```
// RentGen.java - GENERATE DATA FOR THE RENTAL PROFILE REPORT
//
// MODULE INDEX
// NAME                 CONTENTS
// Event                Event structure
// EventComp            Event comparator
// main                 Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 12-09-13     JS      Original
//
//-----------------------------------------------------------------------------

// IMPORTS

import java.util.*;
import java.text.*;
import java.io.*;

//-----------------------------------------------------------------------------

public class RentGen {

    //-------------------------------------------------------------------------

    // DEFINITIONS

    final static int CUST_CNT = 150;            // Number of customers
    final static int VEHICLE_CNT = 75;          // Number of vehicles
    final static int BASE_YEAR = 2012;          // Base date
    final static int BASE_MONTH = 1-1;
    final static int BASE_DAY = 1;
    final static int END_YEAR = 2013;           // End date
    final static int END_MONTH = 9-1;
    final static int END_DAY = 30;

    //-------------------------------------------------------------------------

    // CAR TYPES

    final static String VEHICLE_TYPES[] = {
        "Toyota Camry",
        "Honda Accord",
        "Honda Civic",
        "Naza Forte",
        "Proton Waja",
        "Proton Persona",
        "Proton Gen 2",
        "Proton Saga"
    };

    //-------------------------------------------------------------------------

    // RENTALS

    final static int VEHICLE_RENTALS[] = {
        550,
        550,
        350,
```

```
    250,
    200,
    180,
    160,
    160
};

//---------------------------------------------------------------------

// CUSTOMER NAMES

final static String CUST_NAME_FIRST[] = {
    "Aaliyah", "Aaron", "Abigail", "Adam", "Addison", "Adrian", "Aiden",
    "Alex", "Alexa", "Alexander", "Alexandra", "Alexis", "Alice",
    "Allison", "Alyssa", "Amelia", "Andrew", "Anna", "Annabelle",
    "Anthony", "Aria", "Arianna", "Asher", "Aubrey", "Audrey", "Austin",
    "Ava", "Avery", "Bailey", "Bella", "Benjamin", "Bentley", "Blake",
    "Brandon", "Brayden", "Brianna", "Brody", "Brooke", "Brooklyn",
    "Caleb", "Cameron", "Caroline", "Carson", "Carter", "Charlie",
    "Charlotte", "Chase", "Chloe", "Christian", "Christopher", "Claire",
    "Clara", "Cole", "Colin", "Colton", "Connor", "Cooper", "Daniel",
    "David", "Declan", "Dominic", "Dylan", "Easton", "Eleanor", "Elena",
    "Eli", "Eliana", "Elijah", "Elise", "Elizabeth", "Ella", "Ellie",
    "Emily", "Emma", "Ethan", "Eva", "Evan", "Evelyn", "Gabriel",
    "Gabriella", "Gavin", "Gianna", "Grace", "Grayson", "Hailey", "Hannah",
    "Harper", "Harrison", "Hayden", "Henry", "Hudson", "Hunter", "Ian",
    "Isaac", "Isabella", "Isabelle", "Isaiah", "Isla", "Jace", "Jack",
    "Jackson", "Jacob", "Jake", "James", "Jasmine", "Jason", "Jayden",
    "Jeremiah", "John", "Jonathan", "Jordan", "Joseph", "Joshua", "Julia",
    "Julian", "Kaitlyn", "Kate", "Kayla", "Kaylee", "Kendall", "Kennedy",
    "Kylie", "Landon", "Lauren", "Layla", "Leah", "Leo", "Levi", "Liam",
    "Lillian", "Lily", "Lincoln", "Logan", "Lucas", "Lucy", "Luke", "Lyla",
    "Mackenzie", "Madelyn", "Madison", "Makayla", "Maria", "Mason",
    "Matthew", "Max", "Maya", "Mia", "Micah", "Michael", "Mila", "Miles",
    "Molly", "Morgan", "Natalie", "Nathan", "Nathaniel", "Nevaeh",
    "Nicholas", "Noah", "Nolan", "Nora", "Oliver", "Olivia", "Owen",
    "Paige", "Parker", "Peyton", "Piper", "Quinn", "Reagan", "Reese",
    "Riley", "Ruby", "Ryan", "Ryder", "Sadie", "Samantha", "Samuel",
    "Sarah", "Savannah", "Scarlett", "Sean", "Sebastian", "Sienna",
    "Sophia", "Sophie", "Stella", "Sydney", "Taylor", "Thomas", "Tristan",
    "Tyler", "Victoria", "Violet", "William", "Wyatt", "Xavier", "Zachary",
    "Zoe"
};
final static String CUST_NAME_LAST[] = {
    "Alexander", "Ali", "Anderson", "Brown", "Campbell", "Clark", "Clarke",
    "Cox", "Davies", "Davis", "Doherty", "Driscoll", "Edwards", "Evans",
    "Graham", "Green", "Griffiths", "Hall", "Hamilton", "Hughes",
    "Jackson", "James", "Jenkins", "Johnson", "Johnston", "Jones", "Kelly",
    "Khan", "Lewis", "MacDonald", "Martin", "Mason", "McLaughlin",
    "Mitchell", "Moore", "Morgan", "Morrison", "Moss", "Murphy", "Murray",
