```java
/*
 * MBall.java
 *
 * Use the Java 3D API to write a simple video game -
 * Very primitive version of Monkey Ball - "roll"
 * around moving platforms.
 *
 * The philosophy of the program is to simply use
 * Java3D to render the scene and do all of the game
 * logic "manually".
 *
 * Use the arrow keys to roll a ball around various
 * platforms and collect points.
 *
 * Rewritten to have a single behavior to compute next
 * frame.
 * Include score display and change view button
 *
 * written by mike slattery - nov 2003
 * ( mikes (a) mscs.mu.edu )
 */

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.util.Enumeration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;

class Slab extends Box
{
        static final Color3f black = new Color3f(0.0f, 0.0f, 0.0f);
        static final Color3f white = new Color3f(1.0f, 1.0f, 1.0f);

        Slab(float xw, float yw, float zw)
        {
           // Construct a slab object with width xw, height yw,
           // and depth zw.
           //
           super(xw/2.0f, yw/2.0f, zw/2.0f, null);
           Appearance app = new Appearance();
           Color3f color = new Color3f(0.0f, 1.0f, 0.0f);
           app.setMaterial(new Material(color, black, color, white, 80.0f));
           setAppearance(app);
        }

        Slab getNode()
        {
                // the Slab is its own top-level node
                return this;
        }
}

class MoveableSlab
{
        // I find myself leaning toward placing this slab in
        // a given location (rather than using addAt() as I
        // have been doing
        //
        // For now, all slabs will move back and forth.  One
        // could imagine using different types of Alphas.
        //
        int i;
        Vector3f base;
        Vector3f move;
        Vector3f pos;
        Alpha timer;
        TransformGroup tg;
        Transform3D t3d;
        Slab s;
        Rectangle2D.Float bounds;
        Target targets[];
        Ball b = null; // Occasional reference to the ball
        MBall app;

        MoveableSlab(float xw, float yw, float zw, Vector3f b, Vector3f m, long t, MBa
ll mb)
        {
                // Create a slab with x-width xw, y-height yw, and z-depth zw starting
                // at position b and moving along vector m.  It takes t milliseconds
                // to move each direction (with a pause at each end).
                s = new Slab(xw,yw,zw);
                // Set the bounding rectangle slightly larger than the slab
                bounds = new Rectangle2D.Float(-xw/2.0f-0.1f, -zw/2.0f-0.1f, xw+0.2f,
zw+0.2f);
                base = b;
                move = m;
                app = mb;
                pos = new Vector3f(b);
                // Create the timer to move the slab back and forth
                timer = new Alpha(-1, Alpha.INCREASING_ENABLE | Alpha.DECREASING_ENABL
E,
                        0l, 0l, t, 0l, t/4, t, 0l, t/4);
                // Set up an initial transform for this object
                t3d = new Transform3D();
                t3d.set(base);
                tg = new TransformGroup(t3d);
                tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
                tg.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);
                tg.addChild(s);
        // Then setup targets
        float xoff = xw/4.0f;
        float zoff = zw/4.0f;
        targets = new Target[4];
        targets[0] = new Target(-xoff, -zoff, app);
        targets[1] = new Target(-xoff, zoff, app);
        targets[2] = new Target(xoff, -zoff, app);
        targets[3] = new Target(xoff, zoff, app);
        for (i=0; i<targets.length; i++)
                        tg.addChild(targets[i].getNode());
        }

        TransformGroup getNode()
        {
                return tg;
        }

        void updateSlab()
        {
                // Compute the current position of the slab and update
```

```
		// the transform group
		float tick = timer.value();
		pos.scaleAdd(tick, move, base);
		t3d.set(pos);
		tg.setTransform(t3d);
	}

	void attachBall(Ball b1)
	{
		// Record that the ball is on this slab
		b = b1;
	}

	void releaseBall()
	{
		// The ball has left this slab
		b = null;
	}

	void checkTargets(Rectangle2D r)
	{
		for (i = 0; i < targets.length; i++)
		{
			if (targets[i].active && r.intersects(targets[i].bounds))
				targets[i].hit();
		}
	}

}

class Target
{
	boolean active = true;
	Rectangle2D.Float bounds;
	BranchGroup bg;
	MBall app;

	Target(float x, float z, MBall mb)
	{
		Vector3f pos;
		TransformGroup tg;
		Transform3D t3d;

		app = mb;
		// Set up a target at location x,z relative to
		// the appropriate slab (we just work rel to origin
		// here - we never know which slab we're on).
