Execute the program like below:

"Usage: java -cp SparkApplication.jar SparkApplication input_stream_dir output_dir"

```
java -cp SparkApplication.jar SparkApplication /Upgrad/Project/spark_stream
/Upgrad/Project/spark_output
```

Logic of Spark Streaming Application:

File: SparkApplication.java

1) Create spark context and javastreaming context with batch interval as 1 minute
2) Read the input stream as String from json file in input directory specified as program arguments (i.e arg[0])
3) Use the JSONParser to get the array of lines (individual stock data) in the string format
4) Get the individual stock data by parsing the line string and convert to JavaDStream<Stock>

Problem 1: Calculate the simple moving average closing price of the four stocks in a 5-minute sliding window for the last 10 minutes.

1) Get individual stocks close, open value and use it has value to create StockAvgInfo object (with count as 1) and use stock symbol as key to form the JavaPairDStream<String, StockAvgInfo>

2) Do reduceByKeyAndWindow() to aggregate the close and open values for each stock (key with stock symbol). Specify the window interval as 10 and sliding interval as 5

3) Do mapToPair() transformation on each stock entry in aggregate stock RDD stock_window_agg_data to do the average on close using count value

4) Store the result in output directory specified as input parameters args[1] and create file “prob1_avg_close + timestamp”

Problem 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

1) Get individual stocks close, open value and use it has value to create StockAvgInfo object (with count as 1) and use stock symbol as key to form the JavaPairDStream<String, StockAvgInfo>

2) Do reduceByKeyAndWindow() to aggregate the close and open values for each stock (key with stock symbol). Specify the window interval as 10 and sliding interval as 5

3) Do mapToPair() transformation on each stock entry in aggregate stock RDD stock_window_agg_data to get the profit by subtracting the average on open from average on close and store it as <symbol, profit> pair Dstream stock_profit_info

4) Exchange key/value pair in stock_profit_info to sort by profit

5) Save the result by descending order of profit in output directory specified as input parameters args[1] and create file "prob2_max_profit + timestamp"

Problem 3: Find out the Relative Strength Index or RSI of the four stocks in a 5-minute sliding window for the last 10 minutes.

1) Do mapToPair() on stocks RDD and get the profit/loss value of each stock in tuple2<symbol, StockProfitInfo>
2) Do reduceByKeyAndWindow() to aggregate the profit and loss values for each stock (key with stock symbol). Specify the window interval as 10 and sliding interval as 5
3) Do mapToPair() to get the Average Gain and Average Loss for each stock in StockProfitInfo for the given window. <symbol, StockProfitInfo>
4) Do mapToPairt() to get the RSI for each stock and return the <symbol, rsi> tuple as output.
    a. RSI = 100 if Average Loss is zero
    b. RSI = 0 if Average Profit is zero
    c. Rest based on formula

5) Save the result without sorting in output directory specified as input parameters args[1] and create file “prob3_stock_rsi + timestamp”

Problem 4: Calculate the trading volume of the four stocks every 10 minutes and decide which stock to purchase out of the four stocks

1) Do mapToPair() on stocks RDD and get the volume value of each stock in tuple2<symbol, Integer>
2) Do reduceByKeyAndWindow() to aggregate the volume for each key (stock symbol in this case)
3) Exchange key/value pair to <volume_sum, Symbol> to sort by max_volume
4) Save the result by descending order of profit in output directory specified as input parameters args[1] and create file “prob4_max_vol + timestamp”