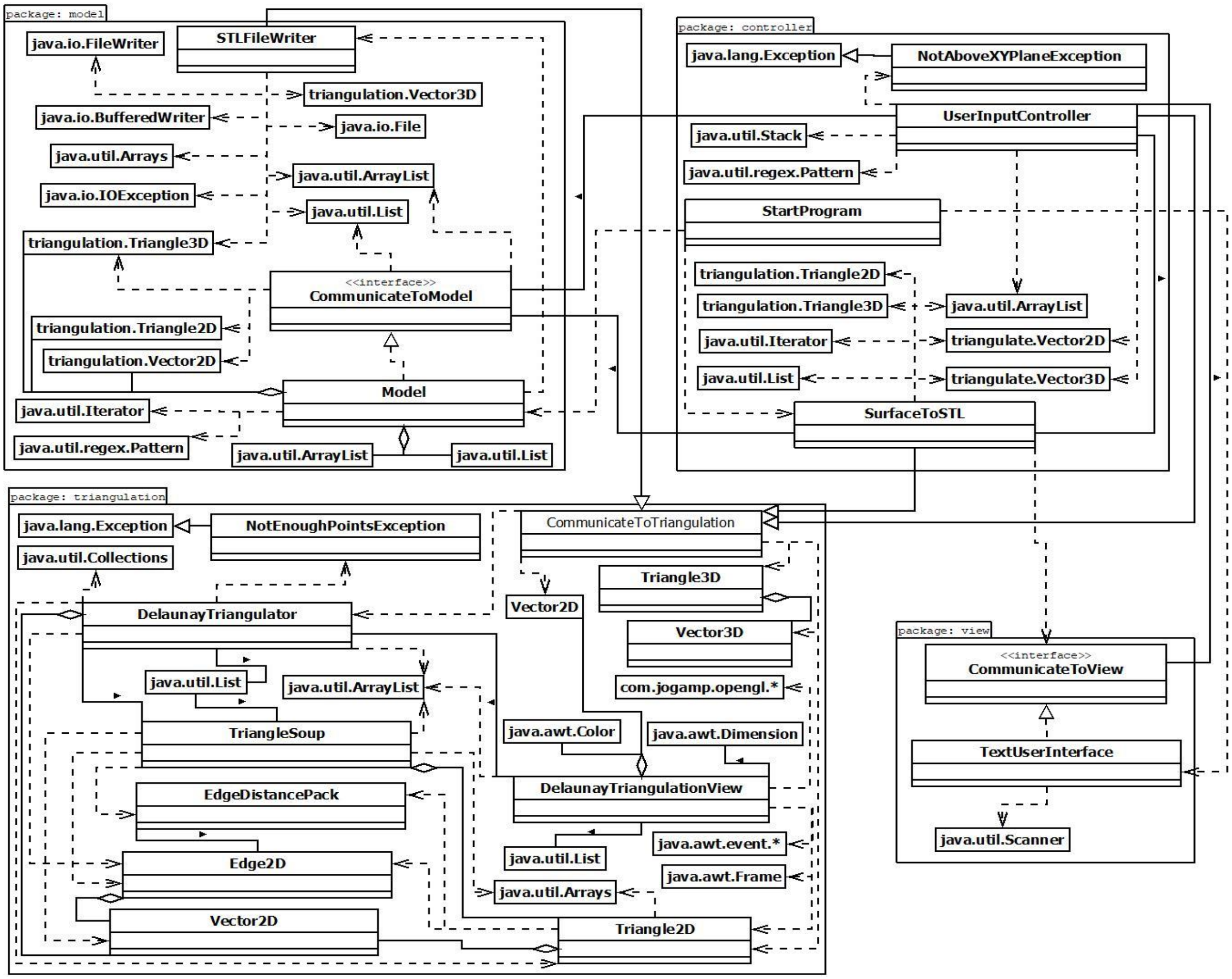
Architecture

The high-level design of the application is shown in the diagram(s) below.

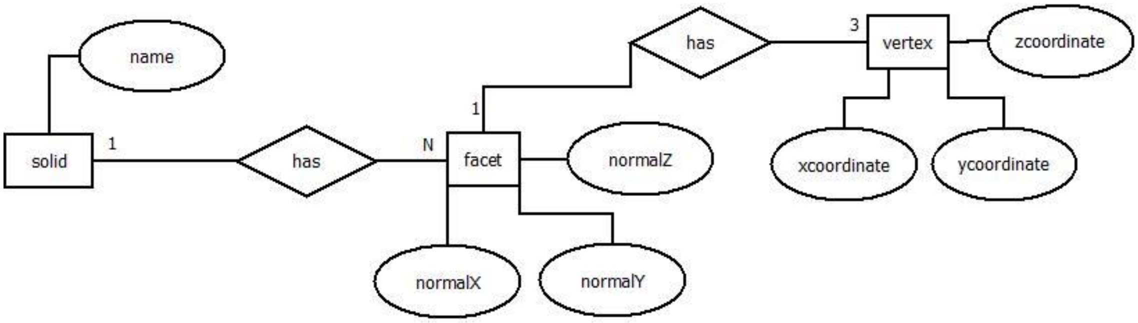


Data

This section contains a description of the logical and physical view of the data used by the application.

Logical Data Model

The logical data model is shown below.



The following data rules are derived from the logical data model.

- A solid has a name and one or more facets.
- A facet has 3 vertices and a normalX, normalY, and normalZ.
- A vertex has one xcoordinate, one ycoordinate, and one zcoordinate.

Physical Data Model

The physical data model, shown below, shows how the logical data elements will be represented in a persistent data storage facility (STL file for Requirement 3).

See http://www.fabbers.com/tech/STL_Format for the STL file format specifications and rules. The ASCII format for the STL file and the list of rules to be followed when creating the STL file are outlined below.

```
solid name
   facet normal normalX normalY normalZ
      outer loop
         vertex xcoordinate ycoordinate zcoordinate
         vertex xcoordinate ycoordinate zcoordinate
         vertex xcoordinate ycoordinate zcoordinate
      endloop
   endfacet
   facet normal  normalX normalY normalZ
      outer loop
         vertex xcoordinate ycoordinate zcoordinate
         vertex xcoordinate ycoordinate zcoordinate
         vertex xcoordinate ycoordinate zcoordinate
      endloop
   endfacet
   ...
endsolid name
```

STL File Rules:

1. The 3D object shall be in the all-positive octant so that no vertices are non-positive.
2. Each triangle T shall have three adjacent triangles such that each adjacent triangle shares exactly two vertices with T.
3. The facets/triangles approximating the 3D object shall be listed in counterclockwise order as seen from the outside of the 3D object.
4. The normal vectors shall be of unit length (length of 1) and point outward from the surface.
5. Triangles should be listed in ascending z-value order to optimize performance.

Internal Data Structures

During execution of the application, certain data elements shall be stored in non-persistent storage using the following types of data structures (Requirement 1).

- ArrayList<String> for the equation.
- char for the choice of bound (‘c’ for circle, ‘t’ for triangle, ‘r’ for rectangle).
- ArrayList<Vector2D> for the bound (circle, triangle, or rectangle) where each point is a Vector2D.
- List<Triangle2D> for the triangles on the bound.
- List<Triangle3D> for the triangles approximating the entire 3D object.
- List<Vector3D> for the normal vectors for each triangle approximating the 3D object.
- int for the resolution of the 3D object.
- String for the name of the solid.

Data Dictionary

The table below describes the characteristics of each attribute shown in the logical data model.

