Department of Computer Science
Technical University of Cluj-Napoca

Structure of Computer Systems

Laboratory activity 2019-2020

Project title: Flame Cutting Machine

Name: Elekes Lukács Roland
Group: 30434
Email: lukyelekes@gmail.com

Contents

Chapter 1

Introduction

1.1 Context

The goal of this project is to design a program that controls a machine by numerical commands. The machine is a Flame Cutting Machine, which is based on the commands received from the program, it moves on two directions and cuts the objects. This will be simulated on the screen.

This application can be used by people who want to visualize the trajectory of cutting and to check the final result on the given set of numerical commands. The simulation can be visualized on a graphical user interface, where the animation will present the movement of the machine.

1.2 Specification

The flame cutting machine simulation is written in Java object oriented programming language (using IntelliJ IDE), and the visualization is realized with Java Swing. The user should specify the commands in a simple text file, that will be the input file. Commands should be simple having 3 simple parameters, what type of trajectory to do (line or arc), the start point and end point. Points are descried in 2D, where O(0,0) is the upper left corner of the screen.

The user can command the following segments:

1. horizontal line
2. vertical line
3. semiarc in horizontal direction
4. semiarc in vertical direction

It is the user responsibility what commands gives to the machine.

Remark. *When a user defines an arc, he can specify the direction to be clockwise or not.*

1.3 Objectives

The main objective is to design and implement the simulation of a machine described before. This objective leads to secondary objectives:

1. Reading numerical commands from external files

2. Processing numerical commands
3. Using graphics
4. Creating animations

Chapter 2

Bibliographic Study

The research stage of the project consisted mainly of finding ways to implement the specification of the problem. The simplest way to simulate such a cutter machine was to draw the corresponding lines of the movements.

For this I have read about and analized the Java Swing library which is suitable for creating a graphical user interface. For drawing I looked for classes and methods that help drawing lines or arcs. Having these methods I had to adjust the written command's parameters to the method parameters, to make as simple as possible to use the application for a new user, thus the user does not have to think about special parameters like upper left point of the rectangle in which the arc is, angles, etc.

Chapter 3

Analysis

The Flame Cutting Machine can do a set of operations and they have to be defined. The numerical command should contain this information somehow, so this lead to the conclusion that we have define a format that contains the type of the movement that the machine will do and from which to which point. The set of commands for a simulation should be stored in a text file that is accessible for the program.

Before drawing on the screen, all the data has to be processed and adjusted to the methods that maintain the graphics part.
In the processing phase the data should be extracted from the text file and processed according to what type of command is the actual one. After this step the commands should be put in a set with adjusted parameters for the drawing methods.

For drawing an empty window is used, so anywhere on the screen lines and arcs can be drawn depending on the commands. Every command is drawn and animated separately to visuallize the functioning of the machine. An animation is a rapidly and repeatedly drawn images, in our case line sequences, so a line or an arc must be divided into small sequences, that are drawn successively one after another.

The direction of the movements can be normal (when starting point is closer to origo) or reverse (when ending point is closer to the origo). This is important when we create the animation, since we have to adjust the drawing methods to start points.

Chapter 4

Design

The format of the command in text files are simple, having only three parameters it is not difficult to write such parameters for a user. The format is the following:

