>

<symbol> ( </symbol>

<symbol> - </symbol>

<integerConstant> 2 </integerConstant>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> i </identifier>

<symbol> = </symbol>

<identifier> i </identifier>

<symbol> | </symbol>

<identifier> j </identifier>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<symbol> } </symbol>

</tokens>

INPUT FILE: (Square.jack, Square)

// This file is part of www.nand2tetris.org

// and the book "The Elements of Computing Systems"

// by Nisan and Schocken, MIT Press.

// File name: projects/10/Square/Square.jack

// (same as projects/09/Square/Square.jack)

/\*\* Implements a graphical square. \*/

class Square {

   field int x, y; // screen location of the square's top-left corner

   field int size; // length of this square, in pixels

   /\*\* Constructs a new square with a given location and size. \*/

   constructor Square new(int Ax, int Ay, int Asize) {

      let x = Ax;

      let y = Ay;

      let size = Asize;

      do draw();

      return this;

   }

   /\*\* Disposes this square. \*/

   method void dispose() {

      do Memory.deAlloc(this);

      return;

   }

   /\*\* Draws the square on the screen. \*/

   method void draw() {

      do Screen.setColor(**true**);

      do Screen.drawRectangle(x, y, x + size, y + size);

      return;

   }

   /\*\* Erases the square from the screen. \*/

   method void erase() {

      do Screen.setColor(**false**);

      do Screen.drawRectangle(x, y, x + size, y + size);

      return;

   }

    /\*\* Increments the square size by 2 pixels. \*/

   method void incSize() {

      if (((y + size) < 254) & ((x + size) < 510)) {

         do erase();

         let size = size + 2;

         do draw();

      }

      return;

   }

   /\*\* Decrements the square size by 2 pixels. \*/

   method void decSize() {

      if (size > 2) {

         do erase();

         let size = size - 2;

         do draw();

      }

      return;

   }

   /\*\* Moves the square up by 2 pixels. \*/

   method void moveUp() {

      if (y > 1) {

         do Screen.setColor(**false**);

         do Screen.drawRectangle(x, (y + size) - 1, x + size, y + size);

         let y = y - 2;

         do Screen.setColor(**true**);

         do Screen.drawRectangle(x, y, x + size, y + 1);

      }

      return;

   }

   /\*\* Moves the square down by 2 pixels. \*/

   method void moveDown() {

      if ((y + size) < 254) {

         do Screen.setColor(**false**);

         do Screen.drawRectangle(x, y, x + size, y + 1);

         let y = y + 2;

         do Screen.setColor(**true**);

         do Screen.drawRectangle(x, (y + size) - 1, x + size, y + size);

      }

      return;

   }

   /\*\* Moves the square left by 2 pixels. \*/

   method void moveLeft() {

      if (x > 1) {

         do Screen.setColor(**false**);

         do Screen.drawRectangle((x + size) - 1, y, x + size, y + size);

         let x = x - 2;

         do Screen.setColor(**true**);

         do Screen.drawRectangle(x, y, x + 1, y + size);

      }

      return;

   }

   /\*\* Moves the square right by 2 pixels. \*/

   method void moveRight() {

      if ((x + size) < 510) {

         do Screen.setColor(**false**);

         do Screen.drawRectangle(x, y, x + 1, y + size);

         let x = x + 2;

         do Screen.setColor(**true**);

         do Screen.drawRectangle((x + size) - 1, y, x + size, y + size);

      }

      return;

   }

}

*OUTPUT FILE : (Square.xml, Square)*

<tokens>

<keyword> class </keyword>

<identifier> Square </identifier>

<symbol> { </symbol>

<keyword> field </keyword>

<keyword> int </keyword>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> ; </symbol>

<keyword> field </keyword>

<keyword> int </keyword>

<identifier> size </identifier>

<symbol> ; </symbol>

<keyword> constructor </keyword>

<identifier> Square </identifier>

<identifier> new </identifier>

<symbol> ( </symbol>

<keyword> int </keyword>

<identifier> Ax </identifier>

<symbol> , </symbol>

<keyword> int </keyword>

<identifier> Ay </identifier>

<symbol> , </symbol>

<keyword> int </keyword>

<identifier> Asize </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> x </identifier>

<symbol> = </symbol>

<identifier> Ax </identifier>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> y </identifier>

<symbol> = </symbol>

<identifier> Ay </identifier>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> size </identifier>

<symbol> = </symbol>

<identifier> Asize </identifier>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> draw </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<keyword> this </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> dispose </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Memory </identifier>

<symbol> . </symbol>

<identifier> deAlloc </identifier>

<symbol> ( </symbol>

<keyword> this </keyword>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> draw </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<keyword> true </keyword>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> erase </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<keyword> false </keyword>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> incSize </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<symbol> ( </symbol>

<symbol> ( </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> &lt; </symbol>

<integerConstant> 254 </integerConstant>

<symbol> ) </symbol>

<symbol> & </symbol>

<symbol> ( </symbol>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> &lt; </symbol>

