```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;
        }

        public static JFKTerminal gridToTerminal(int grid){
            for(JFKTerminal terminal : values()){
                if(terminal.mapGrid == grid) return terminal;
            }
            return NOT_A_TERMINAL;
        }
    }

    /* Main body of the execution containing the data importation, the handling
    of user choice of the task to execute, the stream processing logic and the
    production of the output.
     */
    public static void main(String[] args) throws Exception {

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);

        // get the taxi ride data stream form the file
        DataStream<TaxiRide> rides = env.addSource(
                new TaxiRideSource(
                        "/Users/nicolovendramin/flinkLab/flink-java-project/src/main/" +
                                "java/org/apache/flink/quickstart/data/nycTaxiRides.gz",
                        60, // Watermark
                        2000));

        // Reading from System.in to know which one of the tasks we want to print.
        Scanner reader = new Scanner(System.in);

        // We keep reading until we get a valid choice
        int n = -1;
        while(n<0 || n>3) {
            System.out.println(
                    "Enter the number of the task you want to print " +
                            "[1 -> terminal visit per hour ," +
                            " 2-> busiest terminal per hour ," +
                            " 3 -> busiest terminal in exit per hour," +
                            " 0 -> all of them]: ");

            n = reader.nextInt(); // Scans the next token of the input as an int.
        }
        reader.close();

        // Generating the result of task1
        DataStream<Tuple3<JFKTerminal, Integer, Integer>>
```

```
terminal_rides = rides
                .filter(new JFKFilter()) // filtering rides starting or arriving in JFK terminals
                .map(new TerminalPresenceTimeMapper()) // mapping each ride to the leave-arrive events
                .keyBy(2) // grouping the result by hour of the day
                .keyBy(0) // grouping inside each single grouping by terminal
                .timeWindow(Time.hours(1))  // defining a one hour time window
                .sum(1); // summing over the grouping in the desired time window

        // Generating the result of task2 from previous exercise
        DataStream<Tuple3<JFKTerminal, Integer, Integer>> terminal_rides_max = terminal_rides
                .keyBy(2) // grouping the previous result by hour of the day
                .timeWindowAll(Time.hours(1)) // selecting a time window of one hour across all nodes
                .max(1); // picking the max with respect to the counting

        // Generating the result of task2 considering only trips leaving the terminal
        DataStream<Tuple3<JFKTerminal, Integer, Integer>> terminal_rides_max_leaving = rides
                .filter(new StartRideFilter()) // filtering on start rides
                .filter(new JFKFilter()) // filtering rides leaving from JFK terminals
                .map(new TerminalPresenceTimeMapper()) // mapping each ride to the leave events
                .keyBy(2) // grouping the result by hour of the day
                .keyBy(0) // grouping inside each single grouping by terminal
                .timeWindow(Time.hours(1))  // defining a one hour time window
                .sum(1) // summing over the grouping in the desired time window
                .keyBy(2) // grouping the previous result by hour of the day
                .timeWindowAll(Time.hours(1)) // selecting a time window of one hour across all nodes
                .max(1); // picking the max with respect to the counting

```

```java
        // printing only the required results
        if(n == 1 || n == 0){
            terminal_rides.print();
        }
        if(n == 2 || n == 0) {
            terminal_rides_max.print();
        }
        if(n == 3 || n == 0) {
            terminal_rides_max_leaving.print();
        }

        env.execute();

    }

    /*
    Filters only the start events or end events in a JFKTerminal.
    Keeps only those taxi rides that are start events or end events having, respectively as
    a starting or ending location, one of the terminals of the JFK Airport.
     */
    public static class JFKFilter implements FilterFunction<TaxiRide> {

        @Override
        public boolean filter(TaxiRide taxiRide) throws Exception {


            JFKTerminal terminal;

            // If the record is a start event
            if(taxiRide.isStart)
                // consider as location the starting location
                terminal = JFKTerminal.gridToTerminal(GeoUtils.mapToGridCell(taxiRide.startLon, taxiRide.startLat));
            // If the record is an end event
            else
                // consider as location the ending location
                terminal = JFKTerminal.gridToTerminal(GeoUtils.mapToGridCell(taxiRide.endLon, taxiRide.endLat));

            // the condition to filter  is that the location is a terminal of JFK Airport
            boolean condition = terminal != JFKTerminal.
```

```
NOT_A_TERMINAL;

            if(condition)
                return true;
            else return false;
        }
    }

    /*
    Filters only those rides the start ride events
    */
    public static class StartRideFilter implements FilterFunction<TaxiRide> {

        @Override
        public boolean filter(TaxiRide taxiRide) throws Exception {

            return taxiRide.isStart;
        }
    }

    /*
    This mapper maps each event to its Terminal, 1, hour tuple.
     */
    public static final class TerminalPresenceTimeMapper implements MapFunction<TaxiRide, Tuple3<JFKTerminal, Integer, Integer>> {

        @Override
        public Tuple3<JFKTerminal, Integer, Integer> map(TaxiRide taxiRide) throws Exception {

            int grid = 0;
            long millis = 0;

            // If the record is a start event
            if(taxiRide.isStart) {
                //  the cell is extracted from the starting location
                grid = GeoUtils.mapToGridCell(taxiRide.startLon, taxiRide.startLat);
                //  the time is extracted from the starting time
                millis = taxiRide.startTime.getMillis();
            }
            // If the record is an end event
            else{
                //  the cell is extracted from the end
```

```java
location
        grid = GeoUtils.mapToGridCell(taxiRide.endLon, taxiRide.endLat);
        //  the time is extracted from the end time
        millis = taxiRide.endTime.getMillis();
      }

      // We set up a calendar to be able to extract the hour of the day basing on the unix timestamp
      Calendar calendar = Calendar.getInstance();
      calendar.setTimeZone(TimeZone.getTimeZone( "America/New_York" ));
      calendar.setTimeInMillis(millis);

      // We return a tuple including the terminal of the record, the number of events (1), and the hour of the day
      return new Tuple3<>(JFKTerminal.gridToTerminal(grid), 1, calendar.get(Calendar.HOUR_OF_DAY));

    }

  }

}


```