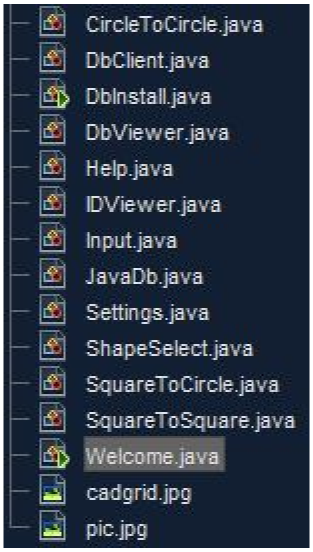
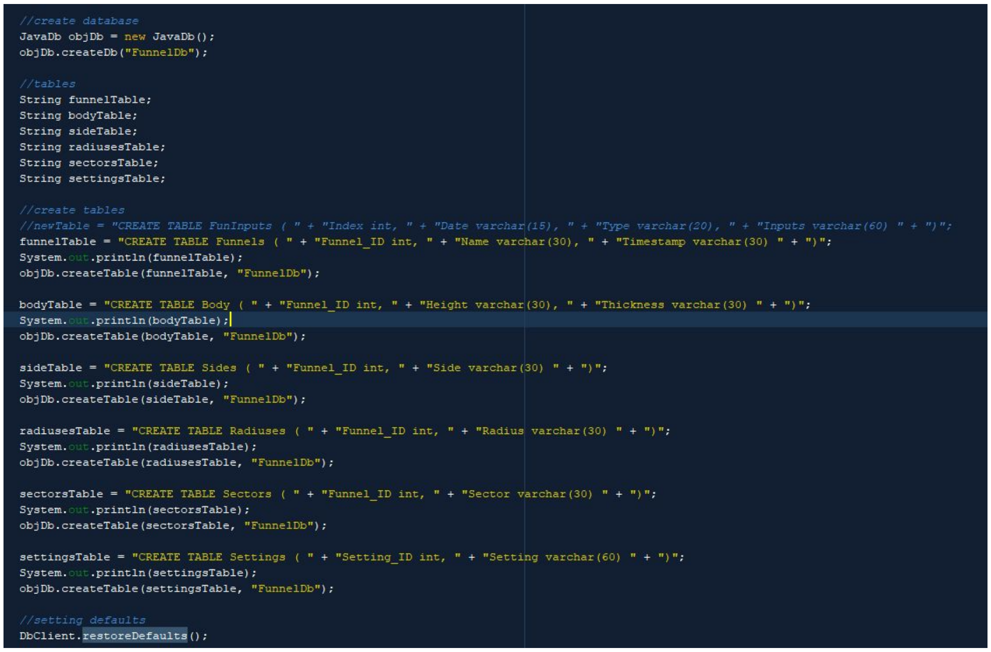
Criterion C

Overall Structure and Classes



```
CircleToCircle.java
DbClient.java
DbInstall.java
DbViewer.java
Help.java
IDViewer.java
Input.java
JavaDb.java
Settings.java
ShapeSelect.java
SquareToCircle.java
SquareToSquare.java
Welcome.java
cadgrid.jpg
pic.jpg
```

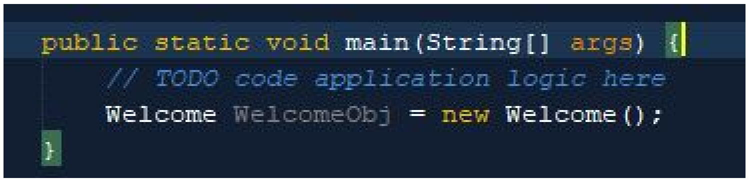
FunnelFinder is a Java based OOP Script generation program written using Netbeans IDE, and designed for AutoCad users to automate the creation of funnels in AutoCad.



```
//create database
JavaDb objDb = new JavaDb();
objDb.createDb("FunnelDb");

//tables
String funnelTable;
String bodyTable;
String sideTable;
String radiusesTable;
String sectorsTable;
String settingsTable;

//create tables
//newTable = "CREATE TABLE FunInputs ( " + "Index int, " + "Date varchar(15), " + "Type varchar(20), " + "Inputs varchar(60) " + ")";
funnelTable = "CREATE TABLE Funnels ( " + "Funnel_ID int, " + "Name varchar(30), " + "Timestamp varchar(30) " + ")";
System.out.println(funnelTable);
objDb.createTable(funnelTable, "FunnelDb");

bodyTable = "CREATE TABLE Body ( " + "Funnel_ID int, " + "Height varchar(30), " + "Thickness varchar(30) " + ")";
System.out.println(bodyTable);
objDb.createTable(bodyTable, "FunnelDb");

sideTable = "CREATE TABLE Sides ( " + "Funnel_ID int, " + "Side varchar(30) " + ")";
System.out.println(sideTable);
objDb.createTable(sideTable, "FunnelDb");

radiusesTable = "CREATE TABLE Radiuses ( " + "Funnel_ID int, " + "Radius varchar(30) " + ")";
System.out.println(radiusesTable);
objDb.createTable(radiusesTable, "FunnelDb");

sectorsTable = "CREATE TABLE Sectors ( " + "Funnel_ID int, " + "Sector varchar(30) " + ")";
System.out.println(sectorsTable);
objDb.createTable(sectorsTable, "FunnelDb");

settingsTable = "CREATE TABLE Settings ( " + "Setting_ID int, " + "Setting varchar(60) " + ")";
System.out.println(settingsTable);
objDb.createTable(settingsTable, "FunnelDb");

//setting defaults
DbClient.restoreDefaults();
```

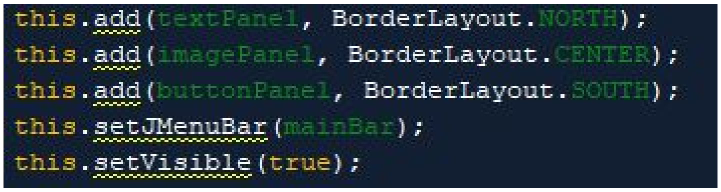
DbInstall class constructs an object of the JavaDb class and creates the database and the 6 tables. Defaults are assigned to the Settings table using the restoreDefaults() method from the DbClient class.

To open the program, the user runs the Welcome class, containing a main method that opens the welcome class frame with a constructor. All frames are opened with a constructor.

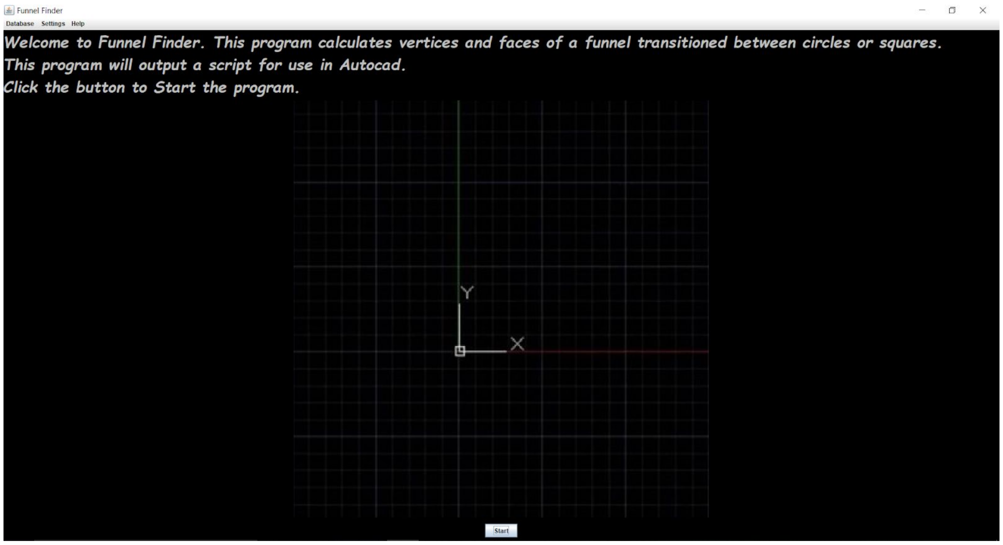


```
public static void main(String[] args) {
    // TODO code application logic here
    Welcome WelcomeObj = new Welcome();
}
```

The welcome frame is separated up into menubar, labels, image, and buttons. The welcome class introduces the user to the basic structure of my modular GUI. MenuBar is on the top, buttons are on the button, labels/fields/images, are in the middle.