Attribute	**Type**	**Size (bytes)**	**Format/Range**	**Structure**
name	String	2*length	max 80 bytes, must be nonempty	String
normalX	single precision float	4	of the form 1.23456E+789	double
normalY	single precision float	4	of the form 1.23456E+789	double
normalZ	single precision float	4	of the form 1.23456E+789	double
xcoordinate	single precision float	4	non-negative of the form 1.23456E+789	double
ycoordinate	single precision float	4	non-negative of the form 1.23456E+789	double
zcoordinate	single precision float	4	non-negative of the form 1.23456E+789	double

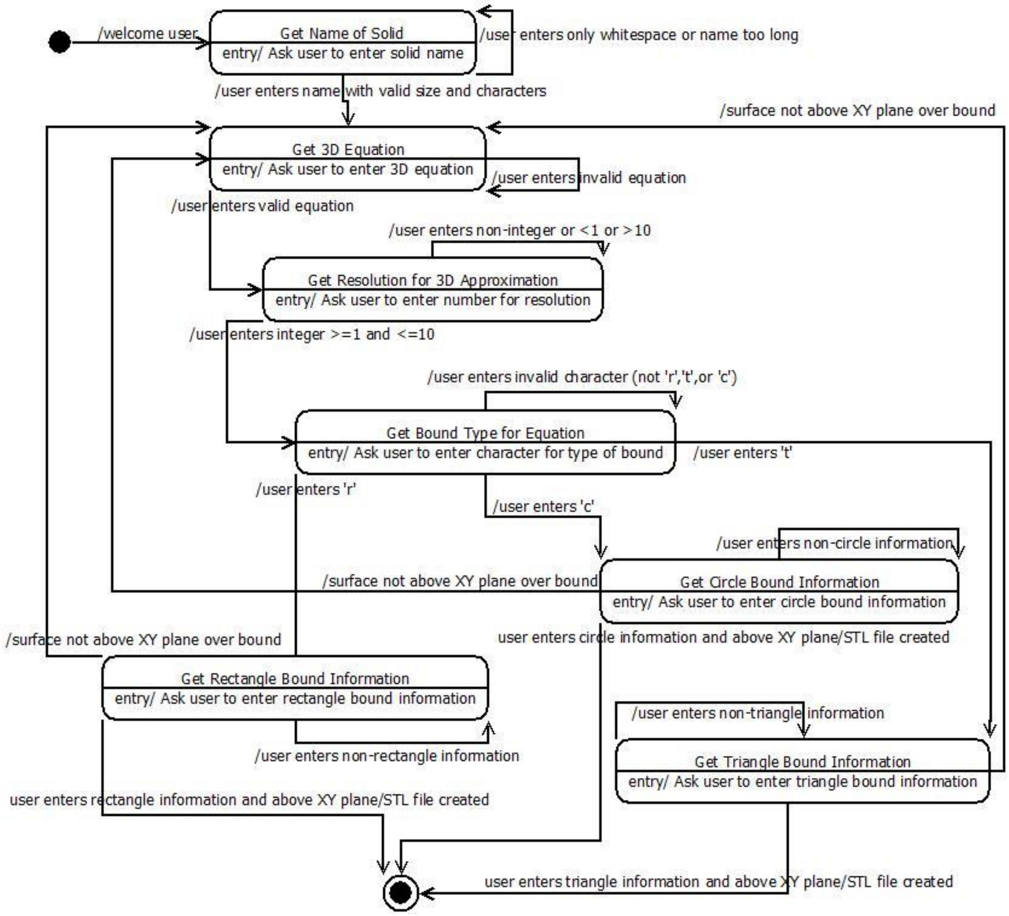
Interfaces

This section contains a description of interfaces that must be supported by the application.

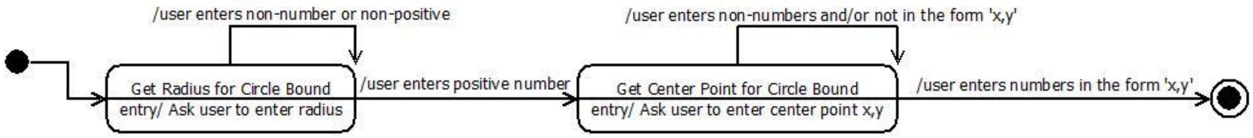
Human-computer Interaction

The user interactions with the application are shown below. The high-level state machine diagram is followed by the more specific state machine diagrams for the circle, rectangle, and triangle bound cases.

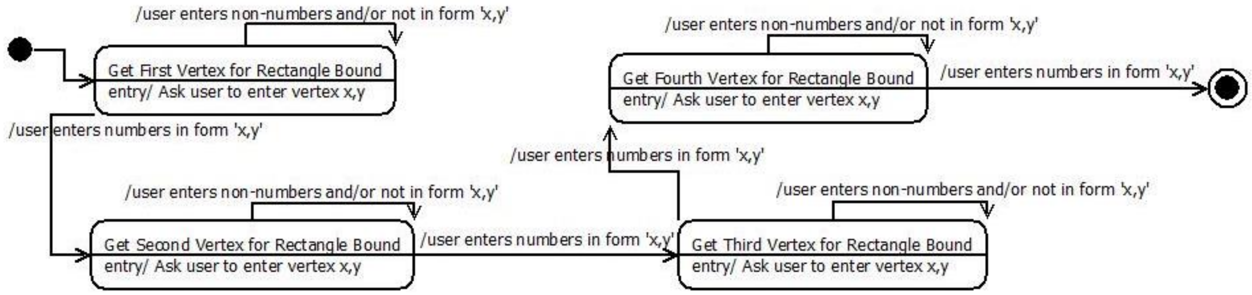
High-Level:


/welcome user
Get Name of Solid
entry/ Ask user to enter solid name
/user enters only whitespace or name too long
/user enters name with valid size and characters
/surface not above XY plane over bound
Get 3D Equation
entry/ Ask user to enter 3D equation
/user enters invalid equation
/user enters valid equation
/user enters non-integer or <1 or >10
Get Resolution for 3D Approximation
entry/ Ask user to enter number for resolution
/user enters integer >=1 and <=10
/user enters invalid character (not 'r','t',or 'c')
Get Bound Type for Equation
entry/ Ask user to enter character for type of bound
/user enters 't'
/user enters 'r'
/user enters 'c'
/user enters non-circle information
/surface not above XY plane over bound
Get Circle Bound Information
entry/ Ask user to enter circle bound information
/surface not above XY plane over bound
user enters circle information and above XY plane/STL file created
Get Rectangle Bound Information
entry/ Ask user to enter rectangle bound information
/user enters non-triangle information
Get Triangle Bound Information
entry/ Ask user to enter triangle bound information
/user enters non-rectangle information
user enters rectangle information and above XY plane/STL file created
user enters triangle information and above XY plane/STL file created

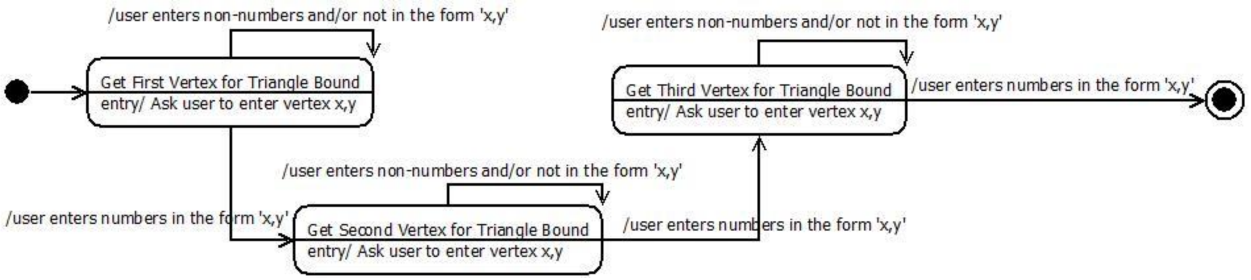
Circle:


/user enters non-number or non-positive
/user enters non-numbers and/or not in the form 'x,y'
Get Radius for Circle Bound
entry/ Ask user to enter radius
/user enters positive number
Get Center Point for Circle Bound
entry/ Ask user to enter center point x,y
/user enters numbers in the form 'x,y'

Rectangle:

