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

import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;

import player.*;
import edu.neu.ccs.demeterf.demfgen.lib.*;
import gen.*;
import edu.neu.ccs.evergreen.ir.Relation;
import edu.neu.ccs.satsolver.*;
/** Class for finishing a list of derivatives */
public class FinishAgent implements PlayerI.FinishAgentI{
   static int randomTries = 10;
   static Sign neg = new Neg();
   static Sign pos = new Pos();

   public FinishedProduct finishDerivative(Derivative d) {
      System.out.println("ding! fries are done");
      Date d_start = new Date();
      long starttime = d_start.getTime();

      Assignment rAssign;

      if(Util.getUniqueVariables(d).length() > 6) {rAssign = bestRandomAssignment(d);}
      else {rAssign = bestBruteAssignment(d);}
      double rQuality = computeQuality(d.optraw.inner().instance.cs, rAssign);
      //double rQuality = 1.0; //Admin gives correct number either way
      System.out.println("Bought [" + d.type.instances.top().r.v + " " + ((d.type.instances.length() == 2
)? d.type.instances.lookup(1).r.v : "") +
            "] from " + d.seller.id + " at " + d.price.val + " Finished for " + rQuality);


      Date d_end = new Date();
      long endtime = d_end.getTime();
      System.out.println("Finish Time: " + (endtime - starttime) + " ms");

      return new FinishedProduct(new IntermediateProduct(rAssign), new Quality(rQuality));
   }

   /** Returns the best assignment found using getRandomAssignment, given a Derivative **/
   public Assignment bestRandomAssignment(Derivative d) {
      List<Constraint> lcon = d.optraw.inner().instance.cs;
      Date starttime = new Date();
      InputInitial input = new InputInitial(d);
      Date endtime = new Date();

      Assignment best = null;
      double bestQ = 0;
      for (int i = 0; i < randomTries; i++) {
         Assignment maybebest = getRandomAssignment(input);
         double maybeQ = computeQuality(lcon, maybebest);
         if (maybeQ > bestQ) {
            best = maybebest;
            bestQ = maybeQ;
         }
      }
      //System.out.println("INPUTINITIAL TIME: " + (endtime.getTime() - starttime.getTime()));
      return best;
   }

   /** Gets a random assignment using an optimally biased coin from a given Derivative */
   private Assignment getRandomAssignment(InputInitial i){
      Derivative d = i.d;
      Date starttime = new Date();
      OutputI o = Util.getBiasForFinishing(i);
      Date endtime = new Date();
      //System.out.println("GBFF time: " + (endtime.getTime() - starttime.getTime()));
      double bmax = o.getMaxBias();
      List<Literal> lol = List.<Literal>create();

      List<Variable> unique = Util.getUniqueVariables(d);

      Iterator<Variable> iter = unique.iterator();
      Variable vTemp;

      while(iter.hasNext()){
         vTemp = iter.next();
         boolean push = !lol.contains(new Literal(neg, vTemp)) && !lol.contains(new Literal(pos, vTemp)
);
         if(push)lol = lol.push(new Literal((Util.coinFlip(bmax)?pos:neg), vTemp));
      }
      //System.out.println("Get Single Random Assignment Time: " + (endtime - starttime) + " ms");

      return new Assignment(lol);
   }

   /** Returns the guaranteed best brute force assignment, given a Derivative.
    * This was inspired by Lichwalla.
    **/
   public Assignment bestBruteAssignment(Derivative d) {
      List<Constraint> lcon = d.optraw.inner().instance.cs;
      Date starttime = new Date();

      ArrayList<Assignment> assignments = generateBruteAssignments(d);
      Iterator<Assignment> iter = assignments.iterator();

      Assignment best = null;
      double bestQ = 0;
      while(iter.hasNext()) {
         Assignment maybebest = iter.next();
         double maybeQ = computeQuality(lcon, maybebest);
         if (maybeQ > bestQ) {
            best = maybebest;
            bestQ = maybeQ;
         }
      }
      Date endtime = new Date();
      System.out.println("Brute Time: " + (endtime.getTime() - starttime.getTime()));
      return best;
   }

   /** Generates all possible assignments given a Derivative d **/
   public ArrayList<Assignment> generateBruteAssignments(Derivative d) {
      List<Variable> lov = Util.getUniqueVariables(d);
      ArrayList<Assignment> alist = new ArrayList<Assignment>();
      for(int i = 0; i < Math.pow(2, lov.length()); i++) {
         alist.add(generateThisAssignment(lov, i));
      }
      return alist;
   }

   /** Generates this assignment 'number', like a binary number.
    * Assignment 0 is all zeros, or 'falses'.
    * Assignment 1 is all zeroes ('falses') except the last one, which is 1, or 'true'.
    * Assignment 2 is all falses except the second to last one, which is true.
    * etc.
    **/
   public Assignment generateThisAssignment(List<Variable> lov, int n) {
      List<Literal> lol = List.<Literal>create();
      int[] binary = Util.generateBinary(n, lov.length());
      Iterator<Variable> iter = lov.iterator();

      for(int i = 0; i < lov.length(); i++){
         Variable v = iter.next();
         if(binary[i] == 0) {
            lol = lol.push(new Literal(neg,v));
         }
         else {
            lol = lol.push(new Literal(pos,v));
         }
      }
      return new Assignment(lol);
   }