<integerConstant> 510 </integerConstant>

<symbol> ) </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> erase </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> size </identifier>

<symbol> = </symbol>

<identifier> size </identifier>

<symbol> + </symbol>

<integerConstant> 2 </integerConstant>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> draw </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> decSize </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> size </identifier>

<identifier> &gt; </identifier>

<integerConstant> 2 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> erase </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> size </identifier>

<symbol> = </symbol>

<identifier> size </identifier>

<symbol> - </symbol>

<integerConstant> 2 </integerConstant>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> draw </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> moveUp </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> y </identifier>

<identifier> &gt; </identifier>

<integerConstant> 1 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<keyword> false </keyword>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<symbol> ( </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> - </symbol>

<integerConstant> 1 </integerConstant>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> y </identifier>

<symbol> = </symbol>

<identifier> y </identifier>

<symbol> - </symbol>

<integerConstant> 2 </integerConstant>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<keyword> true </keyword>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<integerConstant> 1 </integerConstant>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> moveDown </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<symbol> ( </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> &lt; </symbol>

<integerConstant> 254 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<keyword> false </keyword>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<integerConstant> 1 </integerConstant>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> y </identifier>

<symbol> = </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<integerConstant> 2 </integerConstant>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<keyword> true </keyword>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<symbol> ( </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> - </symbol>

<integerConstant> 1 </integerConstant>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> moveLeft </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> x </identifier>

<identifier> &gt; </identifier>

<integerConstant> 1 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<keyword> false </keyword>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> - </symbol>

<integerConstant> 1 </integerConstant>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> x </identifier>

<symbol> = </symbol>

<identifier> x </identifier>

<symbol> - </symbol>

<integerConstant> 2 </integerConstant>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<keyword> true </keyword>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<integerConstant> 1 </integerConstant>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> moveRight </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> &lt; </symbol>

<integerConstant> 510 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<keyword> false </keyword>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<integerConstant> 1 </integerConstant>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> x </identifier>

<symbol> = </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<integerConstant> 2 </integerConstant>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> setColor </identifier>

<symbol> ( </symbol>

<keyword> true </keyword>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Screen </identifier>

<symbol> . </symbol>

<identifier> drawRectangle </identifier>

<symbol> ( </symbol>

<symbol> ( </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> - </symbol>

<integerConstant> 1 </integerConstant>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> , </symbol>

<identifier> x </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> , </symbol>

<identifier> y </identifier>

<symbol> + </symbol>

<identifier> size </identifier>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<symbol> } </symbol>

</tokens>

INPUT FILE: (SquareGame.jack, Square)

// This file is part of www.nand2tetris.org

// and the book "The Elements of Computing Systems"

// by Nisan and Schocken, MIT Press.

// File name: projects/10/Square/SquareGame.jack

// (same as projects/09/Square/SquareGame.jack)

/\*\*

 \* Implements the Square Dance game.

 \* This simple game allows the user to move a black square around

 \* the screen, and change the square's size during the movement.

 \* When the game starts, a square of 30 by 30 pixels is shown at the

 \* top-left corner of the screen. The user controls the square as follows.

 \* The 4 arrow keys are used to move the square up, down, left, and right.

 \* The 'z' and 'x' keys are used, respectively, to decrement and increment

 \* the square's size. The 'q' key is used to quit the game.

 \*/

class SquareGame {

   field Square square; // the square of this game

   field int direction; // the square's current direction:

                        // 0=none, 1=up, 2=down, 3=left, 4=right

   /\*\* Constructs a new Square Game. \*/

   constructor SquareGame new() {

      // Creates a 30 by 30 pixels square and positions it at the top-left

      // of the screen.

      let square = Square.new(0, 0, 30);

      let direction = 0;  // initial state is no movement

      return this;

   }

   /\*\* Disposes this game. \*/

   method void dispose() {

      do square.dispose();

      do Memory.deAlloc(this);

      return;

   }

   /\*\* Moves the square in the current direction. \*/

   method void moveSquare() {

      if (direction = 1) { do square.moveUp(); }

      if (direction = 2) { do square.moveDown(); }

      if (direction = 3) { do square.moveLeft(); }

      if (direction = 4) { do square.moveRight(); }

      do Sys.wait(5);  // delays the next movement

      return;

   }

   /\*\* Runs the game: handles the user's inputs and moves the square accordingly \*/

   method void run() {

      var char key;  // the key currently pressed by the user

      var boolean exit;

      let exit = **false**;

      while (~exit) {

         // waits for a key to be pressed

         while (key = 0) {

            let key = Keyboard.keyPressed();

            do moveSquare();