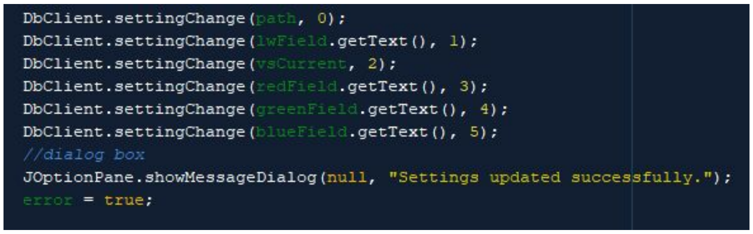


```
this.add(textPanel, BorderLayout.NORTH);
this.add(imagePanel, BorderLayout.CENTER);
this.add(buttonPanel, BorderLayout.SOUTH);
this.setJMenuBar(mainBar);
this.setVisible(true);
```

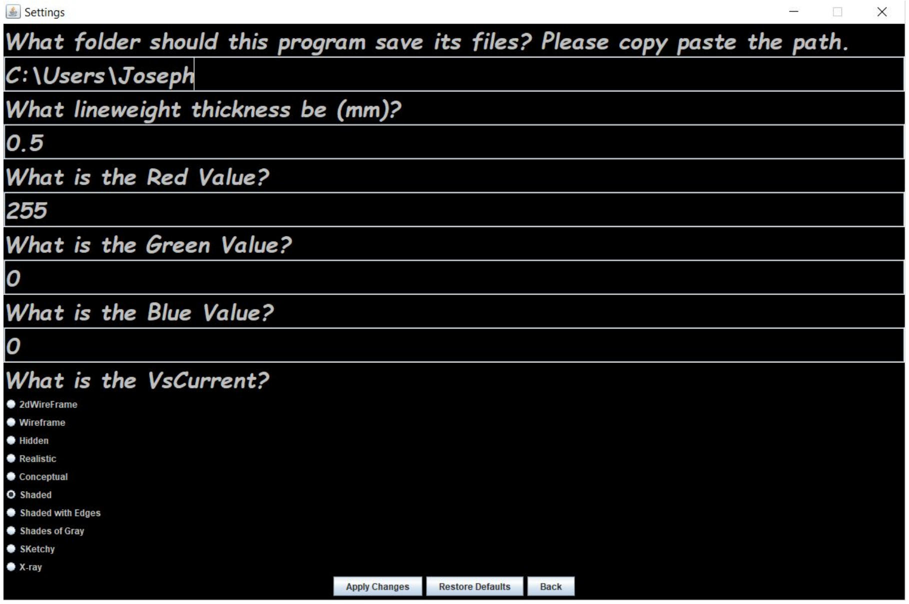


Welcome to Funnel Finder. This program calculates vertices and faces of a funnel transitioned between circles or squares.
This program will output a script for use in Autocad.
Click the button to Start the program.

Before the user creates any funnel scripts, they update the Settings table which determines how and where the script is generated. The Settings frame is accessed by using the settings menu option in the settings menu bar. The Settings frame will display the current settings values in the fields. The user can enter new settings or restore the settings defaults. Before any values are changed, invalid values such as non existent path, invalid RGB values, or out of scope values are checked for.



```
DbClient.settingChange(path, 0);
DbClient.settingChange(lwField.getText(), 1);
DbClient.settingChange(vsCurrent, 2);
DbClient.settingChange(redField.getText(), 3);
DbClient.settingChange(greenField.getText(), 4);
DbClient.settingChange(blueField.getText(), 5);
//dialog box
JOptionPane.showMessageDialog(null, "Settings updated successfully.");
error = true;
```



What folder should this program save its files? Please copy paste the path.

C:\Users\Joseph

What lineweight thickness be (mm)?

0.5

What is the Red Value?

255

What is the Green Value?

0

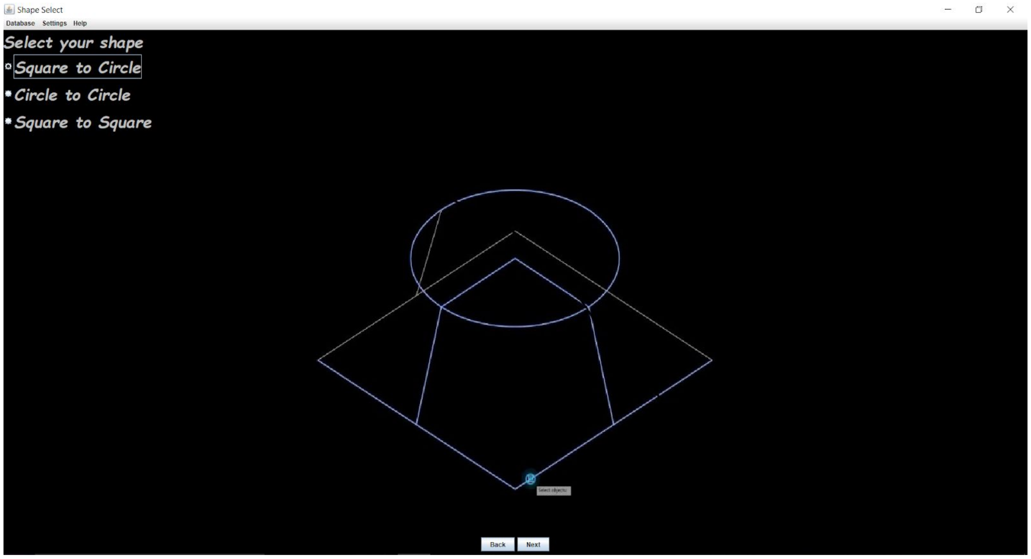
What is the Blue Value?

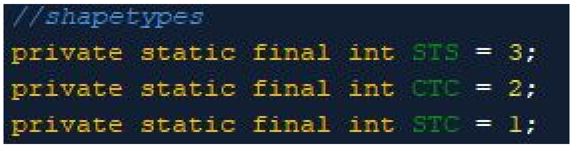
0

What is the VsCurrent?

● 2dWireFrame
● Wireframe
● Hidden
● Realistic
● Conceptual
○ Shaded
● Shaded with Edges
● Shades of Gray
● SKetchy
● X-ray

When the user clicks the Start Button from the welcome Frame, Welcome frame is disposed and ShapeSelect frame is opened. The ShapeSelect frame contains 3 JRadioButtons, each allowing the selection of one of the 3 shape types (Square To Circle (STC), Circle To Circle (CTC), Square To Square(STS)). When Next Button is clicked, the type is passed on as an integer to the Inputs class.



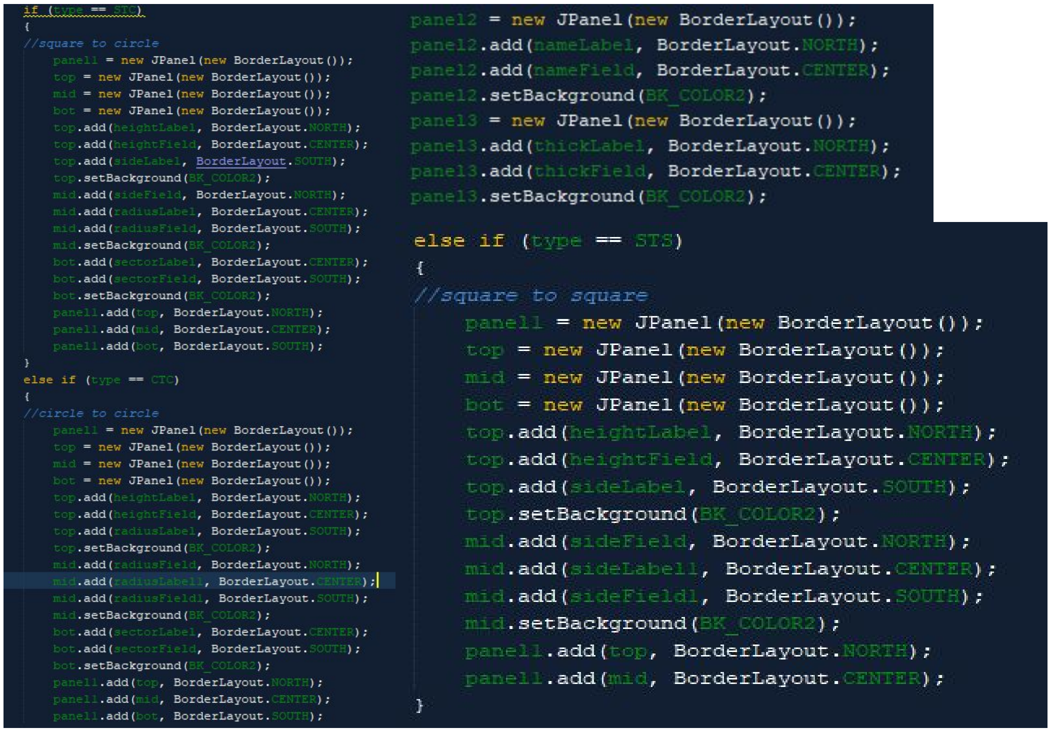


```
//shapetypes
private static final int STS = 3;
private static final int CTC = 2;
private static final int STC = 1;
```

The inputs class will construct the frame differently depending on the ShapeType received.