		pos = new Vector3f(x, 0.8f, z);
		t3d = new Transform3D();
		t3d.set(pos);
		tg = new TransformGroup(t3d);

		// The purpose of this branch is simply to
		// be able to leave the scene graph when hit.
		bg = new BranchGroup();
		bg.setCapability(BranchGroup.ALLOW_DETACH);
		bg.addChild(tg);

		bounds = new Rectangle2D.Float(x-0.2f, z-0.2f, 0.4f, 0.4f);

		// Set up a TransformGroup for the interpolator to spin
		TransformGroup objSpin = new TransformGroup();
		objSpin.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
		tg.addChild(objSpin);

		// A ColorCube will be the visible target
		objSpin.addChild(new ColorCube(0.2));

		// Create a new Behavior object that will
		// spin the cube and add it into
		// the scene graph.
		Transform3D yAxis = new Transform3D();
		Alpha rotationAlpha = new Alpha(-1, 2000);

		RotationInterpolator rotator =
			new RotationInterpolator(rotationAlpha, objSpin);

		// a bounding sphere specifies a region a behavior is active
		// create a sphere centered at the origin with radius of 1
		BoundingSphere bounds = new BoundingSphere();
		rotator.setSchedulingBounds(bounds);
		objSpin.addChild(rotator);
	}

	void hit()
	{
		// This method gets called when the ball
		// runs into us.  Remove the target and
		// add points to the score.
		active = false;
		//System.out.println("Hit");
		bg.detach();
		app.addScore(100);
	}

	BranchGroup getNode()
	{
		return bg;
	}
}

class Ball
{
	int i;
	Vector3f rel_pos;
	Vector3f glob_pos;
	float rotation;
	Vector3f vel_prime;
	Vector3f velocity;
	Vector3f fall;
	Point2D.Float rel_pt;
	TransformGroup tg;
	Transform3D t3d;
	Transform3D rot3d;
	boolean keygo = false;
	boolean keyleft = false;
	boolean keyright = false;

	boolean goflag;
	boolean dead = false;
	MoveableSlab slabs[];
	MoveableSlab current_slab = null;
	MBall app;
```

```
	Rectangle2D.Float bounds;
	float coords[] = new float[3];

	final float twopi = 2.0f*(float)Math.PI;

	Ball(Vector3f p, float ang, float s, MoveableSlab sarray[], MBall mb)
	{
		// Create a ball at global position p.
		// ang measures radians (counterclockwise when viewed
		// from above) off of negative z axis.
		// s is the speed.
		// sarray[] is an array of all the moving slabs in
		// the game world.
		glob_pos = p;
		rotation = ang;
		vel_prime = new Vector3f(0.0f, 0.0f, -s);
		fall = new Vector3f(0.0f, -1.5f*s, 0.0f);
		slabs = sarray;
		app = mb;
		velocity = new Vector3f();
		rot3d = new Transform3D();
		rot3d.rotY(rotation);
		rot3d.transform(vel_prime, velocity);
		rel_pos = new Vector3f();
		rel_pt = new Point2D.Float();
		bounds = new Rectangle2D.Float();
		t3d = new Transform3D();
		t3d.set(glob_pos);
		tg = new TransformGroup(t3d);
		tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
		// Set up the actual geometric ball (Sphere)
		Appearance appear = new Appearance();
		Material material = new Material();
		material.setShininess(50.0f);
		material.setDiffuseColor(0.0f, 0.0f, 1.0f);
		appear.setMaterial(material);
		Sphere sph = new Sphere(0.4f, appear);
	tg.addChild(sph);
	}

	void updateBall(float dt)
	{
		// dt is seconds elapsed time since previous frame

		if (dead)
			return;

		if (current_slab != null)
		{
			goflag = false;
			if (keygo)
			{
				rel_pos.scaleAdd(dt, velocity, rel_pos);
				goflag = true;
			}
			if (keyleft)
			{
				rotation += dt*0.75f;
				if (rotation > twopi);
					rotation -= twopi;
				rot3d.rotY(rotation);
				rot3d.transform(vel_prime, velocity);
				tg.setTransform(rot3d);
			}
			if (keyright)
			{
				rotation -= dt*0.75f;
				if (rotation < 0.0f);
					rotation += twopi;
				rot3d.rotY(rotation);
				rot3d.transform(vel_prime, velocity);
				tg.setTransform(rot3d);
			}
			// Now recompute the global position of
			// the ball
			computeGlobal();

			// Check to see if we've fallen off the
			// slab or hit any targets on
			// this slab. We only need to check this
			// if we've actually moved relative to
			// the slab.