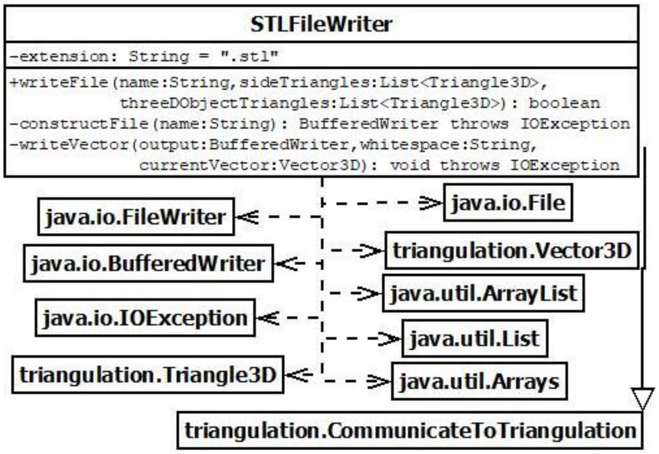

/user enters non-numbers and/or not in form 'x,y'
/user enters non-numbers and/or not in form 'x,y'
Get First Vertex for Rectangle Bound
entry/ Ask user to enter vertex x,y
Get Fourth Vertex for Rectangle Bound
entry/ Ask user to enter vertex x,y
/user enters numbers in form 'x,y'
/user enters numbers in form 'x,y'
/user enters numbers in form 'x,y'
/user enters non-numbers and/or not in form 'x,y'
/user enters non-numbers and/or not in form 'x,y'
Get Second Vertex for Rectangle Bound
entry/ Ask user to enter vertex x,y
/user enters numbers in form 'x,y'
Get Third Vertex for Rectangle Bound
entry/ Ask user to enter vertex x,y

Triangle:

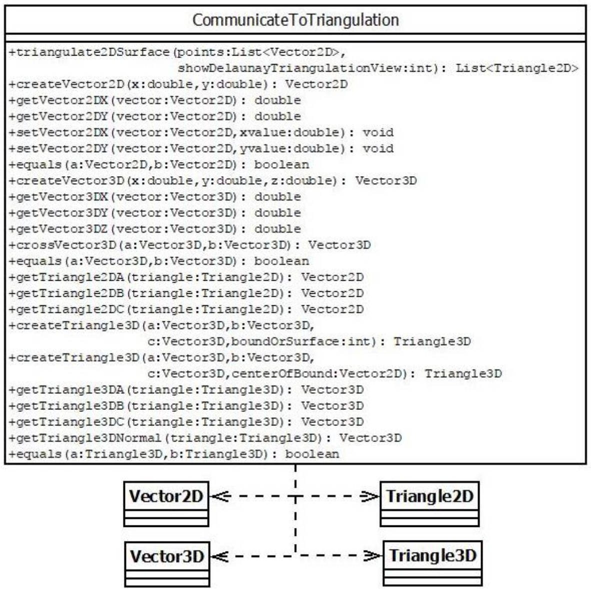

/user enters non-numbers and/or not in the form 'x,y'
Get First Vertex for Triangle Bound
entry/ Ask user to enter vertex x,y
/user enters numbers in the form 'x,y'
/user enters non-numbers and/or not in the form 'x,y'
Get Second Vertex for Triangle Bound
entry/ Ask user to enter vertex x,y
/user enters numbers in the form 'x,y'
/user enters non-numbers and/or not in the form 'x,y'
Get Third Vertex for Triangle Bound
entry/ Ask user to enter vertex x,y
/user enters numbers in the form 'x,y'

Communication with External Entities

The way in which the application will communicate with the 3D Printer via STL file is shown below via Class Diagram (Requirement 3).


STLFileWriter
-extension: String = ".stl"
+writeFile(name:String,sideTriangles:List<Triangle3D>, threeDObjectTriangles:List<Triangle3D>): boolean
-constructFile(name:String): BufferedWriter throws IOException
-writeVector(output:BufferedWriter,whitespace:String, currentVector:Vector3D): void throws IOException
java.io.FileWriter
java.io.File
java.io.BufferedWriter
triangulation.Vector3D
java.util.ArrayList
java.io.IOException
java.util.List
triangulation.Triangle3D
java.util.Arrays
triangulation.CommunicateToTriangulation

The way in which the application will communicate with the third party triangulation package code is shown below via Class Diagram (Requirement 2).


CommunicateToTriangulation
+triangulate2DSurface(points:List<Vector2D>, showDelaunayTriangulationView:int): List<Triangle2D>
+createVector2D(x:double,y:double): Vector2D
+getVector2DX(vector:Vector2D): double
+getVector2DY(vector:Vector2D): double
+setVector2DX(vector:Vector2D,xvalue:double): void
+setVector2DY(vector:Vector2D,yvalue:double): void
+equals(a:Vector2D,b:Vector2D): boolean
+createVector3D(x:double,y:double,z:double): Vector3D
+getVector3DX(vector:Vector3D): double
+getVector3DY(vector:Vector3D): double
+getVector3DZ(vector:Vector3D): double
+crossVector3D(a:Vector3D,b:Vector3D): Vector3D
+equals(a:Vector3D,b:Vector3D): boolean
+getTriangle2DA(triangle:Triangle2D): Vector2D
+getTriangle2DB(triangle:Triangle2D): Vector2D
+getTriangle2DC(triangle:Triangle2D): Vector2D
+createTriangle3D(a:Vector3D,b:Vector3D, c:Vector3D,boundOrSurface:int): Triangle3D
+createTriangle3D(a:Vector3D,b:Vector3D, c:Vector3D,centerOfBound:Vector2D): Triangle3D
+getTriangle3DA(triangle:Triangle3D): Vector3D
+getTriangle3DB(triangle:Triangle3D): Vector3D
+getTriangle3DC(triangle:Triangle3D): Vector3D
+getTriangle3DNormal(triangle:Triangle3D): Vector3D
+equals(a:Triangle3D,b:Triangle3D): boolean
Vector2D
Triangle2D
Vector3D
Triangle3D

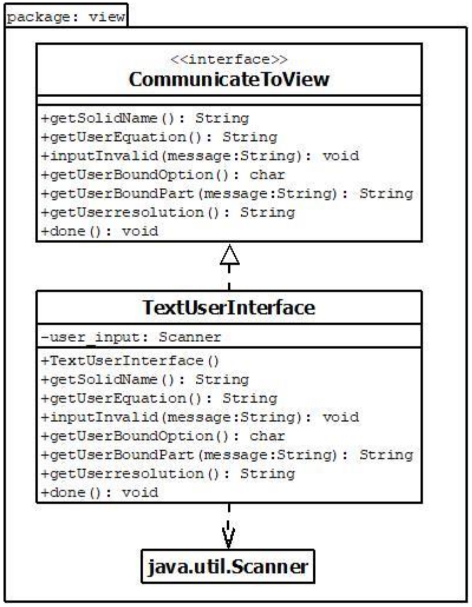
Components

This section contains a more detailed description of each component described in the Architecture section.