Different shape types require different inputs, so the frame needs different labels and fields.

All shape types need height, name, and thickness. However, in addition to that, STC needs radius, side, sectors, STS needs 2 sides, and CTC needs 2 radiuses and sectors.

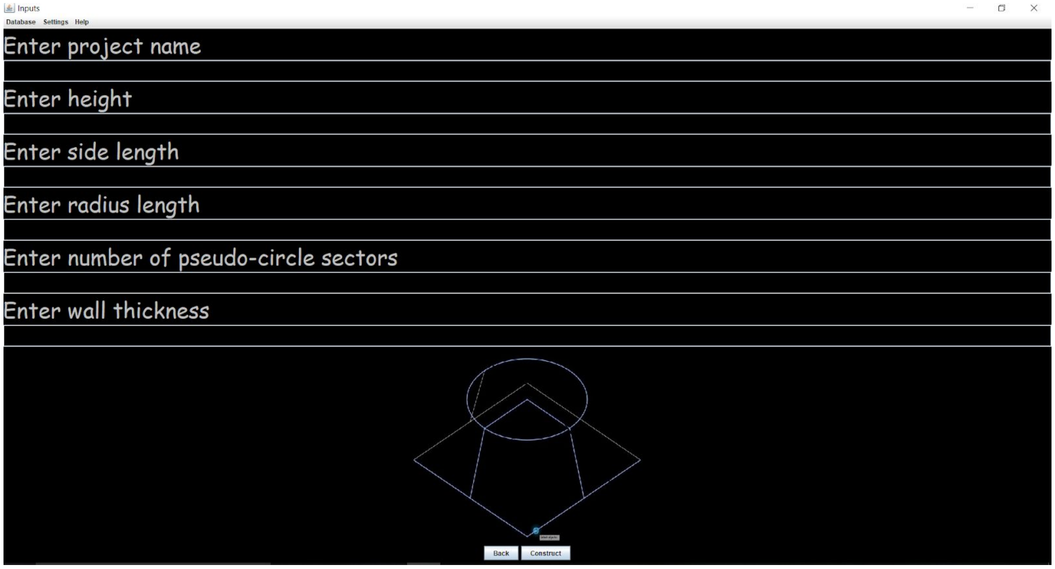


```
if (type == STC)
{
//square to circle
    panel1 = new JPanel(new BorderLayout());
    top = new JPanel(new BorderLayout());
    mid = new JPanel(new BorderLayout());
    bot = new JPanel(new BorderLayout());
    top.add(heightLabel, BorderLayout.NORTH);
    top.add(heightField, BorderLayout.CENTER);
    top.add(sideLabel, BorderLayout.SOUTH);
    top.setBackground(BK_COLOR2);
    mid.add(sideField, BorderLayout.NORTH);
    mid.add(radiusLabel, BorderLayout.CENTER);
    mid.add(radiusField, BorderLayout.SOUTH);
    mid.setBackground(BK_COLOR2);
    bot.add(sectorLabel, BorderLayout.CENTER);
    bot.add(sectorField, BorderLayout.SOUTH);
    bot.setBackground(BK_COLOR2);
    panel1.add(top, BorderLayout.NORTH);
    panel1.add(mid, BorderLayout.CENTER);
    panel1.add(bot, BorderLayout.SOUTH);
}
else if (type == CTC)
{
//circle to circle
    panel1 = new JPanel(new BorderLayout());
    top = new JPanel(new BorderLayout());
    mid = new JPanel(new BorderLayout());
    bot = new JPanel(new BorderLayout());
    top.add(heightLabel, BorderLayout.NORTH);
    top.add(heightField, BorderLayout.CENTER);
    top.add(radiusLabel, BorderLayout.SOUTH);
    top.setBackground(BK_COLOR2);
    mid.add(radiusField, BorderLayout.NORTH);
    mid.add(radiusLabel1, BorderLayout.CENTER);
    mid.add(radiusField1, BorderLayout.SOUTH);
    mid.setBackground(BK_COLOR2);
    bot.add(sectorLabel, BorderLayout.CENTER);
    bot.add(sectorField, BorderLayout.SOUTH);
    bot.setBackground(BK_COLOR2);
    panel1.add(top, BorderLayout.NORTH);
    panel1.add(mid, BorderLayout.CENTER);
    panel1.add(bot, BorderLayout.SOUTH);
```

```
panel2 = new JPanel(new BorderLayout());
panel2.add(nameLabel, BorderLayout.NORTH);
panel2.add(nameField, BorderLayout.CENTER);
panel2.setBackground(BK_COLOR2);
panel3 = new JPanel(new BorderLayout());
panel3.add(thickLabel, BorderLayout.NORTH);
panel3.add(thickField, BorderLayout.CENTER);
panel3.setBackground(BK_COLOR2);
```

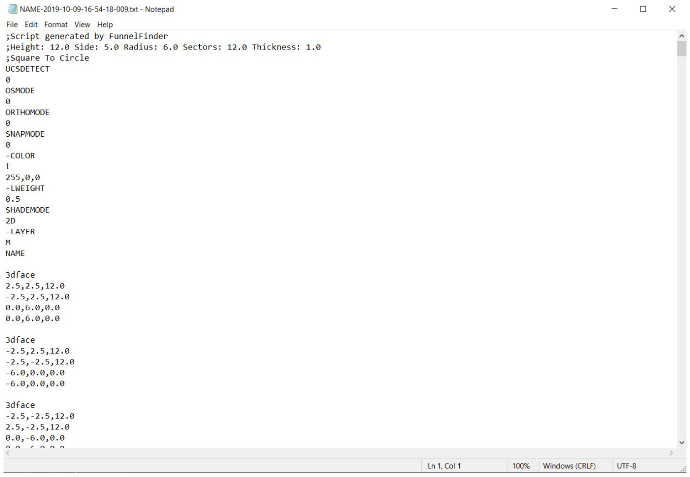
```
else if (type == STS)
{
//square to square
    panel1 = new JPanel(new BorderLayout());
    top = new JPanel(new BorderLayout());
    mid = new JPanel(new BorderLayout());
    bot = new JPanel(new BorderLayout());
    top.add(heightLabel, BorderLayout.NORTH);
    top.add(heightField, BorderLayout.CENTER);
    top.add(sideLabel, BorderLayout.SOUTH);
    top.setBackground(BK_COLOR2);
    mid.add(sideField, BorderLayout.NORTH);
    mid.add(sideLabel1, BorderLayout.CENTER);
    mid.add(sideField1, BorderLayout.SOUTH);
    mid.setBackground(BK_COLOR2);
    panel1.add(top, BorderLayout.NORTH);
    panel1.add(mid, BorderLayout.CENTER);
}
```

The Input frame below was constructed with a shapeType of 1 (STC).



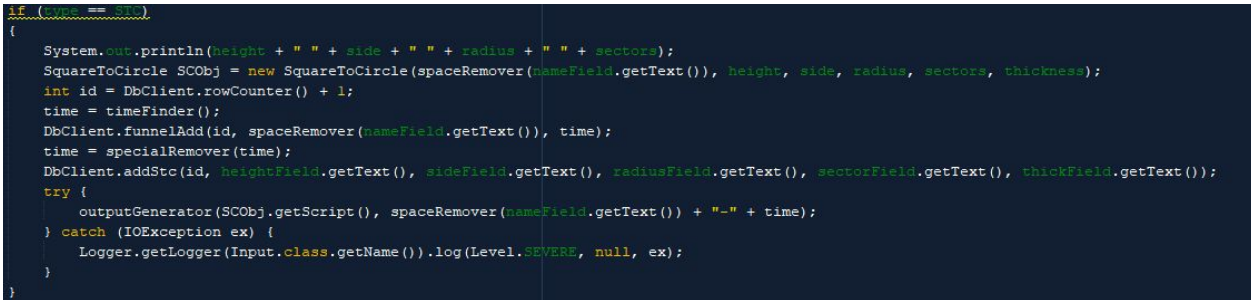
When the user clicks Construct, first the values are proofread for invalid values, such as negative numbers, special characters, or values too long. After that, a respective funnel object will be constructed (using the funnel finder algorithm), the values will be added to the proper tables, and the script will be extracted from the funnel object and added to a file (as a .scr but can be viewed as a .txt). The file will be saved in the path contained within the settings table database.

