OBJECT BEHAVIOR

1. A *Behavior* is a Java class used to change scene graphs

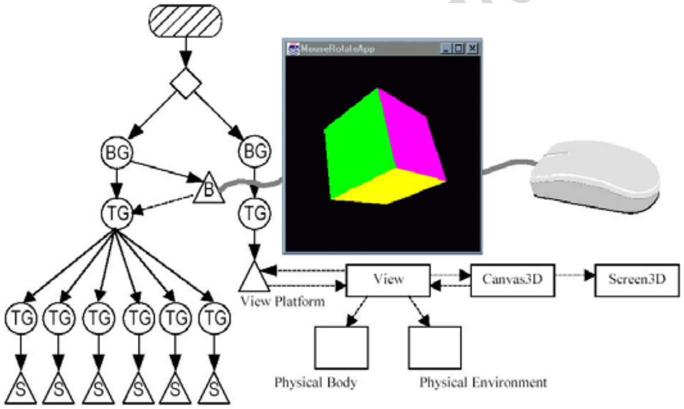
- In a broad sense, a Java application is a behavior.
- Java 3D also provides a `Behavior` class as a base class for smaller components that change the scene
  - ○ An application extends Behavior further to create one or more behaviors to change scene content
  - ○ Often one behavior for each shape being animated

Class Hierarchy

```
java.lang.Object
 └─ javax.media.j3d.SceneGraphObject
     └─ javax.media.j3d.Node
         └─ javax.media.j3d.Leaf
             └─ javax.media.j3d.Behavior
```

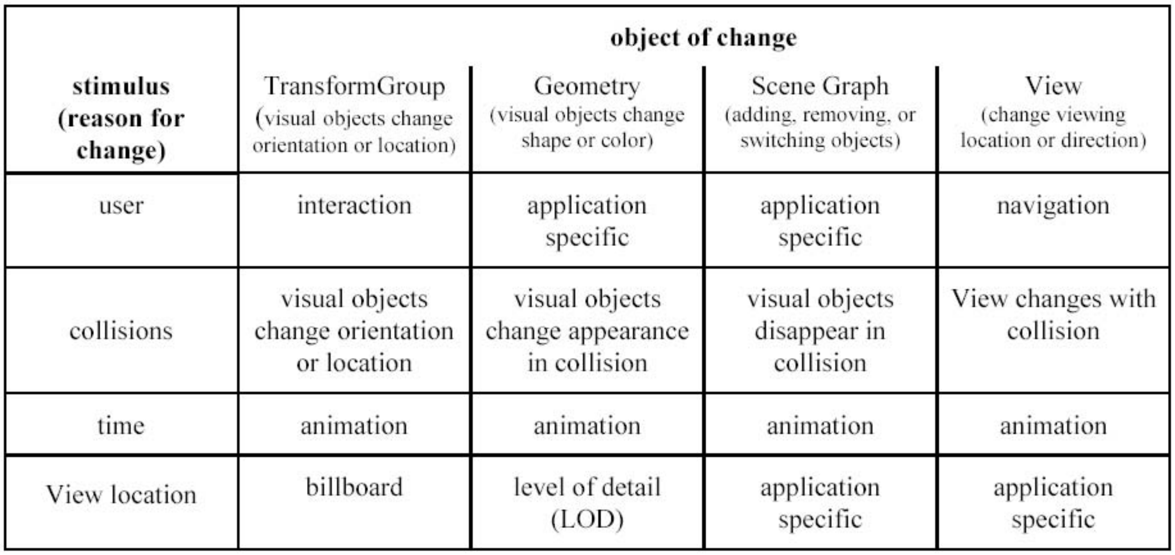
2. Java 3D behavior support:

- Supports arbitrary content changes via Java methods
- Schedules behaviors to run only when necessary
- Enables independent behaviors run in parallel
- Provides animation execution independent of host speed



