Assignement: Java Beans and Reflection API

Exercise 1

Since you are a pretty busy person you cannot always take care of the flowers in your garden. Indeed, to avoid further "planticides" you decide to build an irrigation system for your flowers.

The system is composed by

1. a soil moisture sensor, reporting on the relative humidity of the soil;
2. a controller that starts and stops the irrigation based on the data from the sensor and on the user input;
3. a graphical dashboard that visualises information from the sensor and allows to manually control the irrigation system.

Each of the three items just listed has to be implmented as a JavaBean, according to the following specifications.

Part 1: The Sensor

The soil moisture sensor has to be implemented as a non-visual Java bean called `MoistureSensor`, in a new NetBeans project named `MoistureSensor` and in the package `moisturesensor`.

This bean has a `boolean` property `decreasing`, an `int` property `currentHumidity` which takes values in the range [0,100], and a method `void start()` that activates the sensor.

When activated, the sensor reads periodically, ie. every second, the current humidity from the soil and makes it available as the value of `currentHumidity`. If `decreasing` is `true` the humidity must decrease progressively (but it never becomes negative), and if `decreasing` is `false` the humidity increases (but it never passes 100). Use a generator of random numbers to simulate this behaviour in some way.

Both properties `currentHumidity` and `decreasing` must be *bound properties*, ie, they must must fire `PropertyChange` events to inform registered listeners when their value is changed.

Export the bean `MoistureSensor` in the `jar` file named `MoistureSensor.jar`.

Part 2: The Controller

The controller is a non-visual Java bean called `Controller`, belonging to a project with the same name in a package called `controller`.

It implements locally the irrigation logic, using a `boolean` property `on` and an `int` property `locHumidity`. Using events, the controller starts the irrigation (setting `on` to `true`) as soon as the value of `locHumidity` falls under 30%, and it stops it (setting `on` to `false`) when the humidity reaches 90%.

Part 3: The Graphical Dashboard

Import the Sensor and the Controller beans from their jar files inside the palette of NetBeans. Create the project `IrrigationDashboard` and a class `DashboardFrame` that extends `JFrame`.

Add to this frame two `JLabel`, one `JButton`, the `MoistureSensor` and the `Controller`. Using events, the dashboard must connect the `Controller` with the `MoistureSensor`, exploiting the other beans to start the sensor and to display its status. More precisely, when the user clicks on the button the `MoistureSensor` starts sensing, and the two labels show the values of the properties `currentHumidity` and `decreasing` of the sensor.

To do that:

1. link the property `currentHumidity` of `MoistureSensor` with the property `text` of one of the `JLabel` and with the property `locHumidity` of `Controller`
2. link the property `decreasing` of `MoistureSensor` with the property `text` of the other `JLabel`;
3. link the change of the `on` property of the `Controller` with the value of `decreasing` of `MoistureSensor`.

Export the project in the jar file `IrrigationDashboard.jar`.

Solution format: Three adequately commented source files (`MoistureSensor.java`, `Controller.java` and `IrrigationDashboard.java`) and the three corresponding jar files.

Exercise 2: Using the Java Reflection API

Write a Java program `CheckNPE` that takes as command line arguments the names of other Java classes (either fully qualified names, like `java.lang.String`, or simple names, like `MyJavaClass`; in the latter case it is assumed that file `MyJavaClass.class` is in the same directory of `CheckNPE.class`).

For each class `C` passed as argument, the program behaves as follows. For each method or constructor declared in `C` which has at least one parameter of a reference type, `CheckNPE` checks if it is **NPE sensible**, that is, if invoking the constructor or method with default parameter values a `NullPointerException` is thrown.

The default value of a numeric primitive type parameter is `0`, of a boolean parameter is `false`, of a reference type parameter is `null`.

For each class name passed as parameter, `CheckNPE` must print for each declared constructor or method (1) the name, (2) the list of parameter types and, only if it has at least one parameter of a reference type, (3) if it is NPE sensible or not.

Solution format: An adequately commented source java file.

Exercise 3: [Optional] Manual irrigation

Reusing as much as possible the code developed for Exercise 1 (exploting inheritance or reuse of beans) develop a dashboard providing all the features of Part 3 of Exercise 1, and in addition a button supporting *manual irrigation*. When the user clicks the button, the value of the `on` property

of the `Controller` is flipped, but only if the value of its `locHumidity` permits it: `false` becomes `true` (irrigation starts) only if `locUmidity` is smaller than 60%, and `true` becomes `false` (irrigation stops) only if the humidity is larger than 50%.

The constraints just described have to be implemented by exploiting the `VetoableChangeListener` pattern. In the documentation of the exercise, describe precisely the design choices including (1) who plays the role of vetoing the change when the condition is not satisfied, and (2) how did you reuse the code of Exercise 1

Solution format: Adequately commented source java files, including the required documentation, and jar file of the project.

Date: 2018-10-23T00:14+0200

Author: Andrea Corradini & Matteo Busi

Org version 7.9.3f with Emacs version 24