NAME-2019-10-09-16-54-18-009.scr   10/9/2019 4:54 PM   AutoCAD Script   21 KB


NAME-2019-10-09-16-54-18-009.txt - Notepad
File Edit Format View Help
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

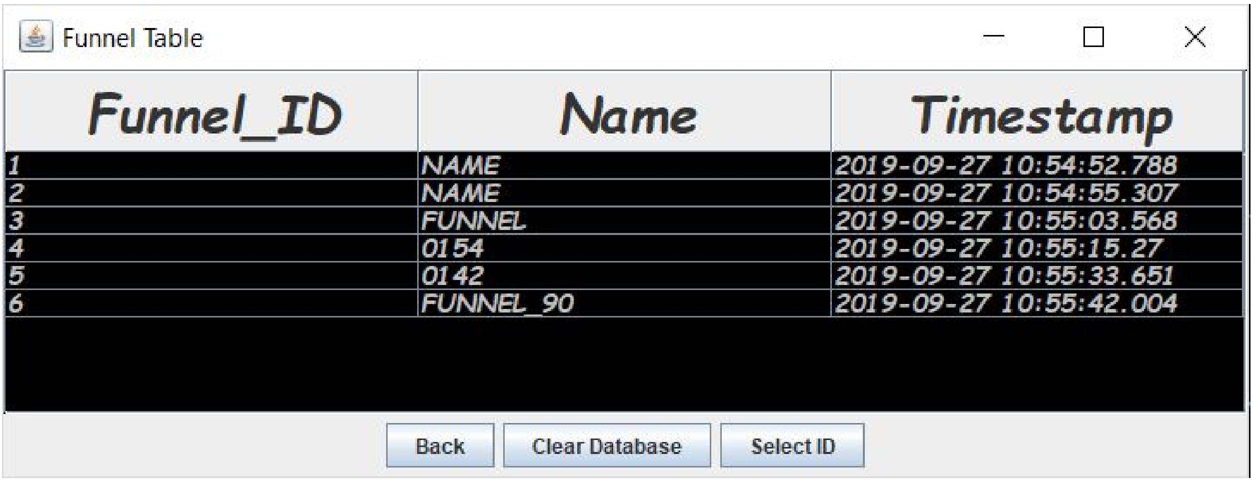
```
;Script generated by FunnelFinder
;Height: 12.0 Side: 5.0 Radius: 6.0 Sectors: 12.0 Thickness: 1.0
;Square To Circle
UCSDETECT
0
OSMODE
0
ORTHOMODE
0
SNAPMODE
0
-COLOR
t
255,0,0
-LWEIGHT
0.5
SHADEMODE
2D
-LAYER
M
NAME

3dface
2.5,2.5,12.0
-2.5,2.5,12.0
0.0,6.0,0.0
0.0,6.0,0.0

3dface
-2.5,2.5,12.0
-2.5,-2.5,12.0
-6.0,0.0,0.0
-6.0,0.0,0.0

3dface
-2.5,-2.5,12.0
2.5,-2.5,12.0
0.0,-6.0,0.0
```

The following Code is an example of this process for the STC funnel type.



```
if (type == STC)
{
    System.out.println(height + " " + side + " " + radius + " " + sectors);
    SquareToCircle SCObj = new SquareToCircle(spaceRemover(nameField.getText()), height, side, radius, sectors, thickness);
    int id = DbClient.rowCounter() + 1;
    time = timeFinder();
    DbClient.funnelAdd(id, spaceRemover(nameField.getText()), time);
    time = specialRemover(time);
    DbClient.addStc(id, heightField.getText(), sideField.getText(), radiusField.getText(), sectorField.getText(), thickField.getText());
    try {
        outputGenerator(SCObj.getScript(), spaceRemover(nameField.getText()) + "-" + time);
    } catch (IOException ex) {
        Logger.getLogger(Input.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

After a funnel has been added to the database, the user can view all the funnels in a table. The DbViewer frame is accessed by using the Database menu option in the settings menu bar.

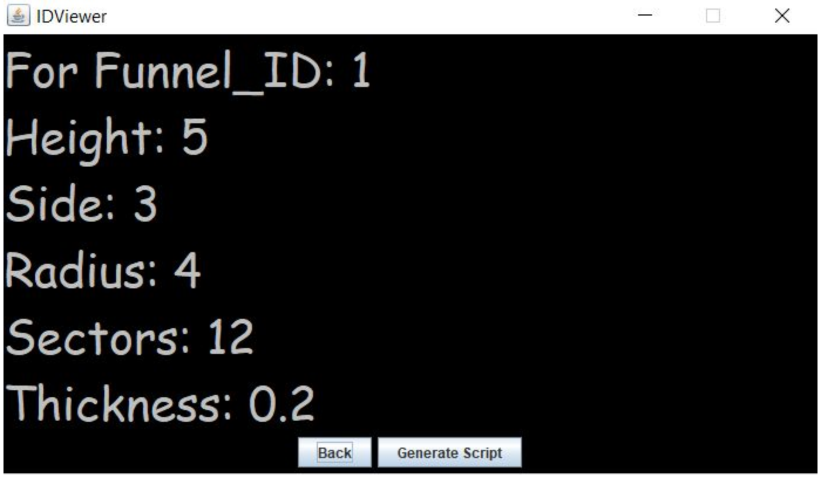


Funnel_ID	Name	Timestamp
1	NAME	2019-09-27 10:54:52.788
2	NAME	2019-09-27 10:54:55.307
3	FUNNEL	2019-09-27 10:55:03.568
4	0154	2019-09-27 10:55:15.27
5	0142	2019-09-27 10:55:33.651
6	FUNNEL_90	2019-09-27 10:55:42.004

The DbViewer frame displays the the Funnels table from the database in a JTable. The user can either clear the database or select an ID. Each calls different methods.

The ID Viewer class retrieves the inputs from the database using the funnel ID searchMethods from dbClient.

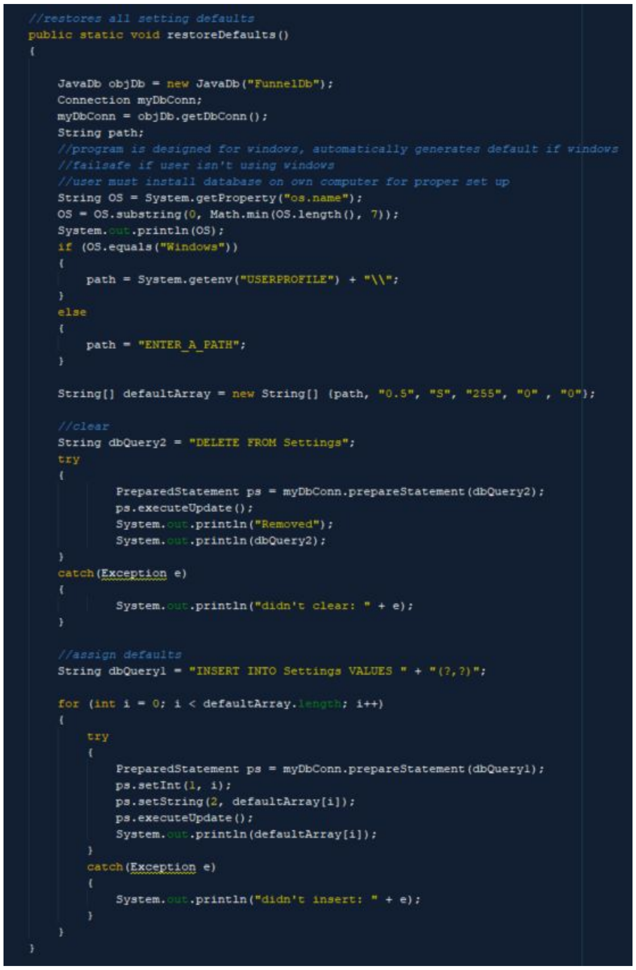
The generate script button can be used to create a funnel input object, and write a script by constructing an input object and accessing the script generation methods.



