```java
package player;

import gen.*;

import java.util.Random;
import java.util.UUID;

import player.playeragent.DeliverAgent;
import player.playeragent.FinishAgent;
import utils.DocumentHandler;
import utils.DerivativesFinder;
import config.GlobalConfig;
import config.PlayerConfig;
import edu.neu.ccs.demeterf.demfgen.lib.List;

/**
 * Various player based utilities.
 */
public class Util {
    public static final double TOLERANCE = 0.0001;

    private static Random rand = new Random();

    /**
     * Get the quality from the best assignment.
     */
    public static double getQuality(Type t) {
        RawMaterialInstance r = new DeliverAgent().rawMaterialInst(t);

        FinishAgent f = new FinishAgent();
        f.bestAssignment(new RawMaterial(r));

        return f.quality;
    }

    /**
     * Perform a bitwise 'AND' test for equality.
     */
    public static boolean reduce(int a, int b) {
        return (a & b) == a;
    }

    /**
     * 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);
    }

    /**
     * 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 raw materials.
     */
    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 maximum number of constraints.
     */
    public static int getMaxRawMaterialLen() {
        return getConfig().MaxRawMaterialLen;
    }

    /**
     * Get the maximum number of derivatives that may be created.
     */
    public static int getMaxCreates() {
        return getConfig().MaxCreates;
    }

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

    /**
     * Find all derivatives for sale.
     */
    public static List<Derivative> forSale(PlayerID id) {
        return DerivativesFinder.findDerivativesForSaleByOthers(
                getStore().stores, id);
    }

    /**
     * Find uniquely typed derivatives for sale.
     */
    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('-', '_');
    }

    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);
    }
}
```