```java
package com.upgrad.driver;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.api.java.function.Function2;
import org.apache.spark.api.java.function.PairFlatMapFunction;
import org.apache.spark.streaming.Durations;
import org.apache.spark.streaming.api.java.JavaDStream;
import org.apache.spark.streaming.api.java.JavaInputDStream;
import org.apache.spark.streaming.api.java.JavaPairDStream;
import org.apache.spark.streaming.api.java.JavaStreamingContext;
import org.apache.spark.streaming.kafka010.ConsumerStrategies;
import org.apache.spark.streaming.kafka010.KafkaUtils;
import org.apache.spark.streaming.kafka010.LocationStrategies;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.upgrad.StockAnalyserProblemStatements;
import com.upgrad.StockAnalyserStock;
import com.upgrad.StockAnalyserStockAverageTuple;
import com.upgrad.constant.StockAnalyserContants;

import scala.Tuple2;

public final class StockAnalyser {
    public static void main(String[] args) throws Exception {
        if (args.length != 3) {
            System.out.println(
                "Following Parameter needed: TopicName, GroupId, BootstrapServer");
            return;
        }

        // Persisting the input arguments into the variables
        String topicName = args[0];
        String groupId = args[1];
        String bootstrapServers = args[2];

        // Create Context. Set a unique name for the application
        SparkConf sparkConf = new SparkConf().setAppName(StockAnalyserContants.APP_NAME
        ).setMaster("local[*]");

        //Create context with a 1 minute batch interval. This is the crux of Spark
        Streaming i.e. defining a micro batch.
        JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.minutes
        (1));
        ssc.sparkContext().setLogLevel("WARN");

        //Split the topics if multiple values are passed. In our case its just a single
        topic.
        Set<String> topicsSet = new HashSet<>(Arrays.asList(topicName.split(",")));

        //Define a new HashMap for holding the Kafka information
```

```
61          Map<String, Object> props = new HashMap<>();
62          props.put("bootstrap.servers", bootstrapServers);
63          props.put("group.id", groupId);
64          props.put("enable.auto.commit", "true");
65          props.put("key.deserializer", StringDeserializer.class);
66          props.put("value.deserializer", StringDeserializer.class);
67
68          // Create direct Kafka stream with brokers and topics.
69          // LocationStrategy with prefer consistent allows partitions to be distributed
            consistently to the spark executors.
70          // ConsumerStrategy allows to subscribe to the Kafka topic.
71          //JavaInputDStream is a continuous input stream associated to the source.
72          JavaInputDStream<ConsumerRecord<String, String>> messages = KafkaUtils.
            createDirectStream(
73                  ssc,
74                  LocationStrategies.PreferConsistent(),
75                  ConsumerStrategies.Subscribe(topicsSet, props));
76
77          //JavaDStream is an internal stream object for processed data.
78          JavaDStream<StockAnalyserStock> stockTest =  messages.map(new Function<
            ConsumerRecord<String,String>, StockAnalyserStock>() {
79              private static final long serialVersionUID = 1L;
80
81              @Override
82              public StockAnalyserStock call(ConsumerRecord<String, String> record) throws
                 Exception {
83                  ObjectMapper mapper = new ObjectMapper();
84
85                  TypeReference<StockAnalyserStock> mapType = new TypeReference<
                    StockAnalyserStock>() {};
86
87                  // Parsing the JSON String
88                  StockAnalyserStock stock = null;
89                  try {
90                      stock = mapper.readValue(record.value(), mapType);
91                  } catch (IOException e) {
92                      e.printStackTrace();
93                  }
94
95                  return stock;
96              }
97          }).cache();
98
99          // Print the raw input
100         stockTest.print();
101
102         JavaPairDStream<String, StockAnalyserStockAverageTuple> pairDStream = stockTest.
            flatMapToPair(
103                 new PairFlatMapFunction<StockAnalyserStock, String,
                    StockAnalyserStockAverageTuple>() {
104                     private static final long serialVersionUID = 67676744;
105                     public Iterator<Tuple2<String, StockAnalyserStockAverageTuple>> call
                        (StockAnalyserStock st)
106                             throws Exception {
107                         List<Tuple2<String, StockAnalyserStockAverageTuple>> list = new
                            ArrayList<Tuple2<String, StockAnalyserStockAverageTuple>>();
108
109                         String symbol = st.getSymbol().toString();
110                         list.add(new Tuple2<String, StockAnalyserStockAverageTuple>(
                            symbol,
111                                 new StockAnalyserStockAverageTuple(1,
112                                         st.getPriceData().getClose(), st.getPriceData().
                                            getClose() - st.getPriceData().getOpen(), st.
                                            getPriceData().getVolume())));
```

```
                        return list.iterator();
                    }
                }).cache();

        JavaPairDStream<String, StockAnalyserStockAverageTuple> result= pairDStream.
        reduceByKeyAndWindow(
                new Function2<StockAnalyserStockAverageTuple,
                StockAnalyserStockAverageTuple, StockAnalyserStockAverageTuple>() {
                    private static final long serialVersionUID = 76761212;
                    public StockAnalyserStockAverageTuple call(
                    StockAnalyserStockAverageTuple st1, StockAnalyserStockAverageTuple
                    st2)
                            throws Exception {
                        st1.setClosePrice(
                                st1.getClosePrice() + st2.getClosePrice());
                        st1.setCount(st1.getCount() + st2.getCount());
                        st1.setProfit(
                                st1.getProfit() + st2.getProfit());
                        st1.setTradingVolume(
                                st1.getTradingVolume() + st2.getTradingVolume());

                        return st1;
                    }
                }, Durations.minutes(5), Durations.minutes(1)).cache();

        // Execute the problem queries
        StockAnalyserProblemStatements probstatements = new
        StockAnalyserProblemStatements();
        probstatements.getMoveAvgClosingPrice(result);
        probstatements.getMaxProfit(result);
        probstatements.getTradingVolume(result);
        ssc.start();
        ssc.awaitTermination();
    }
}
```