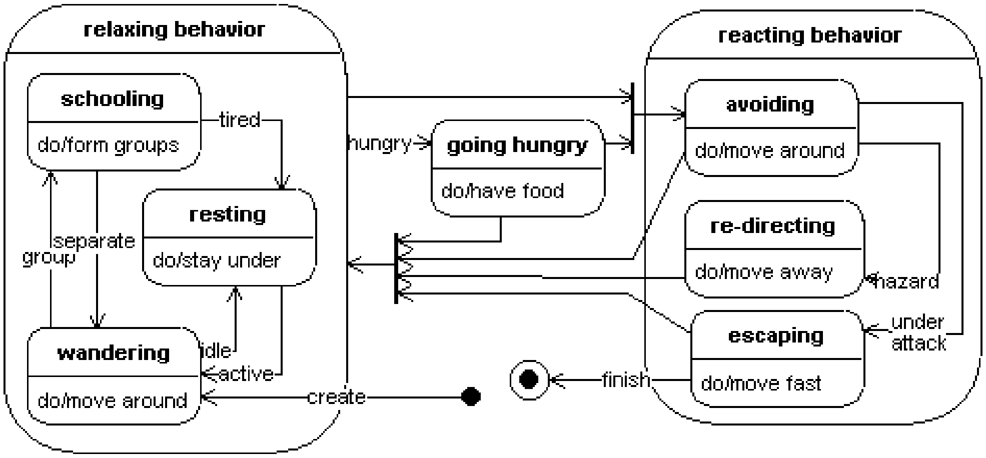
3. Behavior applications

Table 4-1 Applications of Behavior Categorized by Stimulus and Object of Change



stimulus (reason for change)	**object of change** TransformGroup (visual objects change orientation or location)	Geometry (visual objects change shape or color)	Scene Graph (adding, removing, or switching objects)	View (change viewing location or direction)
user	interaction	application specific	application specific	navigation
collisions	visual objects change orientation or location	visual objects change appearance in collision	visual objects disappear in collision	View changes with collision
time	animation	animation	animation	animation
View location	billboard	level of detail (LOD)	application specific	application specific

4. Virtual fish: an example application



5. `Behavior` is an abstract class that defines two methods that must be overridden by a subclass:

- The `initialize` method allows a Behavior object to initialize its internal state and specify its initial wakeup condition(s).
  - ○ A Behavior object is initialized once when the behavior's containing BranchGroup node is added to the virtual universe.
  - ○ A wakeup condition must be set or else the behavior's `processStimulus` method is never executed.
  - ○ Java 3D does not invoke the initialize method in a new thread. Thus, for Java 3D to regain control, the initialize method must not execute an infinite loop; it must return.

- The `processStimulus` method performs its computations and actions (possibly including the registration of state change information that could cause Java 3D to wake other Behavior objects), establishes its next wakeup condition, and finally exits.
  - ○ A typical behavior will modify one or more nodes or node components in the scene graph. These modifications can happen in parallel with rendering.
    - – All modifications to scene graph objects (not including geometry by-reference or texture by-reference) made from the `processStimulus` method of a single behavior instance are guaranteed to take effect in the same rendering frame.

    - All modifications to scene graph objects (not including geometry by-reference or texture by-reference) made from the `processStimulus` methods of the set of behaviors that wake up in response to a wakeup condition by `WakeupOnElapsedFrames(0)` are guaranteed to take effect in the same rendering frame,

  - Other than the above two cases, applications cannot count on behavior execution being synchronized with rendering. In particular, modifications to geometry by-reference or texture by-reference are not guaranteed to show up in the same frame as other scene graph changes.

6. The `Behavior` node also contains an enable flag, a scheduling region, a scheduling interval, and a wakeup condition.

   - The *scheduling region* defines a spatial volume that serves to enable the scheduling of Behavior nodes.

     - A `Behavior` node is active (can receive stimuli) whenever an active `ViewPlatform`'s activation volume intersects a `Behavior` object's scheduling region.

     - Only active behaviors can receive stimuli.

   - The `scheduling interval` defines a partial order of execution for behaviors that wake up in response to the same wakeup condition, i.e., those behaviors that are processed at the same "time".