type start end

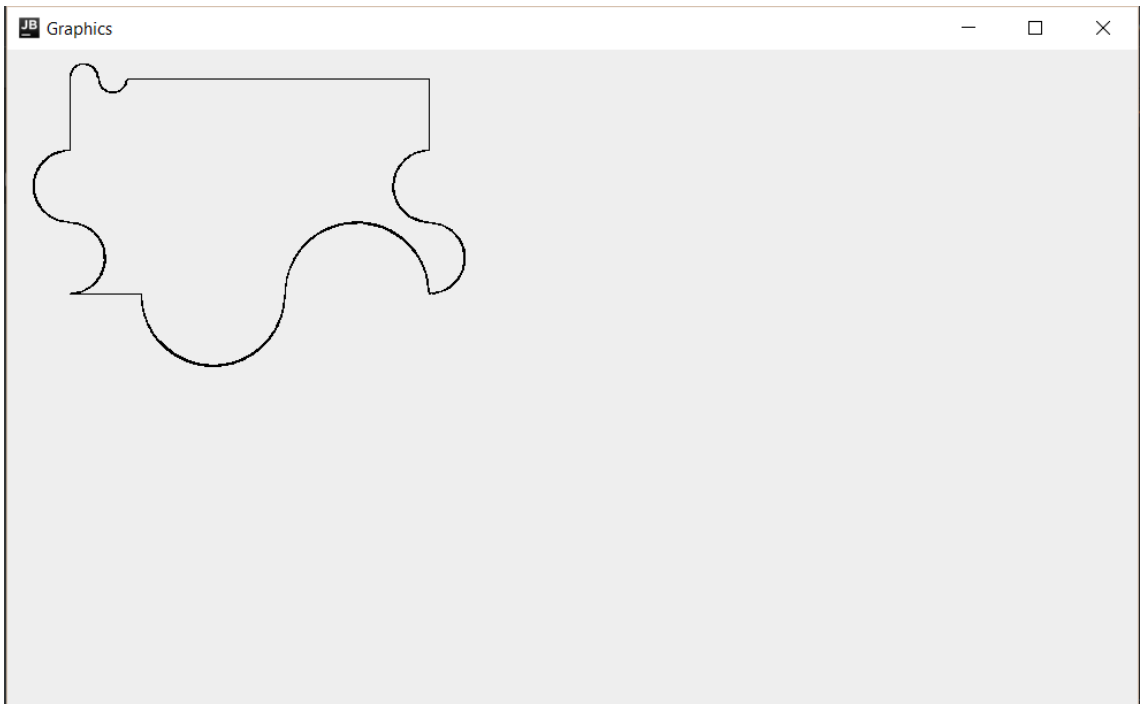
where **type** defines what type of movement should the machine do, **start** is the starting point and **end** is the ending point, both points having two values, X and Y parameters of the point.
Types are marked by one or two characters, they are the following: **l** (line), **ay** (a for arc, yes for clockwise), **an** (a for arc, n for not clockwise). A complete example:

l 50 50 50 100

The design is object-oriented and it is structured on packages. The packages named *models* contains all the classes that represent model from the application. Talking about design, two models would be enough, a Point and a Command consisting of two points, but a third model was created, namely Arc which made the future implementations easier. A Point has two attributes, the X and Y parameters of the point.
A Command contains two points, which are enough for drawing only lines, but additional two attributes were created to specify if the shape is an arc and if it is drawn ciockwise or counterclockwise.
The Arc has in addition to two points several attributes that define a semiarc, however this class has more importance at implementation and not at the design.

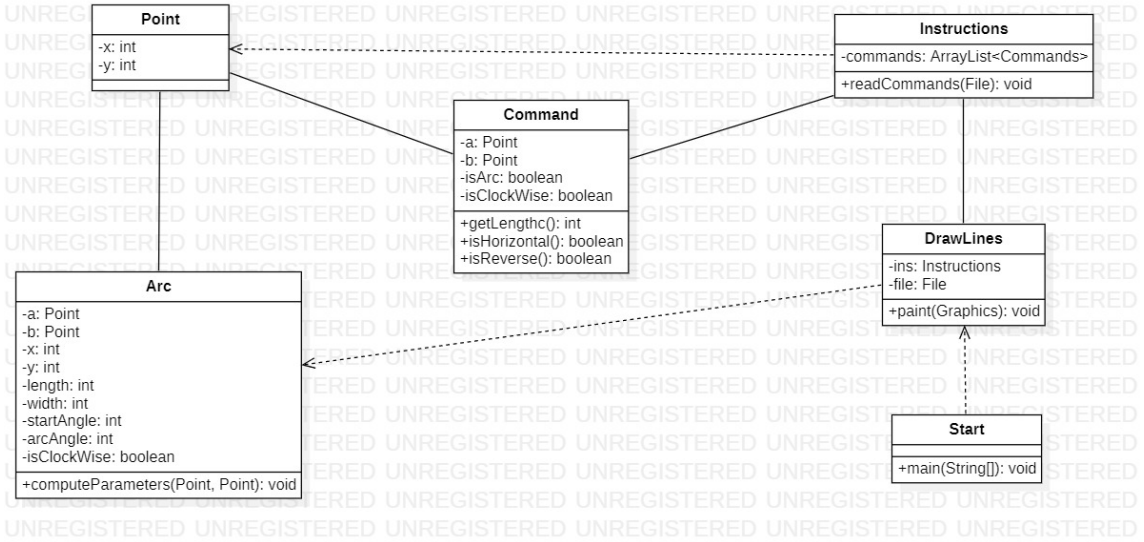
Moreover, a utility class was defined, where we do the processing of data, opening and reading from a file and deciding what type of object to create and add in the set of commands.

