

University of Swansea

FACULTY OF COMPUTER SCIENCE

MSc in Advanced Computer Science

2015/2016

Richard Williams 882230

Donal Evans 404737

Wang Qiru 689404

**“2D Constructive Solid Geometry for CSCM94”**

Supervisor : Dr. Benjamin Mora

Date of Submission : May 2016

# Abstract

This paper will detail the work of Group 4 Solutions and a piece of software which is able to display images using a simple input from a user. This paper uses software engineering principles to understand the user’s requirements and design a working piece of software which fulfils those requirements. The program is written using the java programming language and is done without the help of many of the inbuilt libraries. This paper goes through the techniques used to design and test the software to ensure that it meets the user requirements given. The result was a working program with a user friendly interface which exceeds the user’s initial requirements in several ways and is able to handle many complex user inputs.

# Contents

[Abstract 1](#_Toc450148448)

[Contents i](#_Toc450148449)

[Introduction to the document 1](#_Toc450148450)

[Description of the company 2](#_Toc450148451)

[Quality section on tools 3](#_Toc450148452)

[Deployment environments 4](#_Toc450148453)

[Extra Features 5](#_Toc450148454)

[Schedule 6](#_Toc450148455)

[User Requirements 7](#_Toc450148456)

[Preliminary design 8](#_Toc450148457)

[Test Sections 11](#_Toc450148458)

[Results section 13](#_Toc450148459)

[Conclusion 19](#_Toc450148460)

[Appendix 1 - Minutes 20](#_Toc450148461)

[Appendix 2 - Code 29](#_Toc450148462)

[Package: bean 29](#_Toc450148463)

[Circle.java 29](#_Toc450148464)

[Closedcurve.java 31](#_Toc450148465)

[Difference.java 37](#_Toc450148466)

[Intersection.java 40](#_Toc450148467)

[Mix.java 43](#_Toc450148468)

[Node.java 43](#_Toc450148469)

[Rectangle.java 45](#_Toc450148470)

[Rotate.java 48](#_Toc450148471)

[Scale.java 51](#_Toc450148472)

[Shape.java 55](#_Toc450148473)

[Square.java 55](#_Toc450148474)

[Transform.java 58](#_Toc450148475)

[Translate.java 59](#_Toc450148476)

[Triangle.java 62](#_Toc450148477)

[Union.java 65](#_Toc450148478)

[Package: userinterface 68](#_Toc450148479)

[CodingPanel.java 68](#_Toc450148480)

[DrawingFrame.java 69](#_Toc450148481)

[DrawingPanel.java 81](#_Toc450148482)

[ErrorFrame.java 86](#_Toc450148483)

[HelpFrame.java 88](#_Toc450148484)

[Package: util 92](#_Toc450148485)

[CubicBezler.java 92](#_Toc450148486)

[LoadFile.java 100](#_Toc450148487)

[Parser.java 103](#_Toc450148488)

[SaveFile.java 122](#_Toc450148489)

[Appendix 3 – Contribution Report 124](#_Toc450148490)

# Introduction to the document

This document will describe how Group 4 Solutions have created a working piece of software using good software engineering techniques. First the document will discuss the company set up and how each individual operates within it. It will then continue to describe the tools that were used and how they were quality checked. You will see a full list of user requirements which have been gathered with the client. The document then gets more technical and will show the preliminary design which will include the class diagrams, class members and details of the functions.

There is a section detailing the test suite which can be used to make sure that all the code operates within parameters. We will also have a results section which will detail what the software is capable of, it will use screenshots and we will expand on the processes. Finally, there will be a conclusion which will sum up the process that we have gone through and critically describe what could have gone better or the things that went well.

By the end of this document you will have a strong idea of the process’s that the group encountered and the ways in which each member has contributed to the outcome. You will be able to see all of the minutes for the meetings and also see the way in which our software works.

# Description of the company

Group 4 solutions is a new company made up of four individuals from different backgrounds to provide turnkey solutions to your software needs. We have around ten years coding experience within Java in academia and industry, working on numerous different types of projects. We have a strong background in delivering solutions on time and within budget by using strong software engineering techniques. As a company we believe that to give the client what they need we must first understand the requirements and we take a large amount of time making sure that we know what the requirements are. Using our experience in industry we are able to talk to business’s about their company and help them to accurately describe what they want. Once we know what is required we are able to use our industry and academic experience to find elegant and sophisticated solutions which we are able to deliver on time.

The company is owned and operated by four equal shareholders all with differing experiences which combine to make Group 4 solutions a high value proposition for business’s looking at their software needs. Wang is the technical director and has many years’ experience working on technical projects in Java in academia and industry. He drives the company forward looking at cutting edge techniques to ensure that we deliver the best and most suitable products. Donal has been in academia for many years working on complex problems in physics and maths. He has strong problem solving skills and is our testing director, he is responsible for making sure that all of products have no bugs and fulfil the requirements. Richard has many years’ industrial experience working for the end user, he is skilled in project management and organisation/administration. His main tasks involve liaising with all the parties and ensuring the brief is correct and keeping lines of communication open all the way through the project cycle.

# Quality section on tools

Design for the project will be done in Visual Paradigm, a UML-based software that is widely used and has multiple versions depending on the budget of the user and the intended use to which it will be put, including a free version, which will be used for this project. Visual Paradigm supports multiple types of diagrams and is more than sufficient for the purposes of this project, since it will be unlikely that the program produced will be exceptionally complex.

The language that will be used to program this project is Java, which is one of the most widely used programming languages in the world, being free, extremely well documented and the subject of one of the modules studied by the team in the previous semester, meaning that all team members should share at least a basic level of familiarity with it. Java is also capable of producing programs that will run on multiple different operating systems, which will increase the utility of the final product.

The programming of the project will be completed in Eclipse, a Java-based IDE that is very widely used and with which one member of the team is very familiar, which should prevent time being lost to becoming familiar with the tool. Eclipse is free and well-documented, and has both Mac and Windows versions, which will allow all members of the team to work on the programming using their own computers.

Version control and source code management for the project will be managed via GitHub, a Git repository hosting service that is very widely used, free, and with which a member of the team is very familiar, once again reducing time lost to learning how to use a new tool. GitHub also provides a desktop client for both Windows and Mac which allows synchronization with the remote repository and uploading of local changes, which should enable all members of the team to easily contribute to the code and follow changes made by others.

For GUI design and implementation within the project, the built-in Java AWT library will be used. Being a part of the standard Java API for providing GUIs, AWT is very well-supported and requires no additional installation, which will streamline the process for creating a GUI.

# Deployment environments

The software has been tested on Windows 7, Windows 8 and Windows 10 with the following configurations:

* Java JDK 64-bit (build 1.8.0\_65-b17).
* Eclipse Luna (4.4) and Mars (4.5).

The software is not working on all versions of Mac OS X after OS X 10.8.2 due to Java library compatibility, as Apple controls and distributes modified version of Java on all Macs and stopped supporting AWT since Java 7.

# Extra Features

Within the given timeframe, we were able to meet all user requirements, so we decided to add on more features to further improve our program.

List of extra features:

* Two different approaches for drawing: Area and Pixel by Pixel
* The ability to draw closed curves.
* The ability to choose drawing colours.
* Auto-complete parenthesis
* Error feedback

# Schedule

To enable Group 4 Solutions to meet the deadline and deliver the working software time and consideration were put into the schedule. We followed the suggested schedule rigidly and always tried to get as far ahead as possible to ensure there was extra time in case something was to go wrong. We planned most of the work to be done around 8 weeks;

1. Getting familiar with the group and understanding each person’s role. Discussing the tools which we could all use to progress.
2. Finalise the tools, start working on the preliminary design.
3. Class Diagram, State Diagram and abstract data types completed.
4. Test suite and revised design.
5. Coding.
6. Coding.
7. Coding and debugging.
8. Testing.

# User Requirements

* A 2D CGS model of a shape or set of shapes that is stored internally as a tree.
* The tree shall consist of nodes which shall be either shapes, transforms or mixes. It should be possible to create a new tree, to modify an existing tree by adding or removing nodes within it and to delete an existing tree.
* It should be possible to create new nodes and to delete existing nodes.
* Shapes shall be either circles, squares or triangles defined by their radius, rectangles defined by their horizontal and vertical radii or closed curves constructed from connected vertices and/or Bezier curve segments. All shapes shall have their default position at the origin of the 2D coordinate system in which they exist. A shape node shall always be a leaf node of the tree.
* Transforms shall be either rotation by some angle in radians, scaling by some real number factor in x and y or translation in x and y by some real number amount. It should also be possible to apply a particular transform to a node multiple times with a single instruction. A transform shall be applied to one existing node.
* Mixes shall be either the + (Union), - (Difference) or & (Intersection) of two or more existing nodes.
* There shall be some user interface that allows interacting with a text representation of the tree and also shows a graphical representation of the tree as a 2D image.
* It should be possible to save the current state as a file and to load it back anytime. Each time a saved file is loaded, it should be possible to resume editing the drawing.

# Preliminary design

Given the relatively specific nature of the user requirements in terms of internal data structure, design of the program focused mainly on the mechanisms of class interaction rather than overall structure. The primary consideration during the design phase was what sort of information would need to be passed between classes in order to facilitate drawing. It was decided that each node would contain references to its children (if any) and information relating to itself, rather than explicitly recording all information needed to draw the image represented by the node. It was decided that every node would have a draw() method and a print() method that would recursively traverse the tree in postfix order to allow drawing an image to the screen and printing a text representation of the tree respectively.

In addition to the classes that would comprise the image tree, it was decided that some utility classes were also needed. These consisted of LoadFile and SaveFile, which are responsible for loading and saving trees, Parser, which is responsible for parsing user text input and generating a tree from it, as well as checking that input for syntactic correctness, and CubicBezier, which contains methods needed for handling cubic Bezier curves in the ClosedCurve class. These classes provide various functionality to the program, but are not directly involved in the internal representation of the tree or its drawing.

The final major consideration in designing the program was some form of GUI. It was decided that a main window containing a drawing canvas and a text box for user input would be used, along with Save, Load, Draw, Clear and Help buttons. The Help button would produce a secondary window containing explanations of the various possible instructions and the correct syntax to use them. A mock-up of the proposed layout of the GUI is shown below.

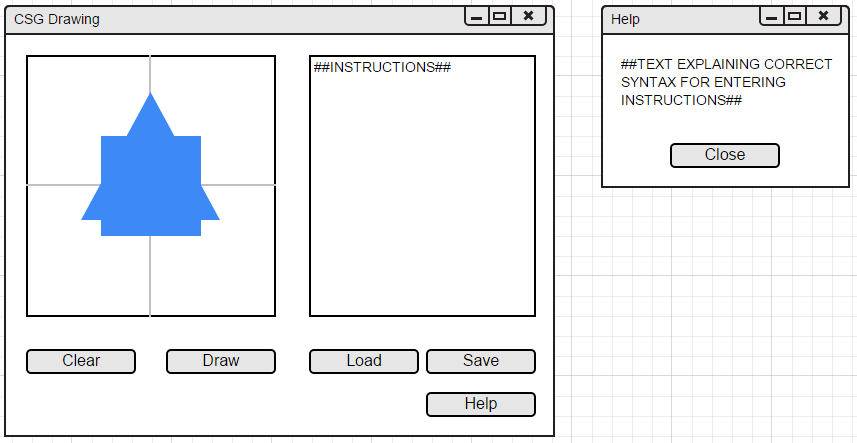
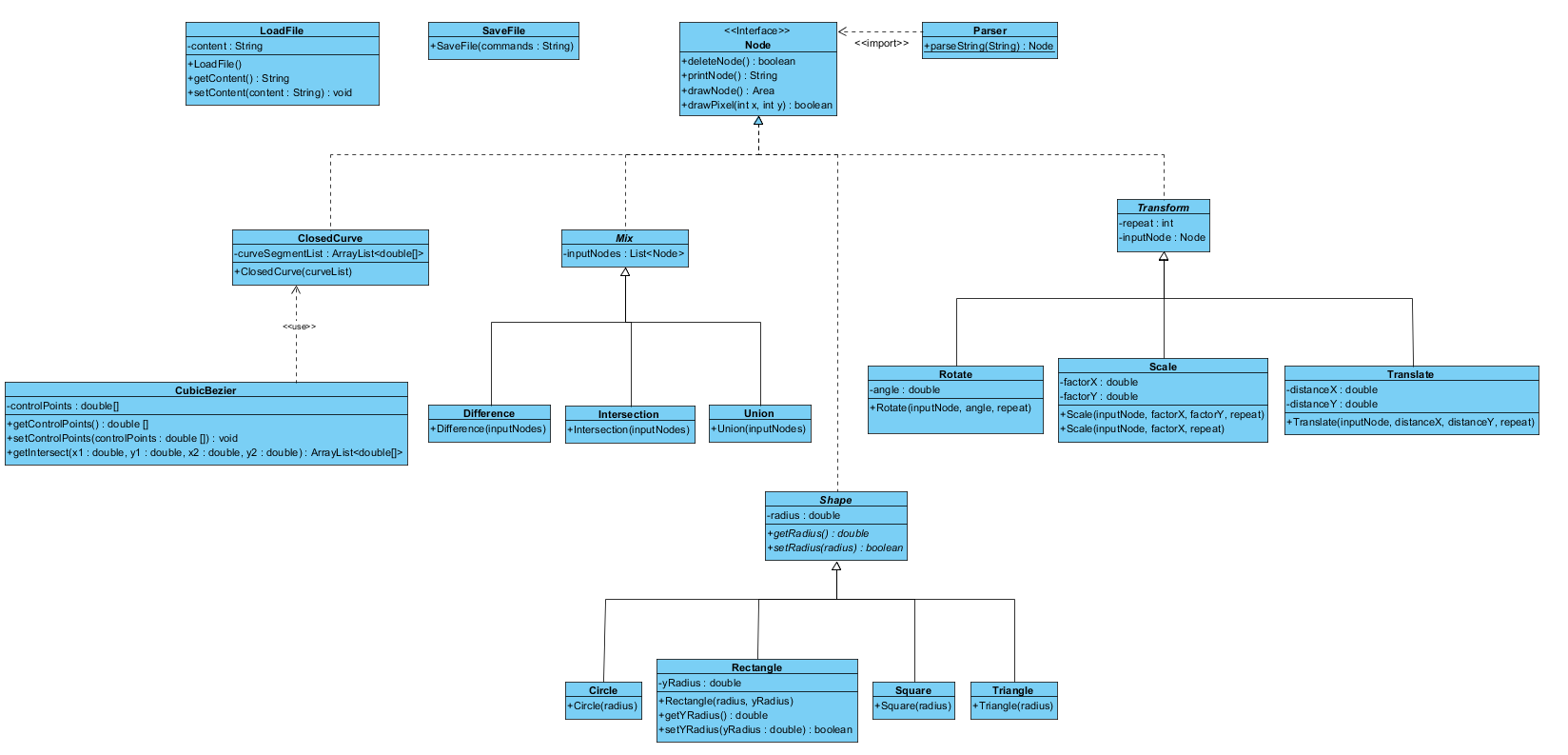


Figure : Proposed GUI design

After the completion of the basic class structure design, a more detailed pass was made and the various methods and member variables that would be required were designed. Two draw() methods were designed for nodes; one employing a “is the given pixel inside the shape?” approach and one using built-in classes in Java that greatly simplified the required mathematics and coding. The latter approach was intended to be attempted first in order to create a working prototype and confirm the viability of the overall design, with the former approach being added later if time allowed. The pixel-by-pixel drawing method would employ a ray casting algorithm and winding number algorithm to determine if a given pixel lies within a complex shape such as a closed curve, and use simple geometry to determine if a given pixel lies within regular polygons. The transforms would be handled using matrix multiplication and the mixes using simple Boolean operators (OR for Union, AND for Intersection, AND NOT for Difference) on the argument nodes.

A class diagram detailing the structure of the program and the methods and member variables for each class is shown below.



# Test Sections

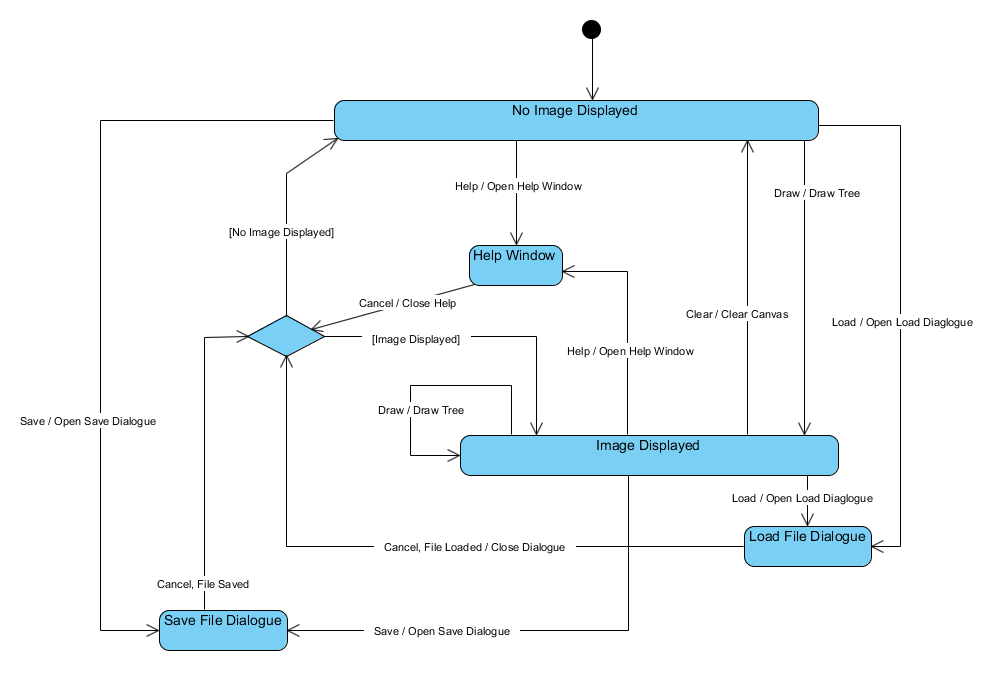
A state diagram representing the GUI, shown below, was created to enable testing of the expected behaviour of the application.

