```java
package info.openrocket.core.simulation.listeners;

import info.openrocket.core.aerodynamics.AerodynamicForces;
import info.openrocket.core.aerodynamics.FlightConditions;
import info.openrocket.core.masscalc.RigidBody;
import info.openrocket.core.models.atmosphere.AtmosphericConditions;
import info.openrocket.core.motor.MotorConfigurationId;
import info.openrocket.core.rocketcomponent.MotorMount;
import info.openrocket.core.rocketcomponent.RecoveryDevice;
import info.openrocket.core.simulation.AccelerationData;
import info.openrocket.core.simulation.FlightEvent;
import info.openrocket.core.simulation.MotorClusterState;
import info.openrocket.core.simulation.SimulationStatus;
import info.openrocket.core.simulation.exception.SimulationException;
import info.openrocket.core.util.BugException;
import info.openrocket.core.util.Coordinate;

/**
 * An abstract base class for implementing simulation listeners. This class
 * implements all
 * of the simulation listener interfaces using methods that have no effect on
 * the simulation.
 * The recommended way of implementing simulation listeners is to extend this
 * class.
 *
 * @author Sampo Niskanen <sampo.niskanen@iki.fi>
 */
public class AbstractSimulationListener implements SimulationListener, SimulationComputationListener,
        SimulationEventListener, Cloneable {

    //// SimulationListener ////

    @Override
    public void startSimulation(SimulationStatus status) throws SimulationException {
        // No-op
    }

    @Override
    public void endSimulation(SimulationStatus status, SimulationException exception) {
        // No-op
    }

    @Override
    public boolean preStep(SimulationStatus status) throws SimulationException {
        return true;
    }

    @Override
    public void postStep(SimulationStatus status) throws SimulationException {
        // No-op
    }

    /**
     * {@inheritDoc}
     * <p>
     * <em>This implementation of the method always returns <code>false</code>.</em>
     */
    @Override
    public boolean isSystemListener() {
        return false;
    }

    //// SimulationEventListener ////

    @Override
    public boolean addFlightEvent(SimulationStatus status, FlightEvent event) throws SimulationException {
        return true;
    }

    @Override
    public boolean handleFlightEvent(SimulationStatus status, FlightEvent event) throws SimulationException {
        return true;
    }

    @Override
    public boolean motorIgnition(SimulationStatus status, MotorConfigurationId motorId, MotorMount mount,
            MotorClusterState instance) throws SimulationException {
        return true;
    }

    @Override
    public boolean recoveryDeviceDeployment(SimulationStatus status, RecoveryDevice recoveryDevice)
            throws SimulationException {
        return true;
    }

    //// SimulationComputationListener ////

    @Override
    public AccelerationData preAccelerationCalculation(SimulationStatus status) throws SimulationException {
        return null;
    }

    @Override
    public AerodynamicForces preAerodynamicCalculation(SimulationStatus status) throws SimulationException {
        return null;
    }

    @Override
    public AtmosphericConditions preAtmosphericModel(SimulationStatus status) throws SimulationException {
        return null;
    }

    @Override
    public FlightConditions preFlightConditions(SimulationStatus status) throws SimulationException {
        return null;
    }

    @Override
    public double preGravityModel(SimulationStatus status) throws SimulationException {
        return Double.NaN;
    }

    @Override
    public RigidBody preMassCalculation(SimulationStatus status) throws SimulationException {
        return null;
    }

    @Override
    public double preSimpleThrustCalculation(SimulationStatus status) throws SimulationException {
        return Double.NaN;
    }

    @Override
    public Coordinate preWindModel(SimulationStatus status) throws SimulationException {
        return null;
    }

    @Override
    public AccelerationData postAccelerationCalculation(SimulationStatus status, AccelerationData acceleration)
            throws SimulationException {
        return null;
    }

    @Override
    public AerodynamicForces postAerodynamicCalculation(SimulationStatus status, AerodynamicForces forces)
            throws SimulationException {
        return null;
    }

    @Override
    public AtmosphericConditions postAtmosphericModel(SimulationStatus status,
            AtmosphericConditions atmosphericConditions) throws SimulationException {
        return null;
    }

    @Override
    public FlightConditions postFlightConditions(SimulationStatus status, FlightConditions flightConditions)
            throws SimulationException {
        return null;
    }

    @Override
    public double postGravityModel(SimulationStatus status, double gravity) throws SimulationException {
        return Double.NaN;
    }

    @Override
    public RigidBody postMassCalculation(SimulationStatus status, RigidBody RigidBody) throws SimulationException {
        return null;
    }

    @Override
    public double postSimpleThrustCalculation(SimulationStatus status, double thrust) throws SimulationException {
        return Double.NaN;
    }

    @Override
    public Coordinate postWindModel(SimulationStatus status, Coordinate wind) throws SimulationException {
        return null;
    }

    @Override
    public AbstractSimulationListener clone() {
        try {
            return (AbstractSimulationListener) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new BugException(e);
        }
    }

}
```