

```

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

import player.*;
import edu.neu.ccs.demeterf.TUCombiner;
import edu.neu.ccs.demeterf.demfgen.lib.*;
import edu.neu.ccs.evergreen.ir.Relation;
import edu.neu.ccs.evergreen.ir.RelationI;
import gen.*;

/** 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){
        double bp = Util.breakEvenPoint(d.type);

        RawMaterialInstance rm = d.optraw.inner().instance;
        Set<ident> idents = TUCombiner.traverse(rm, new CollectIdents());
        Map<ident, Literal> literals = Map.create();

        for (ident i : idents) {
            Sign s;
            if (Util.coinFlip(bp)) {
                s = new Pos();
            } else {
                s = new Neg();
            }
            literals = literals.put(i, new Literal(s, new Variable(i)));
        }

        return new FinishedProduct(new IntermediateProduct(new Assignment(literals.values()),
            new Quality(quality(rm, literals)));
    }

    public static double quality(RawMaterialInstance rm, Map<ident, Literal> literals) {

        int sat = 0;
        int num = rm.cs.length();
        if (num == 0) {
            return 1;
        } else {
            for (Constraint c : rm.cs) {
                RelationI r = new Relation(3, c.r.v);
                int i = 2;
                for (Variable v : c.vs) {
                    int val = literals.get(v.v).value.sign();
                    r = new Relation(3, r.reduce(i--, val));
                }
                if (r.getRelationNumber() == 255) {
                    sat++;
                }
            }
            return ((double) sat) / ((double) num);
        }
    }

    private static class CollectIdents extends TUCombiner<Set<ident>> {

        @Override
        public Set<ident> combine() {
            return Set.<ident>create();
        }

        @Override
        public Set<ident> fold(Set<ident> arg0, Set<ident> arg1) {
            return arg0.union(arg1);
        }

        public Set<ident> combine(ident i) {
            return Set.<ident>create().add(i);
        }
    }
}

```