

# CSCB07 – Software Design

## Assignment

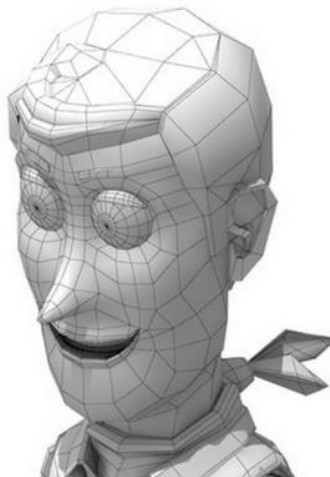
---

### Logistics

- This assignment is worth 10% of the course grade and is due on July 10, 2022.
- It should be done individually and the code you submit must be your own.
- During the tutorial sessions of Week 10 (July 11-15, 2022) and as part of the evaluation, the TAs will conduct short individual interviews to assess your understanding of the submitted code.

### Instructions

In computer graphics, an object is usually represented by dividing its surface into a set of polygons (also called *faces*). Such set is called a mesh.



Graphical elements such as vertices, polygons, and meshes can be transformed (e.g. translated, rotated, etc.) using transformation matrices. For example, to rotate a vertex  $v = (v_x, v_y, v_z)$  about the x-axis with angle  $\theta$ , we can multiply the following matrix by the column vector representing  $v$ :

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$\text{That is, } v \text{ would be transformed into } v' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \cos\theta - v_z \sin\theta \\ v_y \sin\theta + v_z \cos\theta \end{bmatrix}$$

Similarly, the following matrices can be used to rotate  $v$  about the y-axis and the z-axis respectively:

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Several file formats can be used to represent meshes: OBJ, PLY, OFF, STL, FBX, etc. In this assignment, we will only target the OBJ, PLY, and OFF formats.

### **OBJ format**

The OBJ files to be considered consist of two consecutive sets of lines: the first set (lines starting with a “v”) represents the unique vertices contained in the mesh and the second set (lines starting with an “f”) represents the faces of the mesh. For example, the mesh below contains two polygons: the first includes the vertices defined at lines 1, 2, and 3 and the second includes the vertices defined at lines 2, 3, and 4.

```
v 5.1 1.2 0.3
v 4.9 1.5 0.3
v 3.8 1.4 0.5
v 4.1 1.6 0.6
f 1 2 3
f 2 3 4
```

### **PLY format**

The PLY format is similar to OBJ except for the following:

- It has a header that specifies different aspects including the number of vertices and the number of faces. You can assume that the highlighted parts are fixed.
- The lines representing the vertices do not start with a “v”.
- The lines representing the faces do not start with an “f”. Instead, they start with the number of vertices included in the corresponding face.
- Vertex indexing starts at 0 instead of 1. That is, the first line containing a vertex is labelled 0.

```

ply
format ascii 1.0
element vertex 4
property float32 x
property float32 y
property float32 z
element face 2
property list uint8 int32 vertex_indices
end_header
5.1 1.2 0.3
4.9 1.5 0.3
3.8 1.4 0.5
4.1 1.6 0.6
3 0 1 2
3 1 2 3

```

### OFF format

The OFF file format is defined as follows:

- The first line contains the term "OFF"
- The second line consists of three items: the number of vertices, the number of faces, and the number of edges (which can be ignored in this assignment by using the value 0)
- The lines representing vertices are similar to those of the PLY format
- The lines representing faces are similar to those of the PLY format except that each line has an optional component representing the color in RGB format. When color is considered, three extra values are included where each ranges between 0 and 255.

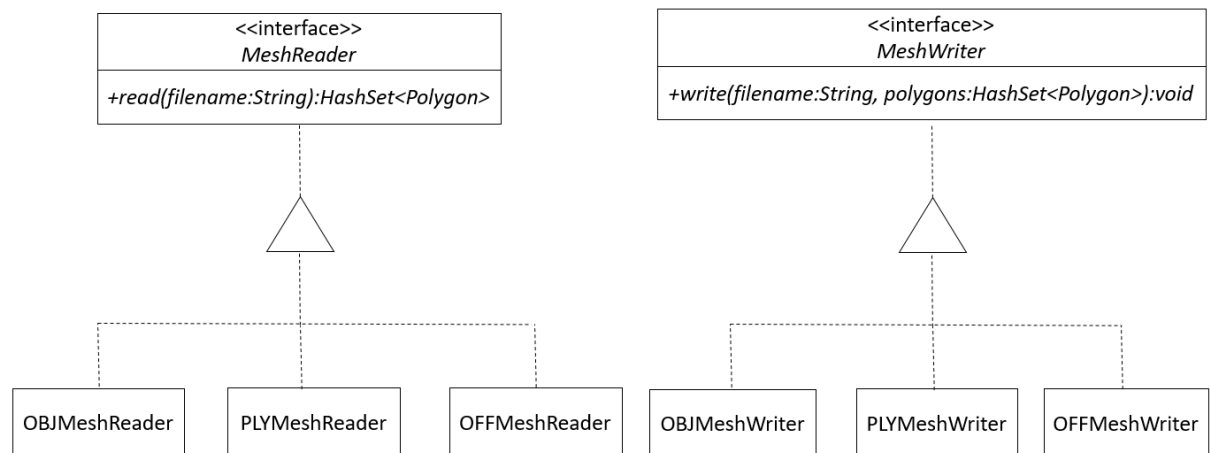
```

OFF
4 2 0
5.1 1.2 0.3
4.9 1.5 0.3
3.8 1.4 0.5
4.1 1.6 0.6
3 0 1 2 220 220 200
3 1 2 3 220 220 200

