Το πρότυπο Παρατηρητής (Observer pattern)

igaviotis@gmail.com

0. Το σενάριο του προβλήματος

Μια συσκευή με διάφορους αισθητήρες (sensor device) καταγράφει ερεθίσματα. Για παράδειγμα, μπορεί να είναι

- ένας μετεωρολογικός σταθμός που καταγράφει θερμοκρασία, υγρασία και ατμοσφαιρική πίεση, ή
- ένας συναγερμός που καταγράφει κίνηση, καπνό, φωτεινότητα.

Γύρω-γύρω υπάρχουν οθόνες (display devices) που προβάλουν τις ενδείξεις που παράγει η συσκευή με τους αισθητήρες. Όταν αλλάζει κάποια τιμή στους αισθητήρες της συσκευής, αυτό πρέπει να αντανακλάται σε όλες τις οθόνες.

Γενικότερα, το πρότυπο Παρατηρητής (Observer pattern ονομάστηκε στη Smalltalk τη δεκαετία του 80) εφαρμόζεται όταν ένα συμβάν, π.χ. μια αλλαγή στην κατάσταση μιας οντότητας, επιβάλει την εκτέλεση κάποιας ενέργειας από τα αντικείμενα που έχουν εγγραφεί συνδρομητές. Παράβαλε με μια συνδρομητική υπηρεσία, όπως τη διανομή ενός περιοδικού: όποτε εκδίδεται νέο τεύχος, αποστέλλεται σε όλους τους συνδρομητές...

Το σενάριο-παράδειγμα θα υλοποιηθεί σε διάφορες εκδόσεις:

1) Ορίζει τις κλάσεις που αναπαριστούν τις οντότητες, τα χαρακτηριστικά και τις λειτουργίες τους και με **απλοϊκό** τρόπο θα ενημερώνονται όλες οι οθόνες μετά από κάθε αλλαγή.
2) Εφαρμόζει τη **βασική** ιδέα του προτύπου Παρατηρητής πάνω στον κώδικα του (1).
3) Μια πιο «**καθαρή**» προσέγγιση που δεν έχει τόσο γενική εφαρμογή όπως η (2)
4) Μια προσαρμοσμένη, **βελτιστοποιημένη** προσέγγιση που αναπόφευκτα είναι πιο περίπλοκη.

Όλες οι εκδόσεις είναι πλήρως λειτουργικές και (υπερ)καλύπτουν το σενάριο.

1 Η απλοϊκή έκδοση

Ξεκινάμε με μια quick&dirty λύση, που κωδικοποιεί τη συσκευή αισθητήρων και τις οθόνες, ενώ όλη η ενημέρωση γίνεται με ευθύνη του «χρήστη», εν προκειμένω στη `main()`.

1.1 Κλάση `SensorData`

Για να μην αναφέρονται διάσπαρτα μέσα στον κώδικα συγκεκριμένα είδη αισθητήρων και οι τιμές που έχουν, χάριν γενίκευσης ορίζω μια βοηθητική κλάση `SensorData` που «πακετάρει» όλα τα δεδομένα που παράγουν οι αισθητήρες.

Στη συγκεκριμένη υλοποίηση έχω περιλάβει δυο αισθητήρες που μετράνε θερμοκρασία και υγρασία. Χρειάζεται ένας κατασκευαστής και κάποιου είδους getter, εδώ έχω μια τυπική `toString()` για να επιστρέφει τις τιμές των αισθητήρων. Κατά περίπτωση, αν στο σενάριο υπάρχει να αλλάζει η τιμή ενός από τους αισθητήρες, μπορείς να γράψεις την κλάση `SensorData` με τη γνωστή «κονσέρβα» που περιλαμβάνει τα πεδία, τον κατασκευαστή, getter και setter μεθόδους.

```
public class SensorData {                    //v1 & v2 & v3

  private int   temperature;
  private float humidity;

  public SensorData(int temperature, float humidity) {
      this.temperature = temperature;
      this.humidity    = humidity;
  }

  public String toString() {
    return "(Temperature:" + this.temperature +
           ", Humidity:"   + this.humidity    + ")";
  }

}
```

1.2 Κλάση `SensorDevice`

Κάθε τέτοια συσκευή έχει τις τιμές των αισθητήρων.

