```java
/* *********************************
 *   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 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)
    {
        long st = System.currentTimeMillis();
        RawMaterial rm = d.optraw.inner();
        Set<ident> idents = TUCombiner.traverse(rm.instance, new CollectIdents());
        List<Literal> literals = List.create();

        System.out.println("<b>Finishing Derivative</b> " + d.name);
        System.out.println("    " + rm.instance.cs.length() + " constraints");

        Iterator<ident> iter = idents.iterator();
        while(iter.hasNext())
        {
                ident id = iter.next();
                gen.Pair<PolynomialI, Double> posPair;
                gen.Pair<PolynomialI, Double> negPair;
                //Variable var = allVars.lookup(v);
                Literal pos = new Literal(new Pos(), new Variable(id));
                Literal neg = new Literal(new Neg(), new Variable(id));

                long temp = System.currentTimeMillis();
                posPair = Utils.PolyAndMaxBias(ShannonFor.shannon(rm, pos));
                temp = System.currentTimeMillis() - temp;
                System.out.println("    shannon: " + temp);
                temp = System.currentTimeMillis();
                negPair = Utils.PolyAndMaxBias(ShannonFor.shannon(rm, neg));

                temp = System.currentTimeMillis() - temp;
                System.out.println("    shannon2: " + temp);
                double biasToUse;

                //TODO: change pickbestassignment return type, make more elegant
                if(Utils.PickBestAssignment(posPair, negPair))
                {
                        biasToUse = posPair.b.doubleValue();
                }
                else
                {
                        biasToUse = negPair.b.doubleValue();
                }

                if(Util.coinFlip(biasToUse))
                {
                        literals = literals.push(pos);
                }
                else
                {
                        literals = literals.push(neg);
                }
        }

        Assignment assignment = new Assignment(literals);

        FinishedProduct fp = new FinishedProduct(new IntermediateProduct(assignment),
                Utils.quality(rm, assignment));

        st = System.currentTimeMillis() -  st;
        System.out.println("    done: " + st);

        return fp;
    }
    */

        public FinishedProduct finishDerivative(Derivative d)
	{
	long st = System.currentTimeMillis();
	RawMaterialInstance rm = d.optraw.inner().instance;
	Set<ident> idents = TUCombiner.traverse(rm, new CollectIdents());
	System.out.println("<b>Finishing</b> " + d.name);
	System.out.println("  Constraints: " + rm.cs.length());
	double bias = Utils.getBMax(d);
	Assignment assign = generateAssignment(idents, bias);
	double quality = Utils.quality(d.optraw.inner(), assign).val;
	List<Relation> rels = Utils.extractRelations(d);
	for(Relation rel : rels)
	{
		System.out.println("  relation #" + rel);
	}

	int iters = 200;
	for (int i = 0; i < iters; i++) {
		Assignment tempAssign = generateAssignment(idents, bias);
		double tempQuality = Utils.quality(d.optraw.inner(), tempAssign).val;
		if (tempQuality > quality) {
			System.out.println("    Better Assignment Found!");
			System.out.println("    Prev: " + quality + " New: " + tempQuality);
			assign = tempAssign;
			quality = tempQuality;
		}
		if(quality == 0.0 && i == 199)
		{
			iters += 200;
			System.out.println("WE'RE GOIN' AGAIN!!!");
		}
	}

	st = System.currentTimeMillis() - st;
	System.out.println("Finished: " + st + " Quality: " + quality + " Price: " + d.price.val);
	double profit = quality - d.price.val;
	if(profit < 0)
	{
		System.out.println("<font color=\"red\">Profit: " + profit + "</font>");
	}
	else
	{
		System.out.println("<font color=\"green\">Profit: " + profit + "</font>");
	}

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

	/* generate a random assignment */
	Assignment generateAssignment(Set<ident> idents, double bias) {
		List<Literal> lits = List.create();

		// Assign a value to each variable and add it to the list of literals
		for (ident i : idents) {
			Sign sign;
			if (Util.coinFlip(bias))
				sign = new Pos();
			else
				sign = new Neg();
			lits = lits.append(new Literal(sign, new Variable(i)));
		}

		return new Assignment(lits);
	}

/*this class is adapted from Bender*/
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);
            }
        }
}
```