```
package player.playeragent;

import edu.neu.ccs.demeterf.demfgen.lib.List;
import gen.*;
import player.*;

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

    /**
     * Calculate the finished product for a given derivative.
     */
    public FinishedProduct finishDerivative(Derivative d) {
        Assignment assignment = bestAssignment(d.optraw.inner());

        Derivative noRaw = new Derivative(d.name, d.seller, d.price, d.type);

        // Simple profit = quality received - price paid
        String diff = Double.toString(quality - d.price.val).substring(0, 6);

        new Printer(Printer.Caller.FINISH).print(diff + ":" + noRaw.print());

        return new FinishedProduct(new IntermediateProduct(assignment),
                new Quality(quality));
    }

    /**
     * Calculate the best assignment.
     */
    public Assignment bestAssignment(RawMaterial f) {
        Assignment result = new Assignment(List.<Literal> create());

        // Give an initial quality
        quality = new Bias(f).breakEven();

        List<Variable> varList = varList(f);

        // Generate assignments for each variable
        for (int j = 0; j < varList.length(); j++) {
            Variable var = varList.lookup(j);

            RawMaterial fx0 = Shannon.cofactor(f, new Literal(new Neg(), var));
            RawMaterial fx1 = Shannon.cofactor(f, new Literal(new Pos(), var));

            double bx0 = new Bias(fx0).breakEven();
            double bx1 = new Bias(fx1).breakEven();

            Sign sign;

            // Use the bigger break even
            if (bx0 > bx1) {
                f = fx0;
                sign = new Neg();

                if (bx0 > quality) {
                    quality = bx0;
                }
            } else {
                f = fx1;
                sign = new Pos();

                if (bx1 > quality) {
                    quality = bx1;
                }
            }

            result.literals = result.literals.append(new Literal(sign, var));
        }

        return result;
    }

    /**
     * Generate the list of variable names from a raw material.
     */
    private List<Variable> varList(RawMaterial rm) {
        List<Variable> result = List.<Variable> create();
```

```
        // Create a set of variables
        for (Constraint c : rm.instance.cs) {
            List<Variable> vars = c.vs;

            for (Variable v : vars) {
                // Don't allow duplicates
                if (!result.contains(v)) {
                    result = result.append(v);
                }
            }
        }

        return result;
    }
}
```