Figure 2: GUI state diagram

From the state diagram it can be seen that the application should start with a blank canvas (no image displayed) and that it should be possible to open the load file dialogue by pressing “load”, the save file dialogue by pressing “save” and the help window by pressing “help” from this state, and to return to it from them. It should also be possible to enter the “image displayed” state by pressing the “Draw” button, providing a valid tree has been entered. Likewise, from the “image displayed” state, it should be possible to open the load file dialogue, the save file dialogue and the help window, and to return to it from them; that is, the displayed image should not be removed when the “load”, “save” or “help” buttons are pressed. It should also be possible to return to the “no image displayed” state by pressing the “clear” button, and to redraw the current image or draw a new image by pressing the “draw” button.

As well as testing the behaviour of the GUI, it is also important to test the operation of the program as a whole by confirming that certain inputs conform to the expected outputs. As there are in theory an infinite number of possible combinations of instructions that can be input by the user of the system, it is not possible to exhaustively test all cases. Furthermore, since the expected output of the system consists of geometric shapes printed to the screen, it is difficult to quantitatively confirm that any given output exactly matches that expected for a given input. However, a qualitative test suite can be created that confirms that the system conforms to the expected behaviour when considering the general nature of the output.

Test1: Square(50) should produce a square at the centre of the drawing area with sides 71 pixels long.

Test2: Circle(50) should produce a circle at the centre of the drawing area with a diameter of 100 pixels.

Test3: Triangle(50) should produce an equilateral triangle at the centre of the drawing area with side length 87 pixels.

Test4: Rotate(Square(50),45) should produce a square with sides 71 pixels long at the centre of the drawing area that has been rotated 45 degrees anti-clockwise.

Test5: Translate(Circle(50),25,25) should produce a circle with a diameter of 100 pixels centred on the point 25 pixels above and 25 pixels to the right of the centre of the drawing area.

Test6: Scale(Triangle(50),2,4) should produce a isosceles triangle with base 173 pixels and height 300 pixels at the centre of the drawing area.

Test7: Union(Triangle(50),Square(30)) should produce an overlaid image of a square with sides 42 pixels long and an equilateral triangle with sides 87 pixels long, both at the centre of the drawing area.

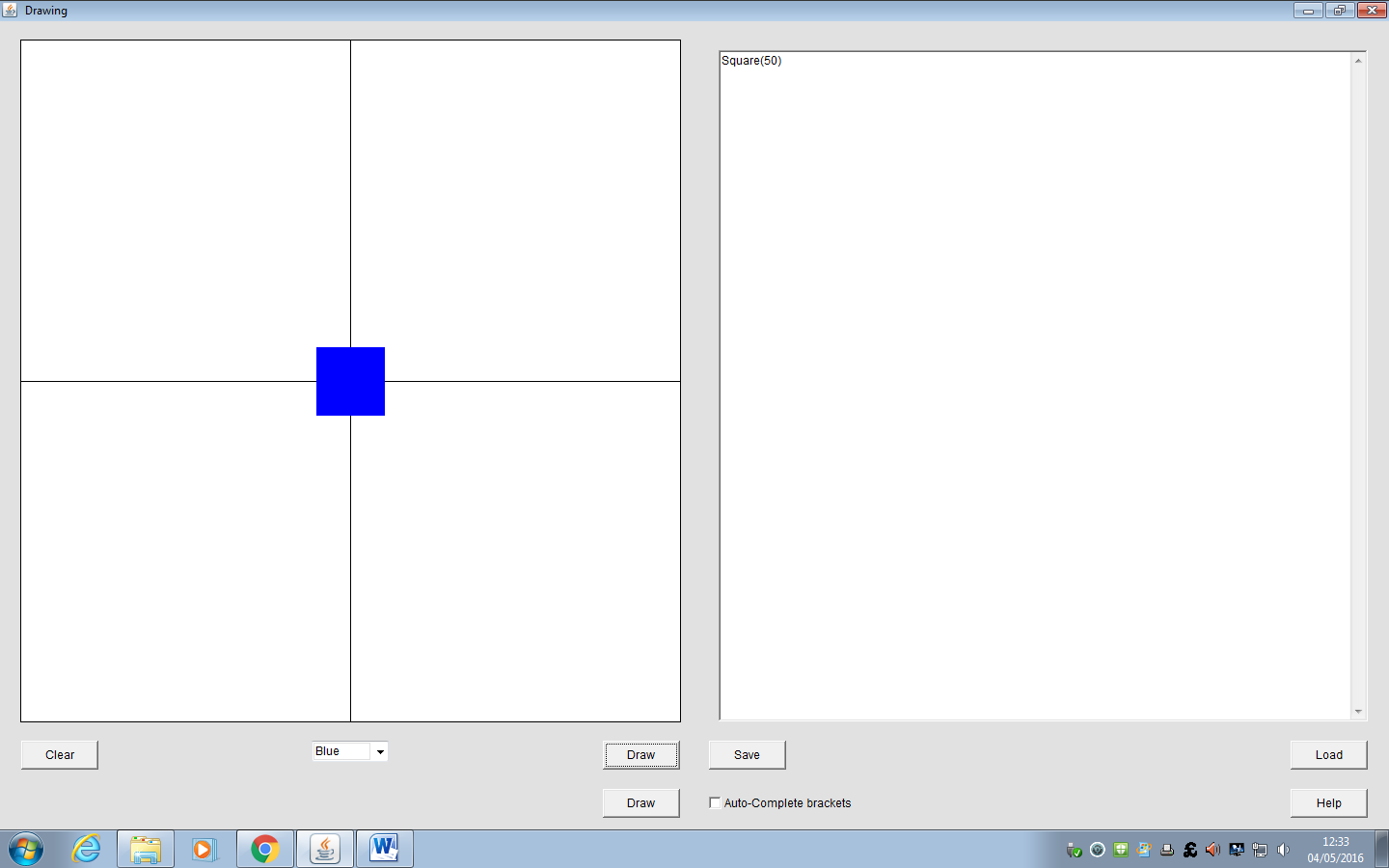
Test8: Intersection(Circle(50),Square(55)) should produce a square with rounded corners, with the diameter of the rounded portion being 100 pixels and the overall height and width of the shape being 85 pixels, positioned at the centre of the drawing area.

Test9: Difference(Circle(50),Square(50)) should produce a circle with diameter 100 pixels with, removed from the centre of it, a square with side length 71 pixels (such that the vertices of the square are on the circumference of the circle), both at the centre of the drawing area.

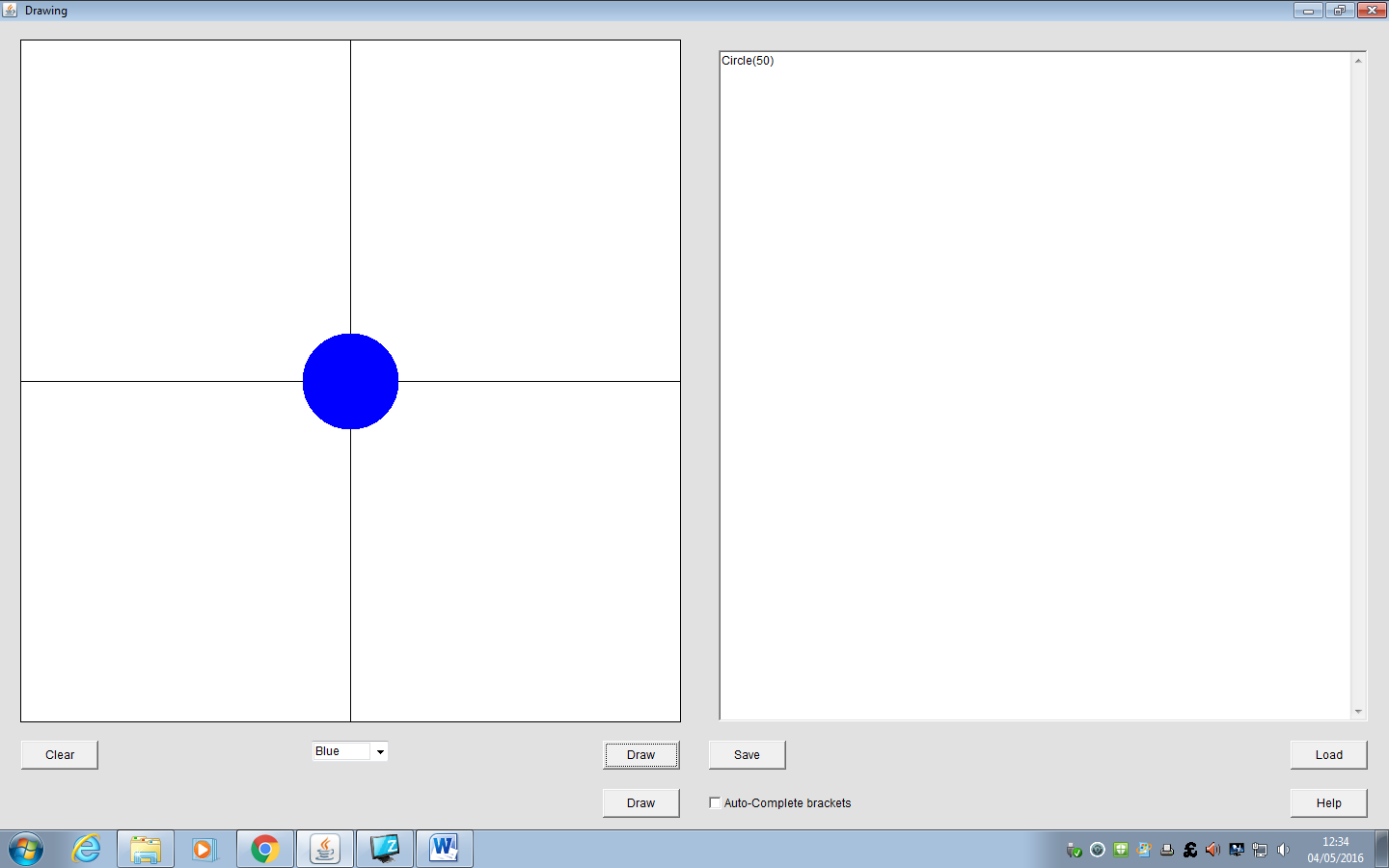
With this very basic test suite it is possible to confirm that each node type can correctly be displayed, and that every type of shape node can be correctly passed to a transform or mix node as an argument. For invalid inputs (such as syntax errors in instructions or arguments outside the valid input domains), error messages will be displayed to the user. A basic implementation of error reporting will indicate only that an error occurred with the input, but more advanced versions will specify the nature of the error.

# Results section

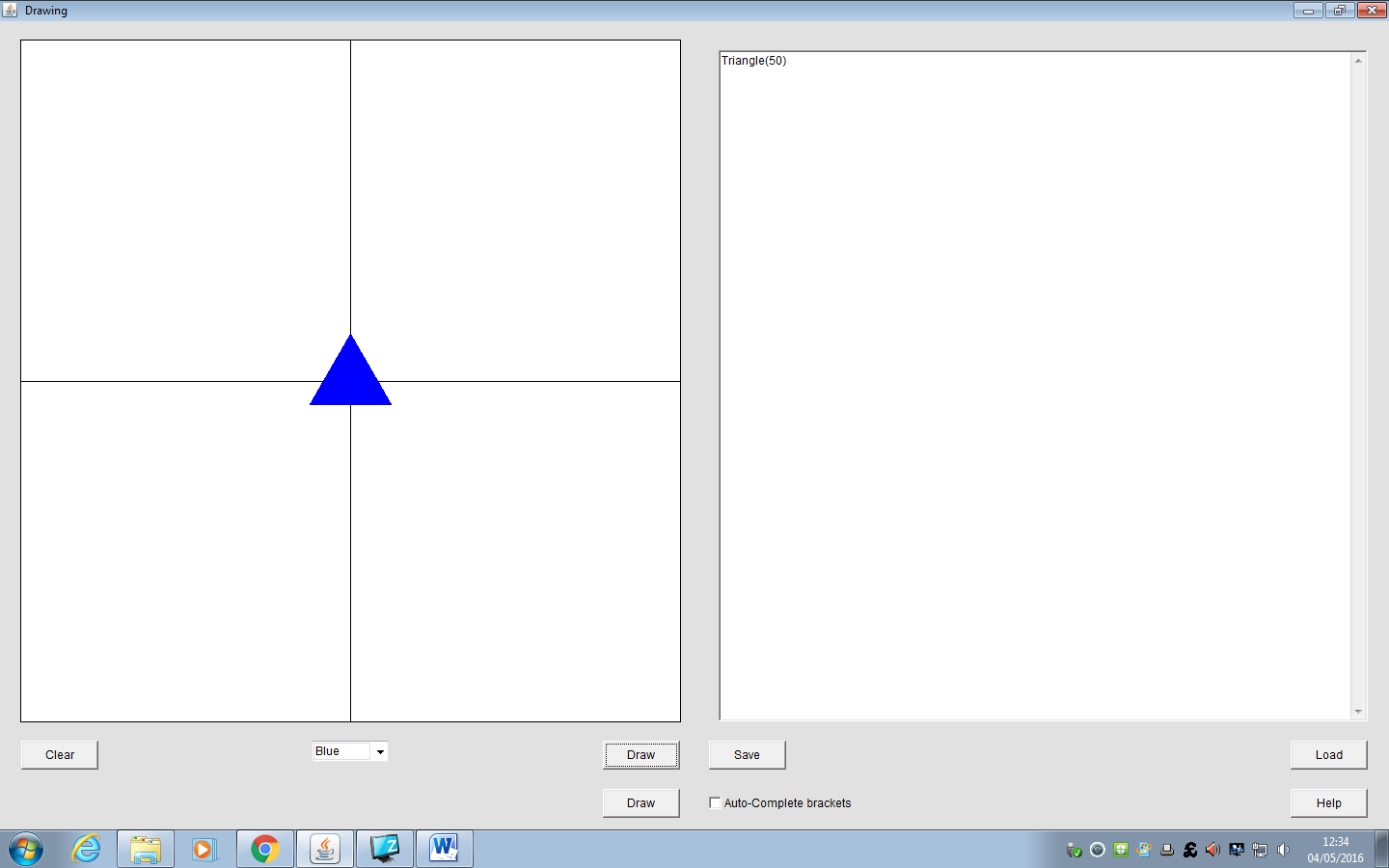
The following section will show with screen shots the results from the test suite. There will be a description of what the image contains.

Test1: Square(50) produced a square at the centre of the drawing area with sides 71 pixels long. 

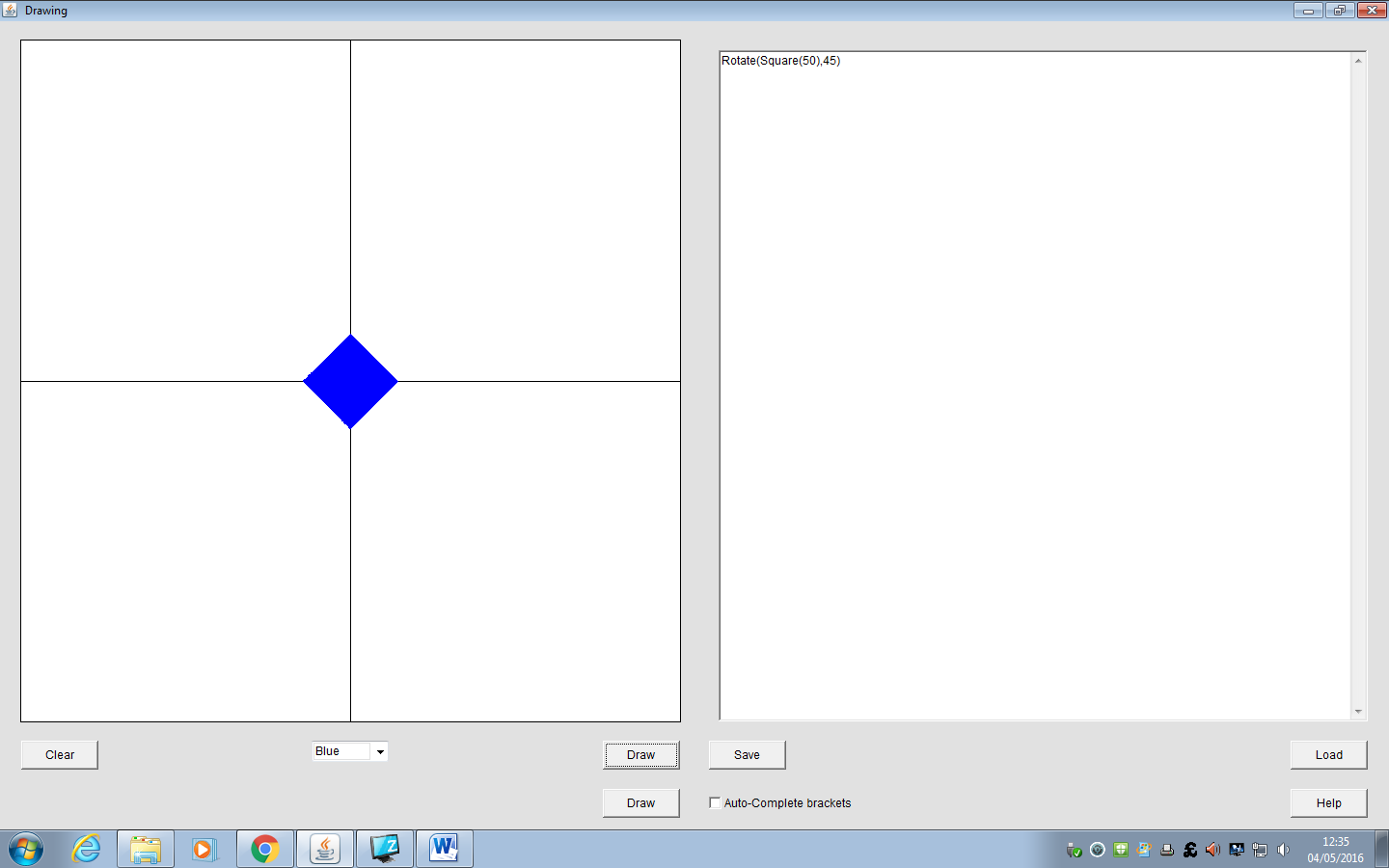
Test2: Circle(50) produced a circle at the centre of the drawing area with a diameter of 100 pixels.