         }

         if (key = 81)  { let exit = **true**; }     // q key

         if (key = 90)  { do square.decSize(); } // z key

         if (key = 88)  { do square.incSize(); } // x key

         if (key = 131) { let direction = 1; }   // up arrow

         if (key = 133) { let direction = 2; }   // down arrow

         if (key = 130) { let direction = 3; }   // left arrow

         if (key = 132) { let direction = 4; }   // right arrow

         // waits for the key to be released

         while (~(key = 0)) {

            let key = Keyboard.keyPressed();

            do moveSquare();

         }

     } // while

     return;

   }

}

*OUTPUT FILE : (SquareGame.xml, Square)*

<tokens>

<keyword> class </keyword>

<identifier> SquareGame </identifier>

<symbol> { </symbol>

<keyword> field </keyword>

<identifier> Square </identifier>

<identifier> square </identifier>

<symbol> ; </symbol>

<keyword> field </keyword>

<keyword> int </keyword>

<identifier> direction </identifier>

<symbol> ; </symbol>

<keyword> constructor </keyword>

<identifier> SquareGame </identifier>

<identifier> new </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> square </identifier>

<symbol> = </symbol>

<identifier> Square </identifier>

<symbol> . </symbol>

<identifier> new </identifier>

<symbol> ( </symbol>

<integerConstant> 0 </integerConstant>

<symbol> , </symbol>

<integerConstant> 0 </integerConstant>

<symbol> , </symbol>

<integerConstant> 30 </integerConstant>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> direction </identifier>

<symbol> = </symbol>

<integerConstant> 0 </integerConstant>

<symbol> ; </symbol>

<keyword> return </keyword>

<keyword> this </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> dispose </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> dispose </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> Memory </identifier>

<symbol> . </symbol>

<identifier> deAlloc </identifier>

<symbol> ( </symbol>

<keyword> this </keyword>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> moveSquare </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> direction </identifier>

<symbol> = </symbol>

<integerConstant> 1 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> moveUp </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> direction </identifier>

<symbol> = </symbol>

<integerConstant> 2 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> moveDown </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> direction </identifier>

<symbol> = </symbol>

<integerConstant> 3 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> moveLeft </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> direction </identifier>

<symbol> = </symbol>

<integerConstant> 4 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> moveRight </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> do </keyword>

<identifier> Sys </identifier>

<symbol> . </symbol>

<identifier> wait </identifier>

<symbol> ( </symbol>

<integerConstant> 5 </integerConstant>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> method </keyword>

<keyword> void </keyword>

<identifier> run </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> var </keyword>

<keyword> char </keyword>

<identifier> key </identifier>

<symbol> ; </symbol>

<keyword> var </keyword>

<keyword> boolean </keyword>

<identifier> exit </identifier>

<symbol> ; </symbol>

<keyword> let </keyword>

<identifier> exit </identifier>

<symbol> = </symbol>

<keyword> false </keyword>

<symbol> ; </symbol>

<keyword> while </keyword>

<symbol> ( </symbol>

<identifier> ~exit </identifier>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> while </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> = </symbol>

<integerConstant> 0 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> key </identifier>

<symbol> = </symbol>

<identifier> Keyboard </identifier>

<symbol> . </symbol>

<identifier> keyPressed </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> moveSquare </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> = </symbol>

<integerConstant> 81 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> exit </identifier>

<symbol> = </symbol>

<keyword> true </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> = </symbol>

<integerConstant> 90 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> decSize </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> = </symbol>

<integerConstant> 88 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> do </keyword>

<identifier> square </identifier>

<symbol> . </symbol>

<identifier> incSize </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> = </symbol>

<integerConstant> 131 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> direction </identifier>

<symbol> = </symbol>

<integerConstant> 1 </integerConstant>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> = </symbol>

<integerConstant> 133 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> direction </identifier>

<symbol> = </symbol>

<integerConstant> 2 </integerConstant>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> = </symbol>

<integerConstant> 130 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> direction </identifier>

<symbol> = </symbol>

<integerConstant> 3 </integerConstant>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> if </keyword>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> = </symbol>

<integerConstant> 132 </integerConstant>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> direction </identifier>

<symbol> = </symbol>

<integerConstant> 4 </integerConstant>

<symbol> ; </symbol>

<symbol> } </symbol>

<keyword> while </keyword>

<symbol> ( </symbol>

<symbol> ~ </symbol>

<symbol> ( </symbol>

<identifier> key </identifier>

<symbol> = </symbol>

<integerConstant> 0 </integerConstant>

<symbol> ) </symbol>

<symbol> ) </symbol>

<symbol> { </symbol>

<keyword> let </keyword>

<identifier> key </identifier>

<symbol> = </symbol>

<identifier> Keyboard </identifier>

<symbol> . </symbol>

<identifier> keyPressed </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<keyword> do </keyword>

<identifier> moveSquare </identifier>

<symbol> ( </symbol>

<symbol> ) </symbol>

<symbol> ; </symbol>

<symbol> } </symbol>

<symbol> } </symbol>

<keyword> return </keyword>

<symbol> ; </symbol>

<symbol> } </symbol>

<symbol> } </symbol>

</tokens>