    "O'Neill", "Owen", "Patel", "Paterson", "Phillips", "Price", "Quinn",
    "Rees", "Reid", "Roberts", "Robertson", "Robinson", "Rodriguez",
    "Rose", "Ross", "Sanders", "Scott", "Smith", "Smyth", "Stewart",
    "Taylor", "Thomas", "Thompson", "Thomson", "Walker", "Watson", "White",
    "Williams", "Wilson", "Wood", "Wright", "Young"
};

//---------------------------------------------------------------------

// VEHICLE STRUCTURE
```

```
public static class Vehicle {
    public int      vehicleType;
    public String   registration;
}

//-----------------------------------------------------------------------

// CUSTOMER STRUCTURE

public static class Cust {
    public String   custName;
    public String   licenceNo;
}

//-----------------------------------------------------------------------

// EVENT STRUCTURE

public static class Event {
    public long     eventTime;
    public String   outlet;
    public String   operation;
    public String   vehicleTag;
    public String   vehicleType;
    public int      odometer;
    public String   licenceNo;
    public String   custName;
    public int      amountPaid;

    // Basic Constructor

    Event ()
    {
        // Empty
    }

    // Copy Constructor

    Event (
        Event       e)
    {
        eventTime = e.eventTime;
        outlet = e.outlet;
        operation = e.operation;
        vehicleTag = e.vehicleTag;
        vehicleType = e.vehicleType;
        odometer = e.odometer;
        licenceNo = e.licenceNo;
        custName = e.custName;
        amountPaid = e.amountPaid;
    }
}

//-----------------------------------------------------------------------

// EVENT COMPARATOR

public static class EventComp implements Comparator<Event> {
    public int
    compare (
        Event       e1,
```

```
        Event       e2)
    {
        if (e1.eventTime < e2.eventTime) return -1;
        if (e1.eventTime > e2.eventTime) return 1;
        return 0;
    }
}

//---------------------------------------------------------------------------

// MAIN LINE

public static void
main (
    String[]        argv)           // Argument values
{
    Random          rand;           // Instance of random number generator
    int             i, j;           // General purpose index
    Cust[]          custArr;        // Customer array
    String[]        tagArr;         // Vehicle tag number array
    Vector<Event>   eventVec;       // Event vector
    Event[]         eventArr;       // Event array
    Event           event;          // Event instance
    int             vi;             // Vehicle type index
    int             ci;             // Customer index
    int             daysRented;     // Days rented
    GregorianCalendar curDate;      // Current date
    GregorianCalendar endDate;      // End date
    GregorianCalendar curTime;      // Current time
    PrintStream     outStream;      // Output stream
    SimpleDateFormat dateFormat;    // Date formatter

    // Initialise the random number generator

    rand = new Random (20360);

    // Create the customer array

    custArr = new Cust [ CUST_CNT ];
    for (i = 0; i < CUST_CNT; i++) {
        do {
            custArr[i] = new Cust();
            custArr[i].custName =
                CUST_NAME_FIRST[rand.nextInt(CUST_NAME_FIRST.length)]+" "+
                CUST_NAME_LAST[rand.nextInt(CUST_NAME_LAST.length)];
            j = 0;
            while (
                j < i &&
                ! custArr[j].custName.equals(custArr[i].custName)
            ) j ++ ;
        } while (j < i);
        do {
            custArr[i].licenceNo = Character.toString (
                (char)('A' + rand.nextInt(26)));
            for (j = 0; j < 6; j++)
                custArr[i].licenceNo += Character.forDigit (
                    rand.nextInt(10), 10);
            j = 0;
            while (
                j < i &&
                ! custArr[j].licenceNo.equals(custArr[i].licenceNo)
```

```
        ) j ++ ;
    } while (j < i);
}

// Initialise the vehicle tag array

tagArr = new String [ VEHICLE_CNT ];

// Initialise the event vector

eventVec = new Vector<Event> ();

// Process each vehicle

for (i = 0; i < VEHICLE_CNT; i++) {

    // Initialise the event structure

    event = new Event ();

    // Select the outlet

    j = rand.nextInt (100);
    if (j < 50)
        event.outlet = "KL Main";