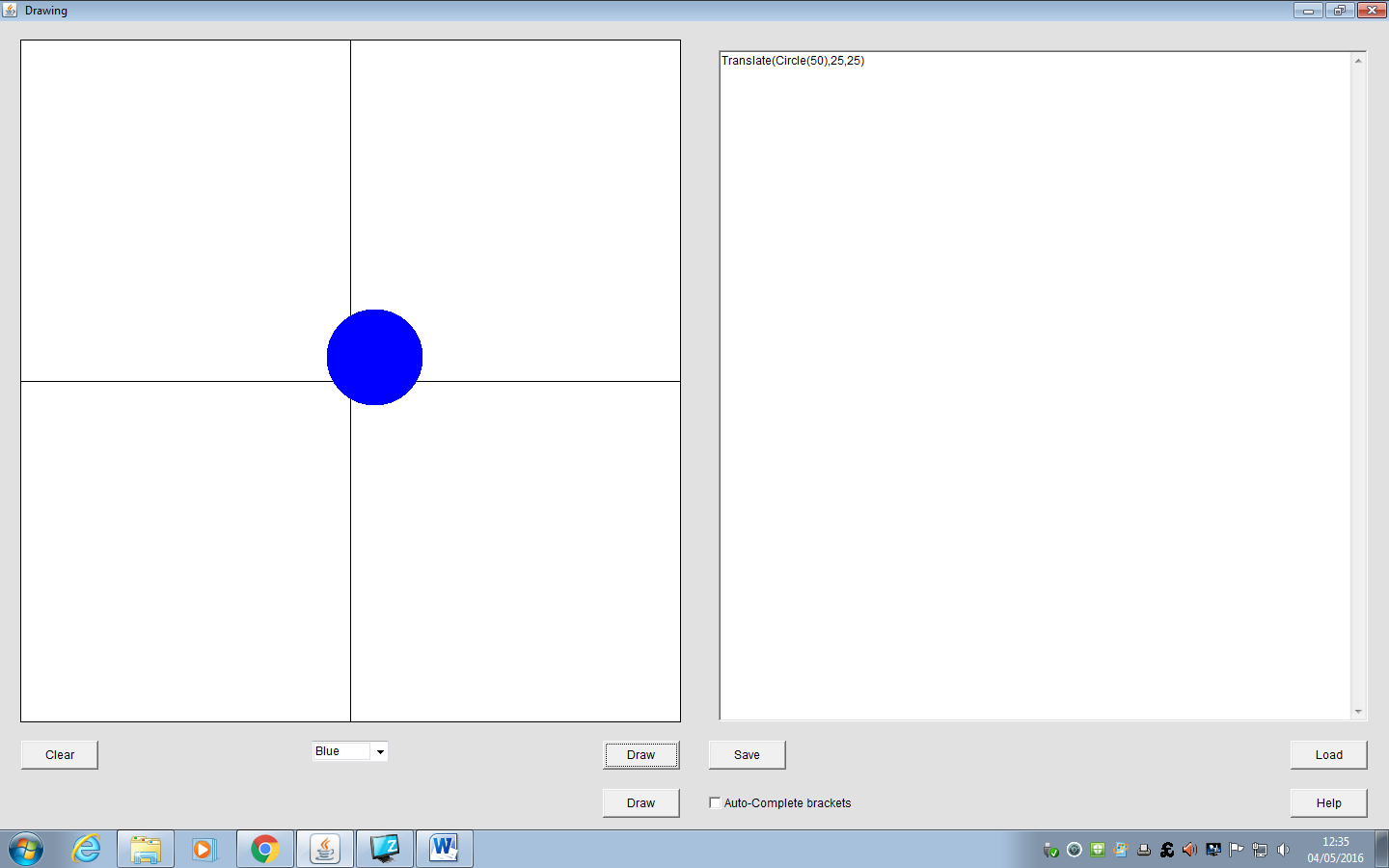
Test3: Triangle(50) produced an equilateral triangle at the centre of the drawing area with side length 87 pixels.



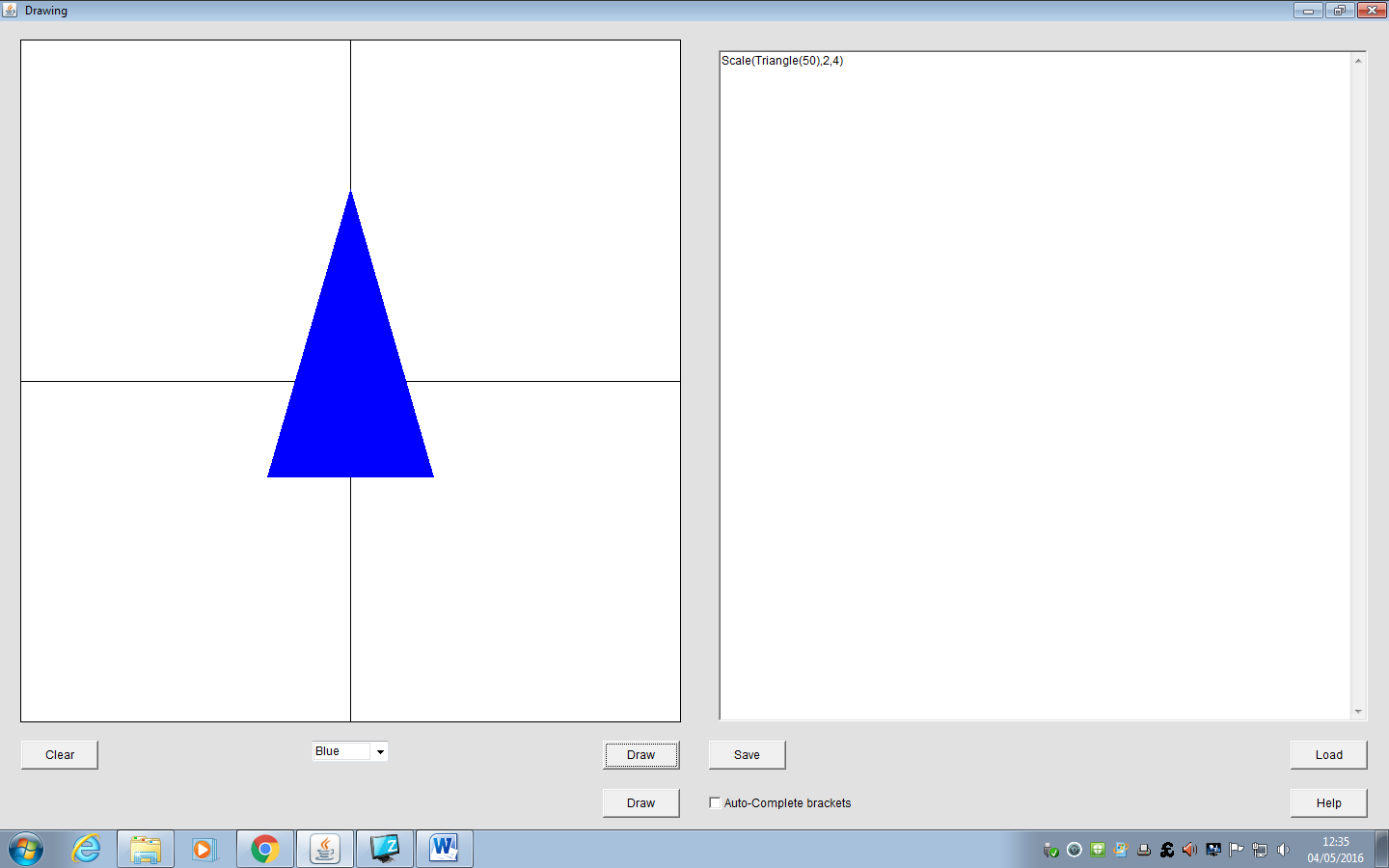
Test4: Rotate(Square(50),45) produced a square with sides 71 pixels long at the centre of the drawing area that has been rotated 45 degrees anti-clockwise.



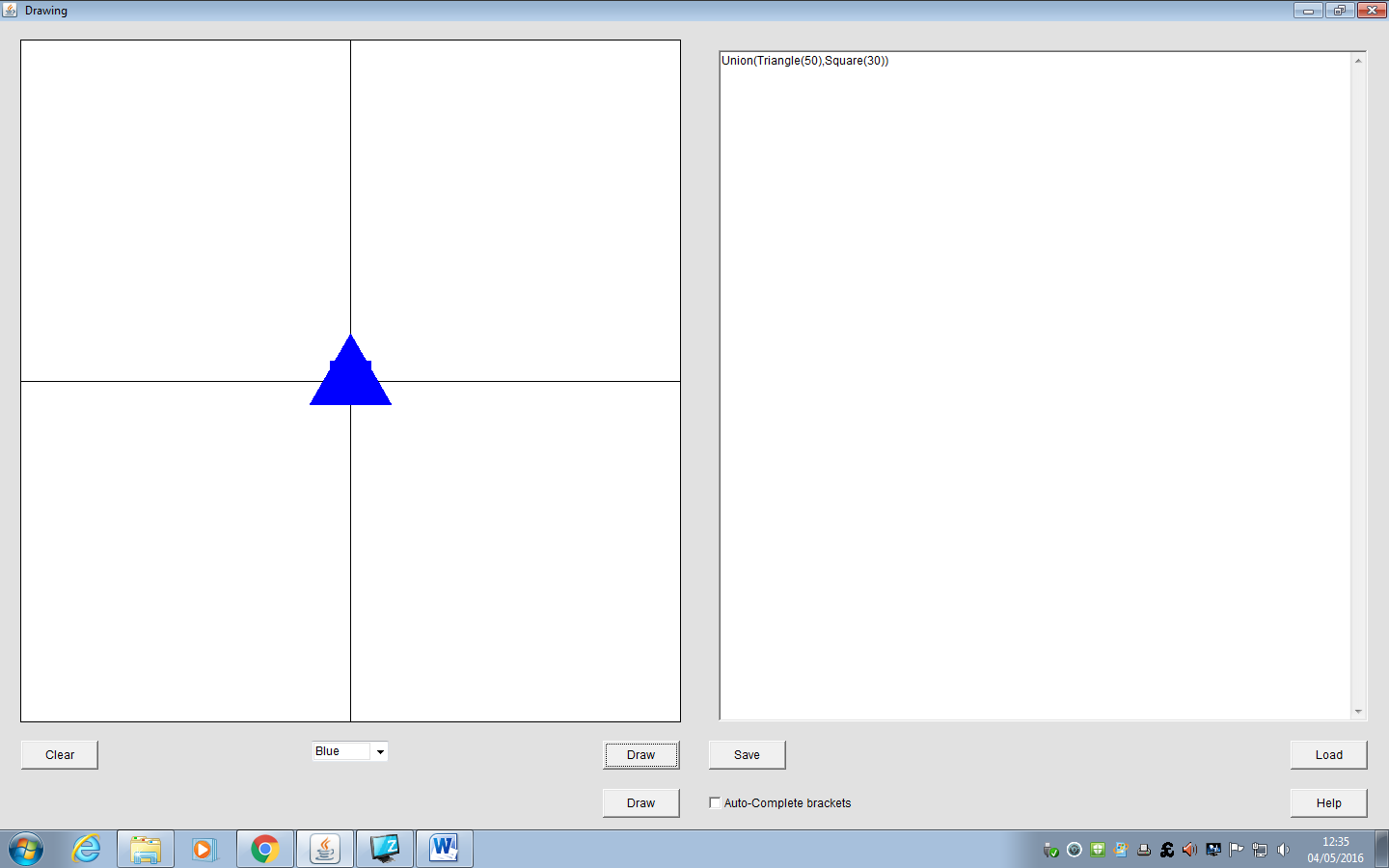
Test5: Translate(Circle(50),25,25) produced a circle with a diameter of 100 pixels centred on the point 25 pixels above and 25 pixels to the right of the centre of the drawing area.



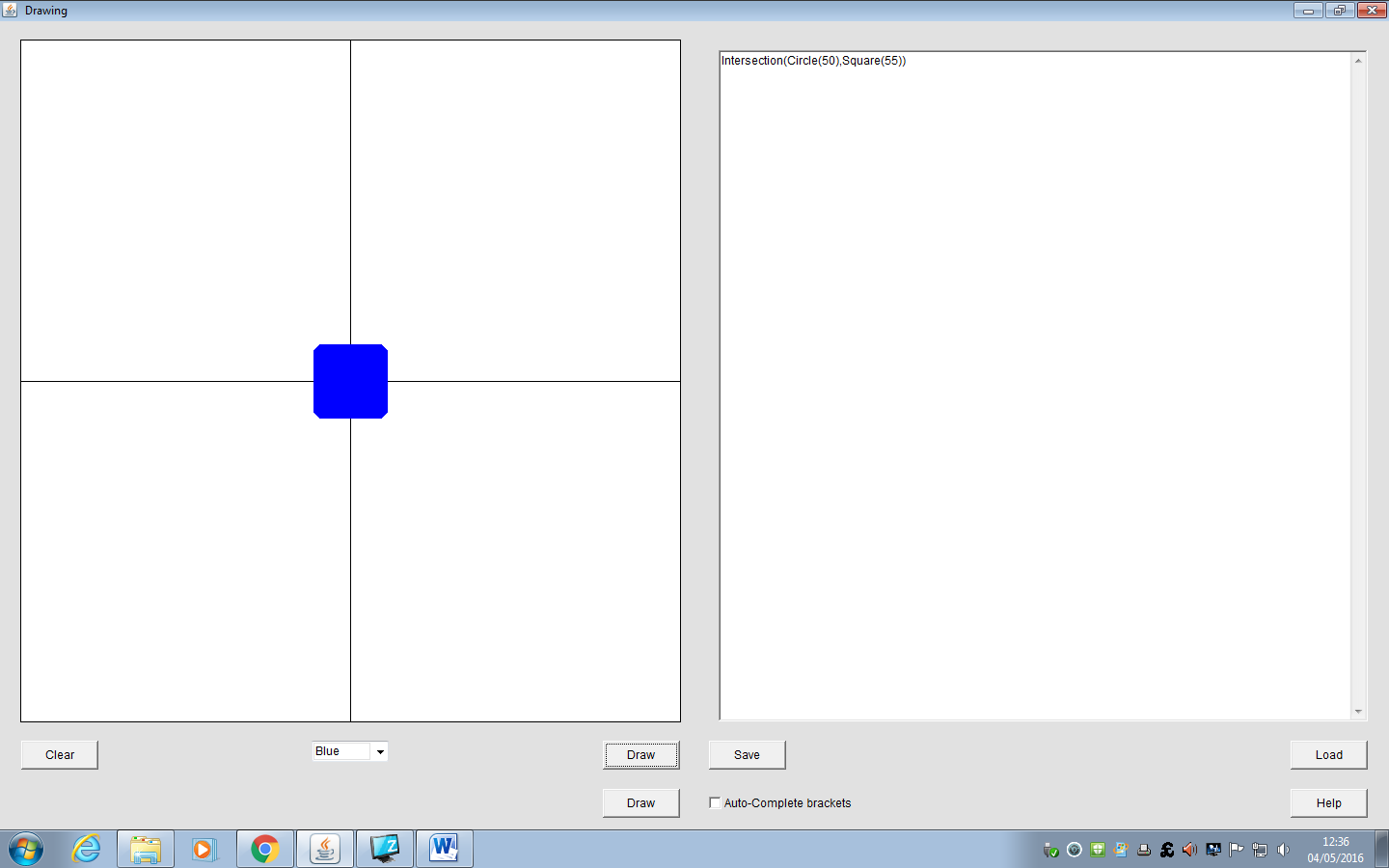
Test6: Scale(Triangle(50),2,4) produced a isosceles triangle with base 173 pixels and height 300 pixels at the centre of the drawing area.



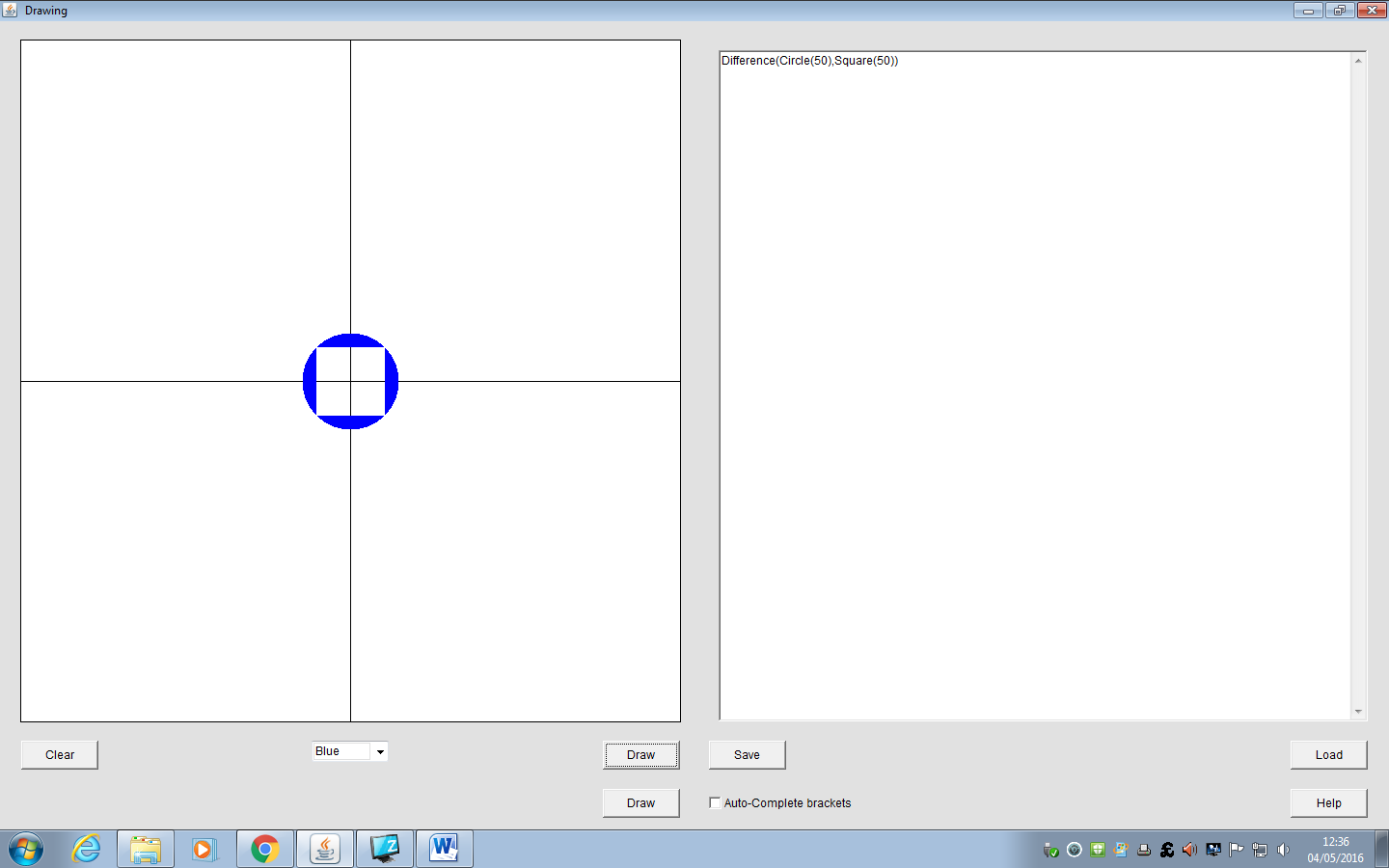
Test7: Union(Triangle(50),Square(30)) produced an overlaid image of a square with sides 42 pixels long and an equilateral triangle with sides 87 pixels long, both at the centre of the drawing area.



