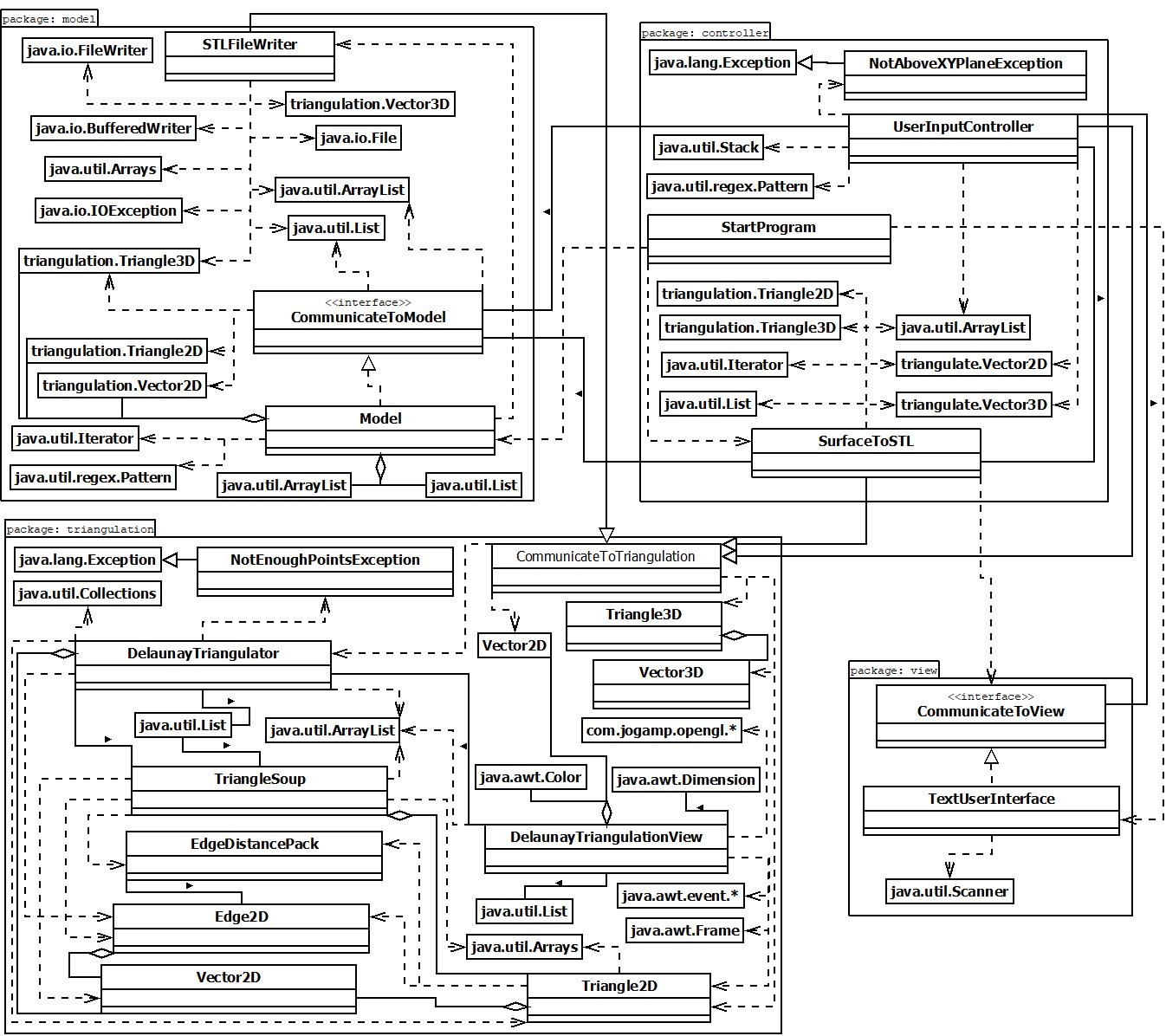
# Architecture

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.