Το μόνο που μπορεί να συμβεί στη συσκευή είναι να αλλάξουν οι τιμές των αισθητήρων της, όταν κάποιος καλέσει τη μέθοδο `setMeasurements`. Στη συγκεκριμένη υλοποίηση τα `sensorData` είναι αμετάβλητα (immutable), αλλά αυτό δεν είναι απαραίτητο.

```
public class SensorDevice {                          //v1

  private SensorData sensorData;

  public void setMeasurements(SensorData changedData) {
      this.sensorData = changedData;
  }

}
```

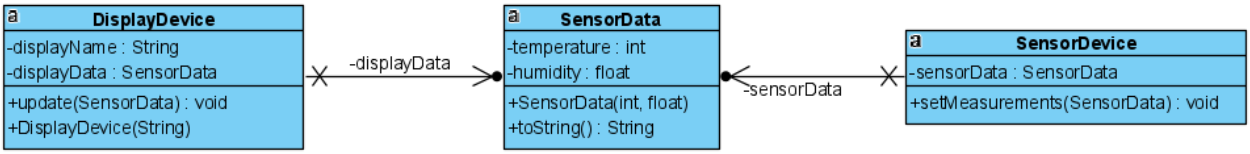
1.3 Κλάση DisplayDevice

Υπάρχουν πολλές οθόνες που προβάλουν τα δεδομένα που παράγει η συσκευή. Για να τις διαφοροποιήσουμε κρατάμε ένα όνομα για την καθεμιά.

Για να ενημερώσει κάποιος τα περιεχόμενα της οθόνης, καλεί τη μέθοδο update. Στην απλούστερη περίπτωση, οι νέες τιμές των αισθητήρων προβάλλονται αυτούσιες, όπως εδώ. Γενικότερα η update μπορεί να προβάλει μια πιο χρήσιμη πληροφορία, πχ. ένα γράφημα, που προκύπτει από τα πρωτογενή δεδομένα των αισθητήρων.

Η δομή της 1ης έκδοσης της λύσης φαίνεται στο διάγραμμα κλάσεων.

```java
public class DisplayDevice {                          //v1

  private final String displayName;
  private SensorData   displayData;

  public DisplayDevice(String displayName) {
    this.displayName = displayName;
    this.displayData = null;
  }

  public void update(SensorData sensorData) {
    this.displayData = sensorData;
    System.out.println(" Device:" + this.displayName +
                       ", Display data:" + this.displayData);
  }

}
```



1.4 Κλάση Main

Για να δοκιμάσουμε την απλοϊκή έκδοση, κατασκευάζω μια συσκευή με αισθητήρες meteoStation και δυο οθόνες.

Έπειτα «πακετάρω» τις ενδείξεις σε αντικείμενα της κλάσης SensorData και ενημερώνεται ο μετεωρολογικός σταθμός. Έπειτα πρέπει να εκτελέσω update() για όλες τις οθόνες.

Όποτε φτάνουν στον μετεωρολογικό σταθμό νέες τιμές, ο προγραμματιστής έχει την ευθύνη να ενημερώνει όλες τις οθόνες.

Το αποτέλεσμα της εκτέλεσης φαίνεται στα γκρι.

```java
public class Main {                                   //v1

  public static void main(String[] args){
    SensorDevice meteoStation = new SensorDevice();
    DisplayDevice masterDisplay = new DisplayDevice("Basement");
    DisplayDevice secondaryDisplay = new DisplayDevice("Attic");

    SensorData reading1 = new SensorData(12, 78.5F);
    meteoStation.setMeasurements(reading1);
    masterDisplay.update(reading1);
    secondaryDisplay.update(reading1);

    SensorData reading2 = new SensorData(10, 75.2F);
    meteoStation.setMeasurements(reading2);
    masterDisplay.update(reading2);
    secondaryDisplay.update(reading2);
  }
}
```

```
Device:Basement, Display data:(Temperature:12, Humidity:78.5)
Device:Attic, Display data:(Temperature:12, Humidity:78.5)
Device:Basement, Display data:(Temperature:10, Humidity:75.2)
Device:Attic, Display data:(Temperature:10, Humidity:75.2)
```

