```java
// MachinesClass.java - MACHINES API CLASS
//
// MAINTENANCE HISTORY
// DATE             PROGRAMMER AND DETAILS
// 23-09-14  MPF    Original
//
//---------------------------------------------------------------------------

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

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

// MACHINES API CLASS

public class MachinesClass
implements Machines {

    private static final boolean VERBOSE = false;
                                // Produce verbose log flag

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

    // SKU LIST CLASS

    private class SkuList
    extends ArrayList<String> {
    }

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

    // INVENTORY MAP CLASS

    private class InventoryMap
    extends TreeMap<AccessoryType, SkuList> {
    }

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

    // ACCESSORY MAP CLASS

    private class AccessoryMap
    extends TreeMap<AccessoryType, String> {
    }

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

    // CAR CLASS

    private class Car {
       public StringcarNo;       // Car serial number
       public AccessoryMap newAccessories = new AccessoryMap();
                                // New accessories that are required
       public AccessoryMap oldAccessories = new AccessoryMap();
                                // Old accessories that in the car
```

```
}

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

// ASSEMBLY STATION CLASS

private class Station
extends ArrayList<Car> {
   public boolean      ready;       // Station ready flag
}

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

// STATION MAP CLASS

private class StationMap
extends TreeMap<Integer, Station> {
}

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

// TEST DATA

InventoryMap    inventoryMap;// Inventory map
StationMap             stationMap;  // Station map
List<Car>       carList;     // Full car list
List<String>    errList;     // Error list
Random          r;           // Randomizer

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

// WRITE INFORMATION TO LOG

private void
log (
   String       msg)         // Message
{
   if (VERBOSE)
       System.out.println (msg);
}

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

// WRITE DEBUG INFORMATION TO LOG

private void
debug (
   String       msg)         // Message
{
   if (VERBOSE)
       System.out.println (msg);
}

//---------------------------------------------------------------------
```

```
// GENERATE RANDOM TEST DATA

public void
genRandData (
   String       file)        // Customisation file
throws IOException
{
   StringBuildersb;          // String builder
   int          stationCnt;  // Number of stations
   List<Station>stationList; // Station list
   Station             station;     // Assembly station
   int          carCnt;             // Number of cars
   int          i;           // General purpose index
   String       errVal;             // Error value
   OutputStreamWriter writer;// Output stream writer

   // Initialise class members

   r = new Random(2728537);
   stationMap = new StationMap();
   inventoryMap = new InventoryMap();
   carList = new ArrayList<Car>();
   errList = new ArrayList<String>();

   // Generate the inventory

   genInventory (AccessoryType.RIM);
   genInventory (AccessoryType.TYRE);
   genInventory (AccessoryType.AUDIO);
   genInventory (AccessoryType.GPS);

   // Generate the assembly stations

   stationList = new ArrayList<Station>();
   stationCnt = r.nextInt(5) + 1;
   while (stationCnt-- > 0) {
       station = new Station();
       stationMap.put (r.nextInt(10000), station);
       stationList.add (station);
   }

   // Generate cars
   sb = new StringBuilder ();
   carCnt = r.nextInt(100) + 5;
   while (carCnt-- > 0) {

       // Create a new car

       Car car = new Car();
       sb.append ((char)('A' + r.nextInt(26)));
       sb.append (carCnt);
       sb.append ((char)('A' + r.nextInt(26)));
       sb.append (r.nextInt(1000000));
       sb.append ((char)('A' + r.nextInt(26)));
       car.carNo = sb.toString();
```

```
        sb.setLength (0);

        // Create accessories for car

        createAccessory (car, AccessoryType.RIM);
        createAccessory (car, AccessoryType.TYRE);
        createAccessory (car, AccessoryType.AUDIO);
        createAccessory (car, AccessoryType.GPS);

        // Add car to random station and car list

        station = stationList.get(r.nextInt(stationList.size()));
        station.add (car);
        carList.add (r.nextInt(carList.size()+1), car);
    }

    // Save customization to file

    writer = new OutputStreamWriter(new FileOutputStream(file));
    for (Car car : carList) {

        // Randomly generate car that does not exist

        if (r.nextInt(10) == 0) {
           errVal = "" + r.nextInt(1000000);
           writer.append (errVal);
           errList.add (errVal);

           for (i = 0; i < 4; i++) {
               writer.append (',');
               errVal = "" + r.nextInt(1000000);
               writer.append (errVal);
               errList.add (errVal);
           }

           writer.append ('\n');
        }

        // Write the car to file

        writer.append (car.carNo);
        writer.append (',');
        writer.append (getNewAccessory(car, AccessoryType.RIM));
        writer.append (',');
        writer.append (getNewAccessory(car, AccessoryType.TYRE));
        writer.append (',');
        writer.append (getNewAccessory(car, AccessoryType.AUDIO));
        writer.append (',');