  - Given a set of behaviors whose wakeup conditions are satisfied at the same time, the behavior scheduler will execute all behaviors in a lower scheduling interval before executing any behavior in a higher scheduling interval.
  - Within a scheduling interval, behaviors can be executed in any order, or in parallel.
- Wakeup conditions control when to wakeup next
  - Respecified on each wakeup
  - A `WakeupCondition` object is an abstract class specialized to different WakeupCriterion objects and to combining objects containing multiple `WakeupCriterion` objects.

- Scheduling bounds control scheduling
  - The behavior scheduler invokes the `processStimulus` method of a Behavior node when an active `ViewPlatform`'s activation volume intersects a Behavior object's scheduling region and all of that behavior's wakeup criteria are satisfied.

7. Creating behaviors

- A behavior can do anything
  - Perform computations
  - Update its internal state
  - Modify the scene graph
  - Start a thread

- For example, a behavior to rotate a radar dish to track an object:
  - On initialization, set initial wakeup criteria
  - Get the objects location
  - Create a transform to re-orient the radar dish
  - Set a `TransformGroup` of the radar dish
  - Set the next wakeup criteria
  - Return

8. Behavior example code

- Extend the `Behavior` class and fill in the `initialize` and `processStimulus` methods

```
public class MyBehavior extends Behavior {
    private WakeupCriterion criteria;
```

```
    public MyBehavior( ) {
        .  .  .  // Do something on construction
        criteria = new WakeupOnAWTEvent( .  .  .  );
    }
    public void initialize( ) {
        .  .  .  // Do something at startup
        wakeupOn( criteria );
    }
    public void processStimulus(Enumeration criteria) {
        .  .  .  // Do something on a wakeup
        wakeupOn( criteria );
    }
}
```

9. Creating behavior scheduling bounds

- A behavior only needs to be scheduled if the viewer is nearby
  - The viewers activation radius intersects its `scheduling bounds`
  - Behavior bounding enables costly behaviors to be skipped if they aren't nearby
- A behaviors scheduling bounds is a bounded volume
  - Sphere, box, polytope, or combination
  - To make a global behavior, use a huge bounding sphere

- By default, behaviors have no scheduling bounds and are never executed!
  - **Common error**: forgetting to set scheduling bounds

10. Anchoring scheduling bounds

- A behavior's bounding volume can be relative to:
  - The behaviors coordinate system
    - Volume centered on origin
    - As origin moves, so does volume
- A `Bounding leaf`'s coordinate system
  - Volume centered on leaf node elsewhere in scene graph
  - As that leaf node moves, so does volume
  - If behavior's origin moves, volume does not

11. Scheduling bounds example code

- Set bounds relative to the behavior's coordinate system

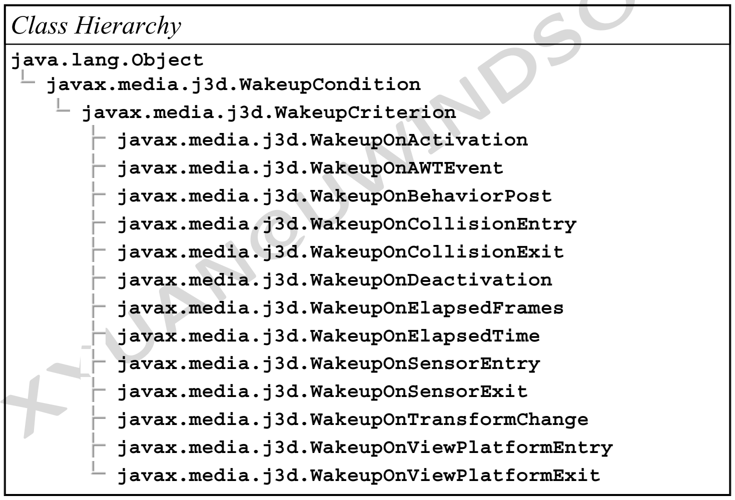
```
Behavior myBeh = new MyBehavior( );
myBeh.setSchedulingBounds( myBounds );
```