2 Η βασική έκδοση

Θέλουμε κάθε φορά που αλλάζει κάποια τιμή στη συσκευή με τους αισθητήρες, κατευθείαν να ενημερώνονται όλες οι οθόνες. Με άλλα λόγια, όταν τροποποιείται η κατάσταση του αντικειμένου SensorDevice, να εκτελείται «αυτόματα» η μέθοδος update σε όλα τα αντικείμενα DisplayDevice.

Σύμφωνα με την ορολογία του προτύπου Παρατηρητής, η κλάση SensorDevice θα γίνει observable και η κλάση DisplayDevice θα γίνει observer. Η σχέση τους είναι ένα-προς-πολλά, δηλαδή ένα observable μπορεί να έχει πολλούς observers. Οι απαραίτητες λειτουργίες τους θα δηλωθούν σε Java interfaces για να μην επιτρέπονται παρεκκλίσεις.

Στη 2η έκδοση η κλάση SensorData παραμένει αναλλοίωτη, όπως είναι στην ενότητα 1.1.

2.1 Κλάση SensorDevice

Κάθε κλάση που θέλει να αυτοαποκαλείται Observable πρέπει να υλοποιεί τις τρεις λειτουργίες που προσθέτουν, αφαιρούν και ειδοποιούν τους observers για τις αλλαγές που συμβαίνουν στην κατάσταση των αντικειμένων της. Για να το επιβάλλω αυτό, έφτιαξα ένα Java interface.

```java
public interface Observable {                         //v2

  public void registerObserver(Observer o);
  public void deleteObserver(Observer o);
  public void notifyObservers();

}
```

Για τη συσκευή με τους αισθητήρες, ο κώδικας της 1ης έκδοσης δεν αλλάζει, εκτός από τις μπλε προσθήκες.

Βασικά χρειάζεται ένα πεδίο, το `observers`, όπου σε μια συλλογή καταχωρούνται τα αντικείμενα που 'παρατηρούν' τη συσκευή.

Για τη διαχείριση της συλλογής που αναπαριστάνεται με μια `ArrayList`, χρειάζεται ο boilerplate κώδικας των τριών μεθόδων που θα τον γλιτώναμε αν το `Observable` ήταν κλάση αντί για interface. Επειδή όμως εν γένει στο πρότυπο Παρατηρητής τα observables ίσως κληρονομούν κάποια άλλη κλάση και η Java δεν υποστηρίζει πολλαπλή κληρονομικότητα, προς το παρόν έχουμε τη `SensorDevice` να υλοποιεί το interface.

Όταν αλλάξει κάποια τιμή αισθητήρα και κληθεί η `setMeasurement()`, εκτός από την αλλαγή της κατάστασης στο εσωτερικό της συσκευής, καλείται και η `notifyObservers()` που αναλαμβάνει να περάσει το μήνυμα σε όλους τους observers, δηλαδή τις οθόνες.

```java
import java.util.ArrayList;                                    //v2

public class SensorDevice implements Observable {

  private SensorData sensorData;
  private ArrayList<Observer> observers = new ArrayList<>();

  public void setMeasurements(SensorData changedData) {
    this.sensorData = changedData;
    this.notifyObservers();
  }

  public void registerObserver(Observer o) {
    this.observers.add(o);
  }

  public void deleteObserver(Observer o) {
    this.observers.remove(o);
  }

  public void notifyObservers() {
    for (Observer observer : this.observers)
      observer.update(this.sensorData);
  }

}
```

2.2 Κλάση `DisplayDevice`

Κάθε observer που σέβεται τον εαυτό του πρέπει να δέχεται ενημερώσεις ότι κάτι άλλαξε στο αντικείμενο που παρακολουθεί, υλοποιώντας τη λειτουργία `update`. Αυτό εξασφαλίζει το ομώνυμο Java interface.

```java
public interface Observer {                                    //v2

  public void update(SensorData sensorData);

}
```

Το μόνο που χρειάζεται να προστεθεί στην κλάση για τις οθόνες είναι η καταχώριση τους ως παρατηρητές της συσκευής.

