```java
// Just importing all the necessary namespaces
package org.apache.flink.quickstart;
import com.dataartisans.flinktraining.exercises.datastream_java.datatypes.TaxiRide;
import com.dataartisans.flinktraining.exercises.datastream_java.sources.TaxiRideSource;
import com.dataartisans.flinktraining.exercises.datastream_java.utils.GeoUtils;
import org.apache.flink.api.common.functions.FilterFunction;
import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.java.tuple.Tuple3;
import org.apache.flink.streaming.api.TimeCharacteristic;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.windowing.time.Time;
import java.util.Scanner;
import java.util.Calendar;
import java.util.TimeZone;


public class HomeworkRight{

    // Already provided util to handle terminals and their location
    public enum JFKTerminal {
        TERMINAL_1(71436),
        TERMINAL_2(71688),
        TERMINAL_3(71191),
        TERMINAL_4(70945),
        TERMINAL_5(70190),
        TERMINAL_6(70686),
        NOT_A_TERMINAL(-1);

        int mapGrid;

        private JFKTerminal(int grid){
            this.mapGrid = grid;
```

```
34          }
35
36          public static JFKTerminal gridToTerminal(int grid){
37              for(JFKTerminal terminal : values()){
38                  if(terminal.mapGrid == grid) return terminal;
39              }
40              return NOT_A_TERMINAL;
41          }
42      }
43
44      /* Main body of the execution containing the data importation, the handling
45      of user choice of the task to execute, the stream processing logic and the
46      production of the output.
47       */
48      public static void main(String[] args) throws Exception {
49
50          StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
51          env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
52
53          // get the taxi ride data stream form the file
54          DataStream<TaxiRide> rides = env.addSource(
55                  new TaxiRideSource(
56                          "/Users/nicolovendramin/flinkLab/flink-java-project/src/main/" +
57                                  "java/org/apache/flink/quickstart/data/nycTaxiRides.gz",
58                          60, // Watermark
59                          2000));
60
61          // Reading from System.in to know which one of the tasks we want to print.
62          Scanner reader = new Scanner(System.in);
63
64          // We keep reading until we get a valid choice
65          int n = -1;
66          while(n<0 || n>3) {
```

```
67              System.out.println(
68                      "Enter the number of the task you want to print " +
69                              "[1 -> terminal visit per hour," +
70                              " 2-> busiest terminal per hour," +
71                              " 3 -> busiest terminal in exit per hour," +
72                              " 0 -> all of them]: ");
73
74              n = reader.nextInt(); // Scans the next token of the input as an int.
75          }
76          reader.close();
77
78          // Generating the result of task1
79          DataStream<Tuple3<JFKTerminal, Integer, Integer>> terminal_rides = rides
80                  .filter(new JFKFilter()) // filtering rides starting or arriving in JFK
    terminals
81                  .map(new TerminalPresenceTimeMapper()) // mapping each ride to the leave-arrive
     events
82                  .keyBy(2) // grouping the result by hour of the day
83                  .keyBy(0) // grouping inside each single grouping by terminal
84                  .timeWindow(Time.hours(1))  // defining a one hour time window
85                  .sum(1); // summing over the grouping in the desired time window
86
87          // Generating the result of task2 from previous exercise
88          DataStream<Tuple3<JFKTerminal, Integer, Integer>> terminal_rides_max = terminal_rides
89                  .keyBy(2) // grouping the previous result by hour of the day
90                  .timeWindowAll(Time.hours(1)) // selecting a time window of one hour across all
     nodes
91                  .max(1); // picking the max with respect to the counting
92
93          // Generating the result of task2 considering only trips leaving the terminal
94          DataStream<Tuple3<JFKTerminal, Integer, Integer>> terminal_rides_max_leaving = rides
95                  .filter(new StartRideFilter()) // filtering on start rides
96                  .filter(new JFKFilter()) // filtering rides leaving from JFK terminals
```

```
 97                  .map(new TerminalPresenceTimeMapper()) // mapping each ride to the leave
    events
 98                  .keyBy(2) // grouping the result by hour of the day
 99                  .keyBy(0) // grouping inside each single grouping by terminal
100                  .timeWindow(Time.hours(1))  // defining a one hour time window
101                  .sum(1) // summing over the grouping in the desired time window
102                  .keyBy(2) // grouping the previous result by hour of the day
103                  .timeWindowAll(Time.hours(1)) // selecting a time window of one hour across
    all nodes
104                  .max(1); // picking the max with respect to the counting
105
106          // printing only the required results
107          if(n == 1 || n == 0){
108              terminal_rides.print();
109          }
110          if(n == 2 || n == 0) {
111              terminal_rides_max.print();
112          }
113          if(n == 3 || n == 0) {
114              terminal_rides_max_leaving.print();
115          }
116
117          env.execute();
