Software Engineering in Java

Swing and MVC

fausto.spoto@univr.it

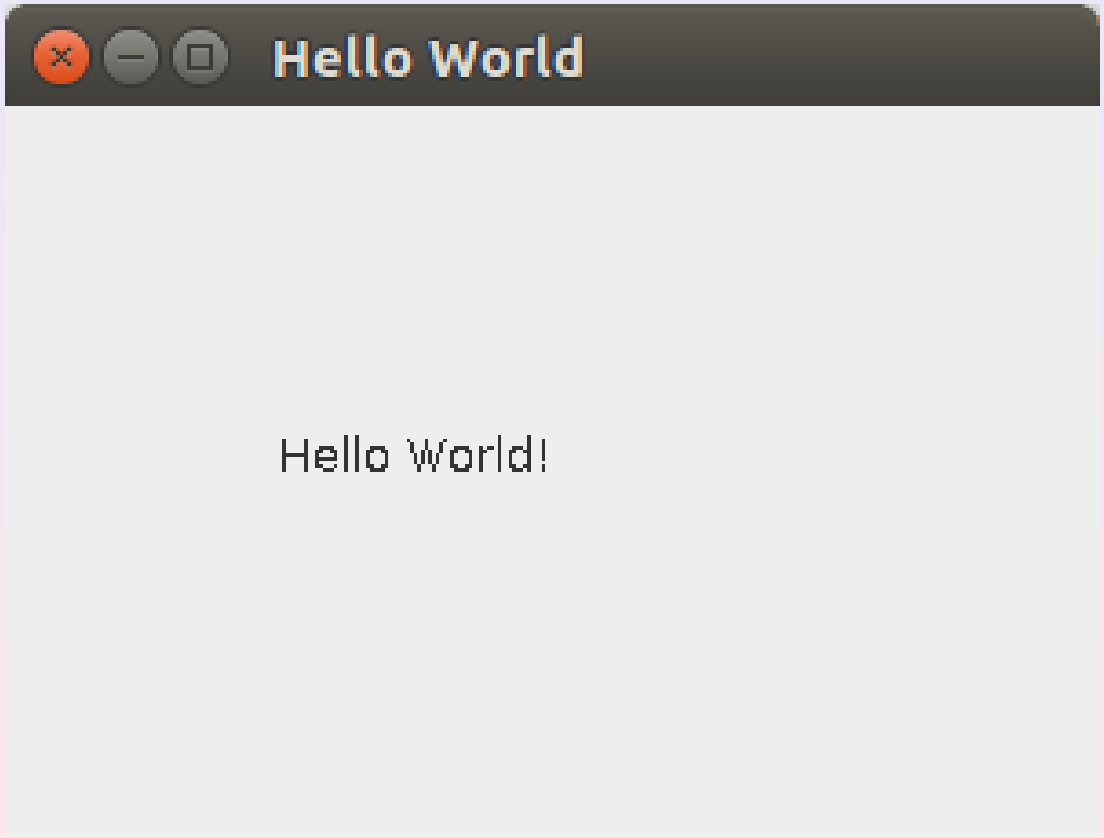
A Design Pattern for Graphical Applications

Graphical applications interact with the user through widgets

- windows
- buttons
- labels
- text fields
- sliders
- menus

The Swing library implements such components through classical design patterns: strategy, composite, decorator ...

A First Example of the Use of Swing



Running Swing Code

Swing calls must happen inside the UI thread:

```
public class HelloWorldMain {

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {

            @Override
            public void run() {
                JFrame frame = new HelloWorldFrame();
                frame.setTitle("Hello World");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
            }
        });
    }
}
```

A Frame implements a Window and Contains Components

```
public class HelloWorldFrame extends JFrame {

    public HelloWorldFrame() {
        add(new HelloWorldComponent());
        pack();
    }
}
```