    else if (j < 85)
        event.outlet = "KLIA";
    else
        event.outlet = "Subang";

    // Generate a vehicle tag number

    do {
        tagArr[i] = "";
        for (j = 0; j < 3; j++)
            tagArr[i] += Character.toString (
                (char)('A' + rand.nextInt(26)));
        tagArr[i] += '-';
        for (j = 0; j < 4; j++)
            tagArr[i] += Character.forDigit (rand.nextInt(10), 10);
        j = 0;
        while (j < i && ! tagArr[j].equals(tagArr[i])) j ++ ;
    } while (j < i);
    event.vehicleTag = tagArr[i];

    // Select the vehicle type

    vi = rand.nextInt (VEHICLE_TYPES.length);
    event.vehicleType = VEHICLE_TYPES[vi];

    // Initialise the odometer reading

    event.odometer = rand.nextInt(60000);

    // Initialise the current date

    curDate = new GregorianCalendar (BASE_YEAR, BASE_MONTH, BASE_DAY);
    endDate = new GregorianCalendar (END_YEAR, END_MONTH, END_DAY);

    // Generate rental sequences
```

```
while (curDate.before (endDate)) {

    // Move the date forward for the vehicle idle time

    curDate.add (Calendar.DATE, 1+rand.nextInt(3));

    // Load the current time

    curTime = curDate;
    curTime.set (Calendar.HOUR_OF_DAY, 8 + rand.nextInt(10));
    curTime.set (Calendar.MINUTE, rand.nextInt(60));
    curTime.set (Calendar.SECOND, rand.nextInt(60));
    event.eventTime = curTime.getTimeInMillis();

    // Set the operation

    event.operation = "OUT";

    // Select the customer

    ci = rand.nextInt (CUST_CNT);
    event.custName = custArr[ci].custName;
    event.licenceNo = custArr[ci].licenceNo;

    // Select the days rented and calculate the payment

    daysRented = 1 + rand.nextInt (12);
    event.amountPaid = (daysRented+1) * VEHICLE_RENTALS[vi];

    // Append the event to the event vector

    eventVec.add (new Event (event));

    // Move the date forward for the vehicle rental time

    curDate.add (Calendar.DATE, daysRented);

    // Only output the "IN" event if it is before the end time

    if (curDate.before (endDate)) {

        // Load the current time

        curTime = curDate;
        curTime.set (Calendar.HOUR_OF_DAY, 8 + rand.nextInt(10));
        curTime.set (Calendar.MINUTE, rand.nextInt(60));
        curTime.set (Calendar.SECOND, rand.nextInt(60));
        event.eventTime = curTime.getTimeInMillis();

        // Set the operation

        event.operation = "IN";
        event.amountPaid = 0;

        // Increment the odometer

        for (j = 0; j < daysRented; j++)
            event.odometer += rand.nextInt (150);

        // Append the event to the event vector
```

```
                    eventVec.add (new Event (event));
                }
            }
        }

        // Sort the event vector

        eventArr = eventVec.toArray(new Event [1]);
        Arrays.sort (eventArr, new EventComp());

        // Initialise the date formatter

        dateFormat = new SimpleDateFormat ("yyyy-MM-dd'\t'HH:mm:ss");

        // Emit the event vector

        try {
            outStream = new PrintStream (new FileOutputStream ("RENTLOG"));
            for (i = 0; i < eventArr.length; i++) {
                curTime = new GregorianCalendar ();
                curTime.setTimeInMillis (eventArr[i].eventTime);
                outStream.print (dateFormat.format (curTime.getTime()));
                outStream.print ("\t");
                outStream.print (eventArr[i].outlet);
                outStream.print ("\t");
                outStream.print (eventArr[i].operation);
                outStream.print ("\t");
                outStream.print (eventArr[i].vehicleTag);
                outStream.print ("\t");
                outStream.print (eventArr[i].vehicleType);
                outStream.print ("\t");
                outStream.print (eventArr[i].odometer);
                outStream.print ("\t");
                outStream.print (eventArr[i].licenceNo);
                outStream.print ("\t");
                outStream.print (eventArr[i].custName);
                outStream.print ("\t");
                outStream.print (eventArr[i].amountPaid);
                outStream.println ();
            }
            outStream.close ();
        } catch (FileNotFoundException e) {
            System.err.println ("Exception: " + e.toString());
            System.exit (1);
        }
    }
}
```