**CSE Programming in C#** **Roger Crawfis**

**The Ohio State University**

# Programming Assignment #2

## A SceneGraph Library

Start with the accompanying worksheet and write a small library (.dll) for a SceneGraph interface (see http://en.wikipedia.org/wiki/Scenegraph). The library will support various nodes (derived from a base type of SceneNode) for drawables, for materials, for transformations and for other state changes. The *class* hierarchy will look like the following:

- ISceneNode
    - IDrawableNode (derived from ISceneNode)
        - DrawableNodeBase (implements IDrawNode)
            - Cube (derived from DrawableNodeBase)
            - Sphere (derived from DrawableNodeBase)
            - Building (derived from DrawableNodeBase)
            - Terrain (derived from DrawableNodeBase)
    - ITransformNode (derived from ISceneNode)
        - Camera (implements ITransformNode)
        - Rotate (implements ITransformNode)
        - Translate (implements ITransformNode)
        - Scale (implements ITransformNode)
        - Perspective (implements ITransformNode)
    - IStateNode (derived from ISceneNode)
        - DrawMode (implements IStateNode)
    - IGroupNode (derived from ISceneNode)

ISceneNode, IDrawableNode, ITransformNode and IStateNode are all ***interfaces***. DrawableNodeBase is an ***abstract*** class and has a partial implementation in it. ISceneNode has several ***behaviors*** defined:

- ISceneNode
    - void Accept(IVisitor visitor)
    - string Name { get; }

You will also create an ***interface*** called IVisitor as part of the Visitor Design Pattern (http://www.dofactory.com/Patterns/Patterns.aspx). Its ***behavior or contract*** is as follows (C++ style):

- Visitor
    - public:
        - virtual void PreVisit(IDrawableNode) = 0;

        - virtual void PostVisit(IDrawableNode) = 0;
        - virtual void PreVisit(ITransformNode) = 0;
        - virtual void PostVisit(ITransformNode) = 0;
        - virtual void PreVisit(IStateNode) = 0;
        - virtual void PostVisit(IStateNode) = 0;
        - virtual void PreVisit(IGroupNode) = 0;
        - virtual void PostVisit(IGroupNode) = 0;

In C++ we would call a class a pure abstract class if all of its members were defined this way (public, virtual and no implementation (=0)), and it contained no fields or other data. We will implement a few of concrete classes that support the IVisitor interface. One will simply walk the tree and print out the names. Another will print out the name as well as the type (using the ToString method of the Type type). It should print this out in a pretty format, with indentation of the children (aka a treeview like display).

- Visitor
    - NameVisitor – prints out the names to the console (implements IVisitor).
    - NameTypeVisitor – prints out the name and the type to the console (implements IVisitor). This was done in the accompanying worksheet.

Other details:

- DrawableNodeBase has an abstract Draw method.
- GroupNode will need an additional method to add ~~and remove~~ children from its collection. You should use a generic collection internally for the children.
- Both IStateNode and ITransformNode have methods for Apply and Unapply.
- For Draw and the Apply/Unapply methods simply print out a message to the console as your implementation of these in your concrete classes.

## A Sample Application

You will also create an additional application (a Console app) that will create the scenegraphs and print them out. For now, we will hard code the creation process. Use good functional decomposition in this sample application. In other words have a separate procedure or method to create each scenegraph. Test your code out with several scenegraphs. Each scenegraph has a single root node. Provide a depth up to seven levels and a fairly good breadth. Test out printing of a sub-graph of your scene. Note: The scene graphs should be **Directed Acyclic Graphs** (DAG), meaning that a child of one of the nodes should not also be an ancestor (parent, grandparent, etc.) of the same node.

## Additional Tasks

1. In the Properties folder, Open the AssemblyInfo.cs file and fill in the Title through the Copyright information. This information will be displayed if you look at the properties of your .exe file.
2. Provide comments explaining your logic. Also, add a block comment to the beginning of the file listing your name, the course and a description of the lab (from above but in a completed tense).
3. Do a ***Build Clean,*** zip up your solution (no .exe files are allowed) and then submit your assignment using submit c459ae directory.