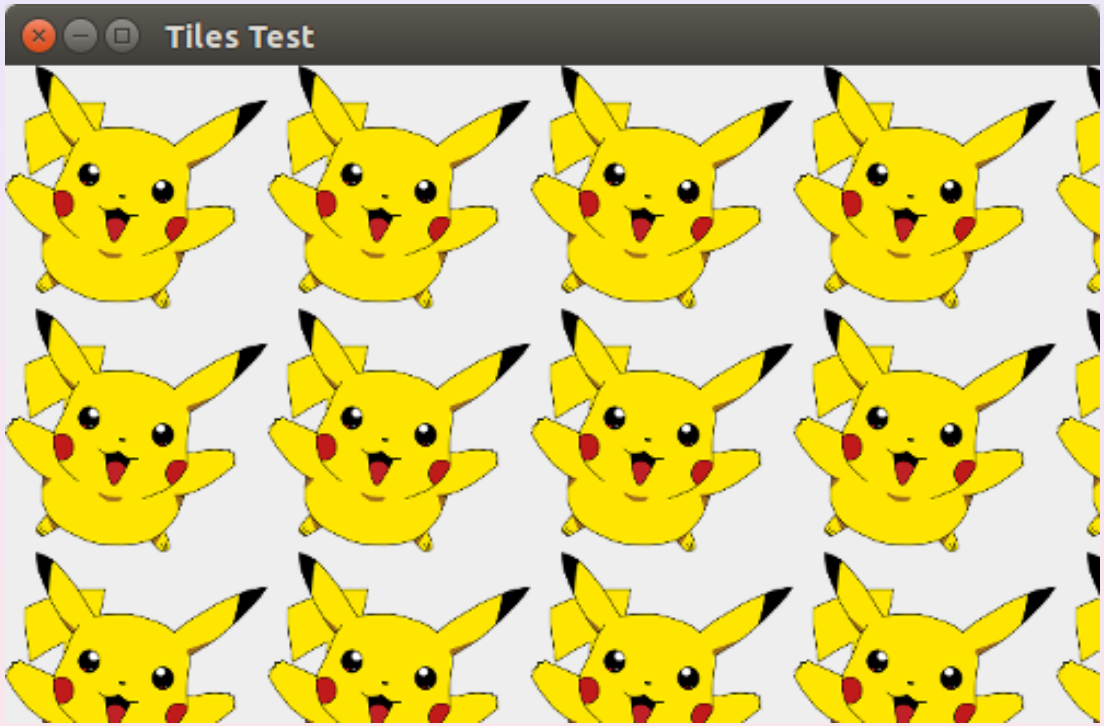
A Component Knows how to Draw Itself

```
public class HelloWorldComponent extends JComponent {
    public final static int MESSAGE_X = 75;
    public final static int MESSAGE_Y = 100;
    public final static int DEFAULT_WIDTH = 300;
    public final static int DEFAULT_HEIGHT = 200;

    @Override
    protected void paintComponent(Graphics g) {
        g.drawString("Hello World!", MESSAGE_X, MESSAGE_Y);
    }

    @Override
    public Dimension getPreferredSize() {
        return new Dimension(DEFAULT_WIDTH, DEFAULT_HEIGHT);
    }
}
```

A Second Example of the Use of Swing



A Second Example of the Use of Swing

```
public class TilesComponent extends JComponent {
    public final static int DEFAULT_WIDTH = 500;
    public final static int DEFAULT_HEIGHT = 300;
    private final Image image;

    public TilesComponent() {
        image = new ImageIcon("img/pika.png").getImage();
    }

    @Override
    public Dimension getPreferredSize() {
        return new Dimension(DEFAULT_WIDTH, DEFAULT_HEIGHT);
    }
```

