```
package test;


import static org.junit.Assert.*;

import java.util.Iterator;

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

import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
public class UtilTest {

        @BeforeClass
        public static void setUpBeforeClass() throws Exception {
        }

        @Before
        public void setUp() throws Exception {
        }

        @Test
        public void testFreshType() throws ParseException {
                //testing contains, really...
                List<Type> typelist = List.create();

                Type t = Type.parse("type classic [1 23]");
                Type t1 = Type.parse("type classic [3 255]");
                Type t2 = Type.parse("type classic [1 22]");
                Type t3 = Type.parse("type classic [23 1]");

                typelist = typelist.push(t2);
                typelist = typelist.push(t1);
                typelist = typelist.push(t);

                assertEquals(true, typelist.contains(t));
                assertEquals(true, typelist.contains(t1));
                assertEquals(true, typelist.contains(t2));
                assertEquals(false, typelist.contains(t3));

        }

        @Test
        public void testGetUniqueVariables() throws ParseException {
            Derivative s = Derivative.parse("deriv[\"s1\" 1 .3 type classic [2 6 22] rm[1: 2 v1 v2 v3 1: 6 v1 v2 v3 1: 22 v2 v4 v7]]"
);
            Derivative s1 = Derivative.parse("deriv[\"s5\" 1 .3 type classic [6] rm[1: 6 v1 v2 v3]]");
            Derivative s2 = Derivative.parse("deriv[\"s3\" 1 .3 type classic [1 125] rm[1: 1 v1 v2 v5 1: 125 v1 v2 v6]]");
            Derivative s3 = Derivative.parse("deriv[\"s4\" 1 .3 type classic [3 7] rm[1: 3 v3 v2 v1 1: 7 v1 v2 v3]]");
            Derivative s4 = Derivative.parse("deriv[\"s5\" 1 .3 type classic [6 38] rm[1: 6 v1 v2 v3 1: 38 v8 v9 v10]]");

            List<Variable> l = List.<Variable> create();
            l = l.push(new Variable(new ident("v1")));
            l = l.push(new Variable(new ident("v2")));
            l = l.push(new Variable(new ident("v3")));
            l = l.push(new Variable(new ident("v4")));
            l = l.push(new Variable(new ident("v7")));
            assertEquals(Util.getUniqueVariables(s), l);

            List<Variable> l1 = List.<Variable> create();
            l1 = l1.push(new Variable(new ident("v1")));
            l1 = l1.push(new Variable(new ident("v2")));
            l1 = l1.push(new Variable(new ident("v3")));
            assertEquals(Util.getUniqueVariables(s1), l1);

            List<Variable> l2 = List.<Variable> create();
            l2 = l2.push(new Variable(new ident("v1")));
            l2 = l2.push(new Variable(new ident("v2")));
            l2 = l2.push(new Variable(new ident("v5")));
            l2 = l2.push(new Variable(new ident("v6")));
            assertEquals(Util.getUniqueVariables(s2), l2);

            List<Variable> l3 = List.<Variable> create();
            l3 = l3.push(new Variable(new ident("v3")));
            l3 = l3.push(new Variable(new ident("v2")));
            l3 = l3.push(new Variable(new ident("v1")));
            assertEquals(Util.getUniqueVariables(s3), l3);
```

```
            List<Variable> l4 = List.<Variable> create();
            l4 = l4.push(new Variable(new ident("v1")));
            l4 = l4.push(new Variable(new ident("v2")));
            l4 = l4.push(new Variable(new ident("v3")));
            l4 = l4.push(new Variable(new ident("v8")));
            l4 = l4.push(new Variable(new ident("v9")));
            l4 = l4.push(new Variable(new ident("v10")));
            assertEquals(Util.getUniqueVariables(s4), l4);

        }

        @Test
        public void testgetImportantTypes() throws ParseException {
                Derivative d1 = Derivative.parse("deriv[\"d1\" 1 .3 type classic [2 6 22]]");
                List<TypeInstance> list = Util.getImportantTypes(d1);
                assertEquals(list.lookup(0).r.v, 2);
                assertEquals(list.length(), 1);

                d1 = Derivative.parse("deriv[\"d1\" 1 .3 type classic [22 6 2]]");
                list = Util.getImportantTypes(d1);