```
For Funnel_ID: 1
Height: 5
Side: 3
Radius: 4
Sectors: 12
Thickness: 0.2
```

Algorithmic Thinking

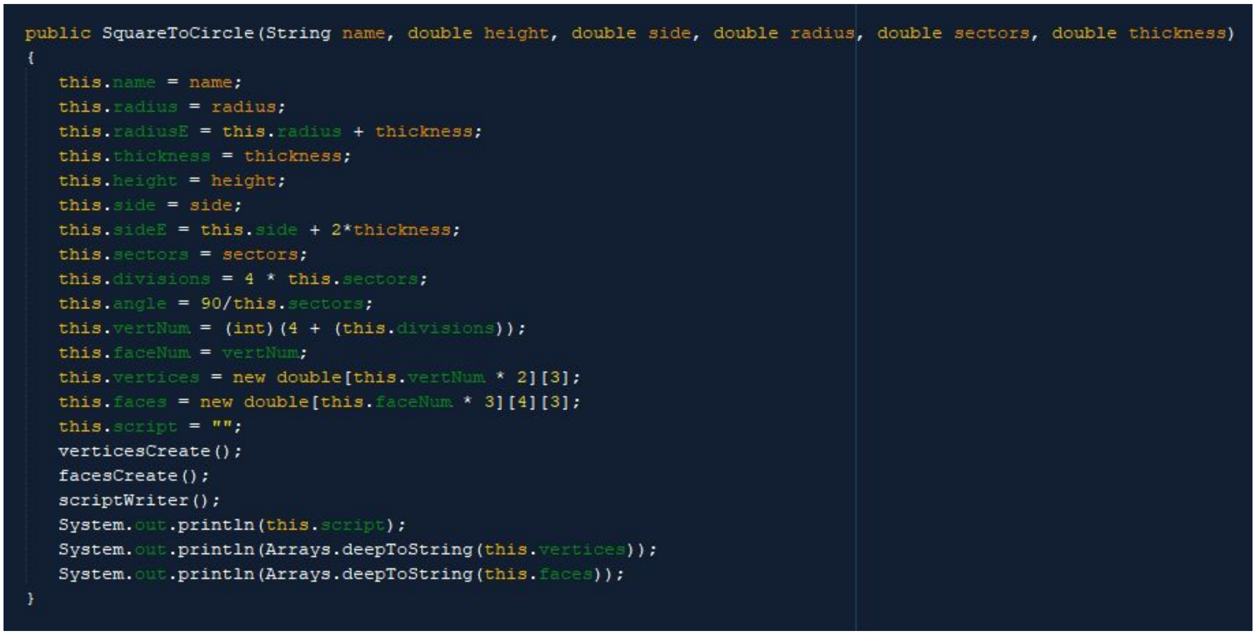
Method restoreDefaults.



```
//restores all setting defaults
public static void restoreDefaults()
{

    JavaDb objDb = new JavaDb("FunnelDb");
    Connection myDbConn;
    myDbConn = objDb.getDbConn();
    String path;
    //program is designed for windows, automatically generates default if windows
    //failsafe if user isn't using windows
    //user must install database on own computer for proper set up
    String OS = System.getProperty("os.name");
    OS = OS.substring(0, Math.min(OS.length(), 7));
    System.out.println(OS);
    if (OS.equals("Windows"))
    {
        path = System.getenv("USERPROFILE") + "\\";
    }
    else
    {
        path = "ENTER_A_PATH";
    }

    String[] defaultArray = new String[] {path, "0.5", "S", "255", "0" , "0"};

    //clear
    String dbQuery2 = "DELETE FROM Settings";
    try
    {
            PreparedStatement ps = myDbConn.prepareStatement(dbQuery2);
            ps.executeUpdate();
            System.out.println("Removed");
            System.out.println(dbQuery2);
    }
    catch(Exception e)
    {
            System.out.println("didn't clear: " + e);
    }

    //assign defaults
    String dbQuery1 = "INSERT INTO Settings VALUES " + "(?,?)";

    for (int i = 0; i < defaultArray.length; i++)
    {
        try
        {
            PreparedStatement ps = myDbConn.prepareStatement(dbQuery1);
            ps.setInt(1, i);
            ps.setString(2, defaultArray[i]);
            ps.executeUpdate();
            System.out.println(defaultArray[i]);
        }
        catch(Exception e)
        {
            System.out.println("didn't insert: " + e);
        }
    }
}
```

Method clears the Settings table and assigns new defaults by reading from an Array. The path default is determined depending on the OS.

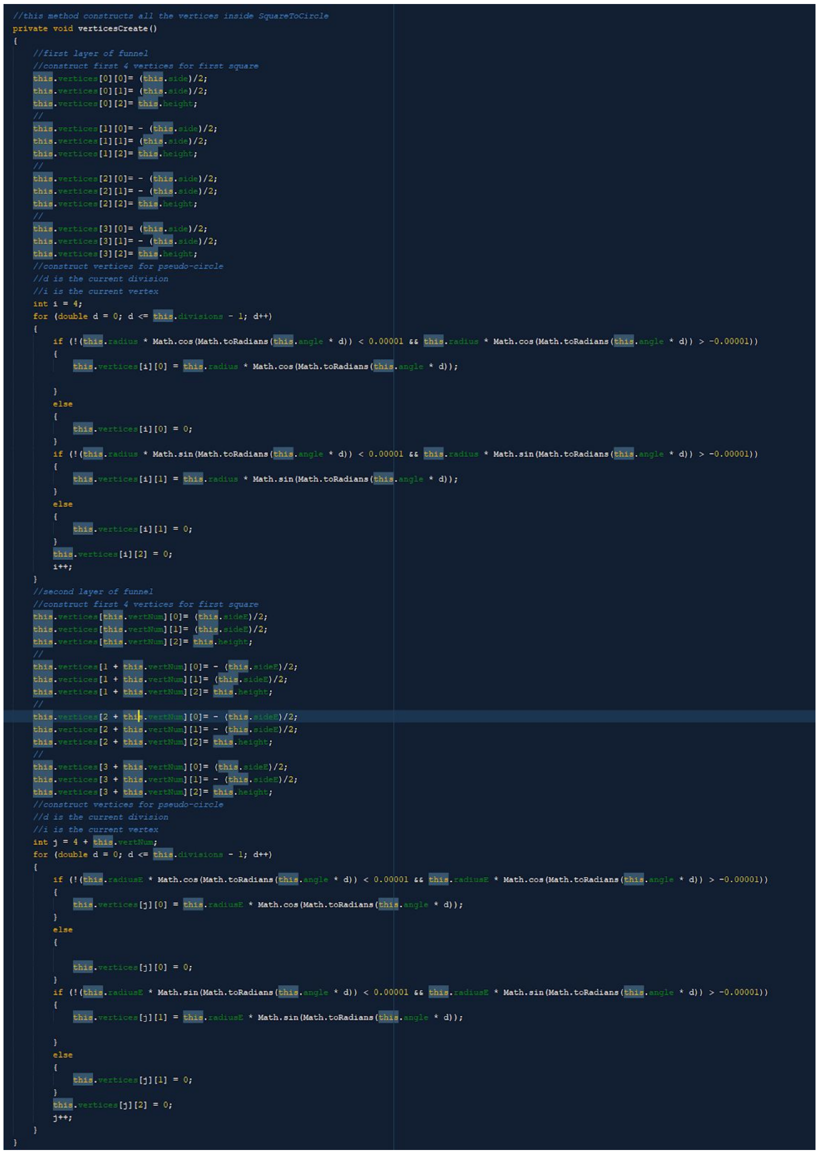
Constructor SquareToCircle



```
public SquareToCircle(String name, double height, double side, double radius, double sectors, double thickness)
{
    this.name = name;
    this.radius = radius;
    this.radiusE = this.radius + thickness;
    this.thickness = thickness;
    this.height = height;
    this.side = side;
    this.sideE = this.side + 2*thickness;
    this.sectors = sectors;
    this.divisions = 4 * this.sectors;
    this.angle = 90/this.sectors;
    this.vertNum = (int)(4 + (this.divisions));
    this.faceNum = vertNum;
    this.vertices = new double[this.vertNum * 2][3];
    this.faces = new double[this.faceNum * 3][4][3];
    this.script = "";
    verticesCreate();
    facesCreate();
    scriptWriter();
    System.out.println(this.script);
    System.out.println(Arrays.deepToString(this.vertices));
    System.out.println(Arrays.deepToString(this.faces));
}
```

Constructor creates a SquareToCircle funnel object. For the 2D array vertices, the first dimension is the number of vertices, and the second dimension is the (x,y,z) of each vertex. For the 3D array faces, the first dimension is the number of faces, the second dimension is the number of vertices per face, and the third dimension is the (x,y,z) of each vertex.

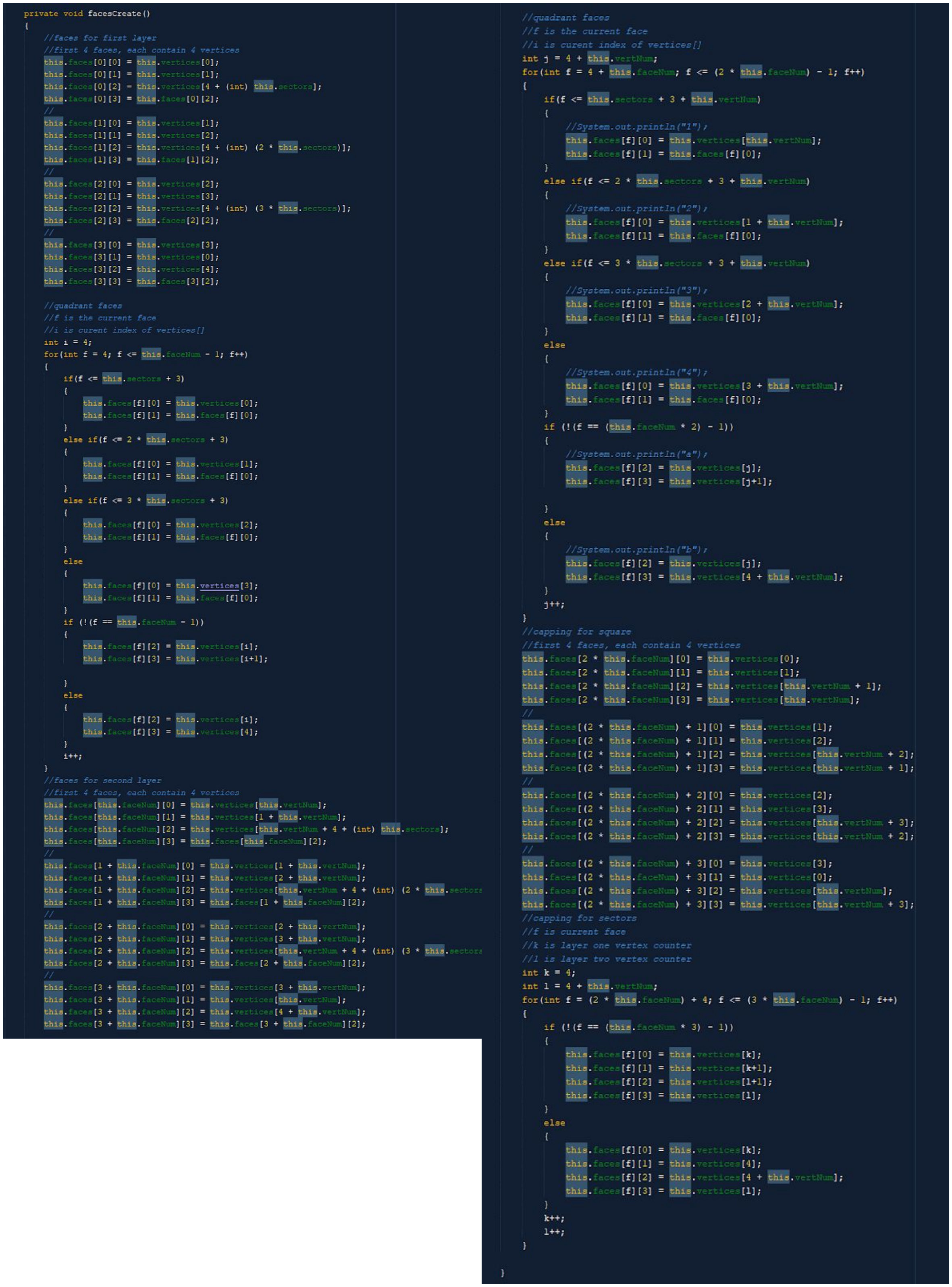
Method verticesCreate.