# LDM v3.jpeg

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 |

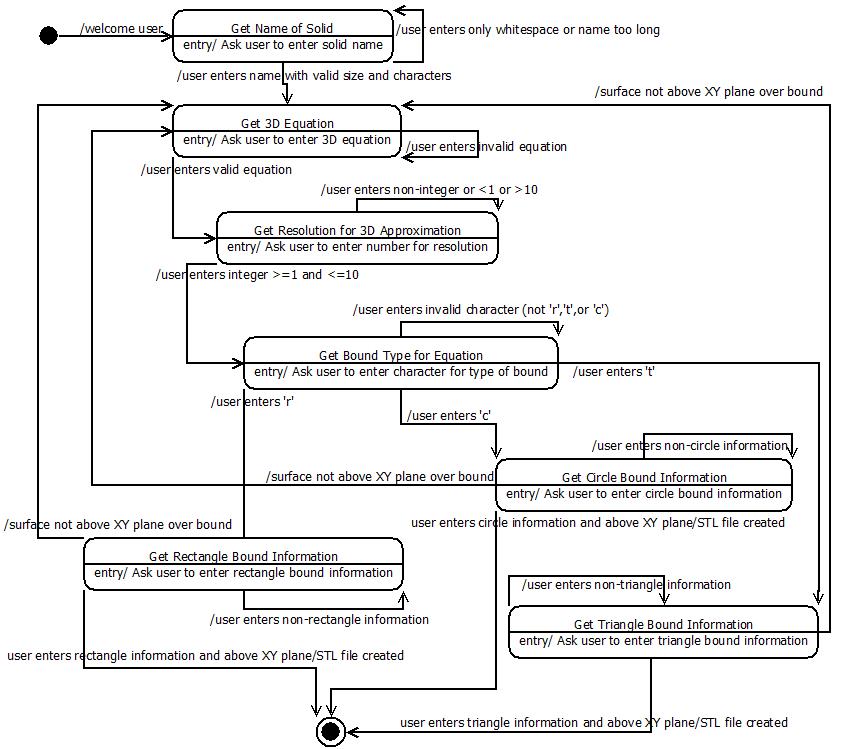
# Interfaces

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

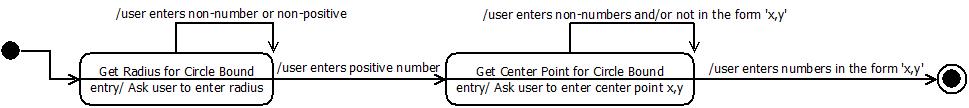
## Human-computer Interaction

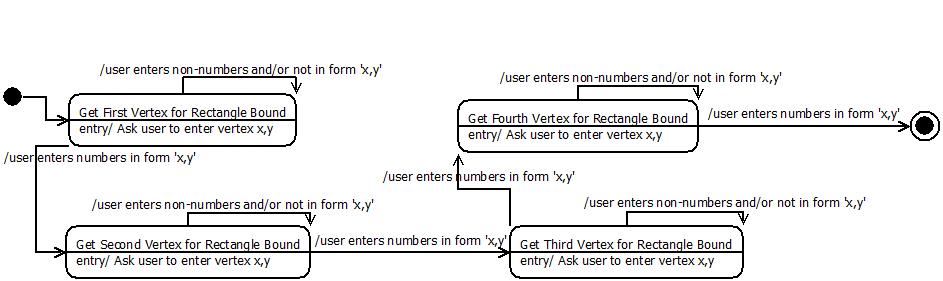
The user interactions with the application are shown below. The high-level state machine diagram is followed by the more specific state machine diagrams for the circle, rectangle, and triangle bound cases.

High-Level:

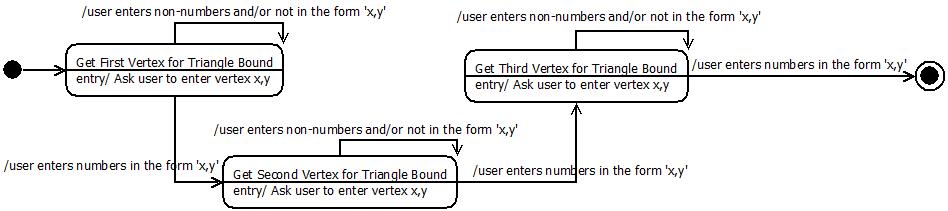


Circle:



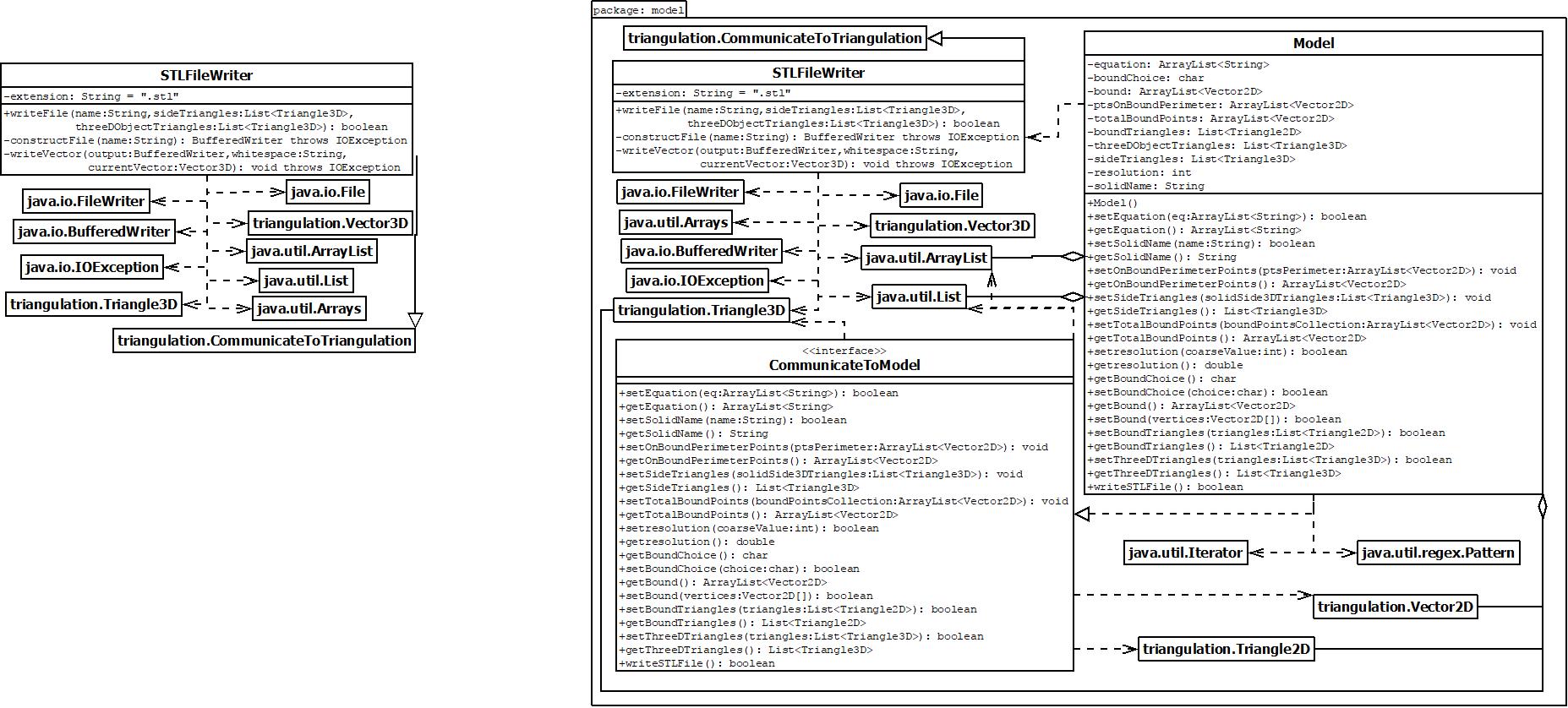
Rectangle:

Triangle:

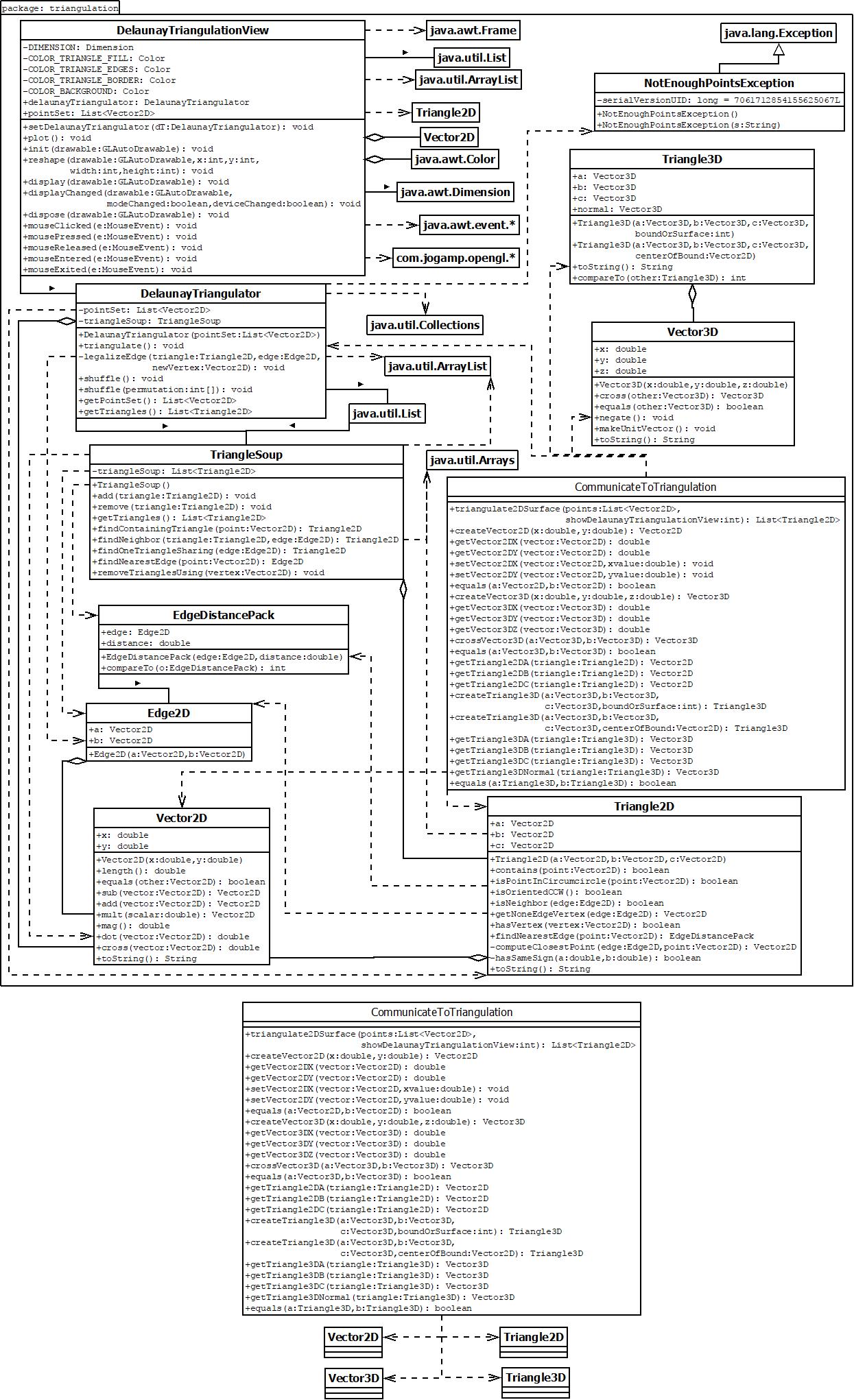


## Communication with External Entities

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



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