Στην τρέχουσα έκδοση *όλα* τα αντικείμενα `DisplayDevice` από κατασκευής είναι παρατηρητές της συσκευής με τους αισθητήρες. Σε εναλλακτικά σενάρια, οι παρατηρητές θα μπορούσαν να καταχωρούνται / αφαιρούνται μέσω των `registerObserver()` / `deleteObserver()` και μάλιστα δυναμικά κατά το χρόνο εκτέλεσης, δηλαδή οποτεδήποτε κατά τη διάρκεια της ζωής τους.

```java
public class DisplayDevice implements Observer {{                    //v2

  private final String displayName;
  private SensorData displayData;

  public DisplayDevice(String displayName, SensorDevice sensorDevice) {
    this.displayName = displayName;
    this.displayData = null;
    sensorDevice.registerObserver(this);
  }

  public void update(SensorData sensorData) {
    this.displayData = sensorData;
    System.out.println(" Device:" + this.displayName +
                       ", Display data:" + this.displayData);
  }

}
```

2.3 Κλάση `Main`

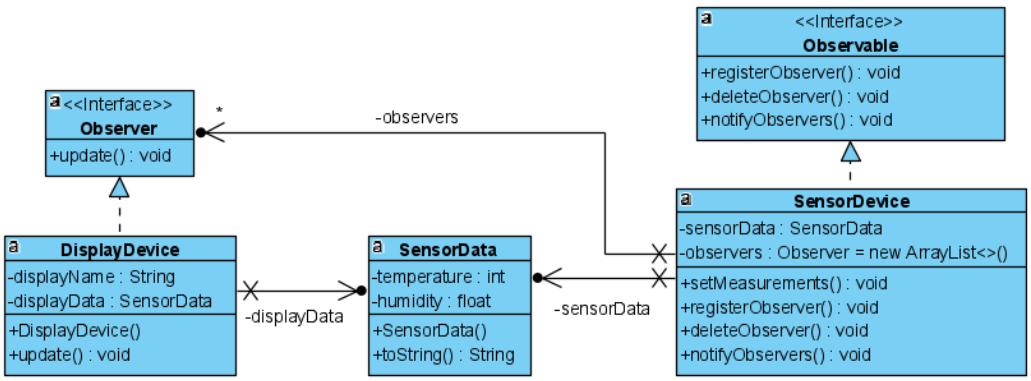
Στη 2η έκδοση που φτιάχνουμε ο κώδικας χρήσης καθαρίζει.

Αρχικά κατασκευάζονται τα αντικείμενα για τη συσκευή και τις οθόνες, όπως ακριβώς και στην 1η έκδοση.

```java
public class Main {                                                  //v2

  public static void main(String[] args){
    SensorDevice meteoStation = new SensorDevice();
    DisplayDevice masterDisplay = new DisplayDevice("Basement", meteoStation);
    DisplayDevice secondaryDisplay = new DisplayDevice("Attic", meteoStation);

    meteoStation.setMeasurements(new SensorData(12, 78.5F));
    meteoStation.setMeasurements(new SensorData(10, 75.2F));
  }

}
```

Έπειτα αρκεί να αλλάξουν οι τιμές των αισθητήρων – αμέσως ενημερώνονται όλες οι οθόνες, χωρίς δεύτερη κουβέντα, δηλαδή χωρίς έξτρα εντολές στον κώδικα χρήσης (αυτές υπάρχουν και εκτελούνται μέσα στον κώδικα του observable. Μια ευθύνη λιγότερη για τον προγραμματιστή, τώρα που εφαρμόστηκε το πρότυπο Παρατηρητής.. Το αποτέλεσμα της εκτέλεσης είναι πανομοιότυπο με αυτό της 1ης έκδοσης.