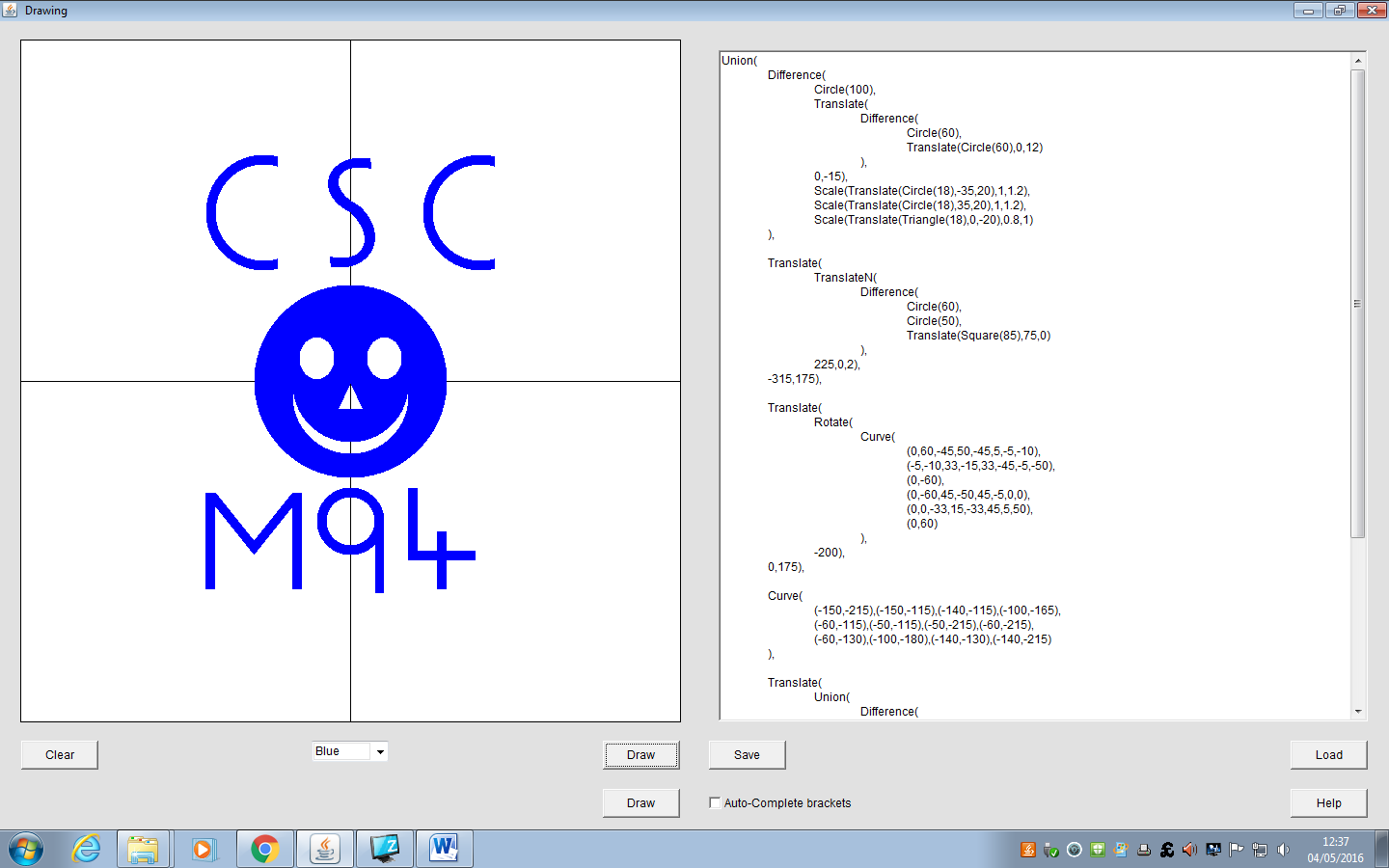
Test8: Intersection(Circle(50),Square(55)) produced a square with rounded corners, with the diameter of the rounded portion being 100 pixels and the overall height and width of the shape being 85 pixels, positioned at the centre of the drawing area.



Test9: Difference(Circle(50),Square(50)) produced a circle with diameter 100 pixels with, removed from the centre of it, a square with side length 71 pixels (such that the vertices of the square are on the circumference of the circle), both at the centre of the drawing area.



This final image showcases what the program can do. It shows all but two of the feautures in the program. The program can run very quickly even with complex code inputs such as this.



# Conclusion

To conclude, this document has gone through Group 4 Solutions efforts to create a working piece of software which not only meets the client’s expectations but exceeds them in some features. We have discussed the company structure and the tools that have been used to create this software. By using good tools that we quality checked it has really enhanced the ability of the group to work together and ensure that our ideas were easily shared. Understanding the user requirements was central to the project design as was using good software engineering techniques to make sure the design was correct before the implementation began. The preliminary design was very important and having the class diagram organised correctly helped to ensure that the implementation went smoothly.

We discussed the test suite that was in place to ensure that all of the elements of the code worked correctly. This is important to ensure that we delivered a working piece of software that would not crash in the user’s environment. The results section showed using screenshots the program working so that you have an idea of the type of work you are able to do with this software. Finally, there is a long list of attachments which detail the meeting held between the group and also the code that was used with a report from each individual member of the group on what they did in the project and what effect that had on the final project.

There were several things to take from this project and they were very important, the way the group did their tasks on time and kept to the schedule. The way the group worked together was great and the communication was constant. Getting the class diagram correct and understanding the project helped so much in the later development stage and meant that the debugging was far easier than I would have been otherwise. Getting a good and thorough understanding of the user requirements enabled us to create a piece of software that works well and fast but did everything it needed to and more.

# Appendix 1 - Minutes

|  |  |
| --- | --- |
|  | Meeting Minutes |

Call to order

**A meeting of**Software Engineering Group 4**was held at**Faraday J**on**04/02/2016**.**

Attendees

**Attendees included**Richard, Donal and Wang**.**

**Members not in attendance included** Donavan.

Reports

Agreed 11am Thursday Faraday J as a regular meeting and discussed the following;

* Read the notes from previous term on software engineering
* Minutes to be taken and typed up and circulated weekly. Using word but sent in PDF.
* We will use the Java language and use Eclipse, version control and git hub.

Strengths and weaknesses of team members

* Donal’s strengths – coding, debugging, team work, problem solving. Physics background good with logic and Maths
* Wang’s strengths – 6 years programming experience, java experience, debugging, coding and software testing.
* Rich’s Strengths – Administration, project management, testing.

Unfinished business

Donavan’s strengths and weaknesses

New business

We agreed to install the software that we are using and familiarise ourselves before the next meeting.

|  |  |
| --- | --- |
|  | Meeting Minutes |

Call to order

**A meeting of**Software Engineering Group 4**was held at**Faraday H**on**11/02/2016**.**

Attendees

**Attendees included**Richard, Donal Donovan and Wang**.**

Reports

* All installed IDE’s and familiarised ourselves with them
* Rich to work on the description of the company
* Donal to work on the list of formal specifications
* Wang to work on the prelim class diagram
* Donovan to work on the introduction and the gant chart
* Started to discuss the interface and how we want it to look text over tick box’s
* Discussed whether the triangles will all be equilateral or not
* Spoke about the origin of the shapes. Centre, bottom left etc
* We will install Doxygen

Unfinished business

Need to ask Ben what is meant by Quality and Tools?

New business

We agreed to get a draft copy of the sections of the report by next Thursdays meeting

Announcements

None

|  |  |
| --- | --- |
|  | Meeting Minutes |

Call to order

**A meeting of**Software Engineering Group 4**was held at**Faraday H and Library**on**18/02/2016**.**

Attendees

**Attendees included** Donal and Wang.

**Members not in attendance**

 Richard and Donovan

Reports

* Discussed Class Diagram
* Tree structure
* Completed User requirements

Unfinished business

Need to ask Ben if scaling operation is uniform or different in x and y? Need clarification on the complexity of the test suite.

New business

Need to install github and WindowBuilder

Donal to work on State Diagramand possibly on the test suite

Wang to work on the class diagram

Announcements

None

|  |  |
| --- | --- |
|  | Meeting Minutes |

Call to order

**A meeting of**Software Engineering Group 4**was held at**Faraday J**on**25/02/2016**.**

Attendees

**Attendees included**Richard, Donal and Wang**.**

**Members not in attendance**

 Donovan

Reports

* All present have installed github and can use it
* Completed prelim state diagram
* Completed prelim class diagram
* All docs now in github including final document and minutes
* Formal user requirements completed
* Chaser email sent to Donovan re gant chart
* We will need to redraw the entire tree each time not just the most recent node

Unfinished business

Need clarification on the complexity of the test suite.

Discussion re class diagram and interfaces

New business

Rich to work on the introduction and the quality and tools section

Announcements

None

|  |  |
| --- | --- |
|  | Meeting Minutes |

Call to order

**A meeting of**Software Engineering Group 4**was held at**Faraday J**on**03/03/2016**.**

Attendees

**Attendees included** Richard and Donal.

**Members not in attendance included**

Wang and Donovan

Reports

* Rich still to complete the intro and quality sections of the final doc. To be completed by the meeting on the 17/3
* Still no reply from Donovan
* Test suite needs to be for most important Class’s and can then be modified for others
* Print save to be added to the class diagram
* Ben signed off on the rest of the class diagram
* Need to redistribute Donovan’s work
* Discussed how busy with other commitments we are for the next couple of weeks and that this may have to take a hit to get past other deadlines
* Clarified that the difference function was (A,B) = A+B

Unfinished business

New business

Announcements

None

|  |  |
| --- | --- |
|  | Meeting Minutes |

Call to order

**A meeting of**Software Engineering Group 4**was held at**Windows Lab**on**10/03/2016**.**

Attendees

**Attendees included**Richard, Wang and Donal**.**

**Members not in attendance**

Donovan

Reports

* Looking at some provisional coding talking about some of the issues with it.
* Discussing issues with rotate function
* Discussing interface, how do we have a text input over the button interface and how it will work.
* Implemented a text box mock up and discussed the syntax to be used
* Talking about whether we use stacks and recursion
* Shapes to have colour once drawn

Unfinished business

* Wang to update the class diagram with parse function and the save, load and print.

New business

* Design complete now so Donal will work on getting a test suite for the 17/3

Announcements

* Wang spoke with Donovan and he advised will be back in from the 14/3

|  |  |
| --- | --- |
|  | Meeting Minutes |

Call to order

**A meeting of**Software Engineering Group 4**was held at**Windows Lab**on**17/03/2016**.**

Attendees

**Attendees included** Richard and Donal.

**Members not in attendance**

Donovan and Wang

Reports

* Wang gave apologies as ill
* No word from Donovan re meeting or the work that he was assigned
* Donal completed the test suite
* Went through the code as it stands and discussed how it works and what could be done

Unfinished business

* Wang to update the class diagram with parse function and the save, load and print.
* Rich to complete the final document sections

New business

* Rich to try and implement a try and catch error system

Announcements

None

|  |  |
| --- | --- |
|  | Meeting Minutes |

Call to order

**A meeting of**Software Engineering Group 4**was held at**Windows Lab**on**14/04/2016**.**

Attendees

**Attendees included**Richard, Wang and Donal**.**

**Members not in attendance**

Donovan

Reports

* Program works and fulfils specification
* Donal may add polygon feature
* Wang may add undo/redo feature
* Rich did quality/intro sections in doc
* Try and catch completed
* Wang finished the diagram
* Donal completed final touches to the code

Unfinished business

* Need to do some testing

New business

* Rich to look at conclusion and results section
* Donal and Wang to look at code and see if any improvements can be made

Announcements

None

|  |  |
| --- | --- |
|  | Meeting Minutes |

Call to order

**A meeting of**Software Engineering Group 4**was held at**Windows Lab**on**22/04/2016**.**

Attendees

**Attendees included**Donal and Wang.

**Members not in attendance**

 Donovan, Richard

Reports

* The thorough error checking in the Parser class was completed
* The error frame was implemented
* It was decided not to proceed with the proposed parser improvement from meeting 8 due to conflicts with the error checking process

Unfinished business

* Splitting the Parser class up into individual methods for each instruction (this may be scrapped depending on time/difficulty)
* Implementing N-sided regular polygons

New business

* Documentation for completed classes

Announcements

None

# Appendix 2 - Code

## Package: bean

### Circle.java

|  |
| --- |
| package bean; |
|  |  |
|  | import java.awt.geom.Area; |
|  | import java.awt.geom.Ellipse2D; |
|  | /\*\* |
|  | \* @class Circle |
|  | \* This Circle class serves as an object. |
|  | \* Is inherited from Shape. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public class Circle extends Shape { |
|  |  |
|  | public Circle(double radius){ |
|  | this.radius=radius; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean deleteNode() { |
|  | // TODO Auto-generated method stub |
|  | return false; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean setRadius(double radius) { |
|  | this.radius=radius; |
|  | return false; |
|  | } |
|  |  |
|  | @Override |
|  | public double getRadius() { |
|  | return this.radius; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public String print() { |
|  | return "Circle(" + this.radius + ")"; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean drawPixel(double x, double y) { |
|  | if ((x\*x) + (y\*y) < this.radius\*this.radius){ |
|  | return true; |
|  | } |
|  | return false; |
|  |  |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public Area draw() { |
|  | Ellipse2D.Double shape = new Ellipse2D.Double(-this.radius,-this.radius,2\*this.radius,2\*this.radius); |
|  | return new Area(shape); |
|  | } |
|  |  |
|  | } |

### Closedcurve.java

|  |
| --- |
| package bean; |
|  |  |
|  | import java.awt.geom.Area; |
|  | import java.awt.geom.CubicCurve2D; |
|  | import java.awt.geom.Path2D; |
|  | import java.util.ArrayList; |
|  | import java.util.Arrays; |
|  |  |
|  | import util.CubicBezier; |
|  | /\*\* |
|  | \* @class ClosedCurve |
|  | \* This ClosedCurve class serves as an implementation of Node. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public class ClosedCurve implements Node{ |
|  |  |
|  | private ArrayList<double[]> curvePoints; |
|  |  |
|  | public ClosedCurve(ArrayList<double[]> curveSegmentList){ |
|  | this.curvePoints = curveSegmentList; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean deleteNode() { |
|  | // TODO Auto-generated method stub |
|  | return false; |
|  | } |
|  |  |
|  | @Override |
|  | public String print() { |
|  | String curveList = Arrays.toString(this.curvePoints.get(0)); |
|  | for (int i = 1; i < this.curvePoints.size(); ++i){ |
|  | curveList = curveList.concat("," + Arrays.toString(this.curvePoints.get(i))); |
|  | } |
|  | //Converting the square brackets from Arrays.toString() into curved brackets |
|  | curveList = curveList.replace("[", "(").replace("]", ")"); |
|  | return "Curve(" + curveList + ")"; |
|  | } |
|  |  |
|  | @Override |
|  | public Area draw() { |
|  | //Create an empty Path2D |
|  | Path2D path = new Path2D.Double(); |
|  | //For every double array in curveList |
|  | for (int i = 0; i < this.curvePoints.size(); ++i){ |
|  | double[] segment = this.curvePoints.get(i); |
|  | //If the current segment is a Bezier curve, add the curve to the path |
|  | if (segment.length == 8){ |
|  | //Create a cubic Bezier curve using the arguments in args[] |
|  | CubicCurve2D curve = new CubicCurve2D.Double(segment[0],segment[1],segment[2],segment[3],segment[4],segment[5],segment[6],segment[7]); |
|  | //Add that curve to the existing path |
|  | path.append(curve, true); |
|  | } |
|  | //If the current segment is the first point in the curve, add a line to that point to the path |
|  | else if (i == 0){ |
|  | path.moveTo(segment[0], segment[1]); |
|  | } |
|  | //If the current segment is a point, add a line to that point to the path |
|  | else { |
|  | path.lineTo(segment[0], segment[1]); |
|  | } |
|  | } |
|  | //Draw the path |
|  | return new Area(path); |
|  | } |
|  |  |
|  | @Override |
|  | public boolean drawPixel(double x, double y) { |
|  | //First generate a list of vertices from the points |
|  | ArrayList<double[]> vertexList = new ArrayList<double[]>(); |
|  |  |
|  | for (int i = 0; i < this.curvePoints.size(); ++i){ |
|  | double[] segment = this.curvePoints.get(i); |
|  | //If segment is a Bezier curve, calculate appropriate line segments and add to vertexList |
|  | if (segment.length == 8){ |
|  | //Create a new CubicBezier |
|  | CubicBezier bezier = new CubicBezier(segment); |
|  | //Calculate the line segments local to the intersection points |
|  | ArrayList<double[]> bezierIntersects = bezier.getIntersect(x, y , x + 1, y ); |
|  | //If the ray intersects the Bezier |
|  | if (bezierIntersects.size() != 0){ |
|  | //Add the start vertex to the vertex list |
|  | vertexList.add(new double[]{segment[0],segment[1]}); |
|  | //Add the intersect vertices to the list |
|  | vertexList.addAll(bezierIntersects); |
|  | //Add the end vertex to the vertex list |
|  | vertexList.add(new double[]{segment[6],segment[7]}); |
|  | } |
|  | else{ |
|  | //Add the start vertex to the vertex list |
|  | vertexList.add(new double[]{segment[0],segment[1]}); |
|  | //Add the end vertex to the vertex list |
|  | vertexList.add(new double[]{segment[6],segment[7]}); |
|  | } |
|  | } |
|  | //If the segment is a lone vertex, add it to the list |
|  | else{ |
|  | vertexList.add(segment); |
|  | } |
|  | } |
|  |  |
|  | //Close the curve by setting the final vertex to be equal to the first |
|  | vertexList.add(vertexList.get(0)); |
|  |  |
|  | //Calculate winding number for a point wrt. the curve |
|  | //This code adapted from <http://geomalgorithms.com/a03-_inclusion.html> |
|  | int winding = 0; |
|  | for (int i = 0; i < vertexList.size() - 1; ++i){ |
|  | double[] thisVertex = vertexList.get(i); |
|  | double[] nextVertex = vertexList.get(i + 1); |
|  | if (thisVertex[1] <= y){ |
|  | if (nextVertex[1] > y){ |
|  | if (isLeft(thisVertex, nextVertex, x, y) > 0){ |
|  | ++winding; |
|  | } |
|  | } |
|  | } |
|  | else if (thisVertex[1] > y){ |
|  | if (nextVertex[1] <= y){ |
|  | if(isLeft(thisVertex, nextVertex, x, y) < 0){ |
|  | --winding; |
|  | } |
|  | } |
|  | } |
|  | } |
|  | if (winding == 0){ |
|  | return false; |
|  | } |
|  | else{ |
|  | return true; |
|  | } |
|  | } |
|  |  |
|  | //Tests if a point (x,y) is to the left of an infinite line passing through points[] |
|  | //Returns >0 if the point is on the left, <0 if the point is on the right and 0 if the point is on the line |
|  | private double isLeft(double[] thisVertex, double[] nextVertex, double x, double y){ |
|  | return ((thisVertex[0] - x) \* (nextVertex[1] - y) - (nextVertex[0] - x)\*(thisVertex[1] - y)); |
|  | } |
|  | } |