        writer.append (getNewAccessory(car, AccessoryType.GPS));
        writer.append ('\n');
    }
    writer.close();
}

//-------------------------------------------------------------------------
```

```
// GET NEW ACCESSORY IN CAR

private String
getNewAccessory (
   Car          car,         // Car
   AccessoryTypeaccessoryType)      // AccessoryType
{
   String       errVal;             // Error value

   // If accessory changed, write it to customisation file

   if (
       ! car.newAccessories.get(accessoryType).equals(
          car.oldAccessories.get(accessoryType))
   ) {
       return car.newAccessories.get(accessoryType);
   }

   // If accessory not changed, randomly write garbage identifier

   else if (r.nextInt(10) == 0) {
       if (r.nextInt(2) == 0)
          errVal = "" + r.nextInt(1000000);
       else
          errVal = "-";
       errList.add (errVal);
       return errVal;
   }

   return "";
}

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

// CREATE ACCESSORY FOR CAR

private void
createAccessory (
   Car          car,         // Car
   AccessoryTypeaccessoryType)      // AccessoryType
{
   SkuList             skuList;     // SKU list
   Accessory    acccesory;   // Accessory

   // Assign a random accessory

   skuList = inventoryMap.get(accessoryType);
   car.newAccessories.put (accessoryType,
       skuList.get(r.nextInt(skuList.size())));

   // Generate with 50% chance that accessory is not changed

   if (r.nextInt(2) == 0) {
       car.oldAccessories.put (accessoryType,
```

```
            car.newAccessories.get(accessoryType));
      } else {
          car.oldAccessories.put (accessoryType,
            skuList.get(r.nextInt(skuList.size())));
      }
   }

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

   // GENERATE TEST INVENTORY

   public void
   genInventory (
      AccessoryTypeaccessoryType)      // Accessory type
   {
      StringBuildersb;           // String builder
      int           skuCnt;             // Number of SKUs
      String        skuId;       // SKU identifier

      inventoryMap.put (accessoryType, new SkuList());

      sb = new StringBuilder ();
      skuCnt = r.nextInt(100) + 5;
      while (skuCnt-- > 0) {
          sb.append ((char)('A' + r.nextInt(26)));
          sb.append (r.nextInt(1000000));
          sb.append ((char)('A' + r.nextInt(26)));
          sb.append (r.nextInt(1000000));
          sb.append ((char)('A' + r.nextInt(26)));

          skuId = sb.toString();
          inventoryMap.get(accessoryType).add(skuId);
          sb.setLength (0);
      }
   }

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

   // VALIDATE TEST RESULT

   public void
   validateResult (
      String        errFile)     // Error file
   throws Exception
   {
      BufferedReader      reader;             // Reader
      String        line;        // Single-line error message
      String        errVal;             // Error value

      // Make sure all accessories are installed correctly

      for (Car car : carList) {
          if (car.oldAccessories.size() != car.newAccessories.size()) {
            log ("Different equipments " + car.carNo);
          } else {
```

```
            validateAccessory (car, AccessoryType.RIM);
            validateAccessory (car, AccessoryType.TYRE);
            validateAccessory (car, AccessoryType.AUDIO);
            validateAccessory (car, AccessoryType.GPS);
        }
    }

    // Make sure all errors are identifieds

    reader = new BufferedReader(new InputStreamReader(
        new FileInputStream(errFile)));
    while ((line = reader.readLine()) != null) {
        errVal = null;
        for (String err : errList) {
            if (line.endsWith (err)) {
                errVal = err;
                break;
            }
        }
        if (errVal == null)
            log ("Bad error " + line);
        else
            errList.remove (errVal);
    }

    for (String err : errList)
        log ("Error skipped " + err);
}

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

// VALIDATE ACCESSORY

public void
validateAccessory (
    Car          car,         // Car
    AccessoryTypeaccessoryType)      // Accessory type
{
    if (car.oldAccessories.containsKey(accessoryType)) {
        if (!car.newAccessories.containsKey (accessoryType)) {
            log ("Not required " + car.oldAccessories.get(accessoryType));
        } else if (
            !car.oldAccessories.get(accessoryType).equals(
                car.newAccessories.get(accessoryType))
        ) {
            log ("Not required " + car.oldAccessories.get(accessoryType));
            log ("Required " + car.newAccessories.get(accessoryType));
        }
    } else if (car.newAccessories.containsKey (accessoryType)) {
        log ("Required " + car.newAccessories.get(accessoryType));
    }
}

//-------------------------------------------------------------------------
```

```
// GET LIST OF SERVICEABLE ASSEMBLY STATIONS

public int[]
getServiceableStations()
{
   int          i;           // General purpose index
   int[]        stationsArr; // Station identifier array

   i = 0;
   stationsArr = new int[stationMap.size()];
   for (Integer stationId : stationMap.keySet())
          stationsArr[i++] = stationId;
   return stationsArr;
}

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

// TEST WHETHER THE ASSEMBLY STATION IS READY FOR THE NEXT REQUEST

public boolean
isStationReady (
   int          stationId)   // Assembly station identifier
{
   if (! stationMap.containsKey (stationId)) {
       log ("Bad station id " + stationId);
       return false;
   }

   if (r.nextInt(10) == 0)