Για καλύτερη επόπτευση του κώδικα της 2ης έκδοσης, να και το διάγραμμά της. Σημειώστε τη χρήση των interface.



3 Η καθαρή έκδοση

Στην έκδοση 3 έχω τροποποιήσει ελαφρώς την υλοποίηση της έκδοσης 2 μετατρέποντας το interface `Observable` σε κλάση, οπότε η κλάση `SensorDevice` κληρονομεί, αντί να υλοποιεί. Αισθητικά αυτό είναι καθαρότερο, διότι ο boilerplate κώδικας με τα μπλε στην ενότητα 2.1 αφαιρείται από την «παραγωγική» κλάση `SensorDevice`.

```
public class SensorDevice extends Observable {    //v3

  private SensorData sensorData;

  public void setMeasurements(SensorData changedData) {
    this.sensorData = sensorData;
    super.notifyObservers(changedData);
  }

}
```

```
import java.util.ArrayList;                    //v3

public class Observable {

  private ArrayList<Observer> observers =
                          new ArrayList<>();

  public void registerObserver(Observer o) {
    this.observers.add(o);
  }

  public void deleteObserver(Observer o) {
    this.observers.remove(o);
  }

  public void notifyObservers(SensorData
                                sensorData) {
    for (Observer observer : this.observers)
      observer.update(sensorData);
  }

}
```

Οι υπόλοιπες τρεις κλάσεις παραμένουν όπως ήταν στην έκδοση 2.

Εν γένει, για την ενσωμάτωση του προτύπου Παρατηρητής με αυτή την προσέγγιση, αρκούν δυο παρεμβάσεις στον κώδικα της εφαρμογής:

- η observable κλάση να καλεί το `super.notifyObservers(changedData)`
- η observer κλάση να έχει εγγραφεί με `sensorDevice.registerObserver(this)` και θα λαμβάνει τις ειδοποιήσεις στην `update()`

4 Η βελτιστοποιημένη έκδοση με ειδοποιήσεις pull

Σε όλες τις υλοποιήσεις μέχρι εδώ, όποτε αλλάζει η κατάσταση της συσκευής με τους αισθητήρες ή γενικότερα του observable, το ίδιο αναλαμβάνει να σπρώξει (push) την ειδοποίηση σε όλους τους observers καλώντας την `update()` τους και αυτή η ειδοποίηση περιλαμβάνει την πλήρη κατάσταση του observable. Ξεκάθαρα υπάρχουν περιθώρια βελτιστοποίησης:

- Μπορεί οι observers να χρειάζεται να ειδοποιηθούν μόνον για συγκεκριμένες αλλαγές της κατάστασης, π.χ. όταν η παλιά και η νέα κατάσταση διαφέρουν πάνω από κάποιο κατώφλι τιμών. Αυτό επιφέρει μείωση του *πλήθους* των μηνυμάτων.
- Μπορεί να μην χρειάζεται να αποσταλεί ολόκληρο το πακέτο της κατάστασης του observable. Αυτό επιφέρει μείωση του *όγκου* των μηνυμάτων.

Στην τρέχουσα έκδοση το observable καθορίζει πότε θα αποσταλεί η ειδοποίηση ότι συνέβη κάποια σημαντική αλλαγή. Οι observers αντλούν την πληροφορία που τους ενδιαφέρει (μοντέλο pull) αντί να είναι παθητικοί αποδέκτες.

Φαντάσου τους συνδρομητές ενός περιοδικού που ενώ ελάμβαναν σώνει και καλά όλα τα νέα τεύχη και με το πλήρες περιεχόμενό τους, τώρα ειδοποιούνται για την έκδοση του νέου τεύχους και επιλέγουν να λάβουν μόνο το περιεχόμενο που τους ενδιαφέρει.

Στη συγκεκριμένη υλοποίηση δεν χρησιμοποιείται η `SensorData` επειδή οι οθόνες μπορούν να παίρνουν μόνο ό,τι χρειάζονται από τις τιμές των αισθητήρων.