View

Class Diagram

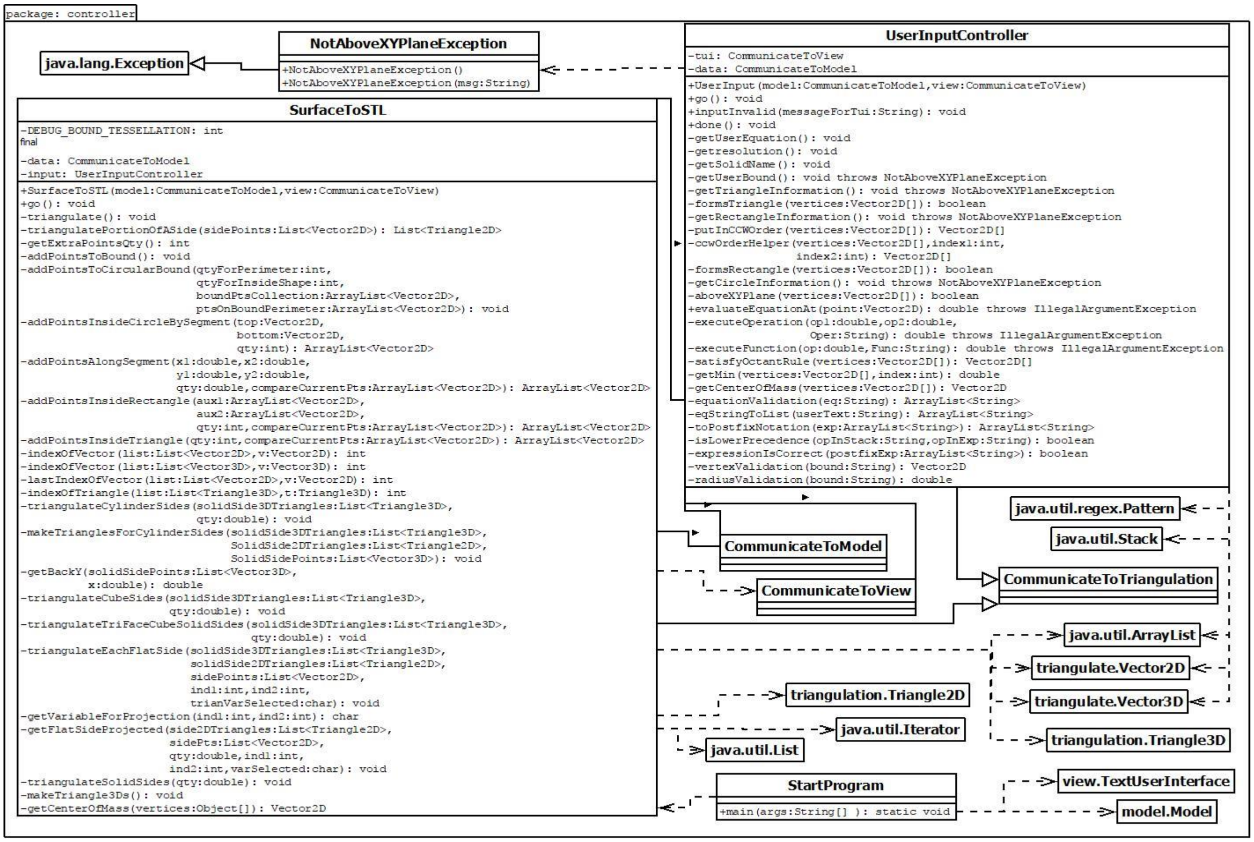
The class diagram for the View component is shown below.


package: view
<<interface>>
CommunicateToView
+getSolidName(): String
+getUserEquation(): String
+inputInvalid(message:String): void
+getUserBoundOption(): char
+getUserBoundPart(message:String): String
+getUserresolution(): String
+done(): void
TextUserInterface
-user_input: Scanner
+TextUserInterface()
+getSolidName(): String
+getUserEquation(): String
+inputInvalid(message:String): void
+getUserBoundOption(): char
+getUserBoundPart(message:String): String
+getUserresolution(): String
+done(): void
java.util.Scanner

Controller

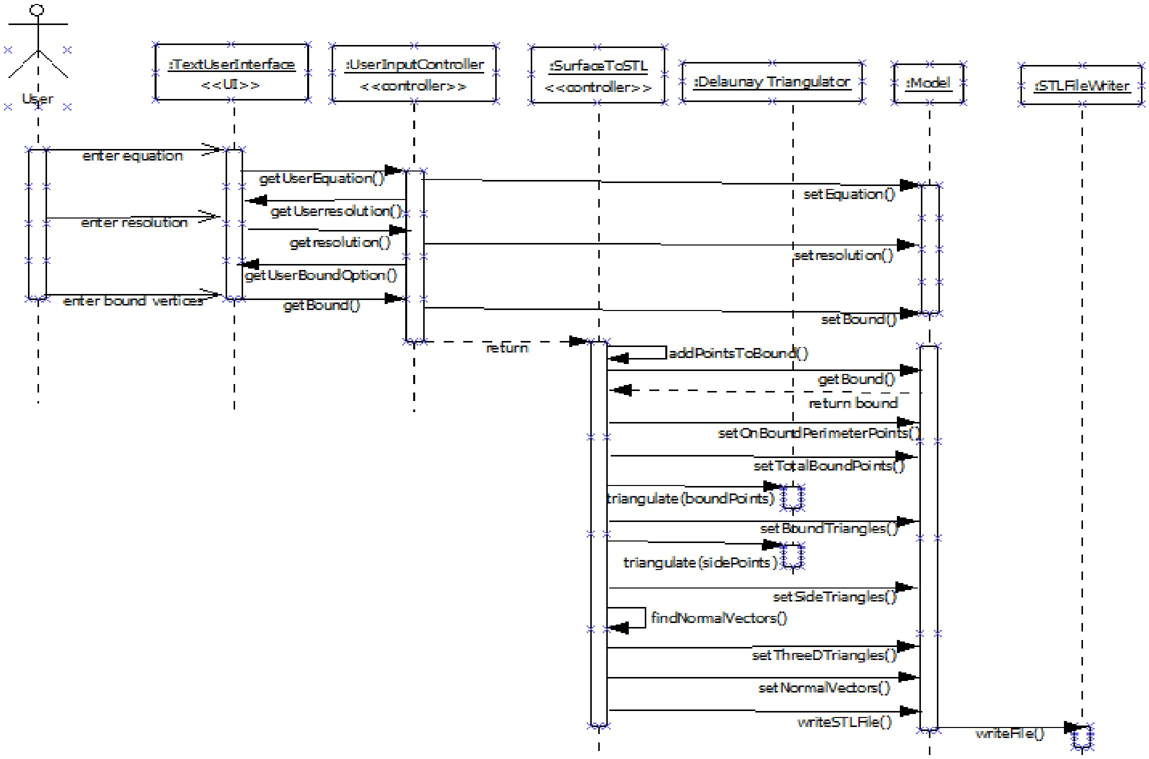
Class Diagram

The class diagram for the Controller component is shown below.