118
119      }
120
121      /*
122      Filters only the start events or end events in a JFKTerminal.
123      Keeps only those taxi rides that are start events or end events having, respectively as
124      a starting or ending location, one of the terminals of the JFK Airport.
125       */
126      public static class JFKFilter implements FilterFunction<TaxiRide> {
127
```

```
128         @Override
129         public boolean filter(TaxiRide taxiRide) throws Exception {
130
131
132             JFKTerminal terminal;
133
134             // If the record is a start event
135             if(taxiRide.isStart)
136                 // consider as location the starting location
137                 terminal = JFKTerminal.gridToTerminal(GeoUtils.mapToGridCell(taxiRide.startLon
    , taxiRide.startLat));
138             // If the record is an end event
139             else
140                 // consider as location the ending location
141                 terminal = JFKTerminal.gridToTerminal(GeoUtils.mapToGridCell(taxiRide.endLon,
    taxiRide.endLat));
142
143             // the condition to filter  is that the location is a terminal of JFK Airport
144             boolean condition = terminal != JFKTerminal.NOT_A_TERMINAL;
145
146             if(condition)
147                 return true;
148             else return false;
149         }
150     }
151
152     /*
153     Filters only those rides the start ride events
154     */
155     public static class StartRideFilter implements FilterFunction<TaxiRide> {
156
157         @Override
158         public boolean filter(TaxiRide taxiRide) throws Exception {
```

```
159
160             return taxiRide.isStart;
161         }
162     }
163
164     /*
165     This mapper maps each event to its Terminal, 1, hour tuple.
166      */
167     public static final class TerminalPresenceTimeMapper implements MapFunction<TaxiRide,
    Tuple3<JFKTerminal, Integer, Integer>> {
168
169         @Override
170         public Tuple3<JFKTerminal, Integer, Integer> map(TaxiRide taxiRide) throws Exception {
171
172             int grid = 0;
173             long millis = 0;
174
175             // If the record is a start event
176             if(taxiRide.isStart) {
177                 //  the cell is extracted from the starting location
178                 grid = GeoUtils.mapToGridCell(taxiRide.startLon, taxiRide.startLat);
179                 //  the time is extracted from the starting time
180                 millis = taxiRide.startTime.getMillis();
181             }
182             // If the record is an end event
183             else{
184                 //  the cell is extracted from the end location
185                 grid = GeoUtils.mapToGridCell(taxiRide.endLon, taxiRide.endLat);
186                 //  the time is extracted from the end time
187                 millis = taxiRide.endTime.getMillis();
188             }
189
190             // We set up a calendar to be able to extract the hour of the day basing on the
```

```
190 unix timestamp
191             Calendar calendar = Calendar.getInstance();
192             calendar.setTimeZone(TimeZone.getTimeZone( "America/New_York" ));
193             calendar.setTimeInMillis(millis);
194 
195             // We return a tuple including the terminal of the record, the number of events (1
    ), and the hour of the day
196             return new Tuple3<>(JFKTerminal.gridToTerminal(grid), 1, calendar.get(Calendar.
    HOUR_OF_DAY));
197 
198         }
199 
200     }
201 
202 }
203 
204 
```