# The 8 Puzzle

## Programming Assignment #1 Exercise 1

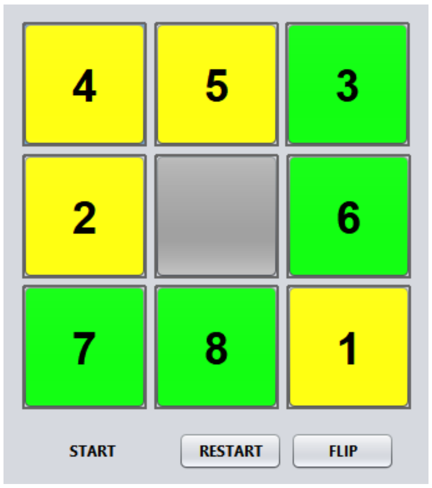
**Emiliano Sescu**

Prepared for Advanced Programming course, Date 11/08/2024

**Exercise text**

The 8 puzzle is a reduced version of the more famous 15 puzzle. Starting from a random configuration of the tiles, the puzzle consists of reaching the final configuration with a sequence of moves. Each move consists of sliding one tile on the hole, thus exchanging the positions of that tile and the hole.

We want to implement the 8 puzzle using *Java Beans*. The project will only provide a usable board to try to solve the puzzle, it will not address its automatic resolution. The system is made of a graphical dashboard, `EightBoard`, containing the board, an `EightController` label, and two buttons: `RESTART` and `FLIP`, as shown in the figure below. The board is made of 9 buttons, which must be instances of a Java bean called `EightTile`, and that we will call tiles in the following. Each tile implements therefore the same logic. Conceptually, the `EightBoard` dashboard displays the board, while the `EightController` label monitors that only legal moves are triggered by clicking the tiles.



## 1 Requirements

### 1.1 EightTile

Tiles must inherit from `JButton`. A tile has (at least) two **private** properties: `Position` and `Label`, which hold integer values in the range [1, 9]. Position is a constant: it is set at startup by the constructor, and it identifies a specific position on the board. `Label` is, as property, **both bound and constrained** (in the JavaBean's sense). The background color of a tile must be grey if the `Label` is 9 (we call this tile the *current hole*), green if `Position` = `Label` (and they are different from 9), and yellow otherwise. The text of the tile must be equal to the value of `Label`, with the exception of the current hole, whose text is empty. Background color and text of each tile must change if `Label` is changed, in order to preserve these constraints. Note that the in this way the final configuration is the only one where all tiles except the hole are green.

When the player clicks on a tile ("I want to slide that tile to the hole"), the `Label` property is changed to 9 (i.e., the cell becomes itself the current hole), ***if this change is not vetoed*** [1]. Only if the change is not vetoed, the tile must pass, in some way, its previous label value to the current hole, which must change its label to that value. If instead the change is vetoed, the tile "flashes", for example by changing the color to red for half a second. Tiles must also provide support for a "restart" action, with a method that takes as argument a permutation of [0,9] and sets the label to an initial value. This action will be triggered by the `Restart` event of the board, thus tiles have to register as listeners for this event.

[1]By the logic of the game, the change of the Label property to 9 should be vetoed if the tile is the current hole, or if the tile is not adjacent to the current hole

## 1.2 EightController

The `EightController` is a bean that has the graphical appearance of a label (e.g., it can extend `JLabel`). It has to check that only legal moves are performed. To this aim, it must be registered as `VetoableChangeListener` to all the tiles. At the beginning the controller displays "**START**". Every time a tile is clicked, it must veto the change if the tile is the hole or it is not adjacent to the hole, displaying "**KO**". Otherwise, it must display "**OK**". The controller must also provide support for a "restart" action that restores its internal information about the configuration when the game is restarted.