package: controller
java.lang.Exception
NotAboveXYPlaneException
+NotAboveXYPlaneException()
+NotAboveXYPlaneException(msg:String)
SurfaceToSTL
-DEBUG_BOUND_TESSELLATION: int
final
-data: CommunicateToModel
-input: UserInputController
+SurfaceToSTL(model:CommunicateToModel,view:CommunicateToView)
+go(): void
-triangulate(): void
-triangulatePortionOfASide(sidePoints:List<Vector2D>): List<Triangle2D>
-getExtraPointsQty(): int
-addPointsToBound(): void
-addPointsToCircularBound(qtyForPerimeter:int,
qtyForInsideShape:int,
boundPtsCollection:ArrayList<Vector2D>,
ptsOnBoundPerimeter:ArrayList<Vector2D>): void
-addPointsInsideCircleBySegment(top:Vector2D,
bottom:Vector2D,
qty:int): ArrayList<Vector2D>
-addPointsAlongSegment(x1:double,x2:double,
y1:double,y2:double,
qty:double,compareCurrentPts:ArrayList<Vector2D>): ArrayList<Vector2D>
-addPointsInsideRectangle(aux1:ArrayList<Vector2D>,
aux2:ArrayList<Vector2D>,
qty:int,compareCurrentPts:ArrayList<Vector2D>): ArrayList<Vector2D>
-addPointsInsideTriangle(qty:int,compareCurrentPts:ArrayList<Vector2D>): ArrayList<Vector2D>
-indexOfVector(list:List<Vector2D>,v:Vector2D): int
-indexOfVector(list:List<Vector3D>,v:Vector3D): int
-lastIndexOfVector(list:List<Vector2D>,v:Vector2D): int
-indexOfTriangle(list:List<Triangle3D>,t:Triangle3D): int
-triangulateCylinderSides(solidSide3DTriangles:List<Triangle3D>,
qty:double): void
-makeTrianglesForCylinderSides(solidSide3DTriangles:List<Triangle3D>,
SolidSide2DTriangles:List<Triangle2D>,
SolidSidePoints:List<Vector3D>): void
-getBackY(solidSidePoints:List<Vector3D>,
x:double): double
-triangulateCubeSides(solidSide3DTriangles:List<Triangle3D>,
qty:double): void
-triangulateTriFaceCubeSolidSides(solidSide3DTriangles:List<Triangle3D>,
qty:double): void
-triangulateEachFlatSide(solidSide3DTriangles:List<Triangle3D>,
solidSide2DTriangles:List<Triangle2D>,
sidePoints:List<Vector2D>,
ind1:int,ind2:int,
trianVarSelected:char): void
-getVariableForProjection(ind1:int,ind2:int): char
-getFlatSideProjected(side2DTriangles:List<Triangle2D>,
sidePts:List<Vector2D>,
qty:double,ind1:int,
ind2:int,varSelected:char): void
-triangulateSolidSides(qty:double): void
-makeTriangle3Ds(): void
-getCenterOfMass(vertices:Object[]): Vector2D
UserInputController
-tui: CommunicateToView
-data: CommunicateToModel
+UserInput(model:CommunicateToModel,view:CommunicateToView)
+go(): void
+inputInvalid(messageForTui:String): void
+done(): void
-getUserEquation(): void
-getresolution(): void
-getSolidName(): void
-getUserBound(): void throws NotAboveXYPlaneException
-getTriangleInformation(): void throws NotAboveXYPlaneException
-formsTriangle(vertices:Vector2D[]): boolean
-getRectangleInformation(): void throws NotAboveXYPlaneException
-putInCCWOrder(vertices:Vector2D[]): Vector2D[]
-ccwOrderHelper(vertices:Vector2D[],index1:int,
index2:int): Vector2D[]
-formsRectangle(vertices:Vector2D[]): boolean
-getCircleInformation(): void throws NotAboveXYPlaneException
-aboveXYPlane(vertices:Vector2D[]): boolean
+evaluateEquationAt(point:Vector2D): double throws IllegalArgumentException
-executeOperation(op1:double,op2:double,
Oper:String): double throws IllegalArgumentException
-executeFunction(op:double,Func:String): double throws IllegalArgumentException
-satisfyOctantRule(vertices:Vector2D[]): Vector2D[]
-getMin(vertices:Vector2D[],index:int): double
-getCenterOfMass(vertices:Vector2D[]): Vector2D
-equationValidation(eq:String): ArrayList<String>
-eqStringToList(userText:String): ArrayList<String>
-toPostfixNotation(exp:ArrayList<String>): ArrayList<String>
-isLowerPrecedence(opInStack:String,opInExp:String): boolean
-expressionIsCorrect(postfixExp:ArrayList<String>): boolean
-vertexValidation(bound:String): Vector2D
-radiusValidation(bound:String): double
java.util.regex.Pattern
java.util.Stack
CommunicateToModel
CommunicateToView
CommunicateToTriangulation
java.util.ArrayList
triangulate.Vector2D
triangulate.Vector3D
triangulation.Triangle2D
java.util.Iterator
triangulation.Triangle3D
java.util.List
StartProgram
+main(args:String[]): static void
view.TextUserInterface
model.Model

Behavior

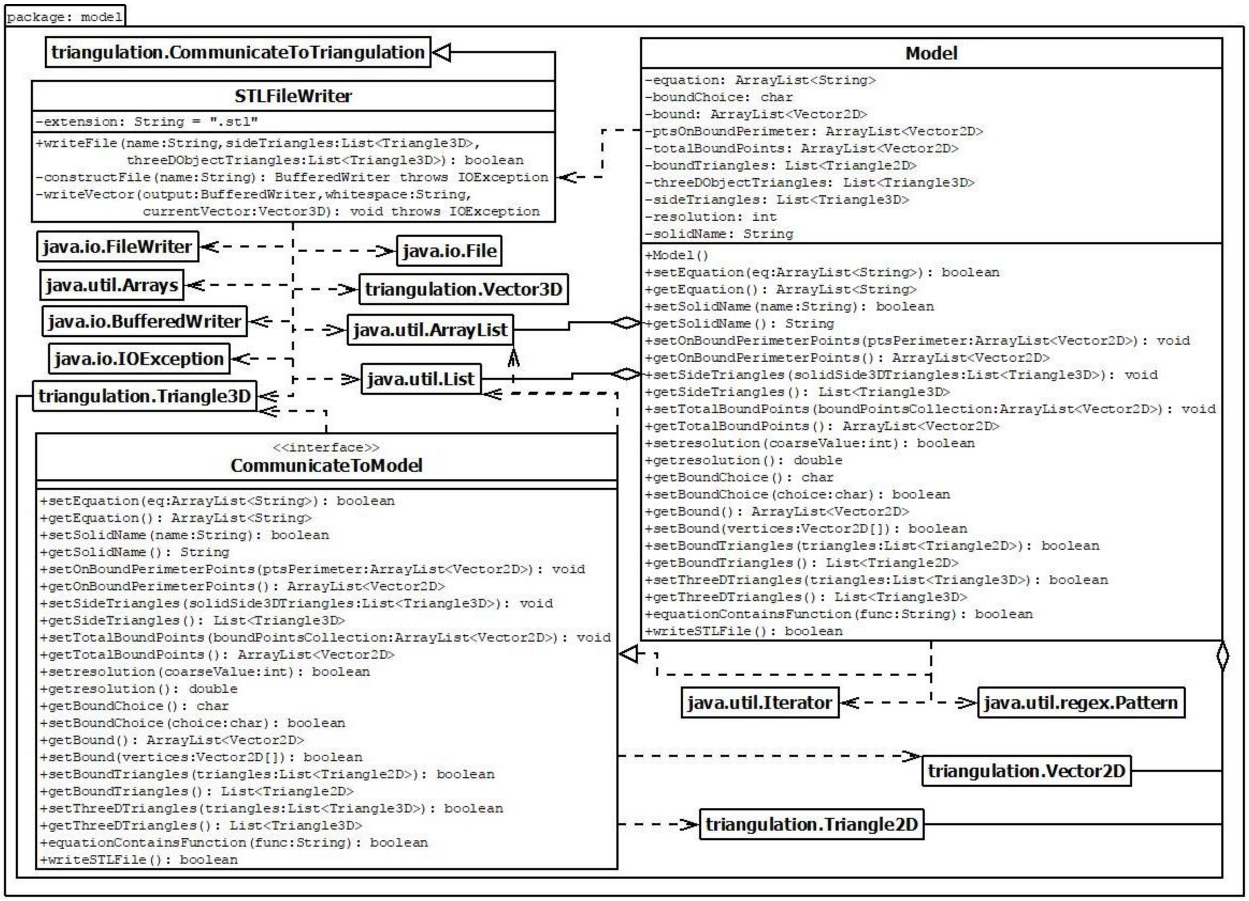
The behavior diagram for the Controller component and overall System interactions is shown below.



