```java
package player.playeragent;

import edu.neu.ccs.demeterf.demfgen.lib.List;
import gen.Derivative;
import gen.Player;
import gen.Price;
import gen.RelationNr;
import gen.Secret;
import gen.Type;
import gen.TypeInstance;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Random;

import player.Minimizer;
import player.PlayerI;
import player.Util;

/**
 * Class for creating a derivative.
 */
public class CreateAgent implements PlayerI.CreateAgentI {
    private static final double FRACTION = 1.0 / 3.0;

    /**
     * Returns a newly created derivative of a different type than already
     * existing derivatives.
     */
    public Derivative createDerivative(Player player, List<Type> existing) {
        Derivative result;
        Type type;

        // Create a unique type
        do {
            type = generate();
        } while (existing.contains(type));

        double q = Util.getQuality(type);

        result = new Derivative(Util.freshName(player), player.id,
                increasePrice(q), type);

        return result;
    }

    /**
     * Generate the type for the derivative.
     */
    private Type generate() {
        List<TypeInstance> relations = List.<TypeInstance> create();

        // Choose [2, 5] independent relations
        int k = new Random().nextInt(4) + 2;

        while (relations.length() < k) {
            // [1, 254]
            int value = new Random().nextInt(254) + 1;

            TypeInstance i = new TypeInstance(new RelationNr(value));

            // A type is a set
            if (!relations.contains(i)) {
                relations = relations.append(i);

                // Minimize what we have so far
                Type simple = new Minimizer().simplify(new Type(relations));

                // Remove the type instance if it is part of a relationship
                if (relations.length() != simple.instances.length()) {
                    relations = relations.remove(i);
                }
            }
        }

        return new Type(new Secret(), shuffle(relations));
    }

    /**
     * Increase the price.
     */
    private Price increasePrice(double p) {
        double price = p + ((1 - p) * FRACTION);

        return new Price(price > 1 ? 1.0 : price);
    }

    /**
     * Shuffle the list.
     */
    private List<TypeInstance> shuffle(List<TypeInstance> input) {
        ArrayList<TypeInstance> list = new ArrayList<TypeInstance>(Arrays
                .asList(input.toArray(new TypeInstance[input.length()])));
        Collections.shuffle(list);

        input = List.<TypeInstance> create(list.toArray(new TypeInstance[list
                .size()]));

        return input;
    }
}
```