```
/* *********************************
 *   DeliverAgent.java
 *     Delivers Raw Materials
 * *********************************/
package player.playeragent;

import player.*;
import edu.neu.ccs.demeterf.demfgen.lib.*;
import gen.*;
import java.util.Iterator;


/** Class for delivering raw material for a derivative */
public class DeliverAgent implements PlayerI.DeliverAgentI{

    /** Adds raw materials to the given derivatives */
    public Derivative deliverRawMaterial(Derivative needRM){
        return needRM.deliver(rawMaterialInst(needRM));
    }

    /** Compute a RawMaterial Instance for the given Derivative */
    private RawMaterialInstance rawMaterialInst(Derivative d){
        //Gets the Relation Number from the Derivative
        int relNum = d.type.instances.top().r.v;

        //Generates the Number of variables
        java.util.Random rand = new java.util.Random();
        int varNum = 10 + rand.nextInt(41);

        //Sets the Number of Variables per Constraint
        int vars = 3;

        //Generates the Constraints
        List<Constraint> constraints = DeliverAgent.getConstraints(relNum, vars, getElems(varNum));

        return new RawMaterialInstance(constraints);
    }

    /** Generates the Number of variables and constraints*/
    private static List<Variable> getElems(int n){
        List<Variable> elems = List.create();
        for (Integer index = n; index > 0; index--){
                elems.append(new Variable(new ident("v"+index.toString())));
        }
        return elems;
    }

    /** Computes a Constraint when given a list of variables*/
    private static List<Constraint> getConstraints(int rNum, int varsPerCon, List<Variable> elems){
        return DeliverAgent.listListVarsToListConstraints(rNum, DeliverAgent.getSymmetric(elems, varsPe
rCon));
    }

    @SuppressWarnings("unchecked")
        private static List<List<Variable>> getSymmetric(List<Variable> vars, int arity){
        List<List<Variable>> constraintargs = List.create();
        if(vars.length() < arity || arity <= 0)
                return constraintargs;
        if(arity == 1){
                Iterator<Variable> iter = vars.iterator();
                while(iter.hasNext()){
                        constraintargs = constraintargs.append(List.create(iter.next()));
                }
                return constraintargs;
        }
        Iterator<Variable> iter = vars.iterator();
        while(iter.hasNext()){
                Variable x = iter.next();
                constraintargs.append(DeliverAgent.addIntoVarLists(x, DeliverAgent.getSymmetric(vars.re
move(x), arity-1)));
        }
        return constraintargs;
        }

    private static List<Constraint> listListVarsToListConstraints(int relNum, List<List<Variable>> bs){
        List<Constraint> lc = List.create();
        while(!bs.isEmpty()){
                lc.append(new Constraint(new Weight(1), new RelationNr(relNum), bs.top()));
                bs.pop();
        }
        return lc;
```

```
    }

        @SuppressWarnings("unchecked")
        private static List<List<Variable>> addIntoVarLists(Variable v, List<List<Variable>> constraint
args){
                List<List<Variable>> retval = List.create();
                Iterator<List<Variable>> iter = constraintargs.iterator();
                while(iter.hasNext()){
                        retval = retval.append(iter.next().append(v));
                }
                return retval;
        }
}
```