                assertEquals(list.lookup(0).r.v, 2);
                assertEquals(list.length(), 1);

                d1 = Derivative.parse("deriv[\"d1\" 1 .3 type classic [1 125]]");
                list = Util.getImportantTypes(d1);
                assertEquals(list.lookup(0).r.v, 1);
                assertEquals(list.length(), 1);

                d1 = Derivative.parse("deriv[\"d1\" 1 .3 type classic [6 38]]");
                list = Util.getImportantTypes(d1);
                assertEquals(list.lookup(0).r.v, 6);
                assertEquals(list.length(), 1);

        }


    @Test
    public void testGetRelationReduction() {
            Pair<Integer, int[]> reduce125 = Util.getRelationReduction(125);
            assertEquals(reduce125.b[0], 1);
            assertEquals(reduce125.b[1], 0);
            assertEquals(reduce125.b[2], 1);
            assertEquals(reduce125.b[3], 1);
            assertEquals(reduce125.b[4], 1);
            assertEquals(reduce125.b[5], 1);
            assertEquals(reduce125.b[6], 1);
            assertEquals(reduce125.b[7], 0);
            Pair<Integer, int[]> reduce22 = Util.getRelationReduction(22);
            assertEquals(reduce22.b[0], 0);
            assertEquals(reduce22.b[1], 1);
            assertEquals(reduce22.b[2], 1);
            assertEquals(reduce22.b[3], 0);
            assertEquals(reduce22.b[4], 1);
            assertEquals(reduce22.b[5], 0);
            assertEquals(reduce22.b[6], 0);
            assertEquals(reduce22.b[7], 0);
            Pair<Integer, int[]> reduce1 = Util.getRelationReduction(1);
            assertEquals(reduce1.b[0], 1);
            assertEquals(reduce1.b[1], 0);
            assertEquals(reduce1.b[2], 0);
            assertEquals(reduce1.b[3], 0);
            assertEquals(reduce1.b[4], 0);
            assertEquals(reduce1.b[5], 0);
            assertEquals(reduce1.b[6], 0);
            assertEquals(reduce1.b[7], 0);
            Pair<Integer, int[]> reduce0 = Util.getRelationReduction(0);
            assertEquals(reduce0.b[0], 0);
            assertEquals(reduce0.b[1], 0);
            assertEquals(reduce0.b[2], 0);
            assertEquals(reduce0.b[3], 0);
            assertEquals(reduce0.b[4], 0);
            assertEquals(reduce0.b[5], 0);
            assertEquals(reduce0.b[6], 0);
            assertEquals(reduce0.b[7], 0);
    }

/*
    @Test
    public void testBreakEven() throws ParseException{
            System.out.println("BreakEven Values");
```

```
            System.out.println("-------------------------------------------------\n");

            Derivative d;
            // Throw out a bunch of breakEven values to make sure this thing works

            for(int i = 1; i < 10; i++){
                    for(int j = i + 1; j < 126; j++){
                            d = Derivative.parse("deriv[\"d1\" 1 .3 type["+i+" "+j+"]]");
                            System.out.print(i + "," + j + " -> " + Util.breakEven(d) + "  \n");
                            System.out.print(Util.getPolyAndBias(d).getPolynomial().toString() + "\n");
                    }
            }

    }
    */


    @Test
    public void testBuyAgent() throws ParseException {
        BuyAgent a = new BuyAgent();
        Derivative d1 = Derivative.parse("deriv[\"d1\" 1 0.0 type classic [1 8]]");
        Derivative d2 = Derivative.parse("deriv[\"d2\" 1 0.1 type classic [2 13]]");
        Derivative d3 = Derivative.parse("deriv[\"d3\" 1 0.2 type classic [4 56]]");
        Derivative d4 = Derivative.parse("deriv[\"d4\" 1 0.3 type classic [67 198]]");
        Derivative d5 = Derivative.parse("deriv[\"d5\" 1 0.4 type classic [33 120]]");
        Derivative d6 = Derivative.parse("deriv[\"d6\" 1 0.25 type classic [33 118]]");
        Derivative d7 = Derivative.parse("deriv[\"d5\" 1 0.4 type classic [33 120]]");
        Derivative d8 = Derivative.parse("deriv[\"d6\" 1 0.25 type classic [33 118]]");
        List<Derivative> testlist1 = List.<Derivative>create();
        List<Derivative> clist1 = List.<Derivative>create();
        testlist1 = testlist1.push(d1);
        testlist1 = testlist1.push(d2);
        testlist1 = testlist1.push(d3);