There is a single class representing the **view** package called DrawLines, this helps creating a window and space for the simulation and it does the graphics and animation part of the simulation.



A picture from the finished simulation and the shape that was cut.

The class diagram is the following:



Chapter 5

Implementation

Point is a simple class having the two attributes and setters/getters.

```
package models;

public class Point {
    private int x;
    private int y;

    public Point (int x, int y){
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
}
```

The Command class is similarly simple, it contains some simple methods that give information of the sequence to be drawn like the length or the its type. The constructor allows us to create the command and know wether it is an arc or not, it is clockwise or not.

```
 public Command(Point a, Point b, boolean arc, boolean isClockWise){
        this.a = a;
        this.b = b;
        isArc = arc;
        this.isClockWise = isClockWise;
    }

public int getLength(){
        if(a.getY() == b.getY()){
            return Math.abs(b.getX() - a.getX());
        }
        else{
            return Math.abs(b.getY() - a.getY());
        }
```

```
    }

    public boolean isHorizontal(){
        if(a.getY() == b.getY()){
            return true;
        }
        else return false;
    }

    public boolean isReverse(){
        if(a.getX() > b.getX() || a.getY() > b.getY()){
            return true;
        }
        return false;
    }

    public boolean isClockWise() {
        return isClockWise;
    }
```

These information are defined in the Instruction class, where we read the commands using BufferedReader[1] and pattern matching[2] to get the numbers from the commands. The type is checked and the type of the sequence to be drawn is defined, after that the numbers are extracted from the line read from the file. Having the numbers and the type, we can build a new command and add it to the set of commands.

```
public class Instructions {
    private ArrayList<Command> commands;

    public Instructions(){
        this.commands = new ArrayList<Command>();
    }

    public void readCommands(File file) throws Exception {
        BufferedReader br = new BufferedReader(new FileReader(file));

        String st;
        Pattern p = Pattern.compile("[0-9]+");

        while((st = br.readLine()) != null){
            Matcher m = p.matcher(st);
            Point a, b;
            int num1 = 0;
            int num2 = 0 ;
            boolean isArc = false;
            boolean isClockWise = true;

            if(st.charAt(0) == 'l'){
                isArc = false;
            }
            if(st.charAt(1) == 'y'){
                isClockWise = true;
            }
```

```
            if(st.charAt(1) == 'n'){
                isClockWise = false;
            }
            if(st.charAt(0) == 'a'){
                isArc = true;
            }

            m.find();
            num1 = Integer.parseInt(m.group());
            m.find();
            num2 = Integer.parseInt(m.group());
            a = new Point(num1, num2);

            m.find();
            num1 = Integer.parseInt(m.group());
            m.find();
            num2 = Integer.parseInt(m.group());
            b = new Point(num1, num2);

            commands.add(new Command(a, b, isArc, isClockWise));
            //commands.add(readLine(st));
        }
    }
}
```

The Arc class computes its parameters based on the two points that are part of the command. This attributes are the parameters of the drawArc() method, which of the following form[3]:

public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)

x - the x coordinate of the upper-left corner of the arc to be drawn.
y - the y coordinate of the upper-left corner of the arc to be drawn.
width - the width of the arc to be drawn.
height - the height of the arc to be drawn.
startAngle - the beginning angle.
arcAngle - the angular extent of the arc, relative to the start angle.

The method that computes this values adjusts them according to the stard and end point of the command.

The DrawLines class contains the method to paint the window. Every command is taken one by one and is drawn on the screen, but they are divided into smaller sequences, and with the use of Thread.sleep() method, an animation was created by the successive drawings.
The methods that are used are drawLine()[4] and drawArc().

A Start class appears also, where the main method is placed to be able to start the application.