### Difference.java

|  |
| --- |
| package bean; |
|  |  |
|  | import java.awt.geom.Area; |
|  | import java.util.ArrayList; |
|  |  |
|  | /\*\* |
|  | \* @class Difference |
|  | \* This Difference class serves as an object. |
|  | \* Is inherited from Mix. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public class Difference extends Mix { |
|  | public Difference(ArrayList<Node> inputNodes){ |
|  | this.inputNodes = inputNodes; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean deleteNode() { |
|  | // TODO Auto-generated method stub |
|  | return false; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | @Override |
|  | public String print() { |
|  | String nodeList = inputNodes.get(0).print(); |
|  | for (int i = 1; i < inputNodes.size(); ++i){ |
|  | nodeList = nodeList.concat("," + inputNodes.get(i).print()); |
|  | } |
|  | return "Difference(" + nodeList + ")"; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean drawPixel(double x, double y) { |
|  | boolean subtractedNodes = false; |
|  | for (int i = 1; i < this.inputNodes.size(); ++i){ |
|  | subtractedNodes = subtractedNodes || this.inputNodes.get(i).drawPixel(x, y); |
|  | } |
|  | return (this.inputNodes.get(0).drawPixel(x, y) && !subtractedNodes); |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public Area draw() { |
|  | Area difference = new Area(this.inputNodes.get(0).draw()); |
|  | for (int i = 1; i < this.inputNodes.size(); ++i){ |
|  | difference.subtract(this.inputNodes.get(i).draw()); |
|  | } |
|  | return difference; |
|  | } |
|  |  |
|  | } |

### Intersection.java

|  |
| --- |
| package bean; |
|  |  |
|  | import java.awt.geom.Area; |
|  | import java.util.ArrayList; |
|  | /\*\* |
|  | \* @class Intersection |
|  | \* This Intersection class serves as an object. |
|  | \* Is inherited from Mix. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public class Intersection extends Mix { |
|  | public Intersection(ArrayList<Node> inputNodes){ |
|  | this.inputNodes = inputNodes; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean deleteNode() { |
|  | // TODO Auto-generated method stub |
|  | return false; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | @Override |
|  | public String print() { |
|  | String nodeList = inputNodes.get(0).print(); |
|  | for (int i = 1; i < inputNodes.size(); ++i){ |
|  | nodeList = nodeList.concat("," + inputNodes.get(i).print()); |
|  | } |
|  | return "Intersection(" + nodeList + ")"; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean drawPixel(double x, double y) { |
|  | boolean intersectingNodes = true; |
|  | for (int i = 0; i < this.inputNodes.size(); ++i){ |
|  | intersectingNodes = intersectingNodes && this.inputNodes.get(i).drawPixel(x, y); |
|  | } |
|  | return intersectingNodes; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public Area draw() { |
|  | Area intersection = new Area(this.inputNodes.get(0).draw()); |
|  | for (int i = 0; i < this.inputNodes.size(); ++i){ |
|  | intersection.intersect(this.inputNodes.get(i).draw()); |
|  | } |
|  | return intersection; |
|  | } |
|  |  |
|  | } |

### Mix.java

|  |
| --- |
| package bean; |
|  |  |
|  | import java.util.ArrayList; |
|  | /\*\* |
|  | \* @class Mix |
|  | \* This Mix class serves as an implementation of Node. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public abstract class Mix implements Node { |
|  |  |
|  | public ArrayList<Node> inputNodes; |
|  |  |
|  | } |

### Node.java

|  |
| --- |
| package bean; |
|  | import java.awt.geom.Area; |
|  | /\*\* |
|  | \* @class Node |
|  | \* This Node class serves as an interface. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public interface Node{ |
|  | public boolean deleteNode(); |
|  |  |
|  | public String print(); |
|  |  |
|  | public Area draw(); |
|  |  |
|  | public boolean drawPixel(double x, double y); |
|  | } |

### Rectangle.java

|  |
| --- |
| package bean; |
|  | import java.awt.Polygon; |
|  | import java.awt.geom.Area; |
|  | /\*\* |
|  | \* @class Rectangle |
|  | \* This Rectangle class serves as an object. |
|  | \* Is inherited from Shape. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public class Rectangle extends Shape { |
|  |  |
|  | public double yRadius; |
|  |  |
|  | public Rectangle(double radius, double yRadius) { |
|  | this.radius = radius; |
|  | this.yRadius = yRadius; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean deleteNode() { |
|  | // TODO Auto-generated method stub |
|  | return false; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean setRadius(double radius) { |
|  | this.radius=radius; |
|  | return false; |
|  | } |
|  |  |
|  | @Override |
|  | public double getRadius() { |
|  | return this.radius; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public String print() { |
|  | return "Rectangle(" + this.radius + "," + this.yRadius + ")"; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean drawPixel(double x, double y) { |
|  | if (x >= -this.radius && x < this.radius && y >= -this.yRadius && y < this.yRadius){ |
|  | return true; |
|  | } |
|  | return false; |
|  |  |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public Area draw() { |
|  | int[] xCoords = {(int) -this.radius, (int) this.radius, (int) this.radius, (int) -this.radius}; |
|  | int[] yCoords = {(int) this.yRadius, (int) this.yRadius, (int) -this.yRadius, (int) -this.yRadius}; |
|  | Polygon shape = new Polygon(xCoords, yCoords, 4); |
|  | return new Area(shape); |
|  | } |
|  | } |

### Rotate.java

|  |
| --- |
| package bean; |
|  |  |
|  | import java.awt.geom.AffineTransform; |
|  | import java.awt.geom.Area; |
|  | /\*\* |
|  | \* @class Rotate |
|  | \* This Rotate class serves as an object. |
|  | \* Is inherited from Transform. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public class Rotate extends Transform { |
|  | public double angle; |
|  |  |
|  | public Rotate(Node inputNode, double angle){ |
|  | this.inputNode = inputNode; |
|  | this.angle = Math.toRadians(angle); |
|  | this.repeat = 1; |
|  | } |
|  |  |
|  | public Rotate(Node inputNode, double angle, int repeat){ |
|  | this.inputNode = inputNode; |
|  | this.angle = Math.toRadians(angle); |
|  | this.repeat = repeat; |
|  | } |
|  |  |
|  | @Override |
|  | public boolean deleteNode() { |
|  | // TODO Auto-generated method stub |
|  | return false; |
|  | } |
|  |  |
|  | @Override |
|  | public Node getInputNode() { |
|  | return this.inputNode; |
|  | } |
|  |  |
|  | @Override |
|  | public String print() { |
|  | if (this.repeat == 1){ |
|  | return "Rotate(" + this.inputNode.print() + "," + Math.toDegrees(this.angle) + ")"; |
|  | } |
|  | else { |
|  | return "RotateN(" + this.inputNode.print() + "," + Math.toDegrees(this.angle) + "," + this.repeat + ")"; |
|  | } |
|  | } |
|  |  |
|  | @Override |
|  | public boolean drawPixel(double x, double y) { |
|  | boolean inShape = false; |
|  | for (int i = 0; i < this.repeat; ++i){ |
|  | inShape = inShape || (this.inputNode.drawPixel(x\*Math.cos((i + 1)\*this.angle) + y\*Math.sin((i + 1)\*this.angle), -x\*Math.sin((i + 1)\*this.angle) + y\*Math.cos((i + 1)\*this.angle))); |
|  | } |
|  | return inShape; |
|  | } |
|  |  |
|  | @Override |
|  | public Area draw() { |
|  | Area initialArea = new Area(this.inputNode.draw()); |
|  | Area rotatedArea = new Area(); |
|  | for (int i = 0; i < this.repeat; ++i){ |
|  | AffineTransform rotate = AffineTransform.getRotateInstance(this.angle); |
|  | initialArea.transform(rotate); |
|  | rotatedArea.add(initialArea); |
|  | } |
|  | return rotatedArea; |
|  | } |
|  |  |
|  | } |

### Scale.java

|  |
| --- |
| package bean; |
|  |  |
|  | import java.awt.geom.AffineTransform; |
|  | import java.awt.geom.Area; |
|  | /\*\* |
|  | \* @class Scale |
|  | \* This Scale class serves as an object. |
|  | \* Is inherited from Transform. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public class Scale extends Transform { |
|  | public double factorX; |
|  | public double factorY; |
|  |  |
|  | public Scale(Node inputNode, double factorX, double factorY){ |
|  | this.inputNode=inputNode; |
|  | this.factorX=factorX; |
|  | this.factorY=factorY; |
|  | this.repeat = 1; |
|  | } |
|  |  |
|  | public Scale(Node inputNode, double factorX, double factorY, int repeat){ |
|  | this.inputNode=inputNode; |
|  | this.factorX=factorX; |
|  | this.factorY=factorY; |
|  | this.repeat = repeat; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean deleteNode() { |
|  | // TODO Auto-generated method stub |
|  | return false; |
|  | } |
|  |  |
|  | @Override |
|  | public Node getInputNode() { |
|  | return this.inputNode; |
|  | } |
|  |  |
|  | @Override |
|  | public String print() { |
|  | String output = "Scale(" + this.inputNode.print() + "," + this.factorX; |
|  | if (this.factorX != this.factorY){ |
|  | output.concat("," + this.factorY); |
|  | } |
|  | if (this.repeat != 1){ |
|  | output.concat("," + this.repeat); |
|  | output.replaceFirst("Scale", "ScaleN"); |
|  | } |
|  | return output.concat(")"); |
|  | } |
|  |  |
|  | @Override |
|  | public boolean drawPixel(double x, double y) { |
|  | boolean inShape = false; |
|  | for (int i = 0; i < this.repeat; ++i){ |
|  | inShape = inShape || (this.inputNode.drawPixel(x/Math.pow(this.factorX,i+1), y/Math.pow(this.factorY,i+1))); |
|  | } |
|  | return inShape; |
|  | } |
|  |  |
|  | @Override |
|  | public Area draw() { |
|  | Area initialArea = new Area(this.inputNode.draw()); |
|  | Area scaledArea = new Area(); |
|  | for (int i = 0; i < this.repeat; ++i){ |
|  | AffineTransform scale = AffineTransform.getScaleInstance(this.factorX, this.factorY); |
|  | initialArea.transform(scale); |
|  | scaledArea.add(initialArea); |
|  | } |
|  | return scaledArea; |
|  | } |
|  |  |
|  | } |

### Shape.java

|  |
| --- |
| package bean; |
|  | /\*\* |
|  | \* @class Shape |
|  | \* This Shape class serves as an implementation of Node. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  | public abstract class Shape implements Node { |
|  |  |
|  | public double radius; |
|  |  |
|  | public abstract boolean setRadius(double radius); |
|  |  |
|  | public abstract double getRadius(); |
|  | } |

### Square.java

|  |
| --- |
| package bean; |
|  |  |
|  | import java.awt.Polygon; |
|  | import java.awt.geom.Area; |
|  | /\*\* |
|  | \* @class Square |
|  | \* This Square class serves as an object. |
|  | \* Is inherited from Shape. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public class Square extends Shape { |
|  |  |
|  | public Square(double radius){ |
|  | this.radius=radius; |
|  | } |
|  |  |
|  | @Override |
|  | public boolean deleteNode() { |
|  | // TODO Auto-generated method stub |
|  | return false; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean setRadius(double radius) { |
|  | this.radius=radius; |
|  | return false; |
|  | } |
|  |  |
|  | @Override |
|  | public double getRadius() { |
|  | return this.radius; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public String print() { |
|  | return "Square(" + this.radius + ")"; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean drawPixel(double x, double y) { |
|  | if (x >= -this.radius/Math.sqrt(2) && x < this.radius/Math.sqrt(2) && y > -this.radius/Math.sqrt(2) && y <= this.radius/Math.sqrt(2)){ |
|  | return true; |
|  | } |
|  | return false; |
|  | } |
|  |  |
|  | @Override |
|  | public Area draw() { |
|  | int[] xCoords = {(int) (-this.radius/Math.sqrt(2)), (int) (this.radius/Math.sqrt(2)), (int) (this.radius/Math.sqrt(2)), (int) (-this.radius/Math.sqrt(2))}; |
|  | int[] yCoords = {(int) (this.radius/Math.sqrt(2)), (int) (this.radius/Math.sqrt(2)), (int) (-this.radius/Math.sqrt(2)), (int) (-this.radius/Math.sqrt(2))}; |
|  | Polygon shape = new Polygon(xCoords, yCoords, 4); |
|  | return new Area(shape); |
|  | } |
|  | } |

### Transform.java

|  |
| --- |
| package bean; |
|  | /\*\* |
|  | \* @class Transform |
|  | \* This Transform class serves as an implementation of Node. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public abstract class Transform implements Node { |
|  |  |
|  | public Node inputNode; |
|  | public int repeat; |
|  |  |
|  | public abstract Node getInputNode(); |
|  | } |

### Translate.java

|  |
| --- |
| package bean; |
|  |  |
|  | import java.awt.geom.AffineTransform; |
|  | import java.awt.geom.Area; |
|  | /\*\* |
|  | \* @class Translate |
|  | \* This Translate class serves as an object. |
|  | \* Is inherited from Transform. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public class Translate extends Transform { |
|  | public double distanceX; |
|  | public double distanceY; |
|  |  |
|  | public Translate(Node inputNode, double distanceX, double distanceY){ |
|  | this.inputNode=inputNode; |
|  | this.distanceX=distanceX; |
|  | this.distanceY=distanceY; |
|  | this.repeat = 1; |
|  | } |
|  |  |
|  | public Translate(Node inputNode, double distanceX, double distanceY, int repeat){ |
|  | this.inputNode=inputNode; |
|  | this.distanceX=distanceX; |
|  | this.distanceY=distanceY; |
|  | this.repeat = repeat; |
|  | } |
|  |  |
|  | @Override |
|  | public boolean deleteNode() { |
|  | // TODO Auto-generated method stub |
|  | return false; |
|  | } |
|  |  |
|  | @Override |
|  | public Node getInputNode() { |
|  | return this.inputNode; |
|  | } |
|  |  |
|  | @Override |
|  | public String print() { |
|  | if (this.repeat == 1){ |
|  | return "Translate(" + this.inputNode.print() + "," + this.distanceX + "," + this.distanceY + ")"; |
|  | } |
|  | else{ |
|  | return "TranslateN(" + this.inputNode.print() + "," + this.distanceX + "," + this.distanceY + "," + this.repeat + ")"; |
|  | } |
|  | } |
|  |  |
|  | @Override |
|  | public boolean drawPixel(double x, double y) { |
|  | boolean inShape = false; |
|  | for (int i = 0; i < this.repeat; ++i){ |
|  | inShape = inShape || (this.inputNode.drawPixel(x - (i+1)\*this.distanceX, y - (i+1)\*this.distanceY)); |
|  | } |
|  | return inShape; |
|  | } |
|  |  |
|  | @Override |
|  | public Area draw() { |
|  | Area initialArea = new Area(this.inputNode.draw()); |
|  | Area translatedArea = new Area(); |
|  | for (int i = 0; i < this.repeat; ++i){ |
|  | AffineTransform translate = AffineTransform.getTranslateInstance(this.distanceX, this.distanceY); |
|  | initialArea.transform(translate); |
|  | translatedArea.add(initialArea); |
|  | } |
|  | return translatedArea; |
|  | } |
|  |  |
|  | } |

### Triangle.java

|  |
| --- |
| package bean; |
|  |  |
|  | import java.awt.Polygon; |
|  | import java.awt.geom.Area; |
|  | /\*\* |
|  | \* @class Triangle |
|  | \* This Triangle class serves as an object. |
|  | \* Is inherited from Shape. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public class Triangle extends Shape{ |
|  |  |
|  | public Triangle(double radius){ |
|  | this.radius=radius; |
|  | } |
|  |  |
|  | @Override |
|  | public boolean deleteNode() { |
|  | // TODO Auto-generated method stub |
|  | return false; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean setRadius(double radius) { |
|  | this.radius=radius; |
|  | return false; |
|  | } |
|  |  |
|  | @Override |
|  | public double getRadius() { |
|  | return this.radius; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | @Override |
|  | public String print() { |
|  | return "Triangle(" + this.radius + ")"; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean drawPixel(double x, double y) { |
|  | if((y < this.radius + Math.sqrt(3)\*x) && (y > -this.radius/2) && (y < this.radius - Math.sqrt(3)\*x)){ |
|  | return true; |
|  | } |
|  | return false; |
|  | } |
|  |  |
|  | @Override |
|  | public Area draw() { |
|  | int[] xCoords = {(int) (-Math.sqrt(3)\*this.radius/2), 0, (int) (Math.sqrt(3)\*this.radius/2)}; |
|  | int[] yCoords = {(int) -this.radius/2, (int) this.radius, (int)-this.radius/2}; |
|  | Polygon shape = new Polygon(xCoords, yCoords, 3); |
|  | return new Area(shape); |
|  | } |
|  |  |
|  | } |