        testlist1 = testlist1.push(d4);
        testlist1 = testlist1.push(d5);
        testlist1 = testlist1.push(d6);

        testlist1 = testlist1.push(d7);
        testlist1 = testlist1.push(d8);
        clist1 = clist1.push(d1);
        clist1 = clist1.push(d2);
        clist1 = clist1.push(d3);
        clist1 = clist1.push(d4);
        clist1 = clist1.push(d5);
        List<Derivative> boughtlist1 = a.buyDerivatives(testlist1, 10);
        //System.out.println(clist1.length() + " is " + boughtlist1.length());


        Iterator<Derivative> big_iter = testlist1.iterator();
        Iterator<Derivative> iter = boughtlist1.iterator();

        Derivative this_d;

        while(iter.hasNext()){
            this_d = iter.next();
            System.out.println("Bought this at " + this_d.price.val + " with breakEven "
                            + Util.breakEven(this_d) + " for profit of " + (Util.breakEven(this_d) - this_d.pri
ce.val));
        }
        assertEquals(clist1.length(),boughtlist1.length());
        /*
        for (int i = 0; i <= clist1.length(); i++) {
                assertEquals(clist1.top().name, boughtlist1.top().name);
                clist1 = clist1.pop();
                boughtlist1 = boughtlist1.pop();
        }
        */
    }

    @Test
    public void testShouldBuy() throws ParseException{
            Derivative d1 = Derivative.parse("deriv[\"d1\" 1 0.9 type classic [190]]");
            Derivative d2 = Derivative.parse("deriv[\"d1\" 1 0.505 type classic [2 22 39]]");
            Derivative d3 = Derivative.parse("deriv[\"d1\" 1 0.505 type classic [3 5]]");
            Derivative d4 = Derivative.parse("deriv[\"d1\" 1 0.505 type classic [37 132]]");
            Derivative d5 = Derivative.parse("deriv[\"d1\" 1 0.505 type classic [7 202]]");
            Derivative d6 = Derivative.parse("deriv[\"d1\" 1 0.505 type classic [5 162]]");
            Derivative d7 = Derivative.parse("deriv[\"d1\" 1 0.505 type classic [51 168]]");
            Derivative d8 = Derivative.parse("deriv[\"d2\" 1 0.505 type classic [3 128]]");
            BuyAgent b = new BuyAgent();
```

```
            assertEquals(b.shouldBuy(d1, 5.0), true);
            assertEquals(b.shouldBuy(d2, 5.0), false);

    }


    @Test
    public void testGetBias() throws ParseException {
            // SatSolverUtil calculateBias
            Derivative s1 = Derivative.parse("deriv[\"s1\" 1 .3 type classic [2 6 22] rm[1: 2 v1 v2 v3 1: 6 v1 v2 v3 1: 22 v2 v4 v7]]"
);
            Derivative s2 = Derivative.parse("deriv[\"s5\" 1 .3 type classic [6] rm[1: 6 v1 v2 v3]]");
            Derivative s3 = Derivative.parse("deriv[\"s3\" 1 .3 type classic [1 125] rm[1: 1 v1 v2 v3 1: 125 v1 v2 v3]]");
            Derivative s4 = Derivative.parse("deriv[\"s4\" 1 .3 type classic [3 7] rm[1: 3 v1 v2 v3 1: 7 v1 v2 v3]]");
            Derivative s5 = Derivative.parse("deriv[\"s5\" 1 .3 type classic [6 38] rm[1: 6 v1 v2 v3 1: 38 v8 v9 v10]]");
            // Util calculateBias
            Derivative u1 = Derivative.parse("deriv[\"s1\" 1 .3 type classic [2 6 22]]");
            Derivative u2 = Derivative.parse("deriv[\"s5\" 1 .3 type classic [6]]");
            Derivative u3 = Derivative.parse("deriv[\"s3\" 1 .3 type classic [1 125]]");
            Derivative u4 = Derivative.parse("deriv[\"s4\" 1 .3 type classic [3 7]]");
            Derivative u5 = Derivative.parse("deriv[\"s5\" 1 .3 type classic [6 38]]");
            /*
          //  System.out.println("o1:------------------------------------------------------------------");
            OutputI o1 = Util.getPolyAndBias(s1);
          //  System.out.println("o2:------------------------------------------------------------------");
            OutputI o2 = Util.getPolyAndBias(s2);
          //  System.out.println("o3:------------------------------------------------------------------");
            OutputI o3 = Util.getPolyAndBias(s3);
          //  System.out.println("o4:------------------------------------------------------------------");
            OutputI o4 = Util.getPolyAndBias(s4);
          //  System.out.println("o5:------------------------------------------------------------------");
