```
/* *********************************
 *   BuyAgent.java
 *     Handles the buying of Derivatives.
 * *********************************/
package player.playeragent;

import player.*;
import edu.neu.ccs.demeterf.demfgen.lib.List;
import edu.neu.ccs.evergreen.ir.Relation;
import gen.*;

/** Class for buying a derivative */
public class BuyAgent implements PlayerI.BuyAgentI{

    /** Returns the profitable derivatives from those on sale */
    public List<Derivative> buyDerivatives(List<Derivative> forSale, double account){
        List<Derivative> dersToBuy = forSale.foldl(new List.Fold<Derivative, BuyChoice>(){
            public BuyChoice fold(Derivative d, BuyChoice choice){
                // Should this one be bought??
                if(shouldBuy(d, choice.account))
                    return choice.buy(d);

                // I guess not...
                return choice;
            }}, new BuyChoice(account)).toBuy;
        List<Derivative> dersToBuyFavFive = List.create();

        int i = 0;
        while(i < dersToBuy.length() && i < 9)
        {
                dersToBuyFavFive = dersToBuyFavFive.append(dersToBuy.lookup(i));
                i++;
        }

        return dersToBuyFavFive;
        //return dersToBuy;
    }

    /** Is the given Derivative profitable? Do I have enough Money?
     *  Returns true if the given Derivative's price is less than the determined
     *  break-even point, and as long as we have enough money in our account.
     */
    /*public boolean shouldBuy(Derivative der, double account) {
        List<Relation> rels = Utils.extractRelations(der);

        // ensure that we do not buy from ourselves!
        if (der.seller.id == Util.getID())
        {
                return false;
        }

        double maxBreakEven = 0, breakEven = 0;
        for(int i = 0;  i < rels.length(); i++)
        {
                breakEven = Utils.getBreakEven(rels.lookup(i));
                if(breakEven > maxBreakEven)
                {
                        maxBreakEven = breakEven;
                }
        }

        if(der.price.val < maxBreakEven && der.price.val <= account)
        {
                return true;
        }
        else
        {
                return false;
        }
    } */
    public boolean shouldBuy(Derivative der, double account) {

        // ensure that we do not buy from ourselves!
        if (der.seller.id == Util.getID())
        {
                return false;
        }

        List<Relation> rels = Utils.extractRelations(der);

        //ensure that we don't fall for a celtics exploit
```

```
        if(rels.length() == 1 && rels.top().getRelationNumber() == 0)
        {
                return false;
        }

        Double breakEven = Utils.getBreakEven(der);

        if(der.price.val < breakEven && der.price.val <= account && der.price.val < 0.98)
        {
                return true;
        }
        else
        {
                return false;
        }
    }

    /** Collects buying decisions based on the shouldBuy "predicate" */
    private class BuyChoice{
        double account;
        List<Derivative> toBuy;
        BuyChoice(double acc){ this(acc, List.<Derivative>create()); }
        BuyChoice(double acc, List<Derivative> buy){ account = acc; toBuy = buy; }
        BuyChoice buy(Derivative der){
            return new BuyChoice(account-der.price.val, toBuy.push(der));
        }
    }
}
```