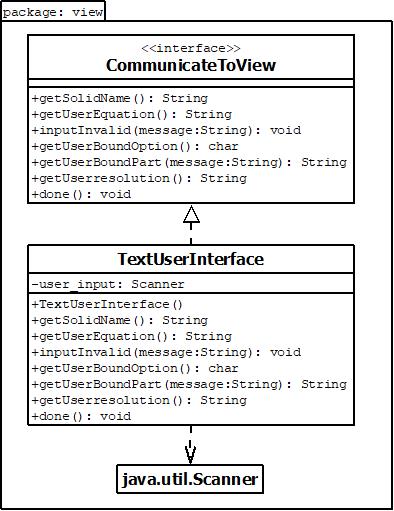
# Components

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

## View

### Class Diagram

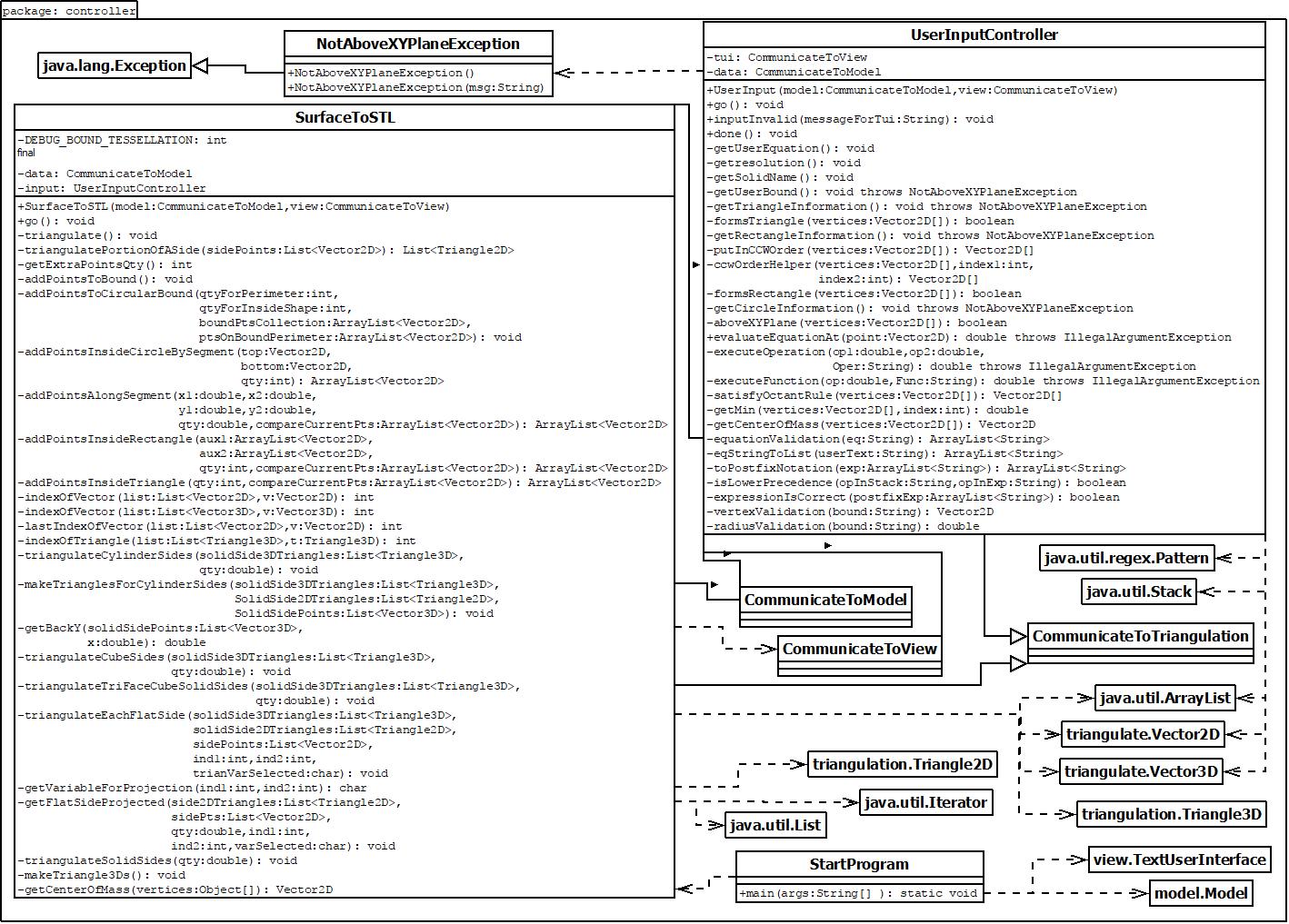
The class diagram for the View component is shown below.