## 1.3 EightBoard

The main class of the application is the `EightBoard` dashboard, that must be defined as extending `JFrame`. It dispays a grid of 3x3 tiles, an `EightController` label, a `RESTART` button, and a `FLIP` button, as shown in the figure above. When the `RESTART` button is clicked, a random configuration (i.e., a random permutation of [0, 9]) is passed to all beans registered as listener to the `Restart` event. At startup the board initializes the grid with the nine `EightTile` beans, passing to them their position as initial value of `Position`. All tiles and the controller are registered as listeners to the `Restart` event, and such an event is fired to complete the initialization of the board with a random configuration. Also, the controller is registered as `VetoableChangeListener` to all the tiles.

## 1.4 Flip button

In the 8 (or 15) puzzle, from half of the initial configurations the final one cannot be reached with any sequence of moves. This can be frustrating for the player. The `Flip` button of the dashboard, when clicked, switches the labels of tiles in position 1 and 2, but only if the hole is in position 9, otherwise it has no effect. Switching the position of two non-hole tiles guarantees that if the previous configuration was not solvable, the new one is.

## 2 Design decisions

Given that "event-based communication is more appreciated", in this solution communications between Java beans are event-based. This is facilitated by two utility classes:

- `PropertyChangeSupport`: Used for **bound** properties, which notify listeners when their values change. It manages listeners and dispatches `PropertyChangeEvents` to inform them of any updates.
- `VetoableChangeSupport`: Used for **constrained** properties, which require approval before changing. It allows listeners to veto a change by throwing a `PropertyVetoException` if the change is not acceptable.

This section will mainly focus on describing how event-based communication was implemented for every bean. To read in depth description of each method, refer to the JavaDoc generated from NetBeans, available in the "target/site/apidocs" folder.

### 2.1 EightBoard

After initializing the components, each tile registers the controller as a `VetoableChangeListener` to handle move vetoes, specifically to veto "*moveCheck*" events. Additionally, each tile registers its adjacent tiles as `PropertyChangeListener` to receive "*labelChange*" events when a tile is clicked, as only adjacent tiles need to be updated based on these changes.

The board itself registers all tiles as `PropertyChangeListener` to handle "*restart*" and "*flip*" events, which are triggered by the `initializeBoard()` and `flipBoard()` methods, respectively. These methods are invoked when the `RESTART` and `FLIP` buttons are clicked.

Moreover, the board registers the controller as a `PropertyChangeListener` to update the controller after a "*restart*" event, since it tracks the hole position. The controller also serves as a `VetoableChangeListener` to validate if a flip operation is permissible.

## 2.2 EightController

The `EightController` Java Bean implements both `VetoableChangeListener` and `PropertyChangeListener`, necessitating the overriding of the following methods:

- `vetoableChange(...)`: it is responsible for vetoing "*moveCheck*" and "*flip*" events. It checks if the proposed changes are permissible (if not, a `PropertyVetoException` is thrown), following the requirements defined in the previous section.
  Note that the label of the tiles are passed as parameters of the fired event.
- `propertyChange(...)`: it updates the controller's internal information about the current hole position when a "*restartHole*" event is fired.

## 2.3 EightTile

The `EightTile` Java Bean implements `PropertyChangeListener`, overriding the `propertyChange(...)` method to manage update events. This method handles the "*labelChange*" event, which is fired after a tile is clicked and the move has been validated by the controller. If the move is vetoed, the `PropertyVetoException` will be catched and the tile flashes red for half a second, if not, it ensures the tile's label is updated accordingly. It also handles the "*restart*" and the "*flip*" events to ensure the tile's labels are correctly adjusted.

## 2.4 Flip button

When the `FLIP` button is clicked, the `flipBoard()` method of the `EightBoard` is invoked. This method first fires a `vetoableChange` event named "*flip*". Since the controller is registered as a `VetoableChangeListener`, it handles the event by checking the current hole position. If the hole is in position 9, the flip is permitted, and the controller updates its label to "FLIPPED". If the hole is not in position 9, the flip is vetoed, the label displays "NOT ALLOWED", and a `PropertyVetoException` is thrown and caught within the `flipBoard()` method.

If the flip is allowed, the method then fires a `PropertyChange` event also named "*flip*". The tiles, registered as `PropertyChangeListener`, receive this event. Only the first and second tiles respond by swapping their labels to reflect the flip request.