			if (goflag)
			{
				rel_pt.setLocation(rel_pos.x, rel_pos.z);
				if (!(current_slab.bounds).contains(rel_pt))
				{
					releaseSlab();
				}
				else
				{
					// Check for targets
					bounds.setRect(rel_pos.x-0.35f,rel_pos.z-0.35f
,0.7f,0.7f);
					current_slab.checkTargets(bounds);
				}
			}
		}
		else
		{
			// If there's no slab under us, we check all the slabs to
			// see if we're about to drop onto one.  If so, we attach
			// to it.  If not, we fall.
			for (i=0; i<slabs.length; i++)
			{
				rel_pos.sub(glob_pos, slabs[i].pos);
				rel_pt.setLocation(rel_pos.x, rel_pos.z);
				if ((slabs[i].bounds).contains(rel_pt))
				{
					if ((rel_pos.y>0.4f) && (rel_pos.y<1.1f))
					{
						setSlab(slabs[i]);
						break;
					}
				}
			}
			if (current_slab == null)
			{
				//Fall a bit
				glob_pos.scaleAdd(dt, fall, glob_pos);
				t3d.set(glob_pos);
				tg.setTransform(t3d);
				// Check for ground
				if (glob_pos.y < 0.0f)
```

```
                                {
                                        dead = true;
                                        //System.out.println("You died");
                                        app.display("You died");
                                }
                                app.checkViewpoint();
                        }
                }
        }

        void computeGlobal()
        {
                // Use the current position of the slab we're on
                // and our relative position to that slab to compute
                // our current global position (in world coordinates).
                if (current_slab == null)
                        throw new Error("computeGlobal called with no slab");
                glob_pos.add(current_slab.pos, rel_pos);
                t3d.set(glob_pos);
                tg.setTransform(t3d);
                app.checkViewpoint();
        }

        void setSlab(MoveableSlab s)
        {
                // glob_pos should be up-to-date when calling this
                // method.  We compute rel_pos relative to the new slab.
                current_slab = s;
                s.attachBall(this);
                rel_pos.sub(glob_pos, current_slab.pos);
                // Force the ball to sit on the surface of the slab
                // (relies on fixed slab thickness and ball radius)
                rel_pos.get(coords);
                coords[1] = 0.9f;
                rel_pos.set(coords);
        }

        void releaseSlab()
        {
                //System.out.println("Slab released");
                current_slab.releaseBall();
                current_slab = null;
        }

        MoveableSlab getSlab()
        {
                return current_slab;
        }

        TransformGroup getNode()
        {
                return tg;
        }
}



public class MBall extends Applet {

     SimpleUniverse simpleU=null;
     int score = 0;
     Label disp;  // display score
     TransformGroup push;
     Transform3D overT3D;
     Transform3D followT3D;
     Button view_button;  // change view mode
     Ball ball;
     MoveableSlab[] slabs;
     Canvas3D canvas3D;

    static final int NEW_OVERLOOK = 1;
    static final int OVERLOOK = 2;
        static final int FOLLOW = 3;
    int view = OVERLOOK;

     public static void addAt(Group root, Node n, Vector3f loc)
     {
       // This is a general utility method for doing translations
       // It's here because I didn't know where else to put it.
       //
       // Add the Node n to the Group root using a translation
       // to place the new group at the loccation specified by loc
       Transform3D t = new Transform3D();
       t.setTranslation(loc);
       TransformGroup TG = new TransformGroup(t);
       TG.addChild(n);
       root.addChild(TG);
     }

     public void addScore(int pts)
     {
        score += pts;
        //System.out.println("Score = "+score);
        disp.setText("Score = "+score);
        if (score == 1200)
                display("You win!");
                //System.out.println("You win!");
     }

        class ComputeFrame extends Behavior {

                // This behavior updates the world
                // for each frame.
                        // Keep frame times
                    long prevTime, currTime;
                float t;
                WakeupOnElapsedFrames stim = new WakeupOnElapsedFrames(0);

                public void initialize()
                {
                        prevTime = 0L;
                        wakeupOn(stim);
                }

                public void processStimulus(Enumeration criteria)
                  {
                    for (int i = 0; i < slabs.length; i++)
                                slabs[i].updateSlab();
                        // Each frame we figure out how much time has
                        // passed and call updateBall().
