```
package player;


import gen.*;
import java.util.Random;
import java.util.UUID;
import utils.DocumentHandler;
import utils.DerivativesFinder;
import config.GlobalConfig;
import config.PlayerConfig;
import edu.neu.ccs.demeterf.demfgen.lib.List;

/* TODO: This should be changed so we can provide a Non-FileSystem
 *         interface to the Administrator.  Many of the functions below
 *         just need to ask the Administrator for values from the store
 *         or accounts
 */

/** Various Player based utilities */
public class Util{
    static Random rand = new Random();
    static boolean debug = true;
    static int playerID;

    public static void setID(int id)
    {
        playerID = id;
    }

    public static int getID()
    {
        return playerID;
    }


    /** Random Double between 0..1 */
    public static double random(){ return rand.nextDouble(); }

    /** Random Integer between 0..(bound-1)*/
    public static int random(int bound){ return rand.nextInt(bound); }

    /** Random coin flip of the given bias */
    public static boolean coinFlip(double bias){ return (Util.random() < bias); }

    /** Random coin flip, bias of 0.5 */
    public static boolean coinFlip(){ return coinFlip(0.5); }

    /** Write a player transaction */
    public static void commitTransaction(PlayerTransaction pTrans){
        String fileName = pTrans.player.name+GlobalConfig.DONE_FILE_SUFFIX;
        DocumentHandler.write(pTrans.print(),(PlayerConfig.BLACKBOARD_PATH+GlobalConfig.SEPAR+fileName)
);
    }

    /** Find the account for the given player */
    public static double getAccount(Player p){ return getAccounts().getAccount(p); }

    /** Find the Derivatives (that Player is selling) that need RawMaterials */
    public static List<Derivative> needRawMaterial(Player player){
        return DerivativesFinder.findDersThatNeedRM(getStore().stores, player);
    }

    /** Find the Derivatives (that Player is buying) that need to be Finished */
    public static List<Derivative> toBeFinished(Player player){
        return DerivativesFinder.findDersThatNeedFinishing(getStore().stores, player);
    }

    /** Get the minimum price decrement when reoffering */
    public static double getMinPriceDec(){ return getConfig().MPD; }

    /** Get the current Types in the Store */
    public static List<Type> existingTypes(){
        return DerivativesFinder.findExistingDerTypes(getStore().stores);
    }

    /** Find all Derivatives ForSale */
    public static List<Derivative> forSale(){
        return DerivativesFinder.findDerivativesForSale(getStore().stores);
    }

    /** Find uniquely typed Derivatives ForSale */
```

```
    public static List<Derivative> uniquelyTyped(List<Derivative> forSale){
        return DerivativesFinder.findUniqueDerivatives(forSale);
    }

    /** Get a Fresh Derivative name */
    public static String freshName(Player p){
        UUID newID = UUID.randomUUID();
        return p.name+"_"+newID.toString().replace('-', '_');
    }

    /**
     * Get a Fresh Type, not in the Store. Not really nec. as the Players will be more complex
     * Only returns RelationNr's from 2 - 128, and no odd numbers.
     * */
    public static Type freshType(List<Type> existing){
        Type type = null;
        RelationNr relnum = null;

        // Create a new Type until we get one with an even RelationNr that isn't already in the store
        do {
                relnum = new RelationNr((Util.random(63) + 1) * 2);
                List<Integer> imps = findImplied(relnum.v);
                List<TypeInstance> types = List.create(new TypeInstance(relnum));

                for (int i = 0; i < imps.length(); i++) {
                        TypeInstance ti = new TypeInstance(new RelationNr(imps.lookup(i).intValue()));
                        types = types.append(ti);

                        int addZero = Util.random(100);
                        if(addZero < 75 && addZero > 70)
                        {
                        types = types.append(new TypeInstance(new RelationNr(0)));
                        }
                }

                type = new Type(types);
        }
        while(existing.contains(type));

        if (debug = true) {
                System.out.println("Original relation number is " + relnum);
                System.out.println("Implied numbers are:");
                for (int j = 0; j < type.instances.length(); j++) {
                        System.out.println(type.instances.lookup(j).r.v);
                }
        }

        return type;
    }

    public static List<Integer> findImplied(int num)
    {
            List<Integer> imps = List.create();

            for(int i = 1; i < 8; i++) {
                double power = Math.pow(2.0, (double)i);

                if ((power > num) && (power + num < 127)) {
                        Integer imp = new Integer((int)(power + num));
                        imps = imps.append(imp);
                }
            }

            return imps;
    }

    private static Accounts getAccounts()
    { return DocumentHandler.getAccounts(PlayerConfig.BLACKBOARD_PATH); }
    private static Store getStore()
    { return DocumentHandler.getStore(PlayerConfig.BLACKBOARD_PATH); }
    private static Config getConfig()
    { return DocumentHandler.getConfig(PlayerConfig.BLACKBOARD_PATH); }
}
```