### Union.java

|  |
| --- |
| package bean; |
|  |  |
|  | import java.awt.geom.Area; |
|  | import java.util.ArrayList; |
|  | /\*\* |
|  | \* @class Union |
|  | \* This Union class serves as an object. |
|  | \* Is inherited from Mix. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public class Union extends Mix { |
|  | public Union(ArrayList<Node> inputNodes){ |
|  | this.inputNodes = inputNodes; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean deleteNode() { |
|  | // TODO Auto-generated method stub |
|  | return false; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | @Override |
|  | public String print() { |
|  | String nodeList = this.inputNodes.get(0).print(); |
|  | for (int i = 1; i < this.inputNodes.size(); ++i){ |
|  | nodeList = nodeList.concat("," + this.inputNodes.get(i).print()); |
|  | } |
|  | return "Union(" + nodeList + ")"; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public boolean drawPixel(double x, double y) { |
|  | boolean unionNodes = false; |
|  | for (int i = 0; i < this.inputNodes.size(); ++i){ |
|  | unionNodes = unionNodes || this.inputNodes.get(i).drawPixel(x, y); |
|  | } |
|  | return unionNodes; |
|  | } |
|  |  |
|  |  |
|  | @Override |
|  | public Area draw() { |
|  | Area union = new Area(this.inputNodes.get(0).draw()); |
|  | for (int i = 0; i < this.inputNodes.size(); ++i){ |
|  | union.add(this.inputNodes.get(i).draw()); |
|  | } |
|  | return union; |
|  | } |
|  | } |

## Package: userinterface

### CodingPanel.java

|  |
| --- |
| package userinterface; |
|  | import java.awt.Panel; |
|  | import java.awt.TextArea; |
|  |  |
|  | /\*\* |
|  | \* @class CodingPanel |
|  | \* This CodingPanel class serves as a panel of DrawingFrame. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  |  |
|  | @SuppressWarnings("serial") |
|  | public class CodingPanel extends Panel { |
|  |  |
|  | public TextArea textArea = new TextArea("Triangle(100)", 0, 0, TextArea.SCROLLBARS\_VERTICAL\_ONLY); |
|  |  |
|  | public CodingPanel(){ |
|  | setLayout(null); |
|  | add(textArea); |
|  | } |
|  |  |
|  | public String getText(){ |
|  | return textArea.getText(); |
|  | } |
|  |  |
|  | public void setText(String text){ |
|  | textArea.setText(text); |
|  | } |
|  | } |

### DrawingFrame.java

|  |
| --- |
| package userinterface; |
|  | import java.awt.Button; |
|  | import java.awt.Checkbox; |
|  | import java.awt.Choice; |
|  | import java.awt.Color; |
|  | import java.awt.Frame; |
|  | import java.awt.event.ActionEvent; |
|  | import java.awt.event.ActionListener; |
|  | import java.awt.event.ComponentAdapter; |
|  | import java.awt.event.ComponentEvent; |
|  | import java.awt.event.ItemEvent; |
|  | import java.awt.event.ItemListener; |
|  | import java.awt.event.KeyAdapter; |
|  | import java.awt.event.KeyEvent; |
|  | import java.awt.event.WindowAdapter; |
|  | import java.awt.event.WindowEvent; |
|  | import java.util.ArrayList; |
|  |  |
|  | import util.LoadFile; |
|  | import util.Parser; |
|  | import util.SaveFile; |
|  | import bean.Node; |
|  | /\*\* |
|  | \* @class DrawingFrame |
|  | \* This DrawingFrame class serves as the main frame of this program. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | @SuppressWarnings("serial") |
|  | public class DrawingFrame extends Frame { |
|  | public DrawingFrame(){ |
|  |  |
|  | //Horizontal spacing between elements |
|  | int spacingX = 30; |
|  |  |
|  | //Vertical spacing between elements |
|  | int spacingY = 20; |
|  |  |
|  | //Horizontal size of a button |
|  | int buttonX = 80; |
|  |  |
|  | //Vertical size of a button |
|  | int buttonY = 30; |
|  |  |
|  | setSize(900,700); |
|  | setVisible(true); |
|  | setLayout(null); |
|  | setBackground(new Color(225,225,225)); |
|  | //The canvas on which shapes will be drawn |
|  | DrawingPanel canvas = new DrawingPanel(); |
|  | canvas.setBounds(spacingX, spacingY + 30, this.getWidth()/2 - (3\*spacingX/2), this.getHeight() - (4\*spacingY + 2\*buttonY + 30)); |
|  | canvas.setBackground(Color.white); |
|  | add(canvas); |
|  |  |
|  | //Cheating to get a 1-pixel-wide border around the canvas by creating an all-black canvas behind it |
|  | DrawingPanel canvasOutline = new DrawingPanel(); |
|  | canvasOutline.setBounds(canvas.getX() - 1, canvas.getY() - 1, canvas.getWidth() + 2, canvas.getHeight() + 2); |
|  | canvasOutline.setBackground(Color.black); |
|  | add(canvasOutline); |
|  |  |
|  | //The text box in which instructions will be entered |
|  | CodingPanel textBox = new CodingPanel(); |
|  | textBox.setBounds(canvas.getWidth() + canvas.getX() + spacingX, canvas.getY(), canvas.getWidth(), canvas.getHeight()); |
|  | textBox.textArea.setBounds(10, 10, textBox.getWidth() - 10, textBox.getHeight() - 10); |
|  | add(textBox); |
|  |  |
|  | Button btnClear=new Button(); |
|  | btnClear.setLabel("Clear"); |
|  | btnClear.setBounds(spacingX, canvas.getY() + canvas.getHeight() + spacingY, buttonX, buttonY); |
|  | add(btnClear); |
|  | btnClear.addActionListener(new ActionListener() { |
|  | @Override |
|  | public void actionPerformed(ActionEvent e) { |
|  | canvas.drawAxes(); |
|  | } |
|  | }); |
|  |  |
|  | //Dropdown for choosing drawing colour |
|  | Choice colourChooser = new Choice(); |
|  | colourChooser.add("Black"); |
|  | colourChooser.add("Blue"); |
|  | colourChooser.add("Cyan"); |
|  | colourChooser.add("Dark Gray"); |
|  | colourChooser.add("Gray"); |
|  | colourChooser.add("Green"); |
|  | colourChooser.add("Light Gray"); |
|  | colourChooser.add("Magenta"); |
|  | colourChooser.add("Orange"); |
|  | colourChooser.add("Pink"); |
|  | colourChooser.add("Red"); |
|  | colourChooser.add("Yellow"); |
|  | colourChooser.setBounds(spacingX + canvas.getWidth()/2 - buttonX/2, canvas.getY() + canvas.getHeight() + spacingY, buttonX, 20); |
|  | colourChooser.select("Blue"); |
|  | add(colourChooser); |
|  |  |
|  | //Array storing the built-in Java colours as Color objects, for use with colour chooser dropdown |
|  | Color[] colourList = {Color.BLACK,Color.BLUE,Color.CYAN,Color.DARK\_GRAY,Color.GRAY,Color.GREEN,Color.LIGHT\_GRAY,Color.MAGENTA,Color.ORANGE,Color.PINK,Color.RED,Color.YELLOW}; |
|  |  |
|  | //Button for drawing pixel-by-pixel |
|  | Button btnDrawPixel = new Button(); |
|  | btnDrawPixel.setLabel("Draw"); |
|  | btnDrawPixel.setBounds(canvas.getWidth() + canvas.getX() - buttonX, canvas.getY() + canvas.getHeight() + spacingY, buttonX, buttonY); |
|  | add(btnDrawPixel); |
|  | btnDrawPixel.addActionListener(new ActionListener() { |
|  | @Override |
|  | public void actionPerformed(ActionEvent e) { |
|  | Node rootNode = Parser.parse(textBox.getText()); |
|  | ArrayList<String> errList=Parser.returnErrList(); |
|  |  |
|  | if(errList.size()>0){ |
|  | ErrorFrame error = new ErrorFrame(); |
|  | error.setTitle("Error"); |
|  |  |
|  | String helpText = ""; |
|  |  |
|  | for(int i=0;i<errList.size();i++){ |
|  | helpText+= (i+1) + ". " + errList.get(i)+" \n\n"; |
|  | } |
|  |  |
|  |  |
|  | error.init(helpText); |
|  |  |
|  | } |
|  | else { |
|  | canvas.drawPixels(rootNode, colourList[colourChooser.getSelectedIndex()]); |
|  | } |
|  | } |
|  | }); |
|  |  |
|  | //Button for drawing using the Area class |
|  | Button btnDraw = new Button(); |
|  | btnDraw.setLabel("Draw"); |
|  | btnDraw.setBounds(canvas.getWidth() + canvas.getX() - buttonX, canvas.getY() + canvas.getHeight() + 2\*spacingY + buttonY, buttonX, buttonY); |
|  | add(btnDraw); |
|  | btnDraw.addActionListener(new ActionListener() { |
|  | @Override |
|  | public void actionPerformed(ActionEvent e) { |
|  | Node rootNode = Parser.parse(textBox.getText()); |
|  | ArrayList<String> errList=Parser.returnErrList(); |
|  |  |
|  | if(errList.size()>0){ |
|  | ErrorFrame error = new ErrorFrame(); |
|  | error.setTitle("Error"); |
|  |  |
|  | String helpText = ""; |
|  |  |
|  | for(int i=0;i<errList.size();i++){ |
|  | helpText+= (i+1) + ". " + errList.get(i)+" \n\n"; |
|  | } |
|  |  |
|  |  |
|  | error.init(helpText); |
|  |  |
|  | } |
|  | else |
|  | canvas.drawArea(rootNode, colourList[colourChooser.getSelectedIndex()]); |
|  | // } |
|  | } |
|  | }); |
|  |  |
|  | Button btnSave = new Button(); |
|  | btnSave.setLabel("Save"); |
|  | btnSave.setBounds(2\*spacingX + canvas.getWidth(), canvas.getY() + canvas.getHeight() + spacingY, buttonX, buttonY); |
|  | add(btnSave); |
|  | btnSave.addActionListener(new ActionListener() { |
|  | public void actionPerformed(ActionEvent e){ |
|  | SaveFile sf = new SaveFile(textBox.getText()); |
|  | } |
|  | }); |
|  |  |
|  | Button btnLoad = new Button(); |
|  | btnLoad.setLabel("Load"); |
|  | btnLoad.setBounds(2\*canvas.getWidth() + canvas.getX() + spacingX - buttonX, canvas.getY() + canvas.getHeight() + spacingY, buttonX, buttonY); |
|  | add(btnLoad); |
|  | btnLoad.addActionListener(new ActionListener() { |
|  | public void actionPerformed(ActionEvent e){ |
|  | LoadFile lf = new LoadFile(); |
|  | if (lf.getContent() != null){ |
|  | textBox.setText(lf.getContent()); |
|  | } |
|  | } |
|  | }); |
|  |  |
|  | Button btnHelp = new Button(); |
|  | btnHelp.setLabel("Help"); |
|  | btnHelp.setBounds(2\*canvas.getWidth() + canvas.getX() + spacingX - buttonX, canvas.getY() + canvas.getHeight() + 2\*spacingY + buttonY, buttonX, buttonY); |
|  | add(btnHelp); |
|  | btnHelp.addActionListener(new ActionListener() { |
|  | public void actionPerformed(ActionEvent e){ |
|  | HelpFrame help = new HelpFrame(); |
|  | help.setTitle("Help"); |
|  | } |
|  | }); |
|  |  |
|  | Checkbox cbAutoComplete=new Checkbox("Auto-Complete brackets"); |
|  | cbAutoComplete.setBounds(2\*spacingX + canvas.getWidth(), canvas.getY() + canvas.getHeight() + 2\*spacingY + buttonY, 2\*buttonX, buttonY); |
|  | cbAutoComplete.addItemListener(new ItemListener() { |
|  | @Override |
|  | public void itemStateChanged(ItemEvent e) { |
|  | // TODO Auto-generated method stub |
|  | if(cbAutoComplete.getState()){ |
|  | textBox.textArea.addKeyListener(new KeyAdapter() { |
|  | public void keyTyped(KeyEvent e) { |
|  | if (e.getKeyChar()=='(') { |
|  | int newlines = 0; |
|  | String text = textBox.textArea.getText(); |
|  | for (int i = 0; i < textBox.textArea.getCaretPosition() ; ++i){ |
|  | if (text.charAt(i) == '\r'){ |
|  | ++newlines; |
|  | } |
|  | } |
|  | e.consume(); |
|  | textBox.textArea.insert("()", textBox.textArea.getCaretPosition() - newlines); |
|  | textBox.textArea.setCaretPosition(textBox.textArea.getCaretPosition() - (newlines + 1)); |
|  | } |
|  | } |
|  | }); |
|  | } |
|  | else{ |
|  | textBox.textArea.removeKeyListener(textBox.textArea.getKeyListeners()[0]); |
|  | } |
|  |  |
|  | } |
|  | }); |
|  | add(cbAutoComplete); |
|  |  |
|  |  |
|  | //Dynamically resizing and repositioning window elements on window resize |
|  | addComponentListener(new ComponentAdapter(){ |
|  | public void componentResized(ComponentEvent e){ |
|  | canvas.setBounds(spacingX, spacingY + 30, getWidth()/2 - (3\*spacingX/2), getHeight() - (4\*spacingY + 2\*buttonY + 30)); |
|  | canvasOutline.setBounds(canvas.getX() - 1, canvas.getY() - 1, canvas.getWidth() + 2, canvas.getHeight() + 2); |
|  | textBox.setBounds(canvas.getWidth() + canvas.getX() + spacingX, canvas.getY(), canvas.getWidth(), canvas.getHeight()); |
|  | textBox.textArea.setBounds(10, 10, textBox.getWidth() - 10, textBox.getHeight() - 10); |
|  | btnClear.setBounds(spacingX, canvas.getY() + canvas.getHeight() + spacingY, buttonX, buttonY); |
|  | colourChooser.setBounds(spacingX + canvas.getWidth()/2 - buttonX/2, canvas.getY() + canvas.getHeight() + spacingY, buttonX, 20); |
|  | btnDrawPixel.setBounds(canvas.getWidth() + canvas.getX() - buttonX, canvas.getY() + canvas.getHeight() + spacingY, buttonX, buttonY); |
|  | btnDraw.setBounds(canvas.getWidth() + canvas.getX() - buttonX, canvas.getY() + canvas.getHeight() + 2\*spacingY + buttonY, buttonX, buttonY); |
|  | btnLoad.setBounds(2\*canvas.getWidth() + canvas.getX() + spacingX - buttonX, canvas.getY() + canvas.getHeight() + spacingY, buttonX, buttonY); |
|  | btnSave.setBounds(2\*spacingX + canvas.getWidth(), canvas.getY() + canvas.getHeight() + spacingY, buttonX, buttonY); |
|  | btnHelp.setBounds(2\*canvas.getWidth() + canvas.getX() + spacingX - buttonX, canvas.getY() + canvas.getHeight() + 2\*spacingY + buttonY, buttonX, buttonY); |
|  | cbAutoComplete.setBounds(2\*spacingX + canvas.getWidth(), canvas.getY() + canvas.getHeight() + 2\*spacingY + buttonY, 2\*buttonX, buttonY); |
|  | } |
|  | }); |
|  |  |
|  |  |
|  |  |
|  |  |
|  | addWindowListener(new WindowAdapter(){ |
|  | public void windowClosing(WindowEvent e){ |
|  | dispose(); |
|  | System.exit(0); |
|  | } |
|  | }); |
|  | //Drawing axes to the canvas on window initialization. Done last to avoid axes being cleared/overwritten during initialization |
|  | canvas.drawAxes(); |
|  | } |
|  |  |
|  |  |
|  | public static void main(String args[]) |
|  | { |
|  | DrawingFrame main = new DrawingFrame(); |
|  | main.setTitle("Drawing"); |
|  | } |
|  |  |
|  | } |