A Second Example of the Use of Swing

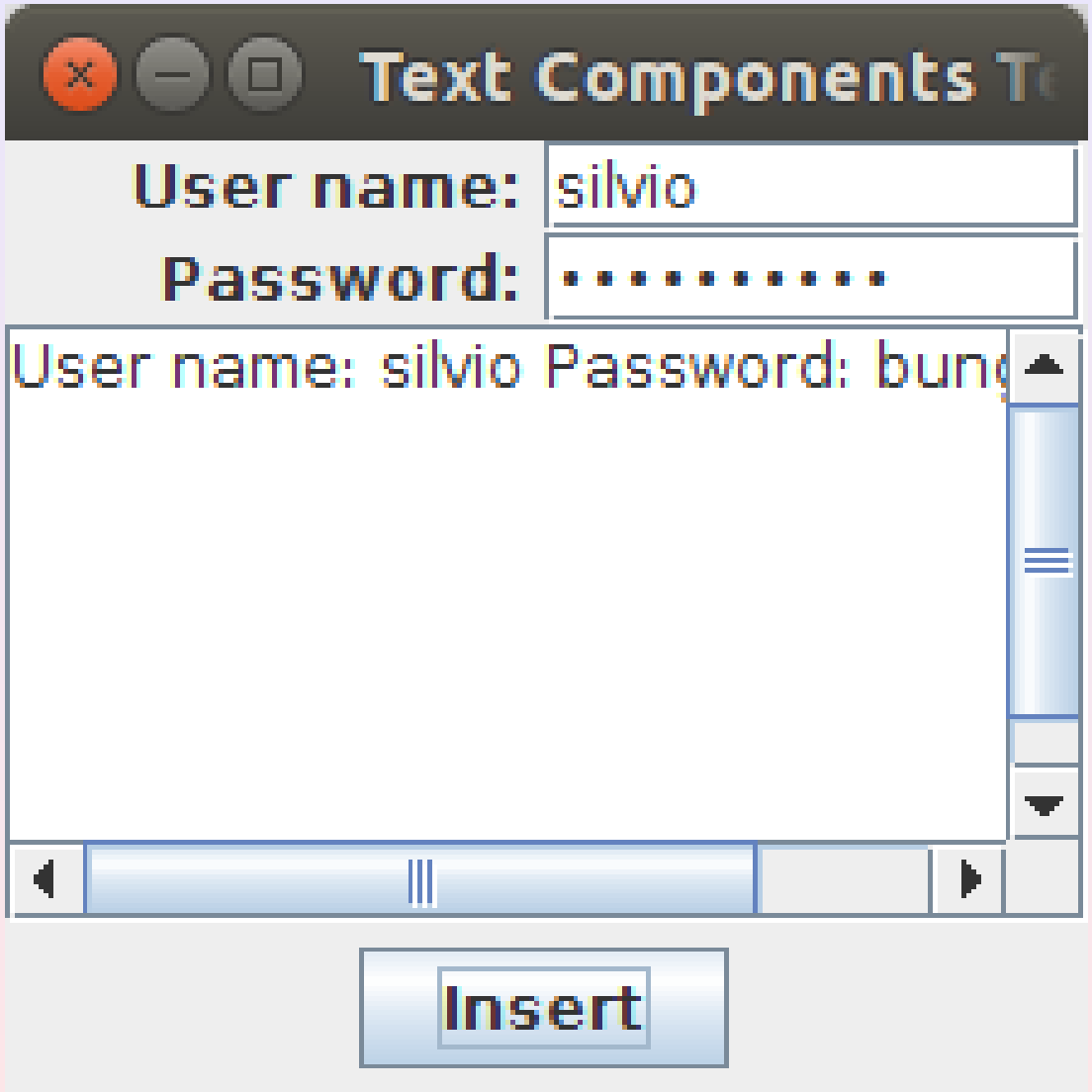
```
@Override
protected void paintComponent(Graphics g) {
    if (image == null)
        return;

    int imageWidth = image.getWidth(this);
    int imageHeight = image.getHeight(this);

    g.drawImage(image, 0, 0, null);

    for (int i = 0; i <= getWidth(); i+= imageWidth)
        for (int j = 0; j <= getHeight(); j+= imageHeight)
            g.copyArea(0, 0, imageWidth, imageHeight, i, j);
}
```

A Third Example of the Use of Swing: Library Components



TextFields, Labels, Container Panels...

```
public class TextComponentFrame extends JFrame {

    private static final long serialVersionUID = 1L;
    public static final int TEXTAREA_ROWS = 8;
    public static final int TEXTAREA_COLUMNS = 20;

    public TextComponentFrame() {
        JTextField textField = new JTextField();
        JPasswordField passwordField = new JPasswordField();

        JPanel northPanel = new JPanel();
        northPanel.setLayout(new GridLayout(2, 2));
        northPanel.add(new JLabel("User name: ", JLabel.RIGHT));
        northPanel.add(textField);
        northPanel.add(new JLabel("Password: ", JLabel.RIGHT));
        northPanel.add(passwordField);

        // a frame has by default the border layout
        add(northPanel, BorderLayout.NORTH);
```

