```
// Station.java - ASSEMBLY STATION CLASS
//
// MAINTENANCE HISTORY
// DATE             PROGRAMMER AND DETAILS
// 23-09-14  MPF    Original
//
//---------------------------------------------------------------------------

import java.util.*;

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

// CLASS DECLARATION

public class Station {

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

    // ASSEMBLY STATION STATE

    public enum StationState {
       EJECTING,
       REMOVING,
       INSTALLING,
       FINISHED,
    }

    int                  stationId;   // Assembly station identifier
    StationState    state;       // Assembly station state
    String          carNo;       // Current car serial number
    Data.AccessoryMap     oldAccessories; // Old accessories
    Data.AccessoryMap     newAccessories; // New accessories

    Data.CarMap           carMap;             // Car map
    Machines        machines;    // Machines API

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

    // CONSTRUCTOR

    public Station(
       int          xStationId,  // Assembly station identifier
       Data.CarMap  xCarMap,     // Car map
       Machines     xMachines)   // Machines API
    {
       stationId = xStationId;
       state = StationState.EJECTING;
       carMap = xCarMap;
       machines = xMachines;
    }

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

    // PROCESS CUSTOMISATION
```

```
public boolean
process ()
{
   // If already finished, just return

   if (state == StationState.FINISHED)
       return true;

   // Process as long as the station is ready

   while (machines.isStationReady(stationId)) {
       switch (state) {
       case EJECTING:
          carNo = machines.getCarNumber (stationId);
          if (carNo == null)
              state = StationState.FINISHED;
          else if (!carMap.containsKey (carNo))
              getNextCar ();
          else {
              oldAccessories = new Data.AccessoryMap();
              for (Accessory acc : machines.getAccessories(stationId))
                 oldAccessories.put (acc.accessoryType, acc.skuId);
              newAccessories = carMap.get(carNo);
              removeNextAccessory ();
          }
          break;

       case REMOVING:
          removeNextAccessory ();
          break;

       case INSTALLING:
          installNextAccessory ();
          break;

       case FINISHED:
          return true;
       }
   }

   return false;
}

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

// GET NEXT CAR

private void
getNextCar ()
{
   carMap.remove (carNo);
   machines.eject (stationId);
   state = StationState.EJECTING;
}
```

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

// REMOVE NEXT ACCESSORY

private void
removeNextAccessory ()
{
   state = StationState.REMOVING;

   // Test if rim needs to be removed

   if (oldAccessories.containsKey (AccessoryType.RIM)) {

       // If either rim or tyre needs to be installed, remove the old rim

       if (
          (
              newAccessories.containsKey (AccessoryType.RIM) &&
              newAccessories.get (AccessoryType.RIM) !=
                 oldAccessories.get (AccessoryType.RIM)
          ) || (
              newAccessories.containsKey (AccessoryType.TYRE) &&
              (
                 ! oldAccessories.containsKey (AccessoryType.TYRE) ||
                 newAccessories.get (AccessoryType.TYRE) !=
                     oldAccessories.get (AccessoryType.TYRE)
              )
          )
       ) {
          // Make sure to put back old rim if no new rim is required

          if (! newAccessories.containsKey (AccessoryType.RIM)) {
              newAccessories.put (AccessoryType.RIM,
                 oldAccessories.get(AccessoryType.RIM));
          }

          // Remove the old rim

          machines.removeAccessory (stationId, AccessoryType.RIM);
          oldAccessories.remove (AccessoryType.RIM);

          return;
       }

       // Do not remove rim that is already installed

       else if (
          newAccessories.containsKey (AccessoryType.RIM) &&
          newAccessories.get (AccessoryType.RIM) ==
              oldAccessories.get (AccessoryType.RIM)
       ) {
          newAccessories.remove (AccessoryType.RIM);
       }
   }
```

```
   if (
       testRemove (AccessoryType.TYRE) &&
       testRemove (AccessoryType.AUDIO) &&
       testRemove (AccessoryType.GPS)
   ) {
       installNextAccessory ();
   }
}

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

// TEST AND REMOVE ACCESSORY IF A DIFFERENT ACCESSORY IS REQUIRED

private boolean
testRemove (
       AccessoryType accType)// Accessory type
{
   if (
       oldAccessories.containsKey (accType) &&
       newAccessories.containsKey (accType)
   ) {
       if (newAccessories.get (accType) == oldAccessories.get (accType))
          newAccessories.remove (accType);
       else {
          machines.removeAccessory (stationId, accType);
          oldAccessories.remove (accType);
          return false;
       }
   }
   return true;
}

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

// INSTALL NEXT ACCESSORY

private void
installNextAccessory ()
{
   state = StationState.INSTALLING;

   if (
       testInstall (AccessoryType.TYRE) &&
       testInstall (AccessoryType.RIM) &&
       testInstall (AccessoryType.AUDIO) &&
       testInstall (AccessoryType.GPS)
   ) {
       getNextCar ();
   }
}

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

// TEST AND INSTALL ACCESSORY IF REQUIRED
```

```
    private boolean
    testInstall (
          AccessoryType accType)// Accessory type
    {
       if (newAccessories.containsKey (accType)) {
          machines.installAccessory (stationId, accType,
             newAccessories.get(accType));
          newAccessories.remove (accType);
          return false;
       }
       return true;
    }
}
```