```java
//this method constructs all the vertices inside SquareToCircle
private void verticesCreate()
{
    //first layer of funnel
    //construct first 4 vertices for first square
    this.vertices[0][0]= (this.side)/2;
    this.vertices[0][1]= (this.side)/2;
    this.vertices[0][2]= this.height;
    //
    this.vertices[1][0]= - (this.side)/2;
    this.vertices[1][1]= (this.side)/2;
    this.vertices[1][2]= this.height;
    //
    this.vertices[2][0]= - (this.side)/2;
    this.vertices[2][1]= - (this.side)/2;
    this.vertices[2][2]= this.height;
    //
    this.vertices[3][0]= (this.side)/2;
    this.vertices[3][1]= - (this.side)/2;
    this.vertices[3][2]= this.height;
    //construct vertices for pseudo-circle
    //d is the current division
    //i is the current vertex
    int i = 4;
    for (double d = 0; d <= this.divisions - 1; d++)
    {
        if (!(this.radius * Math.cos(Math.toRadians(this.angle * d)) < 0.00001 && this.radius * Math.cos(Math.toRadians(this.angle * d)) > -0.00001))
        {
            this.vertices[i][0] = this.radius * Math.cos(Math.toRadians(this.angle * d));

        }
        else
        {
            this.vertices[i][0] = 0;
        }
        if (!(this.radius * Math.sin(Math.toRadians(this.angle * d)) < 0.00001 && this.radius * Math.sin(Math.toRadians(this.angle * d)) > -0.00001))
        {
            this.vertices[i][1] = this.radius * Math.sin(Math.toRadians(this.angle * d));
        }
        else
        {
            this.vertices[i][1] = 0;
        }
        this.vertices[i][2] = 0;
        i++;
    }
    //second layer of funnel
    //construct first 4 vertices for first square
    this.vertices[this.vertNum][0]= (this.sideE)/2;
    this.vertices[this.vertNum][1]= (this.sideE)/2;
    this.vertices[this.vertNum][2]= this.height;
    //
    this.vertices[1 + this.vertNum][0]= - (this.sideE)/2;
    this.vertices[1 + this.vertNum][1]= (this.sideE)/2;
    this.vertices[1 + this.vertNum][2]= this.height;
    //
    this.vertices[2 + this.vertNum][0]= - (this.sideE)/2;
    this.vertices[2 + this.vertNum][1]= - (this.sideE)/2;
    this.vertices[2 + this.vertNum][2]= this.height;
    //
    this.vertices[3 + this.vertNum][0]= (this.sideE)/2;
    this.vertices[3 + this.vertNum][1]= - (this.sideE)/2;
    this.vertices[3 + this.vertNum][2]= this.height;
    //construct vertices for pseudo-circle
    //d is the current division
    //i is the current vertex
    int j = 4 + this.vertNum;
    for (double d = 0; d <= this.divisions - 1; d++)
    {
        if (!(this.radiusE * Math.cos(Math.toRadians(this.angle * d)) < 0.00001 && this.radiusE * Math.cos(Math.toRadians(this.angle * d)) > -0.00001))
        {
            this.vertices[j][0] = this.radiusE * Math.cos(Math.toRadians(this.angle * d));
        }
        else
        {

            this.vertices[j][0] = 0;
        }
        if (!(this.radiusE * Math.sin(Math.toRadians(this.angle * d)) < 0.00001 && this.radiusE * Math.sin(Math.toRadians(this.angle * d)) > -0.00001))
        {
            this.vertices[j][1] = this.radiusE * Math.sin(Math.toRadians(this.angle * d));

        }
        else
        {
            this.vertices[j][1] = 0;
        }
        this.vertices[j][2] = 0;
        j++;
    }
}
```

Method calculates the vertices of a funnel object. In this case, the code is for a SquareToCircle funnel (similar versions of this method in the other 2 classes). First, the vertices for the first square are assigned. Next, a loop is used to count through pseudo circle divisions and calculate the vertex. For the second square and pseudo circle, the process is the same except the array indexes are different.

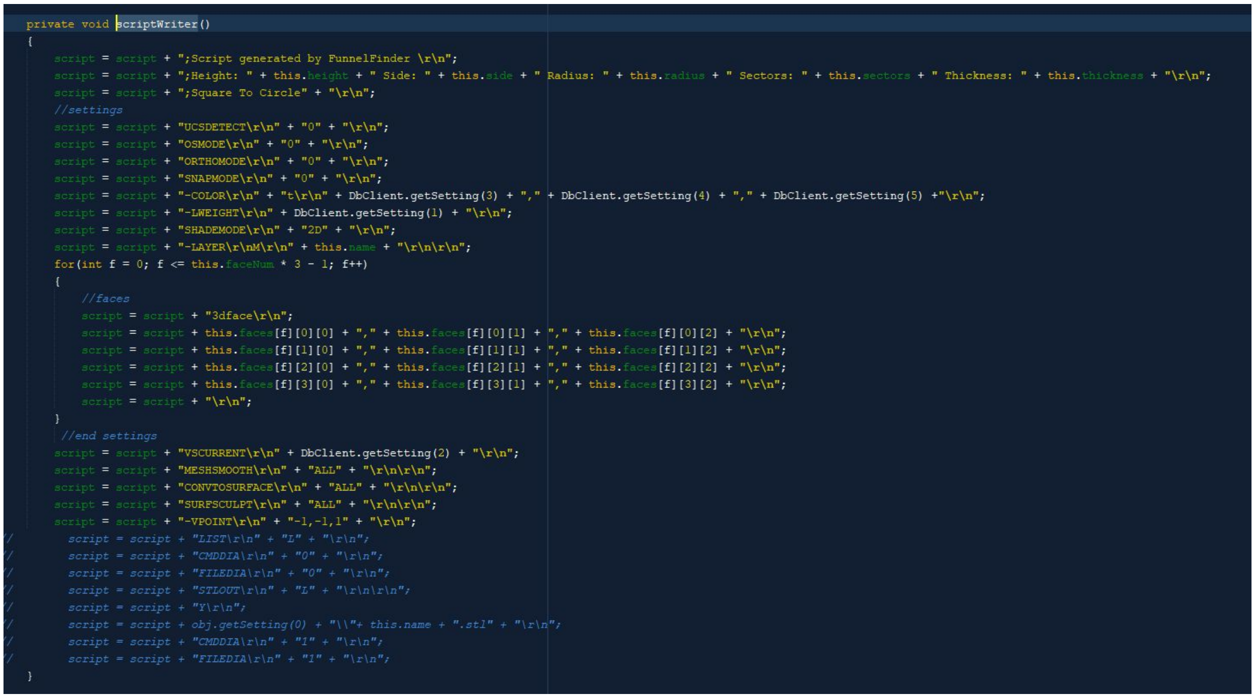
Method facesCreate().