Model

Class Diagram

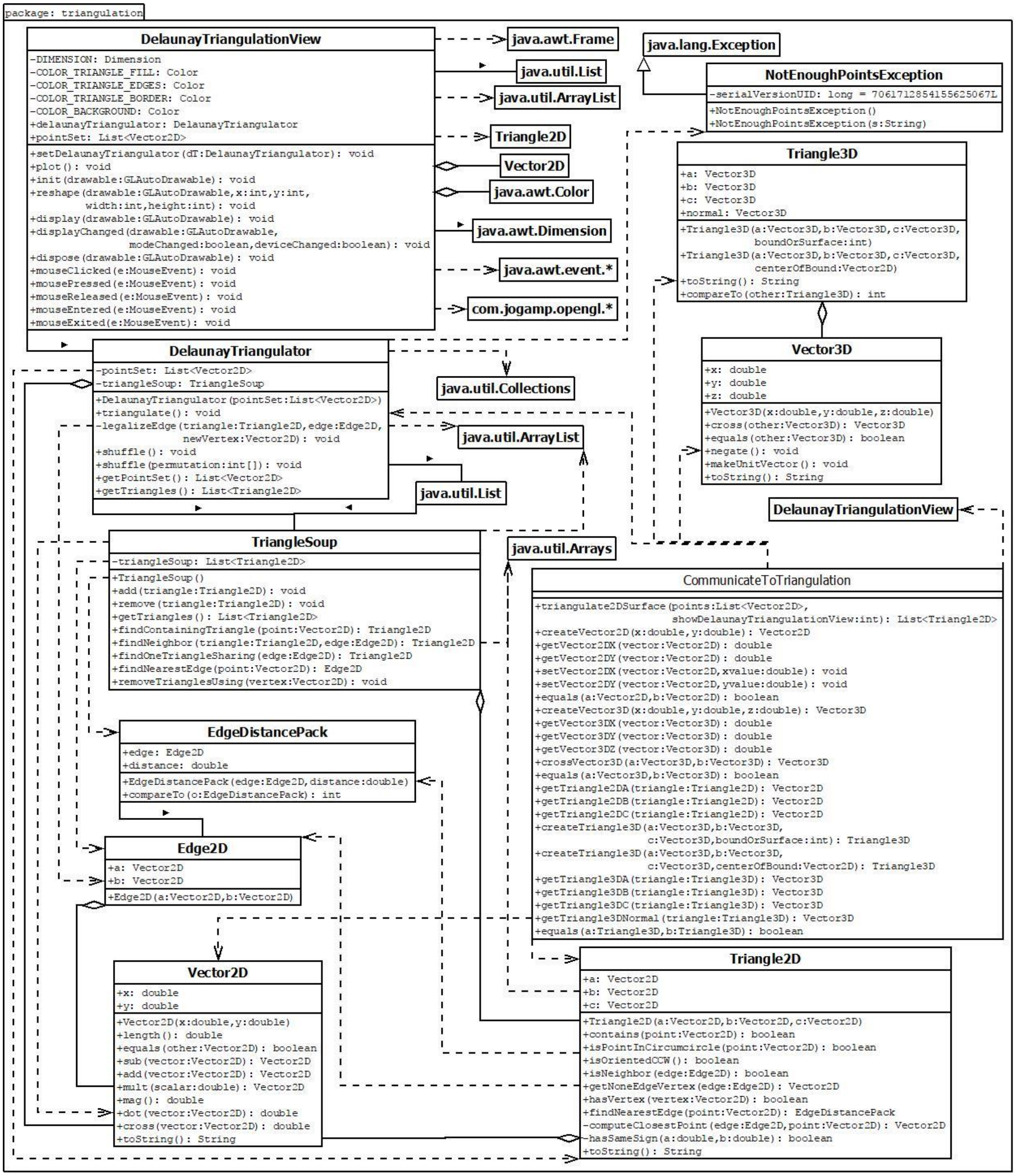
The class diagram for the Model component is shown below.


package: model
triangulation.CommunicateToTriangulation
STLFileWriter
-extension: String = ".stl"
+writeFile(name:String,sideTriangles:List<Triangle3D>, threeDObjectTriangles:List<Triangle3D>): boolean
-constructFile(name:String): BufferedWriter throws IOException
-writeVector(output:BufferedWriter,whitespace:String, currentVector:Vector3D): void throws IOException
java.io.FileWriter
java.io.File
java.util.Arrays
triangulation.Vector3D
java.io.BufferedWriter
java.util.ArrayList
java.io.IOException
java.util.List
triangulation.Triangle3D
<<interface>>
CommunicateToModel
+setEquation(eq:ArrayList<String>): boolean
+getEquation(): ArrayList<String>
+setSolidName(name:String): boolean
+getSolidName(): String
+setOnBoundPerimeterPoints(ptsPerimeter:ArrayList<Vector2D>): void
+getOnBoundPerimeterPoints(): ArrayList<Vector2D>
+setSideTriangles(solidSide3DTriangles:List<Triangle3D>): void
+getSideTriangles(): List<Triangle3D>
+setTotalBoundPoints(boundPointsCollection:ArrayList<Vector2D>): void
+getTotalBoundPoints(): ArrayList<Vector2D>
+setresolution(coarseValue:int): boolean
+getresolution(): double
+getBoundChoice(): char
+setBoundChoice(choice:char): boolean
+getBound(): ArrayList<Vector2D>
+setBound(vertices:Vector2D[]): boolean
+setBoundTriangles(triangles:List<Triangle2D>): boolean
+getBoundTriangles(): List<Triangle2D>
+setThreeDTriangles(triangles:List<Triangle3D>): boolean
+getThreeDTriangles(): List<Triangle3D>
+equationContainsFunction(func:String): boolean
+writeSTLFile(): boolean
Model
-equation: ArrayList<String>
-boundChoice: char
-bound: ArrayList<Vector2D>
-ptsOnBoundPerimeter: ArrayList<Vector2D>
-totalBoundPoints: ArrayList<Vector2D>
-boundTriangles: List<Triangle2D>
-threeDObjectTriangles: List<Triangle3D>
-sideTriangles: List<Triangle3D>
-resolution: int
-solidName: String
+Model()
+setEquation(eq:ArrayList<String>): boolean
+getEquation(): ArrayList<String>
+setSolidName(name:String): boolean
+getSolidName(): String
+setOnBoundPerimeterPoints(ptsPerimeter:ArrayList<Vector2D>): void
+getOnBoundPerimeterPoints(): ArrayList<Vector2D>
+setSideTriangles(solidSide3DTriangles:List<Triangle3D>): void
+getSideTriangles(): List<Triangle3D>
+setTotalBoundPoints(boundPointsCollection:ArrayList<Vector2D>): void
+getTotalBoundPoints(): ArrayList<Vector2D>
+setresolution(coarseValue:int): boolean
+getresolution(): double
+getBoundChoice(): char
+setBoundChoice(choice:char): boolean
+getBound(): ArrayList<Vector2D>
+setBound(vertices:Vector2D[]): boolean
+setBoundTriangles(triangles:List<Triangle2D>): boolean
+getBoundTriangles(): List<Triangle2D>
+setThreeDTriangles(triangles:List<Triangle3D>): boolean
+getThreeDTriangles(): List<Triangle3D>
+equationContainsFunction(func:String): boolean
+writeSTLFile(): boolean
java.util.Iterator
java.util.regex.Pattern
triangulation.Vector2D
triangulation.Triangle2D

Triangulation

Class Diagram

The class diagram for the Triangulation component is shown below.