### DrawingPanel.java

|  |
| --- |
| package userinterface; |
|  | import java.awt.Color; |
|  | import java.awt.Font; |
|  | import java.awt.Graphics2D; |
|  | import java.awt.Panel; |
|  | import java.awt.geom.AffineTransform; |
|  | import java.awt.geom.Area; |
|  | import java.awt.image.BufferedImage; |
|  |  |
|  | import bean.Node; |
|  |  |
|  | /\*\* |
|  | \* @class DrawingPanel |
|  | \* This DrawingPanel class serves as a panel of DrawingFrame. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | @SuppressWarnings("serial") |
|  | public class DrawingPanel extends Panel { |
|  |  |
|  | public DrawingPanel(){ |
|  | } |
|  |  |
|  | //Method for drawing an Area |
|  | public void drawArea(Node rootNode, Color colour){ |
|  | Graphics2D canvasGraphics = (Graphics2D)this.getGraphics(); |
|  |  |
|  | //Drawing the axes |
|  | drawAxes(); |
|  |  |
|  | //Setting the origin to be in the middle of the canvas |
|  | canvasGraphics.translate(this.getWidth()/2, this.getHeight()/2); |
|  |  |
|  | //Flipping the coordinates vertically so that y increases as you go up, not down |
|  | AffineTransform flipVertical = AffineTransform.getScaleInstance(1, -1); |
|  | canvasGraphics.transform(flipVertical); |
|  |  |
|  | canvasGraphics.setColor(colour); |
|  | Area finalArea = rootNode.draw(); |
|  | canvasGraphics.fill(finalArea); |
|  | paint(canvasGraphics); |
|  | } |
|  |  |
|  | public void drawAxes(){ |
|  | Graphics2D axesGraphics = (Graphics2D)this.getGraphics(); |
|  | //Clearing the canvas |
|  | axesGraphics.clearRect(0, 0, getWidth(), getHeight()); |
|  | //Drawing horizontal and vertical lines |
|  | axesGraphics.drawLine(this.getWidth()/2, 0, this.getWidth()/2, this.getHeight()); |
|  | axesGraphics.drawLine(0, this.getHeight()/2, this.getWidth(), this.getHeight()/2); |
|  | axesGraphics.setColor(Color.BLACK); |
|  | paint(axesGraphics); |
|  | } |
|  |  |
|  | //Method for drawing a BufferedImage |
|  | public void drawPixels(Node rootNode, Color colour) { |
|  | //Creating an image that can be drawn pixel by pixel that is the size of the canvas |
|  | BufferedImage pixelCanvas = new BufferedImage(getWidth(), getHeight(), BufferedImage.TYPE\_INT\_ARGB); |
|  | for (int i = 0; i < getHeight(); ++i){ |
|  | for (int j = 0; j < getWidth(); ++j){ |
|  | //Checking if the current pixel should be drawn. Because the origin of pixelCanvas is in the top left |
|  | //corner, pixels are shifted by half the canvas width and height, and the y values are multiplied |
|  | //by -1 to flip the image vertically |
|  | if (rootNode.drawPixel(j - getWidth()/2, - (i - getHeight()/2))){ |
|  | pixelCanvas.setRGB(j, i, colour.getRGB()); |
|  | } |
|  | else{ |
|  | //If current pixel should not be drawn, draw a transparent pixel |
|  | pixelCanvas.setRGB(j, i, 0); |
|  | } |
|  | } |
|  | } |
|  | Graphics2D pixelGraphics = (Graphics2D)this.getGraphics(); |
|  | drawAxes(); |
|  | pixelGraphics.drawImage(pixelCanvas, null, null); |
|  | paint(pixelGraphics); |
|  | } |
|  |  |
|  | public void drawError(String error){ |
|  | Graphics2D errorGraphics = (Graphics2D)this.getGraphics(); |
|  | errorGraphics.clearRect(0, 0, getWidth(), getHeight()); |
|  | Font font = new Font("Courier", Font.PLAIN, 40); |
|  | errorGraphics.setColor(Color.red); |
|  | errorGraphics.setFont(font); |
|  | errorGraphics.drawString(error, 0, this.getHeight()/2); |
|  | paint(errorGraphics); |
|  | return; |
|  |  |
|  | } |
|  | } |

### ErrorFrame.java

|  |
| --- |
| package userinterface; |
|  | import java.awt.Button; |
|  | import java.awt.Color; |
|  | import java.awt.Frame; |
|  | import java.awt.Label; |
|  | import java.awt.TextArea; |
|  | import java.awt.event.ActionEvent; |
|  | import java.awt.event.ActionListener; |
|  | import java.awt.event.WindowAdapter; |
|  | import java.awt.event.WindowEvent; |
|  | /\*\* |
|  | \* @class ErrorFrame |
|  | \* This ErrorFrame class serves as the frame for displaying errors. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | @SuppressWarnings("serial") |
|  | public class ErrorFrame extends Frame { |
|  | public Label label = new Label(""); |
|  |  |
|  | public TextArea ta = new TextArea("",0,0,TextArea.SCROLLBARS\_VERTICAL\_ONLY); |
|  | Button btnClose=new Button(); |
|  | int spacingX = 20; |
|  | int spacingY = 20; |
|  | int buttonX = 80; |
|  | int buttonY = 30; |
|  |  |
|  | public ErrorFrame(){ |
|  | setSize(300,200); |
|  | setVisible(true); |
|  | setLayout(null); |
|  | setResizable(false); |
|  | setBackground(new Color(225,225,225)); |
|  |  |
|  | ta.setEditable(false); |
|  | add(label); |
|  | add(ta); |
|  |  |
|  | btnClose.setLabel("Close"); |
|  | btnClose.setBounds(this.getWidth()/2 - buttonX/2,this.getHeight() - (spacingY + buttonY), buttonX, buttonY); |
|  | add(btnClose); |
|  | btnClose.addActionListener(new ActionListener() { |
|  | @Override |
|  | public void actionPerformed(ActionEvent arg0) { |
|  | dispose(); |
|  | } |
|  | }); |
|  |  |
|  | addWindowListener(new WindowAdapter() |
|  | { |
|  | public void windowClosing(WindowEvent e){ |
|  | dispose();} |
|  | } |
|  | ); |
|  | } |
|  |  |
|  | public void init(String text) { |
|  | ta.setText(text); |
|  | ta.setBounds(spacingX, 2\*spacingY, this.getWidth()-spacingX\*2, this.getHeight() - (4\*spacingY + buttonY)); |
|  | btnClose.setBounds(this.getWidth()/2-buttonX/2,this.getHeight()-50, buttonX, buttonY); |
|  | } |
|  |  |
|  | } |

### HelpFrame.java

|  |
| --- |
| package userinterface; |
|  | import java.awt.Button; |
|  | import java.awt.Color; |
|  | import java.awt.Frame; |
|  | import java.awt.TextArea; |
|  | import java.awt.event.ActionEvent; |
|  | import java.awt.event.ActionListener; |
|  | import java.awt.event.WindowAdapter; |
|  | import java.awt.event.WindowEvent; |
|  | /\*\* |
|  | \* @class HelpFrame |
|  | \* This HelpFrame class serves as the frame for displaying tips. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  | @SuppressWarnings("serial") |
|  | public class HelpFrame extends Frame { |
|  | public HelpFrame(){ |
|  | setSize(520,600); |
|  | setVisible(true); |
|  | setLayout(null); |
|  | setResizable(false); |
|  | setBackground(new Color(225,225,225)); |
|  | int spacingX = 20; |
|  | int spacingY = 20; |
|  | int buttonX = 80; |
|  | int buttonY = 30; |
|  |  |
|  | String helpText = "Syntax for inputting instructions:\n \n" |
|  | + "SHAPES\n" |
|  | + "Square(radius)\n" |
|  | + "Rectangle(width, height)\n" |
|  | + "Circle(radius)\n" |
|  | + "Triangle(radius)\n" |
|  | + "Curve(curve segment 1, curve segment 2,...) \n" |
|  | + "Curve segments are either Bezier curves defined as: (x1, y1, x2, y2, x3, y3, x4, y4) \n" |
|  | + "OR individual points defined as: (x, y).\n" |
|  | + "The Curve command can also be used to draw polygons containing only straight lines between vertices " |
|  | + "if desired. \n" |
|  | + "If two consecutive Bezier curves don't share a start/end point, a line is drawn between them " |
|  | + "to form a continuous shape. \n \n" |
|  | + "TRANSFORMS\n" |
|  | + "Rotate(Node, angle) - Rotation is anticlockwise\n" |
|  | + "Translate(Node, X distance, Y distance)\n" |
|  | + "Scale(Node, factor) OR Scale(Node, X factor, Y factor)\n \n" |
|  | + "REPEAT TRANSFORMS\n" |
|  | + "RotateN(Node, angle, repeats)\n" |
|  | + "TranslateN(Node, X distance, Y distance, repeats)\n" |
|  | + "ScaleN(Node, factor, repeats) OR Scale(Node, X factor, Y factor, repeats)\n \n" |
|  | + "MIXES\n" |
|  | + "Union(Node, Node, Node,...)\n" |
|  | + "Intersection(Node, Node, Node,...)\n" |
|  | + "Difference(Node, Node, Node,...)\n \n" |
|  | + "Example: Difference(Rotate(Square(50),45),Triangle(45))\n" |
|  | + "Produces the Difference of a Square that has been rotated 45 degrees and a Triangle."; |
|  |  |
|  | TextArea textArea = new TextArea(helpText, 0, 0, TextArea.SCROLLBARS\_NONE); |
|  | textArea.setEditable(false); |
|  | textArea.setBounds(spacingX, 2\*spacingY, this.getWidth() - 2\*spacingX, this.getHeight() - (4\*spacingY + buttonY)); |
|  | add(textArea); |
|  |  |
|  | Button btnClose=new Button(); |
|  | btnClose.setLabel("Close"); |
|  | btnClose.setBounds(this.getWidth()/2 - buttonX/2,this.getHeight() - (spacingY + buttonY), buttonX, buttonY); |
|  | add(btnClose); |
|  | btnClose.addActionListener(new ActionListener() { |
|  | @Override |
|  | public void actionPerformed(ActionEvent arg0) { |
|  | dispose(); |
|  | } |
|  | }); |
|  |  |
|  | addWindowListener(new WindowAdapter() |
|  | { |
|  | public void windowClosing(WindowEvent e){ |
|  | dispose();} |
|  | } |
|  | ); |
|  | } |
|  | } |

## Package: util

### CubicBezler.java

|  |
| --- |
| package util; |
|  | /\*\* |
|  | \* @class CubicBezier |
|  | \* This CubicBezier class provides tools for creating and manipulating cubic Bezier curves |
|  | \* Several of the methods implemented in the class are not currently used, but included for completeness |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | import java.awt.geom.CubicCurve2D; |
|  | import java.util.ArrayList; |
|  | import java.util.Arrays; |
|  |  |
|  | //A class for creating and manipulating cubic Bezier curves |
|  |  |
|  | public class CubicBezier{ |
|  | //The control points for the Bezier curve in the form {{x1, x2, x3, x4}, {y1, y2, y3, y4}} |
|  | private double[][] controlPoints; |
|  |  |
|  | public CubicBezier(double[][] controlPoints){ |
|  | this.setControlPoints(controlPoints); |
|  | } |
|  |  |
|  | //Constructor taking control points in the form {x1, y1, x2, y2, x3, y3, x4, y4} |
|  | public CubicBezier(double[] controlPointList){ |
|  | double[][] controlPoints = new double[2][4]; |
|  | for (int i = 0; i < controlPoints[0].length; ++i){ |
|  | controlPoints[0][i] = controlPointList[2\*i]; |
|  | controlPoints[1][i] = controlPointList[1 + 2\*i]; |
|  | } |
|  | this.setControlPoints(controlPoints); |
|  | } |
|  |  |
|  | //Converts the Bezier curve into a polygon with nEdges and returns the vertices as an ArrayList of double arrays |
|  | //Currently naive, needs to implement a better algorithm |
|  | public ArrayList<double[]> toPolygon(int nEdges){ |
|  | ArrayList<double[]> vertices = new ArrayList<double[]>(); |
|  | double stepSize = (double) 1/nEdges; |
|  | for (int i = 0; i < nEdges + 1; ++i){ |
|  | vertices.add(this.getPositionAt(i \* stepSize)); |
|  | } |
|  | return vertices; |
|  | } |
|  |  |
|  | //Calculates the intersection points (x,y) between a line segment and a Bezier curve |
|  | //and outputs points (fx,fy) a small step forwards along the Bezier curve and points(bx, by) |
|  | //a small step backwards along the curve in the form {{bx1, by1}, {fx1, fy1}, {bx2,...},...} |
|  | //linePoints in the form x1, y1, x2, y2 |
|  | public ArrayList<double[]> getIntersect(double x1, double y1, double x2, double y2){ |
|  | //Calculating coefficients for line segment |
|  | double a = x2 - x1; |
|  | double b = y1 - y2; |
|  | double c = y1\*(x1 - x2) + x1\*(y2 - y1); |
|  |  |
|  | //Calculating coefficients for cubic Bezier curve |
|  | double[][] bezierCoeffs = this.getCoeffs(); |
|  |  |
|  | double[] cubicCoeffs = new double[4]; |
|  | cubicCoeffs[0] = a\*bezierCoeffs[1][0] + b\*bezierCoeffs[0][0]; //t^3 terms |
|  | cubicCoeffs[1] = a\*bezierCoeffs[1][1] + b\*bezierCoeffs[0][1]; //t^2 terms |
|  | cubicCoeffs[2] = a\*bezierCoeffs[1][2] + b\*bezierCoeffs[0][2]; //t terms |
|  | cubicCoeffs[3] = a\*bezierCoeffs[1][3] + b\*bezierCoeffs[0][3] + c; //constant terms |
|  |  |
|  | //Calculate the roots of the parametric equation |
|  | double[] roots = this.getCubicRoots(cubicCoeffs); |
|  |  |
|  | //ArrrayList to hold the final results |
|  | ArrayList<double[]> intersections = new ArrayList<double[]>(); |
|  |  |
|  | for (int i = 0; i < roots.length; ++i){ |
|  | double t = roots[i]; |
|  |  |
|  | //Only consider valid roots |
|  | if (t != -1){ |
|  | //The (x,y) position of a point a little further forwards along the Bezier curve |
|  | double[] pointPlus = this.getPositionAt(t + 0.001); |
|  | //The (x,y) position of a point a little further backwards along the Bezier curve |
|  | double[] pointMinus = this.getPositionAt(t - 0.001); |
|  | //An array containing a line segment representing the local direction of the Bezier curve at the intersect |
|  | intersections.add(pointMinus); |
|  | intersections.add(pointPlus); |
|  | } |
|  | } |
|  | return intersections; |
|  | } |
|  | //Calculates the coefficients of the cubic equations in t for the Bezier curve |
|  | //output in the form ((cx1, cy1), (cx2, cy2), (cx3, cy3), (cx4, cy4)) |
|  | public double[][] getCoeffs(){ |
|  | double[][] coeffs = new double[2][4]; |
|  | //First do for the x values, then the y values |
|  | for (int i = 0; i < 2; ++i){ |
|  | coeffs[i][0] = -this.controlPoints[i][0] + 3\*this.controlPoints[i][1] - 3\*this.controlPoints[1][2] + this.controlPoints[i][3]; |
|  | coeffs[i][1] = 3\*this.controlPoints[i][0] - 6\*this.controlPoints[i][1] + 3\*this.controlPoints[1][2]; |
|  | coeffs[i][2] = -3\*this.controlPoints[i][0] + 3\*this.controlPoints[i][1]; |
|  | coeffs[i][3] = this.controlPoints[i][0]; |
|  | } |
|  | return coeffs; |
|  | } |
|  |  |
|  | //Calculating the real roots of a cubic equation in t, returning -1 if root is out of bounds |
|  | public double[] getCubicRoots(double[] cubicCoeffs){ |
|  | //Reversing the order of the cubic coefficients as CubicCurve2D.solveCubic() takes them in the opposite |
|  | //order to the order in which they're generated by getCoeffs() |
|  | double[] eqn = new double[]{cubicCoeffs[3],cubicCoeffs[2],cubicCoeffs[1],cubicCoeffs[0]}; |
|  | //Calculate the number of roots and store any roots found back into eqn. Returns -1 if the equation is a constant |
|  | int numOfRoots = CubicCurve2D.solveCubic(eqn); |
|  |  |
|  | //If the equation is a constant, return an empty array |
|  | if (numOfRoots == -1){ |
|  | return new double[]{}; |
|  | } |
|  | //Create a new array with as many elements as there are roots and store any roots found in it. |
|  | //This is done to prevent empty elements in eqn being mistaken for valid roots with value = 0 |
|  | double[] roots = Arrays.copyOf(eqn, numOfRoots); |
|  | //Discard roots with values of t outside the Bezier curve (0 <= t <= 1) |
|  | for (int i = 0; i < roots.length; ++i){ |
|  | if (roots[i] < 0 || roots[i] > 1){ |
|  | roots[i] = -1; |
|  | } |
|  | } |
|  | return roots; |
|  | } |
|  |  |
|  | //Calculating the roots of a quadratic equation in t, returning an empty array if roots are imaginary |
|  | public double[] getQuadraticRoots (double[] quadCoeffs){ |
|  | if (quadCoeffs[0] == 0){ |
|  | double[] linearCoeffs = Arrays.copyOfRange(quadCoeffs, 1, quadCoeffs.length-1); |
|  | return getLinearRoots(linearCoeffs); |
|  | } |
|  | //The discriminant of the quadratic function |
|  | double discriminant = quadCoeffs[1]\*quadCoeffs[1] - 4 \* quadCoeffs[0]\*quadCoeffs[2]; |
|  | //One real root |
|  | if (discriminant == 0){ |
|  | double root = -quadCoeffs[1]/2\*quadCoeffs[0]; |
|  | return new double[]{root}; |
|  | } |
|  | //Two real roots |
|  | else if (discriminant > 0){ |
|  | double sqrtD = Math.sqrt(discriminant); |
|  | double root1 = (-quadCoeffs[1] - sqrtD)/2\*quadCoeffs[0]; |
|  | double root2 = (-quadCoeffs[1] + sqrtD)/2\*quadCoeffs[0]; |
|  | return new double[]{root1, root2}; |
|  | } |
|  | //Complex roots |
|  | else{ |
|  | return new double[]{}; |
|  | } |
|  | } |
|  |  |
|  | //Calculating the roots of a linear equation in t, returning an empty array if root is imaginary |
|  | public double[] getLinearRoots (double[] linearCoeffs){ |
|  | double t = -linearCoeffs[1]/linearCoeffs[0]; |
|  | if (linearCoeffs[0] == 0){ |
|  | return new double[]{}; |
|  | } |
|  | return new double[]{t}; |
|  | } |
|  |  |
|  | //Calculates the (x,y) position of the point at position t on the Bezier curve |
|  | //given by a\*(1 - t)^3 + 3\*b\*(1 - t)^2\*t + 3\*c\*(1 - t)\*t^2 + d\*t^3 where a,b,c,d are control points |
|  | public double[] getPositionAt(double t){ |
|  | if (t == 0){ |
|  | //Return the start position |
|  | return new double[] {this.controlPoints[0][0], this.controlPoints[1][0]}; |
|  | } |
|  | if (t == 1){ |
|  | //Return the end position |
|  | return new double[] {this.controlPoints[0][3], this.controlPoints[1][3]}; |
|  | } |
|  | double[] coordinates = new double[2]; |
|  | for (int i = 0; i < coordinates.length; ++i){ |
|  | coordinates[i] = this.controlPoints[i][0]\*Math.pow((1-t),3) + 3\*this.controlPoints[i][1]\*(1-t)\*(1-t)\*t |
|  | + 3\*this.controlPoints[i][2]\*(1-t)\*t\*t + this.controlPoints[i][3]\*t\*t\*t; |
|  | } |
|  | return coordinates; |
|  | } |
|  |  |
|  | public double[][] getControlPoints() { |
|  | return controlPoints; |
|  | } |
|  |  |
|  | public void setControlPoints(double[][] controlPoints) { |
|  | this.controlPoints = controlPoints; |
|  | } |
|  |  |
|  | } |

