```
package jp.jaxa.iss.kibo.rpc.GOATS;

import android.util.Log;

import jp.jaxa.iss.kibo.rpc.api.KiboRpcService;
import java.util.concurrent.TimeUnit;

import gov.nasa.arc.astrobee.Result;
import gov.nasa.arc.astrobee.types.Point;
import gov.nasa.arc.astrobee.types.Quaternion;

import org.opencv.core.Mat;

/**
* Class meant to handle commands from the Ground Data System and execute them in Astrobee
*/

public class YourService extends KiboRpcService {

  private final String TAG = this.getClass().getSimpleName();
  @Override
  protected void runPlan1() {
      Log.i(TAG, "start mission");
      // the mission starts
      api.startMission();

      // move to a point
      Point point = new Point(10.71000f, -7.70000f, 4.48000f);
      Quaternion quaternion = new Quaternion(0f, .707f, 0f, .707f);
      Result result = api.moveTo(point, quaternion, false);

      final int LOOP_MAX = 5;

      int loopCounter = 0;
      while (!result.hasSucceeded() && loopCounter < LOOP_MAX) {
          result = api.moveTo(point, quaternion, true);
          ++loopCounter;
      }
      // report point1 arrival
      api.reportPoint1Arrival();

      point = new Point(10.71000f, -7.70000f, 4.48000f);
      quaternion = new Quaternion(-0.100f, .707f, 0f, .707f);
      result = api.moveTo(point, quaternion, false);

      // get a camera image
      Mat image = api.getMatNavCam();

      api.saveMatImage(image, "Target1.png");
```

```
// irradiate the laser
api.laserControl(true);

// take target1 snapshots
api.takeTarget1Snapshot();

// turn the laser off
api.laserControl(false);

/* ******************************************** */
/* write your own code and repair the air leak! */
/* ******************************************** */

point = new Point(11.30000f, -7.70000f, 4.50000f);
quaternion = new Quaternion(0f, 0f, -0.707f, .707f);
result = api.moveTo(point, quaternion, false);

point = new Point(11.30000f, -9.92284f, 4.50000f);
quaternion = new Quaternion(0f, 0f, -0.707f, .707f);
result = api.moveTo(point, quaternion, false);

point = new Point(11.27460f, -9.92284f, 5.29881f);
quaternion = new Quaternion(0f, 0f, -0.707f, 0.707f);
result = api.moveTo(point, quaternion, false);

image = api.getMatNavCam();

api.saveMatImage(image, "Target2.png");

//detect the tag, classify the tag,

// irradiate the laser
api.laserControl(true);

// take target2 snapshots
api.takeTarget2Snapshot();

// turn the laser off
api.laserControl(false);

point = new Point(10.61000f, -9.92284f, 5.31647f);
quaternion = new Quaternion(0f, 0f, -0.707f, 0.707f);
result = api.moveTo(point, quaternion, false);

point = new Point(10.61000f, -7.89178f, 5.31647f);
quaternion = new Quaternion(0f, 0f, -0.707f, 0.707f);
result = api.moveTo(point, quaternion, false);
```

```
        point = new Point(11.27460f, -7.89178f, 4.96538f);
        quaternion = new Quaternion(0f, 0f, -0.707f, 0.707f);
        result = api.moveTo(point, quaternion, false);

        while (!result.hasSucceeded() && loopCounter < LOOP_MAX) {
            result = api.moveTo(point, quaternion, true);
            ++loopCounter;
        }

        // send mission completion
        api.reportMissionCompletion();
    }

    @Override
    protected void runPlan2(){
        // write here your plan 2
    }

    @Override
    protected void runPlan3(){
        // write here your plan 3
    }

    // You can add your method
    private void moveToWrapper(double pos_x, double pos_y, double pos_z,
                               double qua_x, double qua_y, double qua_z,
                               double qua_w){

        final Point point = new Point(pos_x, pos_y, pos_z);
        final Quaternion quaternion = new Quaternion((float)qua_x, (float)qua_y,
                                                     (float)qua_z, (float)qua_w);

        api.moveTo(point, quaternion, true);
    }

    private void relativeMoveToWrapper(double pos_x, double pos_y, double pos_z,
                               double qua_x, double qua_y, double qua_z,
                               double qua_w) {

        final Point point = new Point(pos_x, pos_y, pos_z);
        final Quaternion quaternion = new Quaternion((float) qua_x, (float) qua_y,
                (float) qua_z, (float) qua_w);

        api.relativeMoveTo(point, quaternion, true);
    }

}
```