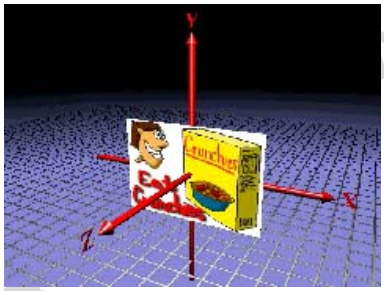
SPECIAL BEHAVIORS

1. Some specialized behaviors are provided by Java 3D
   - *Billboard* auto-rotation of shapes to face the viewer
   - Changing *levels of detail* based on distance to viewer
   - Specialized behaviors are extensions of `Behavior`

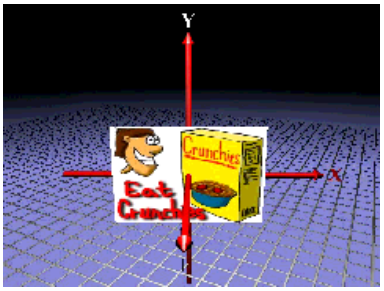
Class Hierarchy

```
java.lang.Object
└ javax.media.j3d.SceneGraphObject
  └ javax.media.j3d.Node
    └ javax.media.j3d.Leaf
      └ javax.media.j3d.Behavior
        ├ javax.media.j3d.Billboard
        └ javax.media.j3d.LOD
          └ javax.media.j3d.DistanceLOD
```


2. A Billboard is a specialized behavior that:
   - Tracks the `ViewPlatform`
   - Generates a rotation about an axis so that the Z-axis points at the platform
   - Writes that transform to a target `TransformGroup`



Viewer steps to the right . . .



. . . and the behavior immediately rotates the shape

3. Billboard rotation can be about:
   - An axis to pivot the `TransformGroup`
   - A point to arbitrarily rotate the `TransformGroup`
     - Rotation makes the groups Y-axis parallel to the viewers Y-axis
4. Methods on `Billboard` set the alignment mode, rotation axis or point, and the target
   - The default alignment mode is about the Y axis
   - Alignment modes include `ROTATE_ABOUT_AXIS` (default) and `ROTATE_ABOUT_POINT`

5. `OritentedShape3D` objects are used to perform the same function as the `Billboard` behavior.
   - Major differences:
     - `OrientedShape3D` object work for applications with more than one view.
     - `OrientedShape3D` objects can be a shared object.
     - `OrientedShape3D` is not a behavior, and therefore it does not have a scheduling bounds to consider.
   - OrientedShape3D can be used for all billboard applications.
     - `Billboard` is not deprecated for the reason of backward compatibility with existing applications.

6. An `OrientedShape3D` either rotates about an axis or a point.
   - The OrientedShape3D object orients itself such that the local positive z-axis of its children face the viewer.
     - If an axis is specified in parallel to the Z axis, the `OrientedShaped3D` will simply not rotate.
   - If the alignment mode is `ROTATE_ABOUT_POINT`, the rotation will be about the specified point, with an additional rotation to align the +y axis of the `TransformGroup` with the +y axis in the `View`.
7. *OrientedShape3DApp* example:


```
1.    public BranchGroup createSceneGraph(SimpleUniverse su) {
2.         // Create the root of the branch graph
3.         BranchGroup objRoot = new BranchGroup();
4.
5.         Vector3f translate = new Vector3f();
6.         Transform3D T3D = new Transform3D();
7.         TransformGroup positionTG = null;
8.         OrientedShape3D orientedShape3D = null;
9.
10.        Geometry treeGeom = createTree();
11.
12.        //specify the position of the trees
13.        float[][] position = {{  0.0f, 0.0f,  -2.0f},
14.                              {-13.0f, 0.0f,  23.0f},
15.                              {  1.0f, 0.0f,  -3.5f}};
16.
17.        // for the positions in the array create a OS3D
18.        for (int i = 0; i < position.length; i++){
19.                translate.set(position[i]);
20.                T3D.setTranslation(translate);
21.                positionTG = new TransformGroup(T3D);
22.
23.                orientedShape3D = new OrientedShape3D();
24.                orientedShape3D.addGeometry(treeGeom);
25.
26.                objRoot.addChild(positionTG);
27.                positionTG.addChild(orientedShape3D);
28.        }
```

8. *Level of Detail* (LOD) varies the level of detail

- LOD is a specialized behavior that:
  - Tracks the `ViewPlatform`
  - Computes a distance to a shape
  - Maps the distance to `Switch` group child choices
- LOD is a technique to improve performance
  - It reduces the complexity of a visual object without affecting the visual result.
  - It helps to save a significant computational time without visual loss of content.
- The `LOD` abstract class generalizes level-of-detail behaviors, and the `DistanceLOD` class implements distance-based switching level-of-detail.

9. Recipe for using a `DistanceLOD` object

1. create a target Switch object(s) with ALLOW_SWITCH_WRITE capability
2. create list of distance thresholds array for the DistanceLOD object
3. create DistanceLOD object using the distance thresholds array
4. set the target switch object for the DistanceLOD object
5. supply a scheduling bounds (or bounding leaf) for the DistanceLOD object
6. assemble the scene graph, including adding children to target Switch object(s)

Figure 5-18 Recipe for Using a DistanceLOD Object to Provide Animation.

10. *DistanceLODApp* example:

```
1.      public BranchGroup createSceneGraph() {
2.          BranchGroup objRoot = new BranchGroup();
3.          BoundingSphere bounds = new BoundingSphere();
4.
5.          // create target TransformGroup with Capabilities
6.          TransformGroup objMove = new TransformGroup();
7.
8.          // create DistanceLOD target object ❶
9.          Switch targetSwitch = new Switch();
10.         targetSwitch.setCapability(Switch.ALLOW_SWITCH_WRITE);
11.
12.         // add visual objects to the target switch ❻
13.         targetSwitch.addChild(new Sphere(.40f, 0, 25));
14.         targetSwitch.addChild(new Sphere(.40f, 0, 15));
15.         targetSwitch.addChild(new Sphere(.40f, 0, 10));
16.         targetSwitch.addChild(new Sphere(.40f, 0,  4));
17.
18.         // create DistanceLOD object
19.         float[] distances = { 5.0f, 10.0f, 20.0f};❷
20.         DistanceLOD dLOD = new DistanceLOD(distances, new Point3f());❸
21.         dLOD.addSwitch(targetSwitch);      ❹
22.         dLOD.setSchedulingBounds(bounds); ❺
23.
24.         // assemble scene graph ❻
25.         objRoot.addChild(objMove);
26.         objMove.addChild(dLOD);          // make the bounds move with object
27.         objMove.addChild(targetSwitch); // must add switch to scene graph
28.
29.         return objRoot;
30.     } // end of CreateSceneGraph method of DistanceLODApp
```