AP Assignment **I** / Ex 1
EightPuzzle with Java Beans

Francesco Lorenzoni

July 2024

Contents

Chapter 1

EightPuzzle with Java Beans

1.1 About folders content

The assignment development was tracked using git. The repository is available at github.com/frenzis01/AP-assignments. The folder of the first exercise of the assignment is `assignment1/ex3_NetBeans`. I initially numerated it as 3 by mistake, but I kept it that way (both in the github repo, and in the email-submitted archive, for "consistency") to avoid erasing the git history.

The folder contains only the class files, for ease of reading. There is also the `.jar` file (renamed to `.raj`) which may be tested as it is. There is also a .zip archive resulting from the export of the NetBeans project, which contains the source code and the NetBeans project files; it may be imported in NetBeans to navigate and test project from the IDE.

1.2 Main Design Choices

Components are exactly the ones required by the assignment: `EightTile`, `EightBoard`, `EightController` and `Flip`, with the only addition of a simple wrapper class `IntWrapper` under the `utils` package.

1.2.1 `position` and `label` attributes

It is important to keep in mind that `EightTile` components are positioned as in Fig. 1.1, mirroring the disposition of the tiles in the assignment description. Every `EightTile` is instantiated with the expected corresponding `position`, i.e. `eightTile6` → `eightTile6.position = 6`.

`EightTiles` *never* "move" in the board and neither their `final` `position` attribute ever changes, only the `label` attribute is changed throughout the game, mimicking the sliding of the tiles.

`label` is private, there is no getter and no setter. It is changed only when invoking `labelRequest()` (which fires a vetoable change) and upon receival of `"swapOK"` and `"restart"` events.

Figure 1.1: `EightTile` components organized in the `EightBoard` grid

1.2.2 Observer pattern

The interaction between components is based on events and listeners, following the Observer pattern. `EightTile.label` is a *Constrained Property* which fires a `PropertyChangeEvent` by invoking `fireVetoableChange(...)` on the `EightController` instance, which is responsible for ensuring that only legal changes are allowed. The three scenarios for a `label` change are:

- *Tile got clicked*: clicked `EightTile` may move to the *hole* (i.e. `EightTile` whose `label == 9`), if it is adjacent to

it.
- ◇ *Flip button got clicked*: tiles in position 1 and 2 are swapped, if the *hole* is in the middle (`position == 5`)
- ◇ *Restart button got clicked*: tiles receive a permutation and infer their new label; in this case, such change is not vetoed.

1.2.3 Events

1.2.3.1 Clicking a Tile

When an `EightTile` gets clicked it attempts to swap its position to the hole. To do so, sets the property `requestedLabel = 9` (i.e. hole), and executes `EightTile.labelRequest(9)` which fires a `PropertyChangeEvent` with `propertyName = "label"`, `oldValue = position` and `newValue = 9`.

> **Bending the rules**
>
> Considering the signature Lst. 1.1 below, we might say that the parameters are not exactly used as their name suggests, and we'd probably be right ☺.
> In fact `oldValue` is used to pass the *position* of the tile, while the `newValue` is used to pass the new *label* of the tile.
> I chose to pass the position in the event to avoid making the Controller check the position of the requesting tile using methods (e.g. `((EightTile)pce.getSource()).getPosition()`) as the assignment requires.
>
> It is needed to wrap position because in case `oldValue == newValue` the event is not fired, and it is not unlikely that `position == newLabel`.

```
// Signature of fireVetoableChange
fireVetoableChange(String propertyName, Object oldValue, Object newValue)
// How it is used in EightTile
this.mVcs.fireVetoableChange(propertyName, new IntWrapper(this.position), newLabel);
```

Listing 1.1: How `fireVetoableChange` is used in `EightTile`

`EightController` which a `vetoableChangeListener`, checks if the requesting tile is adjacent to the hole, and if so, it allows the change, otherwise throws an exception. In case an `EightTile` successfully moves, fires an `ActionEvent("swapOK")`, which is listened by all other tiles, so that the tile —typically the *hole*, unless it's a *Flip*— which has been swapped can update its label. The requested label is obtained by listener by invoking on the event source `getClientProperty("requestedLabel")`.

7 tiles out of 8 listen to the event uselessly, the clicked tile could send it only to the hole, but it would require for it to have a reference to the tile currently holding the hole; for simplicity and readability, I chose to make all tiles listen to the event.

1.3 Flip Button