Text Areas, Scroll Bar Decorators...

```
JTextArea textArea = new JTextArea(TEXTAREA_ROWS, TEXTAREA_COLUMNS);
JScrollPane scrollPane = new JScrollPane(textArea);

add(scrollPane, BorderLayout.CENTER);
```

Interactive Components

```
JButton insertButton = new JButton("Insert");
insertButton.addActionListener(actionListener);
```

A listener specifies the behavior of the click on the button:

```
public interface ActionListener extends EventListener {

    /**
     * Invoked when an action occurs.
     */
    public void actionPerformed(ActionEvent e);

}
```

The Hollywood Principle: Don't call me, I'll call you

Listeners as Explicit Classes (Name Pollution)

```
    JButton insertButton = new JButton("Insert");
    insertButton.addActionListener
        (new MyListener(textField, textArea, passwordField));
}

private class MyListener implements java.awt.event.ActionListener {
    private JTextField textField;
    private JTextArea textArea;
    private JPasswordField passwordField;

    private MyListener(JTextField textField,
            JTextArea textArea, JPasswordField passwordField) {
        this.textField = textField;
        this.textArea = textArea;
        this.passwordField = passwordField;
    }

    @Override
    public void actionPerformed(ActionEvent event) {
        textArea.append("User name: " + textField.getText() +
            " Password: " + new String(passwordField.getPassword()) + "\n");
    }

}
```

Listeners as Local Classes (Still too Long)

```
class MyListener implements java.awt.event.ActionListener {
    @Override
    public void actionPerformed(ActionEvent event) {
        textArea.append("User name: " + textField.getText() +
            " Password: " + new String(passwordField.getPassword()) + "\n");
    }
}

JButton insertButton = new JButton("Insert");
insertButton.addActionListener(new MyListener());
```

Listeners as Anonymous Local Classes (Ok-ish)

```
JButton insertButton = new JButton("Insert");
insertButton.addActionListener(new java.awt.event.ActionListener() {
    @Override
    public void actionPerformed(ActionEvent event) {
        textArea.append("User name: " + textField.getText() +
            " Password: " + new String(passwordField.getPassword()) + "\n");
    }
});
```

Listeners as Lambdas (only Java 8) (great!)

```
JButton insertButton = new JButton("Insert");
insertButton.addActionListener(
    event -> textArea.append("User name: " + textField.getText() +
        " Password: " + new String(passwordField.getPassword()) + "\n"));
```