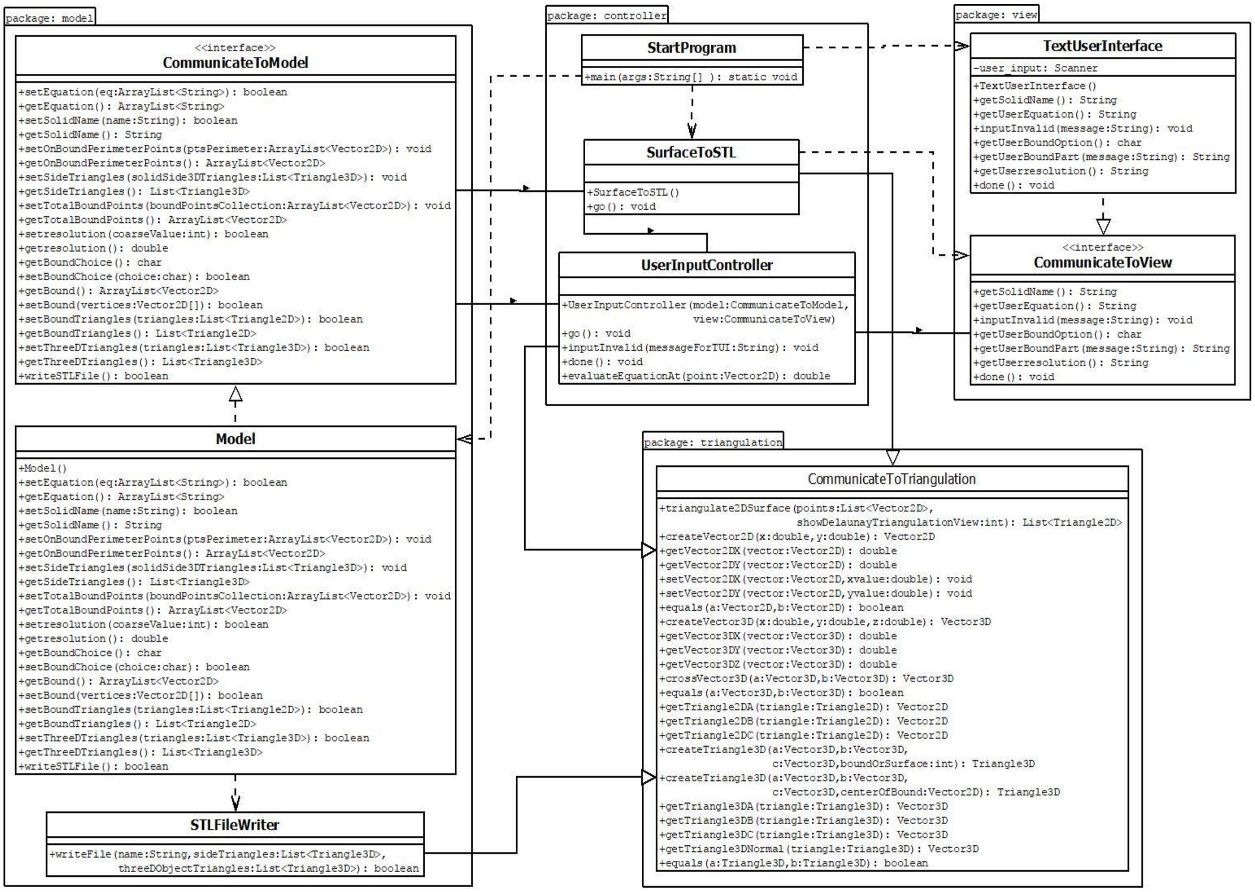

package: triangulation
DelaunayTriangulationView
-DIMENSION: Dimension
-COLOR_TRIANGLE_FILL: Color
-COLOR_TRIANGLE_EDGES: Color
-COLOR_TRIANGLE_BORDER: Color
-COLOR_BACKGROUND: Color
+delaunayTriangulator: DelaunayTriangulator
+pointSet: List<Vector2D>
+setDelaunayTriangulator(dT:DelaunayTriangulator): void
+plot(): void
+init(drawable:GLAutoDrawable): void
+reshape(drawable:GLAutoDrawable,x:int,y:int, width:int,height:int): void
+display(drawable:GLAutoDrawable): void
+displayChanged(drawable:GLAutoDrawable, modeChanged:boolean,deviceChanged:boolean): void
+dispose(drawable:GLAutoDrawable): void
+mouseClicked(e:MouseEvent): void
+mousePressed(e:MouseEvent): void
+mouseReleased(e:MouseEvent): void
+mouseEntered(e:MouseEvent): void
+mouseExited(e:MouseEvent): void
java.awt.Frame
java.util.List
java.util.ArrayList
Triangle2D
Vector2D
java.awt.Color
java.awt.Dimension
java.awt.event.*
com.jogamp.opengl.*
java.lang.Exception
NotEnoughPointsException
-serialVersionUID: long = 7061712854155625067L
+NotEnoughPointsException()
+NotEnoughPointsException(s:String)
Triangle3D
+a: Vector3D
+b: Vector3D
+c: Vector3D
+normal: Vector3D
+Triangle3D(a:Vector3D,b:Vector3D,c:Vector3D, boundOrSurface:int)
+Triangle3D(a:Vector3D,b:Vector3D,c:Vector3D, centerOfBound:Vector2D)
+toString(): String
+compareTo(other:Triangle3D): int
Vector3D
+x: double
+y: double
+z: double
+Vector3D(x:double,y:double,z:double)
+cross(other:Vector3D): Vector3D
+equals(other:Vector3D): boolean
+negate(): void
+makeUnitVector(): void
+toString(): String
DelaunayTriangulator
-pointSet: List<Vector2D>
-triangleSoup: TriangleSoup
+DelaunayTriangulator(pointSet:List<Vector2D>)
+triangulate(): void
-legalizeEdge(triangle:Triangle2D,edge:Edge2D, newVertex:Vector2D): void
+shuffle(): void
+shuffle(permutation:int[]): void
+getPointSet(): List<Vector2D>
+getTriangles(): List<Triangle2D>
java.util.Collections
java.util.ArrayList
java.util.List
DelaunayTriangulationView
TriangleSoup
-triangleSoup: List<Triangle2D>
+TriangleSoup()
+add(triangle:Triangle2D): void
+remove(triangle:Triangle2D): void
+getTriangles(): List<Triangle2D>
+findContainingTriangle(point:Vector2D): Triangle2D
+findNeighbor(triangle:Triangle2D,edge:Edge2D): Triangle2D
+findOneTriangleSharing(edge:Edge2D): Triangle2D
+findNearestEdge(point:Vector2D): Edge2D
+removeTrianglesUsing(vertex:Vector2D): void
java.util.Arrays
CommunicateToTriangulation
+triangulate2DSurface(points:List<Vector2D>, showDelaunayTriangulationView:int): List<Triangle2D>
+createVector2D(x:double,y:double): Vector2D
+getVector2DX(vector:Vector2D): double
+getVector2DY(vector:Vector2D): double
+setVector2DX(vector:Vector2D,xvalue:double): void
+setVector2DY(vector:Vector2D,yvalue:double): void
+equals(a:Vector2D,b:Vector2D): boolean
+createVector3D(x:double,y:double,z:double): Vector3D
+getVector3DX(vector:Vector3D): double
+getVector3DY(vector:Vector3D): double
+getVector3DZ(vector:Vector3D): double
+crossVector3D(a:Vector3D,b:Vector3D): Vector3D
+equals(a:Vector3D,b:Vector3D): boolean
+getTriangle2DA(triangle:Triangle2D): Vector2D
+getTriangle2DB(triangle:Triangle2D): Vector2D
+getTriangle2DC(triangle:Triangle2D): Vector2D
+createTriangle3D(a:Vector3D,b:Vector3D, c:Vector3D,boundOrSurface:int): Triangle3D
+createTriangle3D(a:Vector3D,b:Vector3D, c:Vector3D,centerOfBound:Vector2D): Triangle3D
+getTriangle3DA(triangle:Triangle3D): Vector3D
+getTriangle3DB(triangle:Triangle3D): Vector3D
+getTriangle3DC(triangle:Triangle3D): Vector3D
+getTriangle3DNormal(triangle:Triangle3D): Vector3D
+equals(a:Triangle3D,b:Triangle3D): boolean
EdgeDistancePack
+edge: Edge2D
+distance: double
+EdgeDistancePack(edge:Edge2D,distance:double)
+compareTo(o:EdgeDistancePack): int
Edge2D
+a: Vector2D
+b: Vector2D
+Edge2D(a:Vector2D,b:Vector2D)
Vector2D
+x: double
+y: double
+Vector2D(x:double,y:double)
+length(): double
+equals(other:Vector2D): boolean
+sub(vector:Vector2D): Vector2D
+add(vector:Vector2D): Vector2D
+mult(scalar:double): Vector2D
+mag(): double
+dot(vector:Vector2D): double
+cross(vector:Vector2D): double
+toString(): String
Triangle2D
+a: Vector2D
+b: Vector2D
+c: Vector2D
+Triangle2D(a:Vector2D,b:Vector2D,c:Vector2D)
+contains(point:Vector2D): boolean
+isPointInCircumcircle(point:Vector2D): boolean
+isOrientedCCW(): boolean
+isNeighbor(edge:Edge2D): boolean
+getNoneEdgeVertex(edge:Edge2D): Vector2D
+hasVertex(vertex:Vector2D): boolean
+findNearestEdge(point:Vector2D): EdgeDistancePack
-computeClosestPoint(edge:Edge2D,point:Vector2D): Vector2D
-hasSameSign(a:double,b:double): boolean
+toString(): String

Communication between Components

The ways in which components will communicate with each other are shown below.