- Or relative to a bounding leafs coordinate system

```
TransformGroup myGroup = new TransformGroup( );
BoundingLeaf myLeaf = new BoundingLeaf( bounds );
myGroup.addChild( myLeaf );
.   .   .
Behavior myBeh = new MyBehavior( );
myBeh.setSchedulingBoundingLeaf( myLeaf );
```

12. Waking up a behavior

- `WakeupCriterion` extends `WakeupCondition` to provide multiple ways to wakeup a behavior



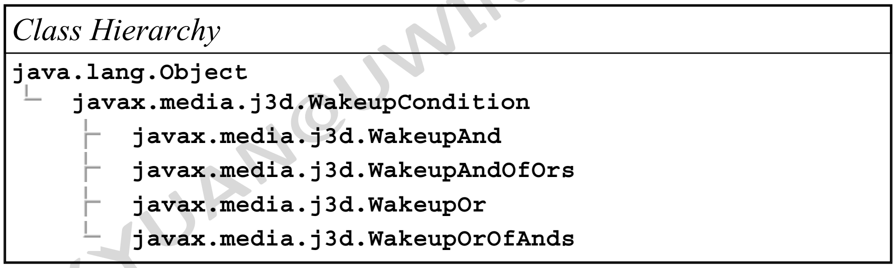
Class Hierarchy

```
java.lang.Object
 └ javax.media.j3d.WakeupCondition
    └ javax.media.j3d.WakeupCriterion
       ├ javax.media.j3d.WakeupOnActivation
       ├ javax.media.j3d.WakeupOnAWTEvent
       ├ javax.media.j3d.WakeupOnBehaviorPost
       ├ javax.media.j3d.WakeupOnCollisionEntry
       ├ javax.media.j3d.WakeupOnCollisionExit
       ├ javax.media.j3d.WakeupOnDeactivation
       ├ javax.media.j3d.WakeupOnElapsedFrames
       ├ javax.media.j3d.WakeupOnElapsedTime
       ├ javax.media.j3d.WakeupOnSensorEntry
       ├ javax.media.j3d.WakeupOnSensorExit
       ├ javax.media.j3d.WakeupOnTransformChange
       ├ javax.media.j3d.WakeupOnViewPlatformEntry
       └ javax.media.j3d.WakeupOnViewPlatformExit
```

- Even when scheduled, a behavior runs only when *wakeup criterion*s are met
  - A specified number of frames/time interval has elapsed
  - A specified AWT event occurs
  - The center of a `ViewPlatform` or a specified Sensor enters/exits a specified region
  - A behavior is activated/deactivated
  - A specified `TransformGroup` node's transform changes
  - A specified `Shape3D` node's Geometry object collides or no longer collides with any other object
  - Movement occurs between a specified `Shape3D` node's Geometry object and any other object with which it collides

  - A specified Behavior object posts a specific event
- Multiple criteria can be AND/ORed to form *wakeup conditions*

13. `WakeupCondition` extends `Object` and provides several subclasses to group wakeup criterion



Class Hierarchy

```
java.lang.Object
 └─ javax.media.j3d.WakeupCondition
     ├─ javax.media.j3d.WakeupAnd
     ├─ javax.media.j3d.WakeupAndOfOrs
     ├─ javax.media.j3d.WakeupOr
     └─ javax.media.j3d.WakeupOrOfAnds
```

14. WakeupCondition example code

- Create AWT event wakeup criterion

```
WakeupCriterion[] onMouseEvents =
   new WakeupCriterion[2];
onMouseEvents[0] =
   new WakeupOnAWTEvent( MouseEvent.MOUSE_PRESSED );
onMouseEvents[1] =
   new WakeupOnAWTEvent( MouseEvent.MOUSE_RELEASED );
```

- Combine together those criterion

```
WakeupCondition onMouse =
   new WakeupOr( onMouseEvents );
```

- Create the behavior

```
Behavior myBeh = new MyBehavior( );
```

- And set the behaviors wakup conditions and scheduling bounds

```
BoundingSphere myBounds = new BoundingSphere(
   new Point3d( ), 1000.0 );
myBeh.setSchedulingBounds( myBounds );
```