                        // Get elapsed time
                        currTime = getView().getCurrentFrameStartTime();
                        t = (currTime-prevTime)/1000.0f;
                        prevTime = currTime;
```

```
                ball.updateBall(t);
                wakeupOn(stim);
             }

    }

    class keyL extends KeyAdapter {
            public void keyPressed(KeyEvent e) {
                    int key = e.getKeyCode();
                    switch (key) {
                      case KeyEvent.VK_UP:
                      case KeyEvent.VK_F: ball.keygo = true; break;
                      case KeyEvent.VK_RIGHT:
                      case KeyEvent.VK_G: ball.keyright = true; break;
                      case KeyEvent.VK_LEFT:
                      case KeyEvent.VK_D: ball.keyleft = true; break;
                    }
            }

            public void keyReleased(KeyEvent e) {
                    int key = e.getKeyCode();
                    switch (key) {
                      case KeyEvent.VK_UP:
                      case KeyEvent.VK_F: ball.keygo = false; break;
                      case KeyEvent.VK_RIGHT:
                      case KeyEvent.VK_G: ball.keyright = false; break;
                      case KeyEvent.VK_LEFT:
                      case KeyEvent.VK_D: ball.keyleft = false; break;
                    }
            }
    }

    class actL implements ActionListener {
      public void actionPerformed(ActionEvent e){
            if (view == FOLLOW)
            {
                    view = NEW_OVERLOOK;
                    view_button.setLabel("Follow view");
                    canvas3D.requestFocus();
            }
            else
            {
                    view = FOLLOW;
                    view_button.setLabel("Over view");
                    canvas3D.requestFocus();
            }
      }
    }

    void display(String s) {
      // Use the TransformGroup push defined below
      // to place a message in front of the player.
      BranchGroup dispGroup = new BranchGroup();
      Text2D shape = new Text2D(s, new Color3f(0.0f,0.0f,0.6f),
                    "Helvetica", 100, Font.BOLD);
      dispGroup.addChild(shape);
      push.addChild(dispGroup);
    }

  public BranchGroup createSceneGraph() {
        // Create the root of the branch graph
        BranchGroup objRoot = new BranchGroup();

        // Set up a white directional light shining
        // down on the scene.
        DirectionalLight lightD = new DirectionalLight();
        BoundingSphere bs = new BoundingSphere(new Point3d(0.0,0.0,0.0),100.0);
        lightD.setInfluencingBounds(bs);
        Vector3f direction = new Vector3f(-1.0f, -1.0f, -0.5f);
        direction.normalize();
        lightD.setDirection(direction);
        lightD.setColor(new Color3f(1.0f, 1.0f, 1.0f));
        objRoot.addChild(lightD);

        // Set the background (essentially the "sky" in
        // the game) to yellow
        //
        Background bg = new Background();
        bg.setColor(1.0f, 1.0f, 0.0f); // yellow
        bg.setApplicationBounds(new BoundingSphere());
        objRoot.addChild(bg);

        // Build a big flat box for the land.
        Color3f black = new Color3f(0.0f, 0.0f, 0.0f);
        Color3f white = new Color3f(1.0f, 1.0f, 1.0f);
        Appearance appL = new Appearance();
        Color3f colorL = new Color3f(0.8f, 0.0f, 0.0f);
        appL.setMaterial(new Material(colorL, black, colorL, white, 80.0f));
        Box land = new Box(30.0f, 0.5f, 30.0f, appL);
        addAt(objRoot, land, new Vector3f(0.0f, -0.5f, 0.0f));

        // Put some objects into the scene - first 1 pillar
        addAt(objRoot, (new Slab(1.0f,12.0f,1.0f)).getNode(), new Vector3f(-6.5f, 6.0f
, 0.0f));
        // then three moving slabs
        slabs = new MoveableSlab[3];
                slabs[0] = new MoveableSlab(8.0f,1.0f,10.0f, new Vector3f(-2.0f, 6.0f,
 0.0f),
                        new Vector3f(4.0f, 0.0f, 0.0f), 2000l, this);
                objRoot.addChild(slabs[0].getNode());
                slabs[1] = new MoveableSlab(8.0f,1.0f,10.0f, new Vector3f(10.0f, 6.0f,
 0.0f),
                        new Vector3f(4.0f, 0.0f, 0.0f), 3000l, this);
                objRoot.addChild(slabs[1].getNode());
                slabs[2] = new MoveableSlab(8.0f,1.0f,10.0f, new Vector3f(6.0f, 2.5f,
-2.0f),
                        new Vector3f(0.0f, 0.0f, 4.0f), 2500l, this);
                objRoot.addChild(slabs[2].getNode());

                // Construct a ball and put it on one of the slabs
                ball = new Ball(new Vector3f(0.0f,6.7f,4.0f),0.0f,1.0f,slabs,this);
                ball.setSlab(slabs[0]);
                objRoot.addChild(ball.getNode());

                // This will move the ViewPlatform back so the
                // objects in the scene can be viewed.