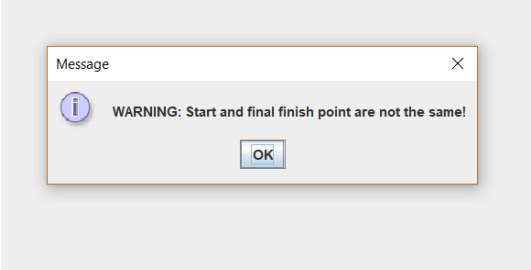
Chapter 6

Testing and Validation

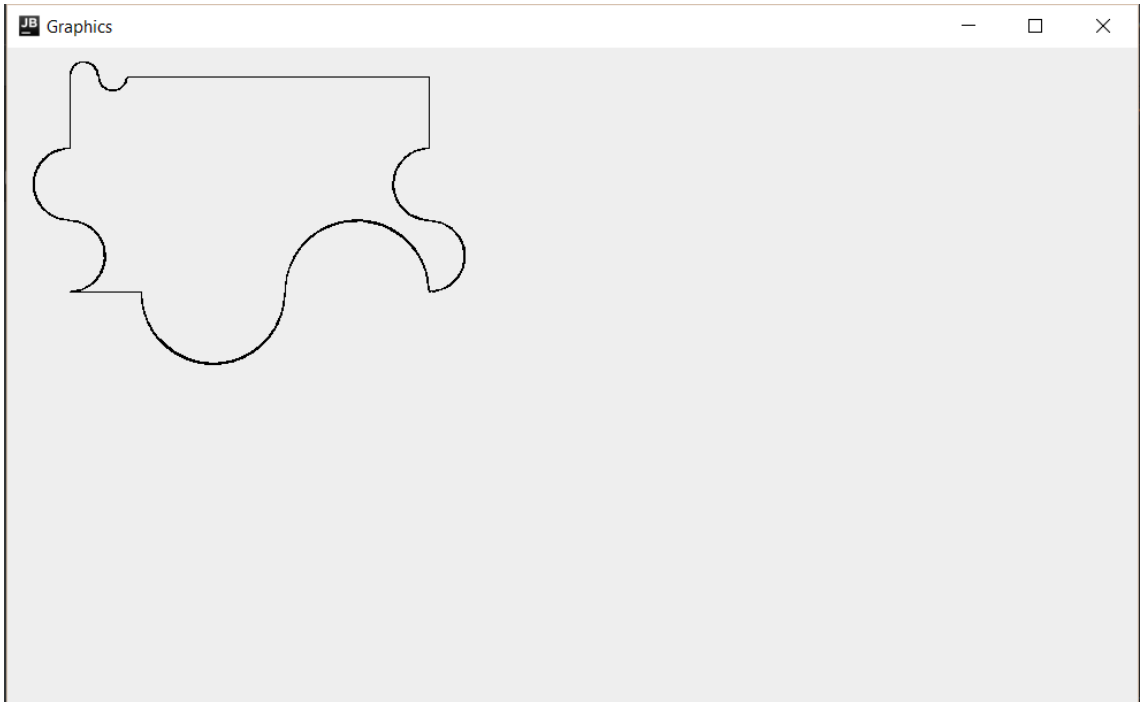
For testing some methods were used, for example toString methods of several classes to check if the objects were created correctly, the numbers were read correctly from the file etc. Outputs on the console were the main testing references during the developement of the application.

At the final stage every possible type of commands were tested, and refactoring was done until the animations were correct.

The user has the responsibility to write correct commands and what type of commands he writes. If the very first point is not the same of the very last point, then only a warning message is shown to the user.



A Message Dialog Box that warns the user, when the start and final point are not the same.



A simulation example that contains every type of commands.

Chapter 7

Conclusion

The objective of the project was to create a simulation for the Flame Cutting Machine, which helped me to be familiarized with some new aspects of programming. Drawing on the screen and animation were completely new concepts, but now I know more about them and how to use them to visualize some movements on the screen using animation.

Another conclusion would be, that after the design the difficulty of the implementation can depend a lot on the programming language that we choose to work with. In Java there are methods and libraries, that helped me reaching my objectives.

It is interesting the some aspects of the design are discovered during implementation, it happens many times that we have to refactor, think again about things or the redesign some moduls.

Future developements:

- Creating GUI for the user
- Interactive system, where the user can give commands from the GUI
- Advanced mode, where the user can specify all the parameters for an arc, thus more complex shapes can be cut

Chapter 8

References

[1]https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html
[2]https://stackoverflow.com/a/13440560/10961194
[3]https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html#drawArc
https://way2java.com/awt-graphics/drawing-arcs/
[4]https://www.geeksforgeeks.org/java-applet-draw-a-line-using-drawline-method/