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

import player.*;
import edu.neu.ccs.demeterf.demfgen.lib.List;
import gen.*;
import java.util.Iterator;


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

        private class DerivativeValue{
                public Derivative d;
                public double be;

                public DerivativeValue(Derivative d, double breakEven){
                        this.d = d;
                        this.be =  breakEven;
                }
        }

        public class DerivativeValueComparator extends List.Comp<DerivativeValue>{
                public boolean comp(DerivativeValue der1, DerivativeValue der2){
                        return (der1.be - der1.d.price.val) / der1.d.price.val < (der2.be - der2.d.pric
e.val) / der2.d.price.val;
                }
        }

    /** Returns the profitable derivatives from those on sale */
    public List<Derivative> buyDerivatives(List<Derivative> forSale, double account){

        //The Iterator for the For Sale List
        Iterator<Derivative> iter = forSale.iterator();

        //The List of Buy Able Derivatives
        List<DerivativeValue> buyAble = List.create();

        //Gets the List of Profitable Derivatives
        while (iter.hasNext()){
                Derivative forSaleDer = iter.next();
                double breakEvenPrice = 0.0;
                //Try to calculate Derivatives break-even price
                try{
                        breakEvenPrice = util.BreakEven.getBreakEvenPoint(util.BreakEven.getRelations(f
orSaleDer));
                }
                catch(Exception e){}
                if (forSaleDer.price.val <= breakEvenPrice){
                        buyAble.push(new DerivativeValue(forSaleDer, breakEvenPrice));
                }
        }

        int buySize = buyAble.length();

        //The List of Derivatives we are going to Buy
        List<Derivative> buying = List.create();

        if(buySize <= 0){
                return buying;
        }

        //Sorts the List by profit per price (profitability)
        buyAble = buyAble.sort(new DerivativeValueComparator());
        //If the list is in the opposite of the desired order (greatest profitability first), flip it.
        if(new DerivativeValueComparator().comp(buyAble.top(), buyAble.lookup(buySize-1)))
                buyAble = buyAble.reverse();

        //The cost counter for all of the buying derivatives
        double currentDerCosts = 0.0;
        //Adds the most profitable derivatives that we can afford to the cart
        while (!buyAble.isEmpty()){
                //If we can afford the derivative add it to the buying list
                if((buyAble.top().d.price.val+currentDerCosts) <= account){
                        //Adds the cost of the derivative to the buying counter
                        currentDerCosts += buyAble.top().d.price.val;
                        //Adds the deriavtive to the buying list
                        buying.push(buyAble.top().d);
```

```
                }
                        buyAble = buyAble.pop();
        }

        return buying;
    }

}
```