Class Diagram

The class diagram that shows the classes that are responsible for the components communicating with each other is shown below.


package: model
<<interface>>
CommunicateToModel
+setEquation(eq:ArrayList<String>): boolean
+getEquation(): ArrayList<String>
+setSolidName(name:String): boolean
+getSolidName(): String
+setOnBoundPerimeterPoints(ptsPerimeter:ArrayList<Vector2D>): void
+getOnBoundPerimeterPoints(): ArrayList<Vector2D>
+setSideTriangles(solidSide3DTriangles:List<Triangle3D>): void
+getSideTriangles(): List<Triangle3D>
+setTotalBoundPoints(boundPointsCollection:ArrayList<Vector2D>): void
+getTotalBoundPoints(): ArrayList<Vector2D>
+setresolution(coarseValue:int): boolean
+getresolution(): double
+getBoundChoice(): char
+setBoundChoice(choice:char): boolean
+getBound(): ArrayList<Vector2D>
+setBound(vertices:Vector2D[]): boolean
+setBoundTriangles(triangles:List<Triangle2D>): boolean
+getBoundTriangles(): List<Triangle2D>
+setThreeDTriangles(triangles:List<Triangle3D>): boolean
+getThreeDTriangles(): List<Triangle3D>
+writeSTLFile(): boolean
Model
+Model()
+setEquation(eq:ArrayList<String>): boolean
+getEquation(): ArrayList<String>
+setSolidName(name:String): boolean
+getSolidName(): String
+setOnBoundPerimeterPoints(ptsPerimeter:ArrayList<Vector2D>): void
+getOnBoundPerimeterPoints(): ArrayList<Vector2D>
+setSideTriangles(solidSide3DTriangles:List<Triangle3D>): void
+getSideTriangles(): List<Triangle3D>
+setTotalBoundPoints(boundPointsCollection:ArrayList<Vector2D>): void
+getTotalBoundPoints(): ArrayList<Vector2D>
+setresolution(coarseValue:int): boolean
+getresolution(): double
+getBoundChoice(): char
+setBoundChoice(choice:char): boolean
+getBound(): ArrayList<Vector2D>
+setBound(vertices:Vector2D[]): boolean
+setBoundTriangles(triangles:List<Triangle2D>): boolean
+getBoundTriangles(): List<Triangle2D>
+setThreeDTriangles(triangles:List<Triangle3D>): boolean
+getThreeDTriangles(): List<Triangle3D>
+writeSTLFile(): boolean
STLFileWriter
+writeFile(name:String,sideTriangles:List<Triangle3D>, threeDObjectTriangles:List<Triangle3D>): boolean
package: controller
StartProgram
+main(args:String[]): static void
SurfaceToSTL
+SurfaceToSTL()
+go(): void
UserInputController
+UserInputController(model:CommunicateToModel, view:CommunicateToView)
+go(): void
+inputInvalid(messageForTUI:String): void
+done(): void
+evaluateEquationAt(point:Vector2D): double
package: view
TextUserInterface
-user_input: Scanner
+TextUserInterface()
+getSolidName(): String
+getUserEquation(): String
+inputInvalid(message:String): void
+getUserBoundOption(): char
+getUserBoundPart(message:String): String
+getUserresolution(): String
+done(): void
<<interface>>
CommunicateToView
+getSolidName(): String
+getUserEquation(): String
+inputInvalid(message:String): void
+getUserBoundOption(): char
+getUserBoundPart(message:String): String
+getUserresolution(): String
+done(): void
package: triangulation
CommunicateToTriangulation
+triangulate2DSurface(points:List<Vector2D>, showDelaunayTriangulationView:int): List<Triangle2D>
+createVector2D(x:double,y:double): Vector2D
+getVector2DX(vector:Vector2D): double
+getVector2DY(vector:Vector2D): double
+setVector2DX(vector:Vector2D,xvalue:double): void
+setVector2DY(vector:Vector2D,yvalue:double): void
+equals(a:Vector2D,b:Vector2D): boolean
+createVector3D(x:double,y:double,z:double): Vector3D
+getVector3DX(vector:Vector3D): double
+getVector3DY(vector:Vector3D): double
+getVector3DZ(vector:Vector3D): double
+crossVector3D(a:Vector3D,b:Vector3D): Vector3D
+equals(a:Vector3D,b:Vector3D): boolean
+getTriangle2DA(triangle:Triangle2D): Vector2D
+getTriangle2DB(triangle:Triangle2D): Vector2D
+getTriangle2DC(triangle:Triangle2D): Vector2D
+createTriangle3D(a:Vector3D,b:Vector3D, c:Vector3D,boundOrSurface:int): Triangle3D
+createTriangle3D(a:Vector3D,b:Vector3D, c:Vector3D,centerOfBound:Vector2D): Triangle3D
+getTriangle3DA(triangle:Triangle3D): Vector3D
+getTriangle3DB(triangle:Triangle3D): Vector3D
+getTriangle3DC(triangle:Triangle3D): Vector3D
+getTriangle3DNormal(triangle:Triangle3D): Vector3D
+equals(a:Triangle3D,b:Triangle3D): boolean

Tessellation Algorithm Overview

This section contains a description of the tessellation algorithm used (Requirement 2).

1. After getting valid user equation, resolution, and bound information, make sure RULE 1 satisfied.
    a. X and Y values:
        i. Find the smallest x and y value for the bound.
        ii. If the x value is negative, move the bound in the positive x direction until the x value in question becomes 0. This entails replacing “x” in the user equation with “(x-insert_abs(minx)_value_here)” -- for example, if the minimum x value was -3, we would replace all occurrences of “x” in the equation with “(x-3)”. This places the most negative x value at 0 and shifts all other x values in the bound in the positive direction so that the entire bound is in the positive octant.
        iii. Repeat the previous step for the minimum y value. This should translate the surface into the positive x and positive y domain.
    b. Z values:

    i. Check that the 3D equation is above the XY plane over the bound (we check a few points on the perimeter of the bound and the center of the bound). If not above the XY plane at any of these points, ask for new input.

2. Add additional points to estimate the bound according to the resolution (High resolution, more points).
3. Tessellate the 2D bound (RULE 2) by giving an external triangulation package the collection of points estimating the bound, resulting in a bunch of 2D triangles identified by their three x,y vertices. For each triangle in the 2D tessellation approximate the bound and the surface:
    a. Store the triangle vertices with z values equal to 1 (RULE 3) and the resulting unit normal vectors (RULE 4) for the bound approximation.
    b. Plug the 3 x,y vertices into the z=f(x,y) function to get the corresponding z values. Create a plane out of the 3 x,y,z vertices (3 non-collinear points create a plane) to approximate the 3D surface in between the 3 points. Get the unit normal vector (RULE 4) for the 3D triangle and store the normal and vertices (RULE 3) for the surface approximation.
4. If the 3D surface does not touch the bound on the XY plane in some place(s), form a side reaching from the bound approximation just above the XY plane up to the surface approximation. Tessellate this side approximation (RULE 2,3), find the unit normal vectors (RULE 4), and store for later.
5. Put each triangle’s vertices and normal vector information into the STL file. Try to satisfy the triangle sorting rule (RULE 5).