            OutputI o5 = Util.getPolyAndBias(s5);
          //  System.out.println("o6:------------------------------------------------------------------");
            OutputI o6 = Util.getPolyAndBias(u1);
          //  System.out.println("o7:------------------------------------------------------------------");
            OutputI o7 = Util.getPolyAndBias(u2);
          //  System.out.println("o8:------------------------------------------------------------------");
            OutputI o8 = Util.getPolyAndBias(u3);
          //  System.out.println("o9:------------------------------------------------------------------");
            OutputI o9 = Util.getPolyAndBias(u4);
          //  System.out.println("o10:-----------------------------------------------------------------");
            OutputI o10 = Util.getPolyAndBias(u5);
            */
          /*
            System.out.println("test calculateBiasForfinishing");
            System.out.println("Type: 2/6/22 MaxBias: " + o1.getMaxBias() + "  should be: ~.333");
            System.out.println("Type: 6 MaxBias: " + o2.getMaxBias() + "  should be: ~.333");
            System.out.println("Type: 1/125 MaxBias: " + o3.getMaxBias() + "  should be: 0.0");
            System.out.println("Type: 3/7 MaxBias: " + o4.getMaxBias() + "  should be: 0.0");
            System.out.println("Type: 6/38 MaxBias: " + o5.getMaxBias() + "  should be: ~.333"+"\n\n");
            System.out.println("test calculateBias");
            System.out.println("Type: 2/6/22 MaxBias: " + o6.getMaxBias() + "  should be: ~.333");
            System.out.println("Type: 6 MaxBias: " + o7.getMaxBias() + "  should be: ~.333");
            System.out.println("Type: 1/125 MaxBias: " + o8.getMaxBias() + "  should be: 0.0");
            System.out.println("Type: 3/7 MaxBias: " + o9.getMaxBias() + "  should be: 0.0");
            System.out.println("Type: 6/38 MaxBias: " + o10.getMaxBias() + "  should be: ~.333"+"\n\n");
           */
            /*
            System.out.println(o1.getPolynomial().toString());
            System.out.println(o2.getPolynomial().toString());
            System.out.println(o3.getPolynomial().toString());
            System.out.println(o4.getPolynomial().toString());
            System.out.println(o5.getPolynomial().toString());
            System.out.println("-------------------------------------------------------------------");
            System.out.println(o6.getPolynomial().toString());
            System.out.println(o7.getPolynomial().toString());
            System.out.println(o8.getPolynomial().toString());
            System.out.println(o9.getPolynomial().toString());
            System.out.println(o10.getPolynomial().toString());
            */

    }


    @Test
    public void testReduceToEnd() throws ParseException {
           FinishAgent fa = new FinishAgent();
       Assignment a = Assignment.parse("[!v1 !v2 v3]");
       Constraint c = Constraint.parse("1: 22 v1 v2 v3");
       assertEquals(1, fa.reduceToEnd(a, c, 22, 3));
```

```
      a = Assignment.parse("[!v1 !v2 v3 v4 v5]");
      c = Constraint.parse("1: 2 v1 v2 v3");
      assertEquals(1, fa.reduceToEnd(a, c, 2, 3));

      a = Assignment.parse("[!v2 !v1 !v3 v4]");
      c = Constraint.parse("1: 2 v1 v2 v3");
      assertEquals(0, fa.reduceToEnd(a, c, 2, 3));

      a = Assignment.parse("[v3 !v2 !v1 v6]");
      c = Constraint.parse("1: 2 v1 v2 v3");
      assertEquals(1, fa.reduceToEnd(a, c, 2, 3));

      a = Assignment.parse("[!v2 !v1 v3]");
      c = Constraint.parse("1: 1 v1 v2 v3");
      assertEquals(0, fa.reduceToEnd(a, c, 1, 3));

      a = Assignment.parse("[!v2 !v1 v3]");
      c = Constraint.parse("1: 6 v1 v2 v3");
      assertEquals(1, fa.reduceToEnd(a, c, 6, 3));

      a = Assignment.parse("[!v2 v1 v3]");
      c = Constraint.parse("1: 127 v1 v2 v3");
      assertEquals(1, fa.reduceToEnd(a, c, 127, 3));

      a = Assignment.parse("[!v2 !v1 !v3]");
      c = Constraint.parse("1: 127 v1 v2 v3");
      assertEquals(1, fa.reduceToEnd(a, c, 127, 3));

        }

   @Test
   public void testDeliverRawMaterial() throws ParseException{