```
private void facesCreate()
{
    //faces for first layer
    //first 4 faces, each contain 4 vertices
    this.faces[0][0] = this.vertices[0];
    this.faces[0][1] = this.vertices[1];
    this.faces[0][2] = this.vertices[4 + (int) this.sectors];
    this.faces[0][3] = this.faces[0][2];
    //
    this.faces[1][0] = this.vertices[1];
    this.faces[1][1] = this.vertices[2];
    this.faces[1][2] = this.vertices[4 + (int) (2 * this.sectors)];
    this.faces[1][3] = this.faces[1][2];
    //
    this.faces[2][0] = this.vertices[2];
    this.faces[2][1] = this.vertices[3];
    this.faces[2][2] = this.vertices[4 + (int) (3 * this.sectors)];
    this.faces[2][3] = this.faces[2][2];
    //
    this.faces[3][0] = this.vertices[3];
    this.faces[3][1] = this.vertices[0];
    this.faces[3][2] = this.vertices[4];
    this.faces[3][3] = this.faces[3][2];

    //quadrant faces
    //f is the current face
    //i is curent index of vertices[]
    int i = 4;
    for(int f = 4; f <= this.faceNum - 1; f++)
    {
        if(f <= this.sectors + 3)
        {
            this.faces[f][0] = this.vertices[0];
            this.faces[f][1] = this.faces[f][0];
        }
        else if(f <= 2 * this.sectors + 3)
        {
            this.faces[f][0] = this.vertices[1];
            this.faces[f][1] = this.faces[f][0];
        }
        else if(f <= 3 * this.sectors + 3)
        {
            this.faces[f][0] = this.vertices[2];
            this.faces[f][1] = this.faces[f][0];
        }
        else
        {
            this.faces[f][0] = this.vertices[3];
            this.faces[f][1] = this.faces[f][0];
        }
        if (!(f == this.faceNum - 1))
        {
            this.faces[f][2] = this.vertices[i];
            this.faces[f][3] = this.vertices[i+1];

        }
        else
        {
            this.faces[f][2] = this.vertices[i];
            this.faces[f][3] = this.vertices[4];
        }
        i++;
    }
    //faces for second layer
    //first 4 faces, each contain 4 vertices
    this.faces[this.faceNum][0] = this.vertices[this.vertNum];
    this.faces[this.faceNum][1] = this.vertices[1 + this.vertNum];
    this.faces[this.faceNum][2] = this.vertices[this.vertNum + 4 + (int) this.sectors];
    this.faces[this.faceNum][3] = this.faces[this.faceNum][2];
    //
    this.faces[1 + this.faceNum][0] = this.vertices[1 + this.vertNum];
    this.faces[1 + this.faceNum][1] = this.vertices[2 + this.vertNum];
    this.faces[1 + this.faceNum][2] = this.vertices[this.vertNum + 4 + (int) (2 * this.sectors
    this.faces[1 + this.faceNum][3] = this.faces[1 + this.faceNum][2];
    //
    this.faces[2 + this.faceNum][0] = this.vertices[2 + this.vertNum];
    this.faces[2 + this.faceNum][1] = this.vertices[3 + this.vertNum];
    this.faces[2 + this.faceNum][2] = this.vertices[this.vertNum + 4 + (int) (3 * this.sectors
    this.faces[2 + this.faceNum][3] = this.faces[2 + this.faceNum][2];
    //
    this.faces[3 + this.faceNum][0] = this.vertices[3 + this.vertNum];
    this.faces[3 + this.faceNum][1] = this.vertices[this.vertNum];
    this.faces[3 + this.faceNum][2] = this.vertices[4 + this.vertNum];
    this.faces[3 + this.faceNum][3] = this.faces[3 + this.faceNum][2];
    //quadrant faces
    //f is the current face
    //i is curent index of vertices[]
    int j = 4 + this.vertNum;
    for(int f = 4 + this.faceNum; f <= (2 * this.faceNum) - 1; f++)
    {
        if(f <= this.sectors + 3 + this.vertNum)
        {
            //System.out.println("1");
            this.faces[f][0] = this.vertices[this.vertNum];
            this.faces[f][1] = this.faces[f][0];
        }
        else if(f <= 2 * this.sectors + 3 + this.vertNum)
        {
            //System.out.println("2");
            this.faces[f][0] = this.vertices[1 + this.vertNum];
            this.faces[f][1] = this.faces[f][0];
        }
        else if(f <= 3 * this.sectors + 3 + this.vertNum)
        {
            //System.out.println("3");
            this.faces[f][0] = this.vertices[2 + this.vertNum];
            this.faces[f][1] = this.faces[f][0];
        }
        else
        {
            //System.out.println("4");
            this.faces[f][0] = this.vertices[3 + this.vertNum];
            this.faces[f][1] = this.faces[f][0];
        }
        if (!(f == (this.faceNum * 2) - 1))
        {
            //System.out.println("a");
            this.faces[f][2] = this.vertices[j];
            this.faces[f][3] = this.vertices[j+1];

        }
        else
        {
            //System.out.println("b");
            this.faces[f][2] = this.vertices[j];
            this.faces[f][3] = this.vertices[4 + this.vertNum];
        }
        j++;
    }
    //capping for square
    //first 4 faces, each contain 4 vertices
    this.faces[2 * this.faceNum][0] = this.vertices[0];
    this.faces[2 * this.faceNum][1] = this.vertices[1];
    this.faces[2 * this.faceNum][2] = this.vertices[this.vertNum + 1];
    this.faces[2 * this.faceNum][3] = this.vertices[this.vertNum];
    //
    this.faces[(2 * this.faceNum) + 1][0] = this.vertices[1];
    this.faces[(2 * this.faceNum) + 1][1] = this.vertices[2];
    this.faces[(2 * this.faceNum) + 1][2] = this.vertices[this.vertNum + 2];
    this.faces[(2 * this.faceNum) + 1][3] = this.vertices[this.vertNum + 1];
    //
    this.faces[(2 * this.faceNum) + 2][0] = this.vertices[2];
    this.faces[(2 * this.faceNum) + 2][1] = this.vertices[3];
    this.faces[(2 * this.faceNum) + 2][2] = this.vertices[this.vertNum + 3];
    this.faces[(2 * this.faceNum) + 2][3] = this.vertices[this.vertNum + 2];
    //
    this.faces[(2 * this.faceNum) + 3][0] = this.vertices[3];
    this.faces[(2 * this.faceNum) + 3][1] = this.vertices[0];
    this.faces[(2 * this.faceNum) + 3][2] = this.vertices[this.vertNum];
    this.faces[(2 * this.faceNum) + 3][3] = this.vertices[this.vertNum + 3];
    //capping for sectors
    //f is current face
    //k is layer one vertex counter
    //l is layer two vertex counter
    int k = 4;
    int l = 4 + this.vertNum;
    for(int f = (2 * this.faceNum) + 4; f <= (3 * this.faceNum) - 1; f++)
    {
        if (!(f == (this.faceNum * 3) - 1))
        {
            this.faces[f][0] = this.vertices[k];
            this.faces[f][1] = this.vertices[k+1];
            this.faces[f][2] = this.vertices[l+1];
            this.faces[f][3] = this.vertices[l];
        }
        else
        {
            this.faces[f][0] = this.vertices[k];
            this.faces[f][1] = this.vertices[4];
            this.faces[f][2] = this.vertices[4 + this.vertNum];
            this.faces[f][3] = this.vertices[l];
        }
        k++;
        l++;
    }

}
```

Method creates faces by reading from the vertices array. In this case, the code is for a SquareToCircle funnel (similar versions of this method in the other 2 classes). First, the faces for the first 4 faces of the first layer are assigned. Next, a loop is used to count through all the other faces for the first layer and assign the vertices to a face. This process is repeated for the second layer. Finally, the two layers are connected using a capping algorithm. This will cap the square, and then the pseudo circle using a loop.

Method scriptWriter().

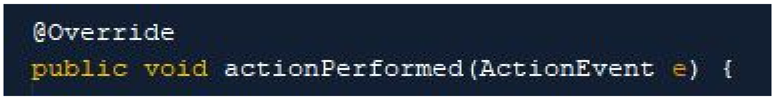