       stationMap.get(stationId).ready = true;
   return stationMap.get(stationId).ready;
}

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

// GET SERIAL NUMBER OF THE CAR AT THE ASSEMBLY STATION

public String
getCarNumber (
   int          stationId)   // Assembly station identifier
{
   Station             station;     // Assembly station

   if (!stationMap.containsKey (stationId)) {
       log ("Bad station id " + stationId);
       return null;
   }

   station = stationMap.get(stationId);
   if (station.size() == 0)
          return null;

   return station.get(0).carNo;
}
```

```
//----------------------------------------------------------------------

// GET LIST OF ACCESSORIES INSTALLED IN THE CAR AT THE ASSEMBLY STATION

public Accessory []
getAccessories (
   int          stationId)   // Assembly station identifier
{
   Station            station;     // Assembly station
   Car          car;         // Car
   Accessory[]  accessories; // Accessories array
   Accessory    accessory;   // Accessory
   int          i;           // General purpose index

   if (!stationMap.containsKey (stationId)) {
       log ("Bad station id " + stationId);
       return null;
   }

   station = stationMap.get(stationId);
   if (station.size() == 0) {
       log ("No car at station " + stationId);
       return null;
   }

   i = 0;
   car = station.get(0);
   accessories = new Accessory[car.oldAccessories.size()];
   for (
       Map.Entry<AccessoryType,String> accessoryEntry :
          car.oldAccessories.entrySet()
   ) {
       accessory = new Accessory();
       accessory.accessoryType = accessoryEntry.getKey();
       accessory.skuId = accessoryEntry.getValue();
       accessories[i++] = accessory;
   }
   return accessories;
}

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

// TEST WHETHER AN SKU IDENTIFIER IS VALID

public boolean
isValidAccessory (
   AccessoryTypeaccessoryType,      // Accessory type
   String       skuId)       // SKU identifier
{
   return inventoryMap.get(accessoryType).contains(skuId);
}

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

// INSTALL AN ACCESSORY IN THE CAR CURRENTLY AT THE ASSEMBLY STATION
```

```
public void
installAccessory (
   int          stationId,   // Assembly station identifier
   AccessoryTypeaccessoryType,      // Accessory type
   String       skuId)       // SKU identifier
{
   Station             station;     // Assembly station
   Car          car;         // Car

   if (!stationMap.containsKey (stationId))
       log ("Bad station id " + stationId);
   else {
       station = stationMap.get(stationId);
       if (station.size() == 0)
          log ("No car at station " + stationId);
       else if (!station.ready)
          log ("Station not ready " + stationId);
       else if (!isValidAccessory (accessoryType, skuId))
          log ("Invalid accessory " + skuId);
       else {
          car = station.get(0);
          if (car.oldAccessories.containsKey(accessoryType))
              log ("Accessory still installed " +
                 car.oldAccessories.get(accessoryType));
          else if (
              accessoryType == AccessoryType.TYRE &&
              car.oldAccessories.containsKey(AccessoryType.RIM)
          )
              log ("Rim not removed");
          else {
              debug ("Installing " + skuId);
              car.oldAccessories.put(accessoryType, skuId);
              station.ready = false;
          }
       }
   }
}

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

// REMOVE AN ACCESSORY FROM THE CAR CURRENTLY AT THE ASSEMBLY STATION

public void
removeAccessory (
   int          stationId,   // Assembly station identifier
   AccessoryTypeaccessoryType)      // Accessory type
{
   Station             station;     // Assembly station
   Car          car;         // Car

   if (!stationMap.containsKey (stationId))
       log ("Bad station id " + stationId);
   else {
       station = stationMap.get(stationId);
```

```
            if (station.size() == 0)
               log ("No car at station " + stationId);
            else if (!station.ready)
               log ("Station not ready " + stationId);
            else {
               car = station.get(0);
               if (!car.oldAccessories.containsKey(accessoryType))
                   log ("Accessory already removed " + stationId);
               else if (
                   accessoryType == AccessoryType.TYRE &&
                   car.oldAccessories.containsKey(AccessoryType.RIM)
               )
                   log ("Rim not removed");
               else {
                   debug ("Removing " + car.oldAccessories.get(accessoryType));
                   car.oldAccessories.remove(accessoryType);
                   station.ready = false;
               }
            }
         }
      }

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

      // EJECT CAR FROM ASSEMBLY STATION AND GET NEXT CAR IN THE QUEUE

      public void
      eject (
         int          stationId)   // Assembly station identifier
      {
         Station             station;     // Assembly station

         if (!stationMap.containsKey (stationId))
             log ("Bad station id " + stationId);
         else {
             station = stationMap.get(stationId);
             if (station.size() == 0)
               log ("No car at station " + stationId);
             else if (!station.ready)
               log ("Station not ready " + stationId);
             else {
               station.remove(0);
               station.ready = false;
             }
         }
      }
}
```