```
package player;

import edu.neu.ccs.demeterf.demfgen.lib.Entry;
import edu.neu.ccs.demeterf.demfgen.lib.List;
import edu.neu.ccs.demeterf.demfgen.lib.Map;
import edu.neu.ccs.demeterf.demfgen.lib.List.Fold;
import edu.neu.ccs.evergreen.ir.Relation;
import gen.*;
import player.playeragent.*;

/** The IAmRobot Player for SDG... */
public class IAmRobot implements PlayerI{
        private BuyAgent buyAgent = new BuyAgent();
    private CreateAgent createAgent = new CreateAgent();
    private DeliverAgent deliverAgent = new DeliverAgent();
    private FinishAgent finishAgent= new FinishAgent();

    /** Invoked (externally) by the IQ evaluator */
        public IAmRobot(){}

    /** Run (externally) by the administrator when it's 'my' turn */
    public static void main(String[] args){
        new PlayerRunner(args[0],new IAmRobot()).main();
    }

    /** Returns the break-even price for the given Derivative */
    public static double getBreakEven(Derivative d) {
        for (TypeInstance typeInst : d.type.instances)
        {
                // Eventually there will be multiple relations, but for now, just
                // return the break even value for the first one
                return getBreakEven(typeInst.r);
        }

        return 1;       // Shouldn't happen
    }
    /** Gets the break-even price for the given relation */
    static double getBreakEven(RelationNr relation) {
        int relationNum = relation.v;
                if (relationNum == 0)
                        return 0;

                SimplePolynomial expectedSatRatioFunc = getPolynomial(relationNum);
                List<Entry<Double, Double>> points = getCriticalPoints(expectedSatRatioFunc);

                Double breakEven = points.foldl(new Fold<Entry<Double, Double>, Double>() {
                        public Double fold(Entry<Double, Double> pair, Double max) {
                                return Math.max(pair.val, max);
                        }
                }, 0.0);

                return breakEven;
    }

    /** Gets the value of b that maximizes the fraction of satisfied constraints within the given deriv
ative */
    public static double getBMax(Derivative d) {
        SimplePolynomial expectedSatRatioFunc = getPolynomial(d.type.instances.top().r.v);
        List<Entry<Double, Double>> points = getCriticalPoints(expectedSatRatioFunc);

        Entry<Double, Double> maxPoint = points.foldl(new Fold<Entry<Double, Double>, Entry<Double, Dou
ble>>() {
                        public Entry<Double, Double> fold(Entry<Double, Double> pair, Entry<Double, Dou
ble> max) {
                                return pair.val > max.val ? pair : max;
                        }
                }, new Entry<Double, Double>(0.0, 0.0));

                return maxPoint.key;
    }

    /** Gets the potential maximums of the polynomial */
    static List<Entry<Double, Double>> getCriticalPoints(SimplePolynomial expectedSatRatioFunc) {
        Map<Double, Double> points = Map.create();

        points = points.put(0.0, expectedSatRatioFunc.solve(0));
        points = points.put(1.0, expectedSatRatioFunc.solve(1));

                double[] bmax = expectedSatRatioFunc.getDerivative().getZeros();
                if (bmax.length == 1)
                {
```

```
                        if (bmax[0] > 0 && bmax[0] < 1)
                                points = points.put(bmax[0], expectedSatRatioFunc.solve(bmax[0]));
                }
                else if (bmax.length == 2)
                {
                        if (bmax[0] > 0 && bmax[0] < 1)
                                points = points.put(bmax[0], expectedSatRatioFunc.solve(bmax[0]));
                        if (bmax[1] > 0 && bmax[1] < 1)
                                points = points.put(bmax[1], expectedSatRatioFunc.solve(bmax[1]));
                }
                return points.toList();
    }

    /** Gets the probability polynomial for the given relation number */
    static SimplePolynomial getPolynomial(int relationNumber) {
        Relation rel = new Relation(3, relationNumber);
        int q0 = rel.q(0);
        int q1 = rel.q(1);
        int q2 = rel.q(2);
        int q3 = rel.q(3);
        return new SimplePolynomial(
                        q3 - q2 + q1 - q0,
                        q2 - (2 * q1) + (3 * q0),
                        q1 - (3 * q0),
                        q0);
    }


    public String getName(){ return "IAmRobot3"; }

        public BuyAgentI getBuyAgent(){ return buyAgent; }
        public CreateAgentI getCreateAgent(){ return createAgent;       }
        public DeliverAgentI getDeliverAgent(){ return deliverAgent;    }
        public FinishAgentI getFinishAgent(){ return finishAgent; }
}
```