```
private void scriptWriter()
{
    script = script + ";Script generated by FunnelFinder \r\n";
    script = script + ";Height: " + this.height + " Side: " + this.side + " Radius: " + this.radius + " Sectors: " + this.sectors + " Thickness: " + this.thickness + "\r\n";
    script = script + ";Square To Circle" + "\r\n";
    //settings
    script = script + "UCSDETECT\r\n" + "0" + "\r\n";
    script = script + "OSMODE\r\n" + "0" + "\r\n";
    script = script + "ORTHOMODE\r\n" + "0" + "\r\n";
    script = script + "SNAPMODE\r\n" + "0" + "\r\n";
    script = script + "-COLOR\r\n" + "t\r\n" + DbClient.getSetting(3) + "," + DbClient.getSetting(4) + "," + DbClient.getSetting(5) +"\r\n";
    script = script + "-LWEIGHT\r\n" + DbClient.getSetting(1) + "\r\n";
    script = script + "SHADEMODE\r\n" + "2D" + "\r\n";
    script = script + "-LAYER\r\nM\r\n" + this.name + "\r\n\r\n";
    for(int f = 0; f <= this.faceNum * 3 - 1; f++)
    {
        //faces
        script = script + "3dface\r\n";
        script = script + this.faces[f][0][0] + "," + this.faces[f][0][1] + "," + this.faces[f][0][2] + "\r\n";
        script = script + this.faces[f][1][0] + "," + this.faces[f][1][1] + "," + this.faces[f][1][2] + "\r\n";
        script = script + this.faces[f][2][0] + "," + this.faces[f][2][1] + "," + this.faces[f][2][2] + "\r\n";
        script = script + this.faces[f][3][0] + "," + this.faces[f][3][1] + "," + this.faces[f][3][2] + "\r\n";
        script = script + "\r\n";
    }
     //end settings
    script = script + "VSCURRENT\r\n" + DbClient.getSetting(2) + "\r\n";
    script = script + "MESHSMOOTH\r\n" + "ALL" + "\r\n\r\n";
    script = script + "CONVTOSURFACE\r\n" + "ALL" + "\r\n\r\n";
    script = script + "SURFSCULPT\r\n" + "ALL" + "\r\n\r\n";
    script = script + "-VPOINT\r\n" + "-1,-1,1" + "\r\n";
//          script = script + "LIST\r\n" + "L" + "\r\n";
//          script = script + "CMDDIA\r\n" + "0" + "\r\n";
//          script = script + "FILEDIA\r\n" + "0" + "\r\n";
//          script = script + "STLOUT\r\n" + "L" + "\r\n\r\n";
//          script = script + "Y\r\n";
//          script = script + obj.getSetting(0) + "\\"+ this.name + ".stl" + "\r\n";
//          script = script + "CMDDIA\r\n" + "1" + "\r\n";
//          script = script + "FILEDIA\r\n" + "1" + "\r\n";
}
```

Method writes the script for the funnel object. Some settings are obtained from the database. A loop is used to read from the faces array and write them into the script.

Techniques Used

Polymorphism

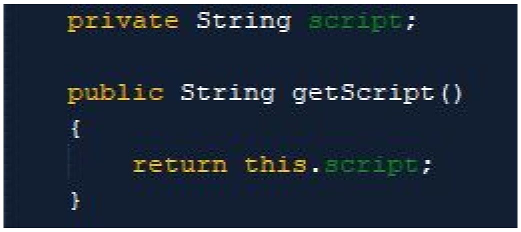


```
@Override
public void actionPerformed(ActionEvent e) {
```

Polymorphism is used in each GUI frame to interact with the user with the override of the actionPerformed method from the JFrame class.

Encapsulation

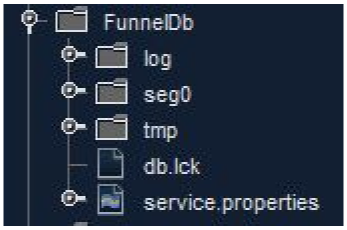
I used encapsulation in each of my funnel object classes. All attributes are private, which controls which ones are accessible. I used mutators to retrieve a generated script from a funnel object. I do not use accessors since I do not need them.



```
    private String script;

    public String getScript()
    {
        return this.script;
    }
```

Apache Derby Database

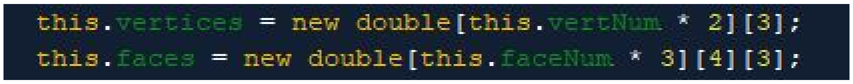
This database is contained locally, and can be easily installed. This is important since database should be installed on the computer it will be used on. It allows the user to store funnel inputs on their computer.



```
FunnelDb
    log
    seg0
    tmp
    db.lck
    service.properties
```

Arrays

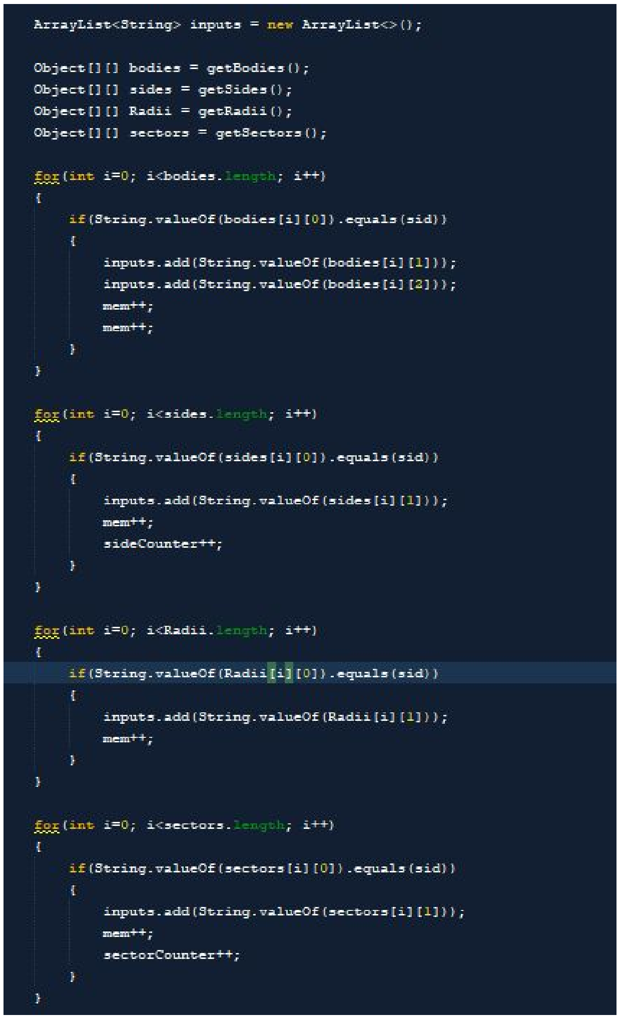
Two arrays are used in each of the three funnel object classes. The first array is used to store the vertices, and the second is used to store the faces. Arrays are used in this case because the number of vertices and faces for each funnel is calculated once (once for EACH object), and never changed.



```java
this.vertices = new double[this.vertNum * 2][3];
this.faces = new double[this.faceNum * 3][4][3];
```

ArrayList

An arraylist is used when searching for funnel inputs in the database since the number of inputs found will vary depending on the funnel type (and this cannot be determined until AFTER the inputs have been accessed).



```java
ArrayList<String> inputs = new ArrayList<>();

Object[][] bodies = getBodies();
Object[][] sides = getSides();
Object[][] Radii = getRadii();
Object[][] sectors = getSectors();

for(int i=0; i<bodies.length; i++)
{
    if(String.valueOf(bodies[i][0]).equals(sid))
    {
        inputs.add(String.valueOf(bodies[i][1]));
        inputs.add(String.valueOf(bodies[i][2]));
        mem++;
        mem++;
    }
}

for(int i=0; i<sides.length; i++)
{
    if(String.valueOf(sides[i][0]).equals(sid))
    {
        inputs.add(String.valueOf(sides[i][1]));
        mem++;
        sideCounter++;
    }
}

for(int i=0; i<Radii.length; i++)
{
    if(String.valueOf(Radii[i][0]).equals(sid))
    {
        inputs.add(String.valueOf(Radii[i][1]));
        mem++;
    }
}

for(int i=0; i<sectors.length; i++)
{
    if(String.valueOf(sectors[i][0]).equals(sid))
    {
        inputs.add(String.valueOf(sectors[i][1]));
        mem++;
        sectorCounter++;
    }
}
```

Tools Used

Java Api

- Swing
    - JFrame
    - JLabel
    - JButton
    - JPanel
    - JTable
    - JRadioButton
    - JScrollPane
    - JOptionPane
- Awt
    - BorderLayout
    - FlowLayout
    - Image
    - Color
    - Font
- Io
    - File
    - Filewriter
    - IOException
- Util
    - Date
    - Arrays

    -

- Sql
    - Timestamp
    - PreparedStatement
    - Connection

Wordcount: 1060