                // We keep that transform in overT3D to use
                // when resetting view.
        ViewingPlatform vp = simpleU.getViewingPlatform();
                TransformGroup vpTrans = vp.getViewPlatformTransform();
                Vector3f translate = new Vector3f(0.0f, 10.0f, 25.0f);
        overT3D = new Transform3D();
        overT3D.rotY(-Math.PI/24.0);
        Transform3D tilt = new Transform3D();
```

```
        tilt.rotX(-Math.PI/15.0);
        overT3D.mul(tilt);
        overT3D.setTranslation(translate);
        vpTrans.setTransform(overT3D);

        // We'll set up a TransformGroup (push)
        // connected to the view platform, where
        // we can display short messages to the
        // user.  Currently, you can only use
        // the display method to do this once.
        // If repeated calls were needed, you'd
        // need to remove existing children before
        // adding the new message (in display()).
                PlatformGeometry pg = new PlatformGeometry();
                Transform3D pushT3D = new Transform3D();
                pushT3D.set(new Vector3f(-0.7f, 0.0f, -3.0f));
                push = new TransformGroup(pushT3D);
                pg.addChild(push);
                push.setCapability(Group.ALLOW_CHILDREN_EXTEND);
        vp.setPlatformGeometry(pg);

       // Create a new Behavior object to update each frame
                ComputeFrame cf = new ComputeFrame();
                BoundingSphere bounds = new BoundingSphere(new Point3d(),100.0);
                cf.setSchedulingBounds(bounds);
        objRoot.addChild(cf);

                return objRoot;
    } // end of createSceneGraph method of MBall

    void checkViewpoint()
        {
                // If the view is currently OVERLOOK, we don't
                // need to do anything.  Otherwise, update
                // view platform appropriately.
                if (view == NEW_OVERLOOK)
                {
                      TransformGroup vpTrans = simpleU.getViewingPlatform().getViewPla
tformTransform();
                      vpTrans.setTransform(overT3D);
                      view = OVERLOOK;
                }
                else if (view == FOLLOW)
                {
                      TransformGroup vpTrans = simpleU.getViewingPlatform().getViewPla
tformTransform();
                      Transform3D T3D =new Transform3D(ball.t3d);
                      T3D.mul(ball.rot3d);
                      T3D.mul(followT3D);
                      vpTrans.setTransform(T3D);
                }
        }

    public MBall() {
        setLayout(new BorderLayout());
        canvas3D = new Canvas3D(SimpleUniverse.getPreferredConfiguration());
        add("Center", canvas3D);
        Panel p = new Panel();
        disp = new Label("Score = 0    ");
        p.add(disp);
        view_button = new Button("Follow view");
        view_button.addActionListener(new actL());
        p.add(view_button);
        add("North", p);

        canvas3D.addKeyListener(new keyL());

        // SimpleUniverse is a Convenience Utility class
        simpleU = new SimpleUniverse(canvas3D);

        BranchGroup scene = createSceneGraph();
                scene.compile();

        simpleU.addBranchGraph(scene);

        // setup the transform followT3D to place
        // the viewpoint "over the shoulder" of
        // the ball
        Vector3f backup = new Vector3f(0.0f, 0.0f, 5.0f);
                followT3D = new Transform3D();
                followT3D.rotX(-Math.PI/12.0);
                Transform3D scoot = new Transform3D();
                scoot.set(backup);
                followT3D.mul(scoot);
    }

    //  The following allows this to be run as an application
    //  as well as an applet

    public static void main(String[] args) {
        Frame frame = new MainFrame(new MBall(), 256, 256);
    } // end of main (method of MBall)

} // end of class MBall
```