```

1. Create a new java project and add to it a package named **assignment**
2. Define a class named **GraphicalObject** representing graphical objects that could be transformed using a matrix. It has the following **void** methods:
  - a. **transform**: an abstract method that takes a two-dimensional matrix of **double** representing the transformation matrix.

- b. **rotateXAxis**: a concrete method that rotates the graphical object about the x-axis. It takes one argument of type **double** representing  $\theta$ .
  - c. **rotateYAxis**: a concrete method that rotates the graphical object about the y-axis. It takes one argument of type **double** representing  $\theta$ .
  - d. **rotateZAxis**: a concrete method that rotates the graphical object about the z-axis. It takes one argument of type **double** representing  $\theta$ .
3. Define class **Vertex** as follows:
- a. It has three fields of type **double** representing the coordinates of the vertex.
  - b. It has a constructor that takes three arguments of type **double** and initializes the coordinates accordingly.
  - c. It inherits from **GraphicalObject**.
  - d. It overrides **hashCode** and **equals**.
  - e. It overrides **toString** by returning a string containing the three coordinates separated by spaces.
4. Define class **Polygon** as follows:
- a. It has one field of type **LinkedHashSet<Vertex>** named **vertices**
  - b. It has a constructor that takes one argument of type **LinkedHashSet<Vertex>** and initializes **vertices** accordingly.
  - c. It inherits from **GraphicalObject**. Transforming a polygon is done by transforming all of its vertices.
  - d. It overrides **hashCode** and **equals**.
5. Define a class named **WrongFormatException** that represents a checked exception to be thrown when attempting to read a file having the wrong format (e.g. when reading a mesh from an OBJ file and encountering a vertex line that doesn't start with a "v")
6. Define interfaces **MeshReader** and **MeshWriter**, as well as classes **OBJMeshReader**, **PLYMeshReader**, **OFFMeshReader**, **OBJMeshWriter**, **PLYMeshWriter**, and **OFFMeshWriter** according to the following UML class diagram. Each of the classes is supposed to provide an implementation that reads/writes polygons as per the corresponding format. When reading a mesh, the format of the corresponding file should be validated using regular expressions and any violation should result in a **WrongFormatException** being thrown.



7. Define class **Mesh** as follows:

- a. It has a field of type **HashSet<Polygon>** named **polygons**
- b. It has a field of type **MeshReader** named **reader**
- c. It has a field of type **MeshWriter** named **writer**
- d. It has a **void** method named **setReader** that takes an argument of type **MeshReader** and sets **reader** accordingly.
- e. It has a **void** method named **setWriter** that takes an argument of type **MeshWriter** and sets **writer** accordingly.
- f. It has a **void** method named **readFromFile** that takes a file name as an argument and uses **reader** to read the polygons from the file.
- g. It has a **void** method named **writeToFile** that takes a file name as an argument and uses **writer** to write the mesh to the file.
- h. It inherits from **GraphicalObject**. Transforming a mesh is done by transforming all of its polygons.
- i. It overrides **hashCode** and **equals**.

8. Write enough JUnit tests to test your code properly and maximize coverage. Test methods that aim at increasing coverage by adding code that is not relevant to the assertion are not allowed.

9. Kindly take note of the following:

- a. Adding extra fields or methods to the aforementioned interfaces and classes is not allowed.
- b. You can test your code using the files "car.obj", "car.ply", "car.off", and "car\_rotated.obj" provided with this assignment.
- c. You can visualize OBJ, PLY, and OFF files online (e.g. <https://3dviewer.net/>) if you do not have an appropriate viewer installed.
- d. You might want to use the following methods when reading files

- i. Method **split** of class **String**
  - ii. Method **parseDouble** of class **Double**
- e. The following code reads a mesh from an OBJ file, rotates it about the y-axis by  $\pi/3$ , and saves the rotated mesh to OBJ and PLY files:

```
public static void main(String [] args) {  
    Mesh mesh = new Mesh();  
    mesh.setReader(new OBJMeshReader());  
    mesh.readFromFile("/Users/rawad/car.obj");  
    mesh.rotateYAxis(Math.PI/3);  
    mesh.setWriter(new OBJMeshWriter());  
    mesh.writeToFile("/Users/rawad/car_rotated.obj");  
    mesh.setWriter(new PLYMeshWriter());  
    mesh.writeToFile("/Users/rawad/car_rotated.ply");  
}
```

### **Submission**

Upload the java files as a single archive file to "Assignment" on Quercus by July 10<sup>th</sup> at the latest.