```java
package player.playeragent;

import java.util.PriorityQueue;

import player.Bias;
import player.PlayerI;
import edu.neu.ccs.demeterf.demfgen.lib.List;
import gen.Derivative;

/**
 * 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> result = List.<Derivative> create();

        PriorityQueue<Profit> q = new PriorityQueue<Profit>();

        // Build the queue of profitable derivatives
        for (Derivative d : forSale) {
            double profit = new Bias(d.type).breakEven() - d.price.val;

            if (profit > 0) {
                q.add(new Profit(profit, d));
            }
        }

        // Most profitable
        while (q.size() > 0) {
            Derivative d = q.poll().d;

            // Don't exceed our account
            if (account > d.price.val) {
                result = result.append(d);
                // For now, only buy one!
                break;
            }
        }

        return result;
    }

    /**
     * Gives us a way to sort for the most profitable derivatives.
     */
    private class Profit implements Comparable<Profit> {
        private final double profit;
        private final Derivative d;

        public Profit(double profit, Derivative d) {
            this.profit = profit;
            this.d = d;
        }

        public int compareTo(Profit p) {
            return new Double(profit).compareTo(new Double(p.profit));
        }
    }
}
```