```
/* *********************************
 *   FinishAgent.java
 *     Finish a given Raw Material
 * *********************************/
package player.playeragent;

import java.util.Iterator;

import player.*;
import edu.neu.ccs.demeterf.demfgen.lib.*;
import gen.*;
import util.BreakEven;
import java.util.Set;

/** Class for finishing a list of derivatives */
public class FinishAgent implements PlayerI.FinishAgentI{

    /** Calculate the finished product for a given Derivative */
    public FinishedProduct finishDerivative(Derivative d){
        if(d.optraw.isSome()){
                RawMaterial raw = d.optraw.inner();
                List<RelationNr> relNrs = Util.getRelationNumbers(raw);
                Set<Variable> vars = Util.getVars(raw);
                double bias = 0.0;
                try{
                        bias = BreakEven.getBreakEvenBias(relNrs);
                }catch(Exception e){
                        return FinishAgent.failure(vars, d.optraw.inner().instance.cs);
                }
                Iterator<Variable> viter = vars.iterator();
                List<Literal> lits = List.create();
                RawMaterial shannonRaw = new RawMaterial(raw.instance);
                while(viter.hasNext()){
                        Variable tempVar = viter.next();
                        RawMaterial rawPos = Util.getShannon(shannonRaw, new Literal(new Pos(), tempVar
));
                        RawMaterial rawNeg = Util.getShannon(shannonRaw, new Literal(new Neg(), tempVar
));
                        double polyPos = util.BreakEven.getRawMaterialLookAhead(rawPos).eval(bias);
                        double polyNeg = util.BreakEven.getRawMaterialLookAhead(rawNeg).eval(bias);
                        if(polyNeg > polyPos){
                                shannonRaw = rawNeg;
                                lits = lits.append(new Literal(new Neg(), tempVar));
                        }else{
                                shannonRaw = rawPos;
                                lits = lits.append(new Literal(new Pos(), tempVar));
                        }
                }
                return new FinishedProduct(new IntermediateProduct(new Assignment(lits)), new Quality(F
inishAgent.evaluateQuality(d.optraw.inner().instance.cs, lits)));
        }else{
                return FinishAgent.failure();
        }
    }

    private static FinishedProduct failure(Set<Variable> vars, List<Constraint> consts){
        List<Literal> lits = List.create();
        Iterator<Variable> viter = vars.iterator();
        while(viter.hasNext()){
                lits = lits.append(new Literal(Util.coinFlip() ? new Pos() : new Neg(), viter.next()));
        }
                return new FinishedProduct(new IntermediateProduct(new Assignment(lits)),
                                new Quality(FinishAgent.evaluateQuality(consts, lits)));
    }

    private static FinishedProduct failure(){
        List<Literal> lits = List.create();
                return new FinishedProduct(new IntermediateProduct(new Assignment(lits)),
                                new Quality(Math.random()));
    }

        private static double evaluateQuality(List<Constraint> constraints, List<Literal> lits){
                double total = constraints.length();
                double satisfied = 0;
                for(int i=0; i < ((int)total); i++){
                        if(FinishAgent.isSatisfied(constraints.lookup(i).r.v,
                                        FinishAgent.getSign(lits, constraints.lookup(i).vs.lookup(0)),
                                        FinishAgent.getSign(lits, constraints.lookup(i).vs.lookup(1)),
                                        FinishAgent.getSign(lits, constraints.lookup(i).vs.lookup(2))))
                                satisfied += 1.0;
                }
                return satisfied/total;
        }

        private static Sign getSign(List<Literal> lits, Variable v){
                int l = lits.length();
                for(int i=0; i<l; i++){
                        if(lits.lookup(i).var.equals(v))
                                return lits.lookup(i).value;
                }
                return null;
        }

        public static boolean isSatisfied(int relation, Sign x, Sign y, Sign z){
                if(relation > 255)
                        return false;
                if(relation >= 128){
                        relation -= 128;
                        if(x.equals(new Pos()) && y.equals(new Pos()) && z.equals(new Pos()))
                                return true;
                }
                if(relation >= 64){
                        relation -= 64;
                        if(x.equals(new Pos()) && y.equals(new Pos()) && z.equals(new Neg()))
                                return true;
                }
                if(relation >= 32){
                        relation -= 32;
                        if(x.equals(new Pos()) && y.equals(new Neg()) && z.equals(new Pos()))
                                return true;
                }
                if(relation >= 16){
                        relation -= 16;
                        if(x.equals(new Pos()) && y.equals(new Neg()) && z.equals(new Neg()))
                                return true;
                }
                if(relation >= 8){
                        relation -= 8;
                        if(x.equals(new Neg()) && y.equals(new Pos()) && z.equals(new Pos()))
                                return true;
                }
                if(relation >= 4){
                        relation -= 4;
                        if(x.equals(new Neg()) && y.equals(new Pos()) && z.equals(new Neg()))
                                return true;
                }
                if(relation >= 2){
                        relation -= 2;
                        if(x.equals(new Neg()) && y.equals(new Neg()) && z.equals(new Pos()))
                                return true;
                }
                if(relation >= 1){
                        if(x.equals(new Neg()) && y.equals(new Neg()) && z.equals(new Neg()))
                                return true;
                }
                return false;
        }
}
```