```
import org.opensourcephysics.numerics.*;

public class TorqueApp extends AbstractSimulation {
   Display3DFrame frame = new Display3DFrame("Rotation test");
   Element body = new ElementBox();          // shows rigid body
   Element shaft = new ElementCylinder();    // shows shaft
   // shows angular velocity of shaft
   Element arrowOmega = new ElementArrow();
   Element arrowL = new ElementArrow(); // shows angular momentum
   Element arrowTorque = new ElementArrow(); // shows torque on shaft
   Group shaftGroup = new Group(); // contains shaft and arrowOmega
   // contains body, arrowL, and arrowTorque
   Group bodyGroup = new Group();
   double theta = 0, omega = 0.1, dt = 0.1;
   // principal moments of inertia
   double Ixx = 1.0, Iyy = 1.0, Izz = 2.0;

   public TorqueApp() {
      frame.setDecorationType(VisualizationHints.DECORATION_AXES);
      body.setSizeXYZ(1.0, 1.0, 0.1); // thin rectangle
      shaft.setSizeXYZ(0.1, 0.1, 0.8);
      arrowL.getStyle().setLineColor(java.awt.Color.MAGENTA);
      arrowTorque.getStyle().setLineColor(java.awt.Color.CYAN);
      bodyGroup.addElement(body);
      bodyGroup.addElement(arrowTorque);
      bodyGroup.addElement(arrowL);
      shaftGroup.addElement(bodyGroup);
      shaftGroup.addElement(arrowOmega);
      shaftGroup.addElement(shaft);
      frame.addElement(shaftGroup);
   }

   void computeVectors() {
      // convert omega to body frame
      double[] omega =
          body.toBodyFrame(new double[] {0, 0, this.omega});
      // L in body frame
      double[] angularMomentum =
           new double[] {omega[0]*Ixx, omega[1]*Iyy, omega[2]*Izz};
      // torque is computed in body frame
      double[] torque = VectorMath.cross3D(omega, angularMomentum);
      arrowL.setSizeXYZ(angularMomentum);
      arrowTorque.setSizeXYZ(torque);
      // position torque arrow at tip of angular momentum
      arrowTorque.setXYZ(angularMomentum);
   }

   public void initialize() {
      omega = control.getDouble("omega");
      arrowOmega.setSizeXYZ(0, 0, omega);
      double tilt = control.getDouble("tilt");
      double cos = Math.cos(tilt/2), sin = Math.sin(tilt/2);
      Transformation rotation = new Quaternion(cos, sin, 0, 0);
      bodyGroup.setTransformation(rotation);
      computeVectors();
   }
```



```
   public void reset() {
      control.setValue("omega", "pi/4");
      control.setValue("tilt", "pi/5");
   }

   protected void doStep() {
      theta += omega*dt;
      double cos = Math.cos(theta/2), sin = Math.sin(theta/2);
      shaftGroup.setTransformation(new Quaternion(cos, 0, 0, sin));
      computeVectors();
   }

   public static void main(String[] args) {
      SimulationControl.createApp(new TorqueApp());
   }
}
```

`TorqueApp` instantiates a shaft group and a body group in its constructor. The shaft group contains visual representations of the shaft, the angular velocity vector, and the body group. The body group contains representations of a thin rectangular sheet, the angular velocity, and the torque. The body group is rotated about the x-axis in the `initialize` method and the shaft group is rotated about the z-axis in the `doStep` method. Because the body group is within the shaft group, the body group also rotates. The `computeVectors` method displays the angular momentum and torque vectors by computing their values in the rotating (body) frame of reference. The torque is computed using a cross produce of the body frame angular velocity and the body frame angular momentum according to (17.21).

A complete description of a rigid body requires three angular orientation variables as well as three angular velocity variables. The orientation is often given in terms of the Euler angles ψ, θ, and ϕ. These angles specify the orientation as a series of three independent rotations about prechosen axes and must be applied in exactly the order given because the rotation matrices do not commute. To use these angles as differential equation state variables, we must be able to calculate their rate as a function of the body frame angular velocity in (17.23). The expression for the angular velocity in the body frame in terms of the Euler angles is (see Goldstein)

$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\end{bmatrix} = \begin{bmatrix}\sin\theta\sin\psi & \cos\psi & 0\\ \sin\theta\cos\psi & -\sin\psi & 0\\ \cos\theta & 0 & 1\end{bmatrix}\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}. \tag{17.24}$$

Unfortunately, the matrix in (17.24) is singular when $\sin\theta = 0$.

Because we must invert this matrix to solve for the rate of the Euler angles, the Euler angle rate equation becomes unstable when θ approaches 0 or π. A better approach for numerical work is to abandon Euler angles and to use *quaternions*.