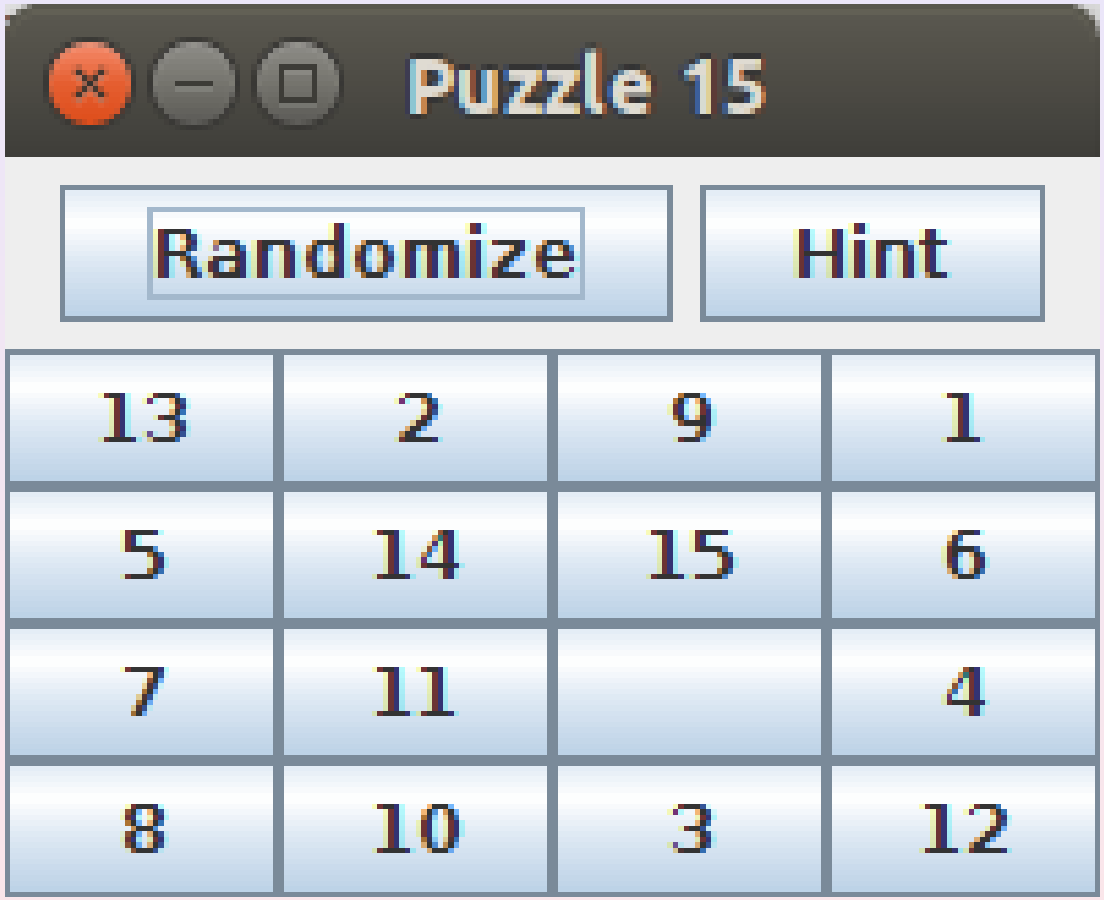
Buttons

```
JPanel southPanel = new JPanel();
JButton insertButton = new JButton("Insert");
// a panel has by default the flow layout
southPanel.add(insertButton);
insertButton.addActionListener(
    event -> textArea.append("User name: " + textField.getText() +
        " Password: " + new String(passwordField.getPassword()) + "\n"));

add(southPanel, BorderLayout.SOUTH);

pack();
```

A Serious Swing Application!