           DeliverAgent dAgent = new DeliverAgent();
           Derivative i1 = Derivative.parse("deriv[\"s1\" 1 .3 type classic [2 6 22]]");
           Derivative i2 = Derivative.parse("deriv[\"s5\" 1 .3 type classic [6]]");
           Derivative i3 = Derivative.parse("deriv[\"s3\" 1 .3 type classic [1 125]]");
           Derivative i4 = Derivative.parse("deriv[\"s4\" 1 .3 type classic [3 7]]");
           Derivative i5 = Derivative.parse("deriv[\"s5\" 1 .3 type classic [6 38]]");


           dAgent.deliverRawMaterial(i1);
           /*
           System.out.println(dAgent.deliverRawMaterial(i2).print());
           System.out.println(dAgent.deliverRawMaterial(i3).print());
           System.out.println(dAgent.deliverRawMaterial(i4).print());
           System.out.println(dAgent.deliverRawMaterial(i5).print());
           */
   }


   @Test
   public void testFinishingAgent() throws ParseException {


      DeliverAgent da = new DeliverAgent();
      FinishAgent fa = new FinishAgent();
      CreateAgent ca = new CreateAgent();
      Derivative d0, d1;
      FinishedProduct f;

      for(int i = 1; i <= 5; i++){
          d0 = ca.createDerivative(new Player(new PlayerID(1), "Woot"), List.<Type> create());
          d1 = da.deliverRawMaterial(d0);
          f = fa.finishDerivative(d1);
          System.out.println(Util.breakEven(d0) + ":" + d0.type.instances.lookup(0).r.v + " " + ((d0.ty
pe.instances.length() == 2)? d0.type.instances.lookup(1).r.v : "")+ " finished @ " + f.quality.val);

      }

    }




   //-------------------------------------------------
   // OBSOLETE CODE:
   /*
        @Test
        public void testComputePolynomial(){
                int [] cat1 = new int[]{0, 3, 0, 0};
```

```
                int [] result1 = new int[] {0, 3, -6, 3};
                for (int i=0; i < 4; i++) {
                        assertEquals(result1[i], Util.computePolynomial(cat1)[i]);
                }
                int [] cat2 = new int[]{0, 0, 0, 0};
                int [] result2 = new int[] {0, 0, 0, 0};
                for (int i=0; i < 4; i++) {
                        assertEquals(result2[i], Util.computePolynomial(cat2)[i]);
                }
                int [] cat3 = new int[]{0, 0, 0, 1};
                int [] result3 = new int[] {0, 0, 0, 1};
                for (int i=0; i < 4; i++) {
                        assertEquals(result3[i], Util.computePolynomial(cat3)[i]);
                }
                int [] cat4 = new int[]{0, 0, 1, 0};
                int [] result4 = new int[] {0, 0, 1, -1};
                for (int i=0; i < 4; i++) {
                        assertEquals(result4[i], Util.computePolynomial(cat4)[i]);
                }
        }
         */
        /*
        @Test
        public void testComputeDeriv() {
                int[] poly1 = new int[] {0,2,2,1};
                int[] deriv1 = new int[] {2,4,3};
                int[] poly2 = new int[] {1,0,0,0};
                int[] deriv2 = new int[] {0,0,0};
                int[] poly3 = new int[] {0,0,0,0};
                int[] deriv3 = new int[] {0,0,0};
                int[] poly4 = new int[] {-1,3,-4,-2};
                int[] deriv4 = new int[] {3,-8,-6};
                for (int i=0; i < 3; i++) {
                        assertEquals(deriv1[i], Util.computeDeriv(poly1)[i]);
                }
                for (int i=0; i < 3; i++) {
                        assertEquals(deriv2[i], Util.computeDeriv(poly2)[i]);
                }
                for (int i=0; i < 3; i++) {
                        assertEquals(deriv3[i], Util.computeDeriv(poly3)[i]);
                }
                for (int i=0; i < 3; i++) {
                        assertEquals(deriv4[i], Util.computeDeriv(poly4)[i]);
                }
        }
        */
        /*@Test
        public void testComputeBMax() {
                assertEquals(1.0/3.0, Util.computeBMax(2));
                assertEquals(0, Util.computeBMax(3));
                assertEquals(0.5, Util.computeBMax(10));
                assertEquals(2.0/3.0, Util.computeBMax(8));
                assertEquals(1.0/3.0, Util.computeBMax(16));
        }
  */
}
```