```java
package player;

import edu.neu.ccs.csu670.components.derivativeminimizer.DerivativeMinimizer;
import edu.neu.ccs.demeterf.demfgen.lib.List;
import gen.RelationNr;
import gen.Type;
import gen.TypeInstance;

import java.util.TreeSet;

/**
 * Given a derivative d, the robot should compute the equivalent derivative
 * simplify(d) where simplify(d) has the minimum number of relations among all
 * equivalent derivatives. Two derivatives d and d1 are equivalent iff
 * break-even(d) = break-even(d1).
 */
public class Minimizer implements DerivativeMinimizer {
    private TreeSet<Integer> relations;
    private TreeSet<Integer> others;
    private TreeSet<Integer> rest;

    /**
     * Simplify (minimize) the given Type.
     */
    public Type simplify(Type t) {
        List<TypeInstance> instances = List.<TypeInstance> create();

        relations = toIntegerSet(t);

        // Test each relation
        while (relations.size() > 0) {
            int first = relations.pollFirst();

            // Either the relation will begin a relationship or be independent
            instances = instances
                    .append(new TypeInstance(new RelationNr(first)));

            // Set to test against
            others = new TreeSet<Integer>(relations);

            // Loop through all others
            findOthers(first);
        }

        return new Type(instances);
    }

    /**
     * Tests all of the other relations. If we find a reduction, we can remove
     * that value from the relations list and find the rest of the reductions.
     */
    private void findOthers(int first) {
        // Test each of the others
        while (others.size() > 0) {
            int next = others.pollFirst();

            // Find a reduction
            if (Util.reduce(first, next)) {
                // Ignore the reduced value in the future
                relations.remove(next);

                rest = new TreeSet<Integer>(others);

                // See if any further reductions occur
                findRest(next);
            }
        }
    }

    /**
     * Tests the further relations. If we find a reduction, we can remove that
     * value from both the relations list and the list of others.
     */
    private void findRest(int next) {
        for (int test : rest) {
            if (Util.reduce(next, test)) {
                // Ignore the further reduction values in the future
                relations.remove(test);
                others.remove(test);

                next = test;
```

```
            }
        }
    }

    /**
     * Convert a Type to a set of relation numbers (integers). This both removes
     * duplicates and accomplishes sorting for us (ascending order).
     */
    private TreeSet<Integer> toIntegerSet(Type t) {
        TreeSet<Integer> result = new TreeSet<Integer>();

        for (TypeInstance i : t.instances) {
            result.add(i.r.v);
        }

        return result;
    }
}
```