Separation of Concerns

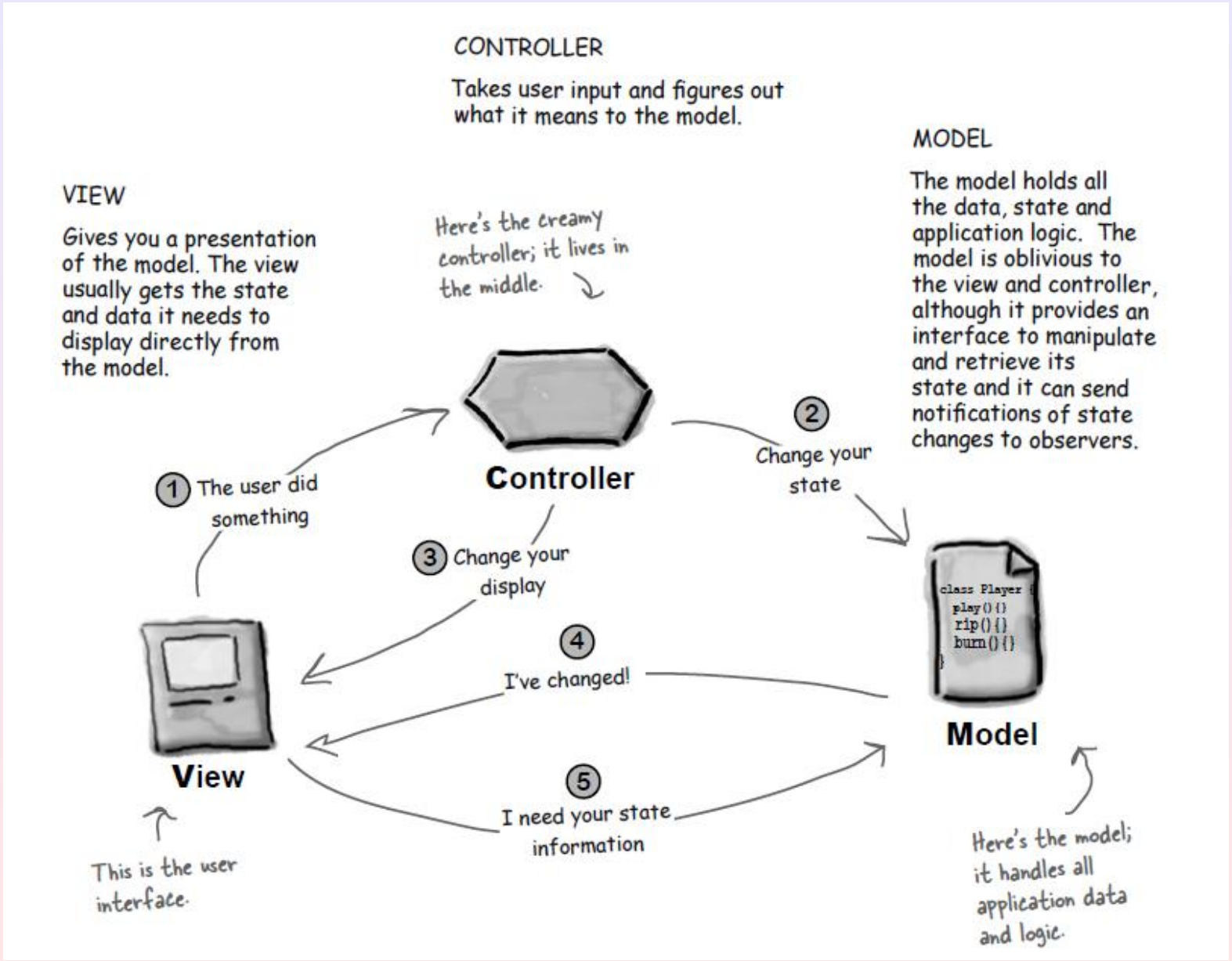
The graphical interface should be kept separate from the logic:

- for desktop
- for Android
- for special accessibility

Data should be kept separate from the logic:

- faster on desktop
- more compact on mobile
- kept in a database
- accessible through a web interface

Model-View-Controller Design Pattern



The Organization into Packages

- `model`: data representation
- `view`: game views
- `controller`: data/view coordination
- `moves`: rules for tile movements
- `ai`: artificial intelligence

Rule of thumb

Classes from one package should only import interfaces from other packages

The Model

```
public interface Model {
    // 5: I need your state information
    public int at(int x, int y);
    public Configuration getConfiguration();

    // 2: change your state
    public void setConfiguration(Configuration configuration);
    public void setView(View listener);
}
```

The View

```
public interface View {
    Model getModel();
    void setController(Controller controller);

    // 3: change your display
    void showSolvedDialog();

    // 4: I've changed
    void onConfigurationChange();
}
```

The Controller

```
public interface Controller {
      // 1: the user did something
      void onClick(int x, int y);
      void randomize();
      void giveHint();
}
```

The Model Contains a Configuration

```
public interface Configuration {

     * Yields the label on the tile at the given coordinates

    int at(int x, int y);

     * Yields a new configuration where a tile has been swapped

    Configuration swap(int fromX, int fromY, int intoX, int intoY);
}
```

Many implementations are possible: bidimensional arrays are the most obvious

The fluent interface supports immutable configurations!

The Model of the Puzzle 1/2

```
public class TilesModel implements Model {
    private Configuration configuration;
    private View view;

    public TilesModel(Configuration configuration) {
        this.configuration = configuration;
    }

    public int at(int x, int y) {
        return configuration.at(x, y);
    }

    public Configuration getConfiguration() {
        return configuration;
    }
```

The Model of the Puzzle 2/2

```
public void setConfiguration(Configuration configuration) {
    if (this.configuration != configuration) {
        this.configuration = configuration;
        if (view != null)
            view.onConfigurationChange();
    }
};

@Override
public void setView(View view) {
    this.view = view;
}
```