## Controller

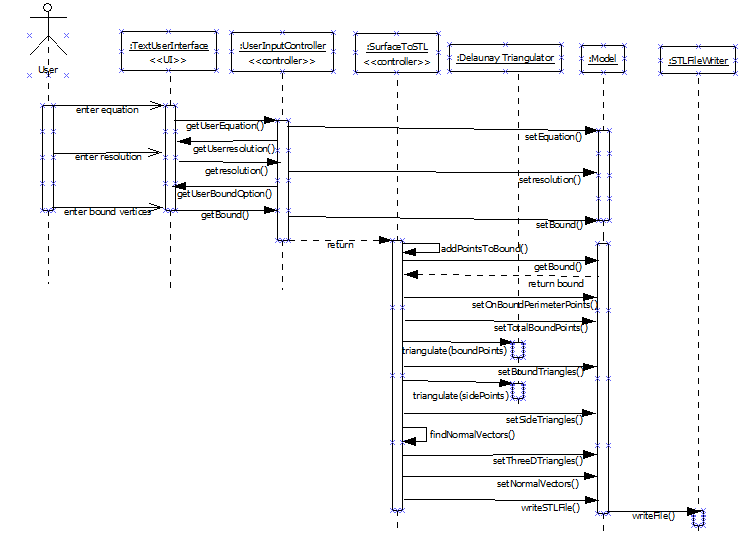
### Class Diagram

The class diagram for the Controller component is shown below.



### Behavior

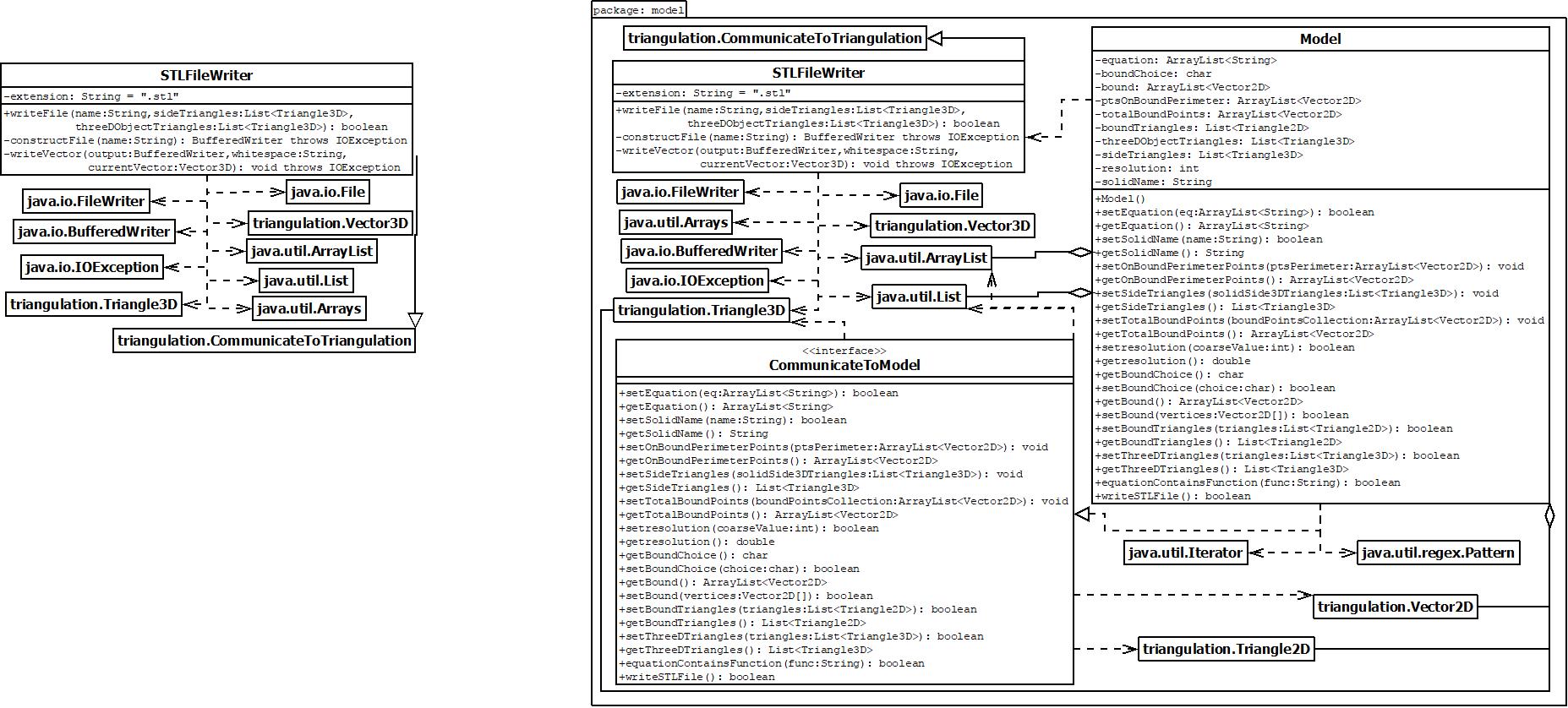
The behavior diagram for the Controller component and overall System interactions is shown below.



