Exercise Sheet 11

January 11: MVC

Exercise 1

The goal of this exercise is to implement a simplistic MVC where the input/output device is just the console! Therefore, implement the following classes or interfaces:

1. A class `Model` that maintains a String as its internal state
2. An interface `Observer` with the usual `public void update(String state)` method
3. A class `ConsoleView` that implements the `Observer` interface and prints to the console whenever receiving an update
4. A class `Controller` that glues together `ConsoleView` and `Model` and is capable of receiving input events from the command line, i.e., standard in. It should read a complete line from standard in and update its model accordingly. In effect it implements a read eval loop. The code starting this loop should be put into a method `start`.

Finally write some test case roughly as follows

```
Model model = new Model();
Controller controllerOne = new Controller(model, new ConsoleView(1));
Controller controllerTwo = new Controller(model, new ConsoleView(2));
controllerOne.setState("3");
controllerTwo.setState("42");
controllerTwo.start();
```

Experiment with multiple views and controllers.

Exercise 2

Download the head first `headfirst.combined.djview` package and compile it! Follow the flow when setting BPM and starting (file → start) the sound! Draw a sequence diagram!

Exercise 3

a) Extend `headfirst.combined.djview` by implementing a second view! It displays the BPM graphically using traffic lights (0–60: red, 61–120: green, 121–180: yellow) and must work simultaneously with `DJView`!

b) Similarly, implement a second view and controller using `javax.swing.JSlider` enabling to set BPM by sliding a knob within a bounded interval. This controller should work side-by-side (i.e. in parallel) with `BeatController` and `DJView`![1]

Exercise 4

Implement the Web Model2 pattern for `DJView` as described/suggested in Head First p. 549ff!

Exercise 5

Have a look at the `http://www.jhotdraw.org/` framework and download the source code! Please answer the following questions!

1. Which classes correspond to models?

[1] You have to solve the problem of the views notifying each other, however if updates always flow through the shared model this should happen automatically!

2. Which classes correspond to views?
3. Which classes correspond to controllers?

Hints

- Consult the literature!
- You can work in pairs, if you want!
- If you want to learn a Java API, look into the java docs!
- Always use the same familiar IDE (suggestion Eclipse)!