4.1 Το υποκείμενο της παρατήρησης

Στη βοηθητική κλάση `Observable` το μόνο που αλλάζει είναι ότι το μήνυμα που αποστέλλεται στους παρατηρητές δεν περιλαμβάνει τις τιμές που άλλαξαν, έτσι ώστε εκείνοι να ζητήσουν τις τιμές που χρειάζονται. Γι' αυτό η `update()` δεν κουβαλάει τις τιμές που άλλαξαν – απλώς ανακοινώνει ότι έγινε κάποια αλλαγή.

```
import java.util.ArrayList;                                    //v4

public class Observable {

  private ArrayList<Observer> observers = new ArrayList<>();

  public void registerObserver(Observer o) {
    this.observers.add(o);
  }

  public void dele
    this.observers
  }

  public void notifyObservers() {
    for (Observer observer : this.observers)
      observer.update(); //send empty message to all
   }

}
```

```
Device:Basement, Display data: (12, 78.5)
Device:Attic, Display data: (12, 78.5)
Device:Basement, Display data: (10, 75.2)
Device:Attic, Display data: (10, 75.2)
```

Ενημερώνουμε και το σχετικό interface.

```
public interface Observer {    //v4

  public void update();

}
```

Καθώς βλέπεις, η κλάση `Observable` είναι πλήρως επαναχρησιμοποιούμενη, δεν εξαρτάται σε τίποτα από το πεδίο εφαρμογής, δηλαδή το χώρο του προβλήματος.

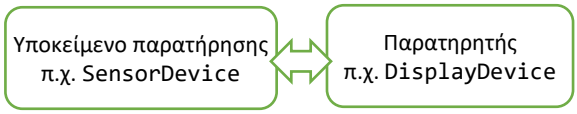
Στην κλάση `SensorDevice` το υποκείμενο της παρατήρησης κάνει μια προσπάθεια να περιορίσει τη φλυαρία του. Αντί να ενοχλεί τους παρατηρητές για οποιαδήποτε αλλαγή στην κατάσταση, στέλνει το (κενό) μήνυμα υπό προϋποθέσεις, π.χ. όταν η υγρασία έχει αλλάξει κατά τουλάχιστον 1 βαθμό ή όταν η θερμοκρασία είναι διαφορετική.

```
public class SensorDevice extends Observable {                 //v4

  private final static float humidityThreshold = 1.0F;

  private int   temperature;
  private float humidity;

  public int getTemperature() { return this.temperature; }

  public float getHumidity() { return this.humidity; }

  public void setMeasurements(int temperature, float humidity) {
    boolean sendUpdate = false;
    if ((Math.abs(this.humidity - humidity) >= humidityThreshold) ||
        (this.temperature != temperature))
      sendUpdate = true;
    this.temperature = temperature;
    this.humidity    = humidity;
    if (sendUpdate) super.notifyObservers();
  }

}
```

4.2 Οι παρατηρητές

Πάμε στην κλάση `DisplayDevice`. Οι οθόνες εδώ διατηρούν μια αναφορά προς τη συσκευή που παρακολουθούν, ώστε να μπορούν να της ζητήσουν τις τιμές που χρειάζονται. Έτσι έχουμε αμφίδρομη σύνδεση μεταξύ του υποκειμένου της παρατήρησης και των παρατηρητών. Πέρα από την ομορφιά της συμμετρίας, αυτό ανοίγει νέες δυνατότητες, π.χ. θα μπορούσε ένας παρατηρητής να βρίσκει ποιοι άλλοι παρατηρητές παρατηρούν το ίδιο υποκείμενο.