## Model

### Class Diagram

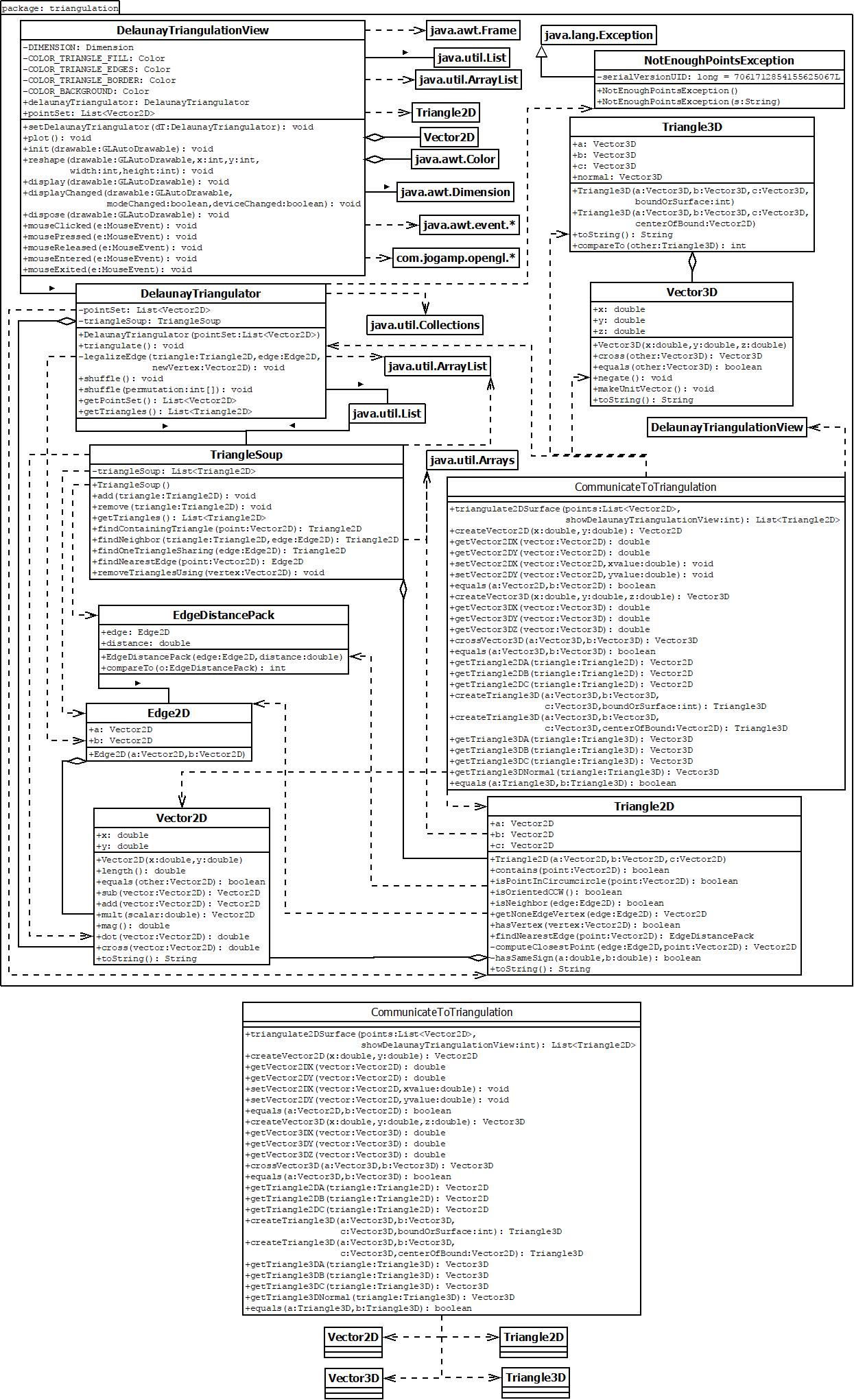
The class diagram for the Model component is shown below.

****

## Triangulation

### Class Diagram

The class diagram for the Triangulation component is shown below.

