```java
package com.upgrad;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.spark.api.java.function.Function2;
import org.apache.spark.api.java.function.PairFlatMapFunction;
import org.apache.spark.streaming.api.java.JavaDStream;
import org.apache.spark.streaming.api.java.JavaPairDStream;

import scala.Tuple2;

public class StockAnalyserProblemStatements  implements Serializable{

    private static final long serialVersionUID = 1L;

    /* Problem Statement 1:
     * Calculate the simple moving average closing price of the four stocks in a 5-minute sliding window for the last 10 minutes.
     * Closing prices are used mostly by the traders and investors as it reflects the price at which the market finally settles down.
     * The SMA (Simple Moving Average) is a parameter used to find the average stock price over a certain period based on a set of
     * parameters. The simple moving average is calculated by adding a stock's prices over a certain period and dividing the
     * sum by the total number of periods. The simple moving average can be used to identify buying and selling opportunities
     *
    */
    public void getMoveAvgClosingPrice(JavaPairDStream<String, StockAnalyserStockAverageTuple> result) {
        JavaPairDStream<String, String> moveAvgClosingPricepair =  result.flatMapToPair(new PairFlatMapFunction<Tuple2<String,StockAnalyserStockAverageTuple>,String, String>() {

            private static final long serialVersionUID = 1L;

            @Override
            public Iterator<Tuple2<String, String>> call(Tuple2<String, StockAnalyserStockAverageTuple> t) throws Exception {
                List<Tuple2<String, String>> list = new ArrayList<Tuple2<String, String>>();

                list.add(new Tuple2<String, String>(t._1, t._2.getAvgMovingClosingPrice()));
                return list.iterator();
            }
        });
        moveAvgClosingPricepair.print();
    }

    /*
     * Problem Statement 2:
     * Find the stock out of the four stocks giving maximum profit (average closing price - average opening price)
     * in a 5-minute sliding window for the last 10 minutes.
    */
    public void getMaxProfit(JavaPairDStream<String, StockAnalyserStockAverageTuple> result){
        // Getting the tuple having maximum stock profit
        JavaDStream<Tuple2<String, StockAnalyserStockAverageTuple>> maxProfit = result.reduce(new Function2<Tuple2<String,StockAnalyserStockAverageTuple>, Tuple2<
```

```
        String,StockAnalyserStockAverageTuple>, Tuple2<String,
        StockAnalyserStockAverageTuple>>() {
51
            private static final long serialVersionUID = 1L;
52
53
            @Override
54
            public Tuple2<String, StockAnalyserStockAverageTuple> call(Tuple2<String,
55          StockAnalyserStockAverageTuple> v1, Tuple2<String,
            StockAnalyserStockAverageTuple> v2)
                    throws Exception {
56              if (v1._2().getProfit()  > v2._2().getProfit())
57                  return v1;
58
59              return v2;
60          }
61      });
62
63      JavaPairDStream<String, String> maxProfitpair =  maxProfit.flatMapToPair(new
64      PairFlatMapFunction<Tuple2<String,StockAnalyserStockAverageTuple>,String,String
        >() {
65
            private static final long serialVersionUID = 1L;
66
67
            @Override
68          public Iterator<Tuple2<String, String>> call(Tuple2<String,
69          StockAnalyserStockAverageTuple> t) throws Exception {
                List<Tuple2<String, String>> list = new ArrayList<Tuple2<String, String
70              >>();
71
                list.add(new Tuple2<String, String>(t._1, t._2.getMaxProfit()));
72              return list.iterator();
73          }
74      });
75
76      maxProfitpair.print();
77  }
78
79  /*
80   * Problem Statement 3:
81   * Calculate the trading volume(total traded volume) of the four stocks every 10
82   minutes and decide which stock to purchase
     * out of the four stocks. Remember to take the absolute value of the volume.
83   * Volume plays a very important role in technical analysis as it helps us to
84   confirm trends and patterns.
     * You can think of volumes as a means to gain insights into how other participants
85   perceive the market.
     * Volumes are an indicator of how many stocks are bought and sold over a given
86   period of time. Higher the volume,
     * more likely the stock will be bought.
87   */
88  public void  getTradingVolume(JavaPairDStream<String, StockAnalyserStockAverageTuple
89  > result){
        // Getting the tuple having maximum stock profit
90      JavaDStream<Tuple2<String, StockAnalyserStockAverageTuple>> maxProfit = result.
91      reduce(new Function2<Tuple2<String,StockAnalyserStockAverageTuple>, Tuple2<
        String,StockAnalyserStockAverageTuple>, Tuple2<String,
        StockAnalyserStockAverageTuple>>() {
92
            private static final long serialVersionUID = 1L;
93
94
            @Override
95          public Tuple2<String, StockAnalyserStockAverageTuple> call(Tuple2<String,
96          StockAnalyserStockAverageTuple> v1, Tuple2<String,
            StockAnalyserStockAverageTuple> v2)
```

```
                    throws Exception {
                if (v1._2().getTradingVolume()  > v2._2().getTradingVolume())
                    return v1;

                return v2;
            }
        });

        JavaPairDStream<String, String> pairTradingVolume =  maxProfit.flatMapToPair(new
         PairFlatMapFunction<Tuple2<String,StockAnalyserStockAverageTuple>,String,String
        >() {

            private static final long serialVersionUID = 1L;

            @Override
            public Iterator<Tuple2<String, String>> call(Tuple2<String,
            StockAnalyserStockAverageTuple> t) throws Exception {
                List<Tuple2<String, String>> list = new ArrayList<Tuple2<String, String
                >>();

                list.add(new Tuple2<String, String>(t._1, t._2.getMaxTradingVolume()));
                return list.iterator();
            }
        });

        pairTradingVolume.print();
    }
}
```