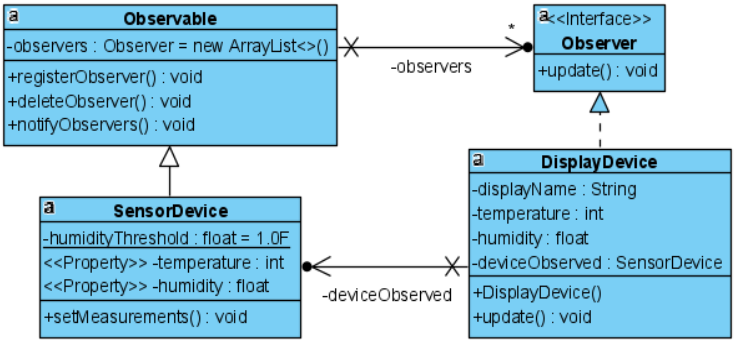
```
public class DisplayDevice implements Observer {           //v4

  private final String displayName;
  private SensorDevice deviceObserved;
  private int          temperature;
  private float        humidity;

  public DisplayDevice(String displayName,
                       SensorDevice deviceObserved) {
    this.displayName = displayName;
    this.deviceObserved = deviceObserved;
    deviceObserved.registerObserver(this);
  }

  public void update() {
    this.temperature = this.deviceObserved.getTemperature();
    this.humidity    = this.deviceObserved.getHumidity();
    System.out.println(" Device:" + this.displayName +
                       ", Display data: (" + this.temperature +
                       ", " + this.humidity + ")");
  }

}
```



Στην κλάση `Main` της 4ης έκδοσης ο μετεωρολογικός σταθμός είναι το υποκείμενο παρατήρησης. Κατασκευάζονται δυο οθόνες που τον παρατηρούν, λαμβάνουν μηνύματα για τις δυο πρώτες αλλαγές και αντλούν τις τιμές που θέλουν.

Η τρίτη `setMeasurements()` αλλάζει ελάχιστα την υγρασία και έτσι δεν πυροδοτεί αποστολή μηνύματος, όπως (δεν) φαίνεται και στα γκρι αποτελέσματα.

Το διάγραμμα δείχνει την κλάση και το interface που περιλαμβάνουν όλο τον boilerplate κώδικα και τις κλάσεις `SensorDevice` (υποκείμενο παρατήρησης) και `DisplayDevice` (παρατηρητές). Στο υποκείμενο καθορίζεται η πολιτική αποστολής των ειδοποιήσεων που πυροδοτούνται με τη `notifyObservers()` και οι παρατηρητές αντλούν τα δεδομένα που τους χρειάζονται, υπό την προϋπόθεση ότι έχουν εγγραφεί στη συνδρομητική υπηρεσία με τη `registerObserver()`.




```java
public class Main {                                    //v4

  public static void main(String[] args){
    SensorDevice meteoStation = new SensorDevice();
    new DisplayDevice("Basement", meteoStation);
    new DisplayDevice("Attic", meteoStation);

    meteoStation.setMeasurements(12, 78.5F);
    meteoStation.setMeasurements(10, 75.2F);
    meteoStation.setMeasurements(10, 75.5F);
  }

}
```

Παρεμπιπτόντως, η Java παρέχει ενσωματωμένες `java.util.Observable` και `Observer` με δυνατότητες push και pull, αλλά είναι υπό απόσυρση (depreciated) ήδη από την έκδοση 9. Σε κάθε περίπτωση, ο κώδικας εδώ επέτρεψε να κατανοήσουμε την εφαρμογή του προτύπου Παρατηρητής και η μελέτη των παραλλαγών μας εξοπλίζει με εναλλακτικές υλοποίησής του ώστε να ταιριάζει στις ιδιαίτερες απαιτήσεις του σεναρίου.

5 Σύνοψη

Το πρότυπο Παρατηρητής (Observer), που αναφέρεται και ως Publish-Subscribe, εφαρμόζεται όταν πολλά αντικείμενα πρέπει να ειδοποιηθούν επειδή συνέβη κάτι. Παρέχει το μηχανισμό για την αποστολή των μηνυμάτων και επιτρέπει την διακίνηση του περιεχομένου, είτε μέσω push από το υποκείμενο της παρατήρησης, είτε μέσω pull από τους παρατηρητές.

Επιτυγχάνει ασθενή σύζευξη μεταξύ των συμμετεχόντων και δυναμική συμμετοχή στην παρακολούθηση.

Αναφορές

- Eric Freeman, Elisabeth Robson, *Head First Design Patterns,* O' Reilly, 2014
- Craig Larman, *Applying UML and patterns,* Pearson, 2004