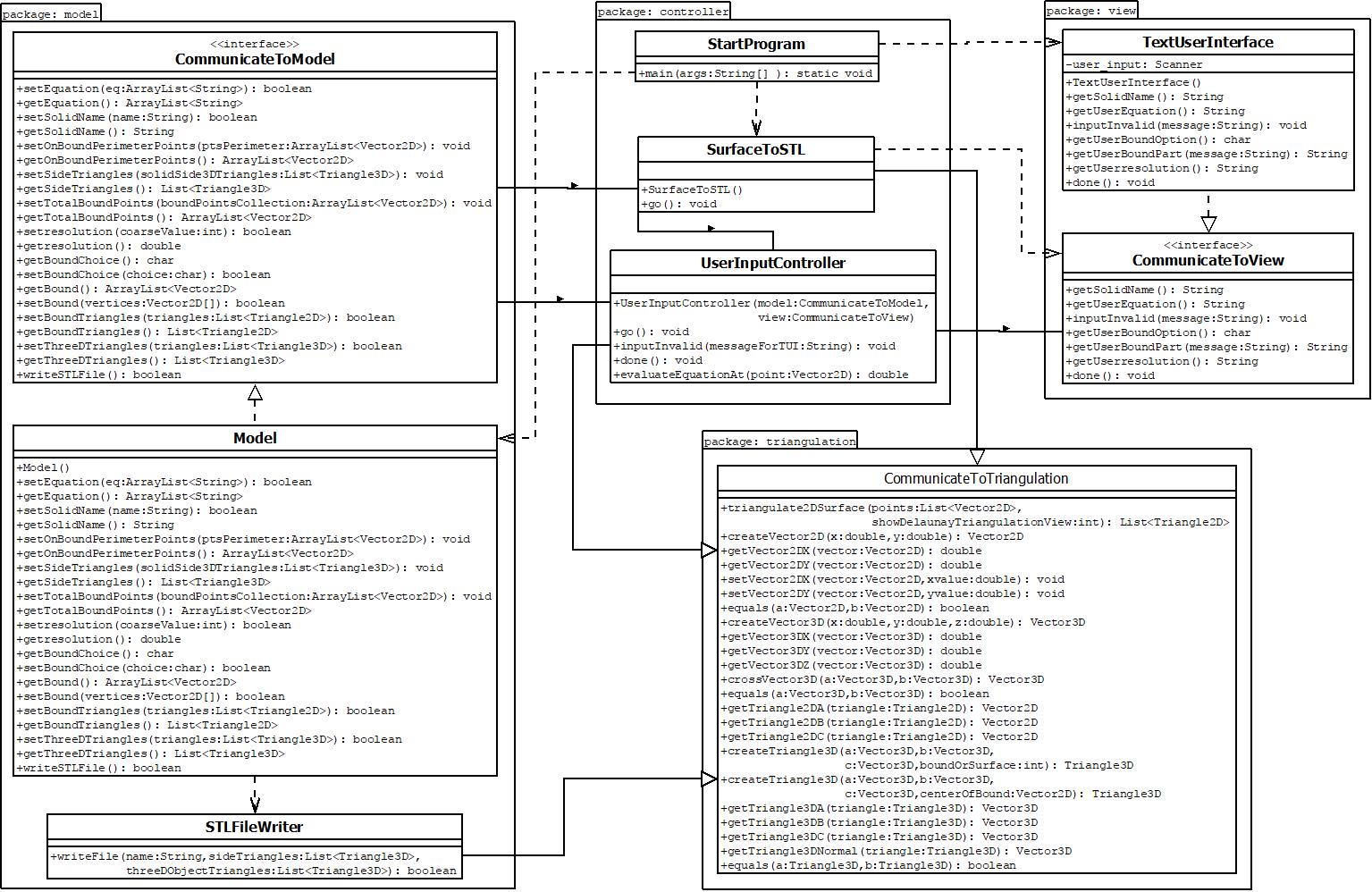


## Communication between Components

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.



# 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.
   1. X and Y values:
      1. Find the smallest x and y value for the bound.
      2. 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.
      3. Repeat the previous step for the minimum y value. This should translate the surface into the positive x and positive y domain.
   2. Z values:
      1. 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:
   1. Store the triangle vertices with z values equal to 1 (RULE 3) and the resulting unit normal vectors (RULE 4) for the bound approximation.
   2. 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).