   /** computes the quality of a given assignment against a given list of constraints of a given relati
on **/
   private double computeQuality(List<Constraint> loc, Assignment a){

      Date d_start = new Date();
      long starttime = d_start.getTime();

      double clength = loc.length();
      double this_w, good_w = 0.0, total_w = 0.0;
      Constraint c;

      for (int i = 0; i < clength; i++) {
         c = loc.top();
         this_w = c.w.v;
         if (reduceToEnd(a, c, c.r.v, 3) == 1) good_w += this_w;
         total_w+= this_w;
         loc = loc.pop();
      }
      double qual = good_w/total_w;


      Date d_end = new Date();
      long endtime = d_end.getTime();
      //System.out.println("Compute Quality: " + (endtime - starttime) + " ms  Quality @ " + qual);
      return qual;
   }

   /** returns either 1 or 0, depending on if this assignment satisfies this constraint */
   public int reduceToEnd(Assignment a, Constraint con, int relation, int rank){
      Relation r = new Relation(rank, relation);
      if(relation == 255) return 1;   //Constraint satisfied
      if(relation == 0) return 0;       //Constraint NOT satisfied
      int value;
      Literal lit = a.literals.top();
      Variable var = lit.var;                   //Current variable to look at
      List<Variable> convarlist = con.vs;
      Assignment new_a = new Assignment(a.literals.pop());      // Remove first variable of assignment
to return to function
      if (convarlist.contains(var)) {
         value =  (lit.value.equals(pos)? 1:0);
         int position = (convarlist.length()-1) - convarlist.index(var); // position is inverted in red
uce
         return reduceToEnd(new_a, con, r.reduce(position, value), rank);
      }
      else {
         return reduceToEnd(new_a, con, relation, rank);
      }
   }

   //-------------------------------------------------------------------
   // OBSOLETE CODE:

   /** Finishes a Derivative deterministically and randomly, and takes the best assignment.
    * Random is pretty much always better, so this function is not used.
    */
   /*
   public FinishedProduct finishDerivativeBothWays(Derivative d) {
      Assignment dAssign = bestDetermAssignment(d);
      Assignment rAssign = bestRandomAssignment(d);

      double dQuality = computeQuality(d.optraw.inner().instance.cs, dAssign);
      double rQuality = computeQuality(d.optraw.inner().instance.cs, rAssign);
      double bestQuality = Math.max(dQuality, rQuality);

      Assignment bestAssignment = (dQuality > rQuality)? dAssign : rAssign;

      System.out.println("Bought [" + d.type.instances.top().r.v + " " + ((d.type.instances.length() ==
 2)? d.type.instances.lookup(1).r.v : "") +
            "] at " + d.price.val + "  Finished at " + bestQuality + ((dQuality > rQuality)? "Determ" :
 "Random") + " determ: "+dQuality);

      return new FinishedProduct(new IntermediateProduct(bestAssignment), new Quality(bestQuality));
   }
   */

   /** Finds the best assignment deterministically. Not a good way to find the best assignment. **/
   /*
   public Assignment bestDetermAssignment(Derivative d){
      List<Variable> lov = Util.getUniqueVariables(d);
      Iterator<Variable> iter = lov.iterator();
      RawMaterial m = d.optraw.inner();
      InputInitial i = new InputInitial(d);
      OutputI o = Util.getBiasForFinishing(i);
      double bMax = o.getMaxBias();
      Variable v;

      //System.out.println(d.price.val);

      List<Literal> lol = List.<Literal> create();
      while(iter.hasNext()){
         v = iter.next();
         lol = lol.append(getThisLiteral(m, v, bMax));
         //System.out.print(lol.lookup(lol.length()-1).display());
      }
      return new Assignment(lol);
   }
   */
   /** returns whether this variable should be set to true (1) or false (0) */
   /*
   public Literal getThisLiteral(RawMaterial m, Variable v, double bMax){
      List<Constraint> loc = m.instance.cs;
      Iterator<Constraint> iter = loc.iterator();
      int ones = 0, zeros = 0;
      Constraint c;

      while(iter.hasNext()){
         c = iter.next();
         if(c.vs.contains(v)){
            if(betterAssignment(c, v, bMax) == 1) ones++;
            else zeros++;
         }
      }

      return new Literal((ones > zeros? pos:neg), v);
   }
   */
   /*
   public int betterAssignment(Constraint c, Variable v, double bMax){
      int toOne, toZero;
      Relation r;
      int this_r;
      int position;
      this_r = c.r.v;
      r = new Relation(3, this_r);
      position = (c.vs.length()-1) - c.vs.index(v); // position is inverted in reduce
      toOne = r.reduce(position, 1);
      toZero = r.reduce(position, 0);
      Polynomial pToOne = Polynomial.getPolynomial(toOne);
      Polynomial pToZero = Polynomial.getPolynomial(toZero);
      //System.out.println("To One : " + Util.pluginToPoly(pToOne, bMax));
      //System.out.println("To Zero : " + Util.pluginToPoly(pToZero, bMax));

      return ((Util.computeMaxY(pToOne, bMax) > Util.computeMaxY(pToZero, bMax))? 1:0);
   }
   */



}
```