### LoadFile.java

|  |
| --- |
| package util; |
|  |  |
|  | import java.awt.FileDialog; |
|  | import java.awt.Frame; |
|  | import java.io.BufferedReader; |
|  | import java.io.FileReader; |
|  | import java.io.IOException; |
|  |  |
|  | /\*\* |
|  | \* @class LoadFile |
|  | \* This LoadFile class initializes a FileDialog for loading existing drawing from a file. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | @SuppressWarnings("serial") |
|  | public class LoadFile extends Frame { |
|  | private String content; |
|  | public String getContent() { |
|  | return content; |
|  | } |
|  |  |
|  | public void setContent(String content) { |
|  | this.content = content; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Load the user commands from a .draw file. |
|  | \*/ |
|  |  |
|  |  |
|  | public LoadFile() { |
|  | FileDialog fc=new FileDialog(this,"Load from a file",0); |
|  | fc.setFile("\*.draw"); |
|  | fc.setVisible(true); |
|  | if(fc.getFile()!=null){ |
|  | try ( |
|  | FileReader fileReader = new FileReader(fc.getDirectory()+fc.getFile()); |
|  | BufferedReader bufferedReader = new BufferedReader(fileReader); |
|  | ) |
|  | { |
|  | String line = null; |
|  | String commands = ""; |
|  | while ((line = bufferedReader.readLine()) != null) { |
|  | commands+=line + "\n"; |
|  | } |
|  | bufferedReader.close(); |
|  | this.setContent(commands); |
|  | } catch (IOException x) { |
|  | System.err.format("IOException: %s%n", x); |
|  | } |
|  | } |
|  | else { |
|  | this.setContent(null); |
|  | } |
|  | } |
|  | } |

### Parser.java

|  |
| --- |
| package util; |
|  |  |
|  | import java.util.ArrayList; |
|  |  |
|  | import bean.Circle; |
|  | import bean.ClosedCurve; |
|  | import bean.Difference; |
|  | import bean.Intersection; |
|  | import bean.Node; |
|  | import bean.Rectangle; |
|  | import bean.Rotate; |
|  | import bean.Scale; |
|  | import bean.Square; |
|  | import bean.Translate; |
|  | import bean.Triangle; |
|  | import bean.Union; |
|  | /\*\* |
|  | \* @class Parser |
|  | \* This Parser class contains all methods for parsing user commands |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | public class Parser { |
|  | private static ArrayList<String> errList = new ArrayList<String>(); |
|  |  |
|  | /\*\* |
|  | \* Parse user input into Nodes |
|  | \* @param input the user input |
|  | \* @return return a Node |
|  | \*/ |
|  |  |
|  | public static Node parse(String input){ |
|  | //Removing all newlines, tabs and whitespace from input string |
|  | String instructions = input.replaceAll("\n", "").replaceAll("\t", "").replaceAll(" ", "").replaceAll("\r", ""); |
|  | //Using getArgList to produce a list of the comma separated arguments |
|  | ArrayList<String> argList = getArgList(instructions); |
|  |  |
|  | if(argList==null){ |
|  | errList.add("Oops, you must've forgentten to enter an opening/closing bracket !"); |
|  | } |
|  | else{ |
|  | //Recursively creating the required nodes |
|  | //SHAPES |
|  | if (instructions.startsWith("Square(")){ |
|  | if (checkArgSize(argList.size(),1,true)){ |
|  | if(checkDouble(argList.get(0))) |
|  | return new Square(Double.parseDouble(argList.get(0))); |
|  | else |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | } |
|  | else |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | } |
|  |  |
|  | else if (instructions.startsWith("Rectangle(")){ |
|  | if (checkArgSize(argList.size(),2,true)){ |
|  | if(checkDouble(argList.get(0))&&checkDouble(argList.get(1))) |
|  | return new Rectangle(Double.parseDouble(argList.get(0)), Double.parseDouble(argList.get(1))); |
|  | else |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | } |
|  | else |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | } |
|  |  |
|  | else if (instructions.startsWith("Triangle(")){ |
|  | if (checkArgSize(argList.size(),1,true)){ |
|  | if(checkDouble(argList.get(0))) |
|  | return new Triangle(Double.parseDouble(argList.get(0))); |
|  | else |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | } |
|  | else |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | } |
|  |  |
|  | else if (instructions.startsWith("Circle(")){ |
|  | if (checkArgSize(argList.size(),1,true)){ |
|  | if(checkDouble(argList.get(0))) |
|  | return new Circle(Double.parseDouble(argList.get(0))); |
|  | else |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | } |
|  | else |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | } |
|  |  |
|  | //A curve segment is defined by 4 points, each with an x and y value (8 values total) or 1 point wit an x ad y value (2 values total) |
|  | //A curve is a sequence of one or more curve segments |
|  | else if (instructions.startsWith("Curve(")){ |
|  |  |
|  | //Create an empty list of curve segment coordinates |
|  | ArrayList<double[]> curveSegmentList = new ArrayList<double[]>(); |
|  |  |
|  | //If this curve consists of more than one curve segment, add each of them to the list |
|  | if (argList.get(0).startsWith("(")){ |
|  | for (int i = 0; i < argList.size(); ++i){ |
|  |  |
|  | //Taking the current curve segment string and converting it to a double array |
|  | ArrayList<String> argList2 = getArgList(argList.get(i)); |
|  | double[] curvePoints = new double[argList2.size()]; |
|  | for (int j = 0; j < argList2.size(); ++j){ |
|  | if (checkArgSize(argList2.size(),2,true)||checkArgSize(argList2.size(),8,true)){ |
|  | if(checkDouble(argList2.get(j))) |
|  | curvePoints[j] = Double.parseDouble(argList2.get(j)); |
|  | else{ |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | break; |
|  | } |
|  | } |
|  | else{ |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | break; |
|  | } |
|  | } |
|  | //Adding the current curve segment to the list |
|  | curveSegmentList.add(curvePoints); |
|  | } |
|  | } |
|  | //If the curve consists of only one curve segment, add it to the list |
|  | else{ |
|  | double[] curvePoints = new double[argList.size()]; |
|  | for (int i = 0; i < argList.size(); ++i){ |
|  | if (checkArgSize(argList.size(),8,true)){ |
|  | if(checkDouble(argList.get(i))) |
|  | curvePoints[i] = Double.parseDouble(argList.get(i)); |
|  | else{ |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | break; |
|  | } |
|  | } |
|  | else{ |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | break; |
|  | } |
|  | } |
|  | curveSegmentList.add(curvePoints); |
|  | } |
|  | return new ClosedCurve(curveSegmentList); |
|  | } |
|  |  |
|  |  |
|  | //TRANSFORMS |
|  | //Repeat transforms take an extra argument (the number of repeats) |
|  | else if (instructions.startsWith("RotateN(")){ |
|  | if (checkArgSize(argList.size(),3,true)){ |
|  | if (checkDouble(argList.get(1)) && checkInteger(argList.get(2))) |
|  | return new Rotate(parse(argList.get(0)),Double.parseDouble(argList.get(1)),Integer.parseInt(argList.get(2))); |
|  | else |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | } |
|  | else |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | } |
|  |  |
|  | else if (instructions.startsWith("Rotate(")){ |
|  | if (checkArgSize(argList.size(),2,true)){ |
|  | if(checkDouble(argList.get(1))) |
|  | return new Rotate(parse(argList.get(0)),Double.parseDouble(argList.get(1))); |
|  | else |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | } |
|  | else |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | } |
|  |  |
|  | else if (instructions.startsWith("TranslateN(")){ |
|  | if (checkArgSize(argList.size(),4,true)){ |
|  | if (checkDouble(argList.get(1)) && checkDouble(argList.get(2)) && checkInteger(argList.get(3))) |
|  | return new Translate(parse(argList.get(0)),Double.parseDouble(argList.get(1)), Double.parseDouble(argList.get(2)),Integer.parseInt(argList.get(3))); |
|  | else |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | } |
|  | else |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | } |
|  |  |
|  | else if (instructions.startsWith("Translate(")){ |
|  | if (checkArgSize(argList.size(),3,true)){ |
|  | if (checkDouble(argList.get(1)) && checkDouble(argList.get(2))) |
|  | return new Translate(parse(argList.get(0)),Double.parseDouble(argList.get(1)), Double.parseDouble(argList.get(2))); |
|  | else |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | } |
|  | else |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | } |
|  |  |
|  | else if (instructions.startsWith("ScaleN(")){ |
|  | //Two different calls for Scale because there are two different Scale functions (uniform and non-uniform) |
|  | if (argList.size() == 3){ |
|  | if (checkDouble(argList.get(1)) && checkInteger(argList.get(2))) |
|  | return new Scale(parse(argList.get(0)),Double.parseDouble(argList.get(1)), Double.parseDouble(argList.get(1)),Integer.parseInt(argList.get(2))); |
|  | else |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | } |
|  | else if (argList.size() == 4){ |
|  | if (checkDouble(argList.get(1)) && checkDouble(argList.get(2)) && checkInteger(argList.get(3))) |
|  | return new Scale(parse(argList.get(0)),Double.parseDouble(argList.get(1)), Double.parseDouble(argList.get(2)),Integer.parseInt(argList.get(3))); |
|  | else |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | } |
|  | else |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | } |
|  |  |
|  | else if (instructions.startsWith("Scale(")){ |
|  | if (argList.size() == 2){ |
|  | if (checkDouble(argList.get(1))) |
|  | return new Scale(parse(argList.get(0)),Double.parseDouble(argList.get(1)), Double.parseDouble(argList.get(1))); |
|  | else |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | } |
|  | else if (argList.size() == 3){ |
|  | if (checkDouble(argList.get(1)) && checkDouble(argList.get(2))) |
|  | return new Scale(parse(argList.get(0)),Double.parseDouble(argList.get(1)), Double.parseDouble(argList.get(2))); |
|  | else |
|  | errList.add("You have entered wrong format of arguments for \""+instructions+"\""); |
|  | } |
|  | else |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | } |
|  |  |
|  | //MIXES |
|  | //Mix nodes take a List of Nodes as their input, so generate a List by parsing each |
|  | //element of argsList into a Node |
|  | else if (instructions.startsWith("Union(")){ |
|  | if (checkArgSize(argList.size(),2,false)){ |
|  | ArrayList<Node> mixNodes = new ArrayList<Node>(); |
|  | for (int i = 0; i < argList.size(); ++i){ |
|  | mixNodes.add(parse(argList.get(i))); |
|  | } |
|  | return new Union(mixNodes); |
|  | } |
|  | else |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | } |
|  |  |
|  | else if (instructions.startsWith("Intersection(")){ |
|  | if (checkArgSize(argList.size(),2,false)){ |
|  | ArrayList<Node> mixNodes = new ArrayList<Node>(); |
|  | for (int i = 0; i < argList.size(); ++i){ |
|  | mixNodes.add(parse(argList.get(i))); |
|  | } |
|  | return new Intersection(mixNodes); |
|  | } |
|  | else |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | } |
|  |  |
|  | else if (instructions.startsWith("Difference(")){ |
|  | if (checkArgSize(argList.size(),2,false)){ |
|  | ArrayList<Node> mixNodes = new ArrayList<Node>(); |
|  | for (int i = 0; i < argList.size(); ++i){ |
|  | mixNodes.add(parse(argList.get(i))); |
|  | } |
|  | return new Difference(mixNodes); |
|  | } |
|  | else |
|  | errList.add("You have entered wrong number of arguments for \""+instructions+"\""); |
|  | } |
|  | else |
|  | errList.add("You have a syntax error near \""+instructions.substring(0, Math.min(12,Math.min(instructions.length(),instructions.indexOf('('))))+"\""); |
|  | } |
|  | return null; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* |
|  | \* Store error occurred when parsing the user input instructions |
|  | \* |
|  | \*/ |
|  |  |
|  | public static ArrayList<String> returnErrList(){ |
|  | ArrayList<String> temp=new ArrayList<String>(errList); |
|  | errList.clear(); |
|  | return temp; |
|  | } |
|  |  |
|  |  |
|  | /\*\* |
|  | \* Split the user input instructions into a well-formatted ArrayList |
|  | \* @param instructions the user input instructions |
|  | \* @return return an ArrayList of String for formatted instructions |
|  | \*/ |
|  | private static ArrayList<String> getArgList(String instructions){ |
|  | ArrayList<String> argList = new ArrayList<String>(); |
|  | if(checkBrackets(instructions)){ |
|  | //Creating a substring of whatever is contained within the outermost pair of brackets |
|  | String argument = instructions.substring(instructions.indexOf('(') + 1,instructions.lastIndexOf(')')); |
|  | int endIndex = argument.length() - 1; |
|  | int pointer = argument.length() - 1; |
|  | int brackets = 0; |
|  | boolean clause = false; |
|  |  |
|  |  |
|  | //For each character in argument, moving from the end to the beginning |
|  | while(pointer > 0){ |
|  | //If a close bracket is seen and we are not currently in a clause, begin a clause and keep |
|  | //track of how many open and close brackets there have been |
|  | if (argument.charAt(pointer) == ')'){ |
|  | clause = true; |
|  | ++brackets; |
|  | } |
|  | if (argument.charAt(pointer) == '('){ |
|  | --brackets; |
|  | } |
|  | //If we have reached the end of the current clause |
|  | if (brackets == 0 && clause){ |
|  | //End the current clause |
|  | clause = false; |
|  | } |
|  | //If we have reached a comma and are not in a clause i.e. we have moved the pointer |
|  | //over one entire argument |
|  | if (argument.charAt(pointer) == ',' && !clause){ |
|  | //Add that argument to the beginning of argList |
|  | argList.add(0, argument.substring(pointer + 1, endIndex + 1)); |
|  |  |
|  | //Move the end pointer to the beginning of the next argument |
|  | endIndex = pointer - 1; |
|  | } |
|  | //Move to the next character in argument |
|  | --pointer; |
|  | } |
|  | //Add the final argument (that doesn't have a comma at the start of it) to argList |
|  | argList.add(0, argument.substring(0, endIndex + 1)); |
|  |  |
|  |  |
|  | return argList; |
|  | } |
|  | else |
|  | return null; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Check if the number of parameters is correct |
|  | \* @param inputSize the number of user input parameters |
|  | \* @param correctSize the correct number of parameter for the Node type |
|  | \* @param fixedArgSize if he number of parameters for the Node type should be fixed |
|  | \* @return return true if the number of parameters is correct |
|  | \*/ |
|  |  |
|  | public static boolean checkArgSize(int inputSize,int correctSize, boolean fixedArgSize){ |
|  | if(fixedArgSize){ |
|  | if (inputSize!=correctSize) |
|  | return false; |
|  | } |
|  | else |
|  | if(inputSize<correctSize) |
|  | return false; |
|  | return true; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Check if the parameter is parsable to Integer |
|  | \* @param str the user input parameter |
|  | \* @return return true if the parameter if parsable |
|  | \*/ |
|  | public static boolean checkInteger(String str){ |
|  | try |
|  | { |
|  | Integer.parseInt(str); |
|  | return true; |
|  | } |
|  | catch( Exception e) |
|  | { |
|  | return false; |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Check if the parameter is parsable to Double |
|  | \* @param str the user input parameter |
|  | \* @return return true if the parameter if parsable |
|  | \*/ |
|  |  |
|  | public static boolean checkDouble(String str){ |
|  | try |
|  | { |
|  | Double.parseDouble(str); |
|  | return true; |
|  | } |
|  | catch( Exception e) |
|  | { |
|  | return false; |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Parse user input into Nodes |
|  | \* @param str the formatted input |
|  | \* @return return true if the number of brackets is correct |
|  | \*/ |
|  |  |
|  | public static boolean checkBrackets(String str){ |
|  | if (str.trim().length() == 0){ |
|  | return false; |
|  | } |
|  | int brackets = 0; |
|  | boolean anyBracket=false; |
|  | for (int i = 0; i < str.length(); i++) |
|  | { |
|  | char current = str.charAt(i); |
|  | if (current == '('){ |
|  | anyBracket=true; |
|  | ++brackets; |
|  | } |
|  | if (current == ')'){ |
|  | anyBracket=true; |
|  | --brackets; |
|  | } |
|  | if (brackets < 0){ |
|  | return false; |
|  | } |
|  | } |
|  | if (brackets != 0){ |
|  | return false; |
|  | } |
|  | if(!anyBracket) |
|  | return false; |
|  | return true; |
|  | } |
|  | } |

### SaveFile.java

|  |
| --- |
| package util; |
|  |  |
|  | import java.awt.FileDialog; |
|  | import java.awt.Frame; |
|  | import java.io.BufferedWriter; |
|  | import java.io.FileWriter; |
|  | import java.io.IOException; |
|  | /\*\* |
|  | \* @class SaveFile |
|  | \* This SaveFile class initializes a FileDialog for saving existing drawing into a file. |
|  | \* |
|  | \* @author Donal Evans |
|  | \* @author Qiru Wang |
|  | \*/ |
|  |  |
|  | @SuppressWarnings("serial") |
|  | public class SaveFile extends Frame { |
|  |  |
|  | /\*\* |
|  | \* Save the user commands into a .draw file |
|  | \* @param commands the user commands |
|  | \*/ |
|  |  |
|  | public SaveFile(String commands) { |
|  | FileDialog fc=new FileDialog(this,"Save to a file",1); |
|  | fc.setVisible(true); |
|  | if(fc.getFile()!=null){ |
|  | String extension = new String(""); |
|  | if (!fc.getFile().endsWith(".draw")){ |
|  | extension = ".draw"; |
|  | } |
|  | try ( |
|  | FileWriter fileWriter = new FileWriter(fc.getDirectory()+fc.getFile()+extension); |
|  | BufferedWriter bufferedWriter = new BufferedWriter(fileWriter); |
|  | ) |
|  | { |
|  | bufferedWriter.write(commands); |
|  | bufferedWriter.close(); |
|  | } catch (IOException x) { |
|  | System.err.format("IOException: %s%n", x); |
|  | } |
|  | } |
|  | } |
|  | } |

# Appendix 3 – Contribution Report