The View of the Puzzle 1/3

```
public class TilesPanel extends JPanel implements View {
    private final JFrame frame;
    private final Model model;
    private Controller controller;
    private final JButton[][] buttons = new JButton[4][4];

    public TilesPanel(Model model, JFrame frame) {
        this.frame = frame;
        this.model = model;

        createButtons();

        model.setView(this);
    }

    @Override
    public Model getModel() {
        return model;
    }
```

The View of the Puzzle 2/3

```
private void createButtons() {
    setLayout(new GridLayout(4, 4));

    for (int y = 0; y < 4; y++)
        for (int x = 0; x < 4; x++)
            add(buttons[x][y] = mkButton(x, y, model.at(x, y)));
}

private JButton mkButton(int x, int y, int value) {
    JButton button = new JButton(value == 0 ? "" : String.valueOf(value));
    button.addActionListener(event -> {
        if (controller != null)
            controller.onClick(x, y);
    });

    return button;
}
```

The View of the Puzzle 3/3

```
@Override
public void setController(Controller controller) {
    this.controller = controller;
}

@Override
public void onConfigurationChange() {
    for (int y = 0; y < 4; y++)
        for (int x = 0; x < 4; x++)
            buttons[x][y].setText(model.at(x, y) == 0 ?
                "" : String.valueOf(model.at(x, y)));
}

@Override
public void showSolvedDialog() {
    new SolvedDialog(frame, controller).setVisible(true);
}
```

The Controller of the Puzzle 1/2

```
public class Puzzle15Controller implements Controller {
    private final View view;
    private final Mover mover;
    private final Solver solver;

    public Puzzle15Controller(View view) {
        this.view = view;
        this.mover = new Mover(view.getModel());
        this.solver = new Solver(view.getModel());

        view.setController(this);
    }

    @Override
    public void onClick(int x, int y) {
        mover.moveAt(x, y);
        if (mover.isSolved())
            view.showSolvedDialog();
    }
```

The Controller of the Puzzle 2/2

```
@Override
public void giveHint() {
    solver.step();
    if (mover.isSolved())
        view.showSolvedDialog();
}

@Override
public void randomize() {
    do {
        mover.randomize();
    }
    while (mover.isSolved());
}
```

Wiring and Plumbing 1/2

```
public class Puzzle15Frame extends JFrame {
    private final TilesModel model = new TilesModel(new ArrayBackedConfiguration());
    private final Controller controller;

    public Puzzle15Frame() {
        setTitle("Puzzle 15");

        View view = addTiles();
        controller = new Puzzle15Controller(view);
        controller.randomize();
        addControlButtons();

        setIconImage(new ImageIcon("img/puzzle15.jpg").getImage());

        pack();
    }
```

Wiring and Plumbing 2/2

```
private void addControlButtons() {
    JPanel panel = new JPanel();

    JButton randomize = new JButton("Randomize");
    randomize.addActionListener(event -> controller.randomize());
    panel.add(randomize);

    JButton hint = new JButton("Hint");
    hint.addActionListener(event -> controller.giveHint());
    panel.add(hint);

    add(panel, BorderLayout.NORTH);
}

private View addTiles() {
    TilesPanel panel = new TilesPanel(model, this);
    add(panel, BorderLayout.CENTER);

    return panel;
}
```

A Bit of Intelligence

How can the computer suggest moves that solve the game?

There are clever and fast techniques taht use relatively complex algorithms. Implementing such algorithms requires time and supporting libraries for special data structures

We follow another approach

We first implement a naive algorithm, that immediately explodes, computationally. Then we improve it through heuristics

How the Working Set of Configurations Works

ws: c_0

c_0 has four successor configurations c_1, c_2, c_3, c_4

How the Working Set of Configurations Works

ws: $\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}, \mathbf{c_4}$

c_1 has two successor configurations c_5, c_6

How the Working Set of Configurations Works

ws: $c_2, c_3, c_4, \mathbf{c_5}, \mathbf{c_6}$

c_2 has six successor configurations $c_7, c_8, c_9, c_{10}, c_{11}, c_{12}$

How the Working Set of Configurations Works

ws: $c_3, c_4, c_5, c_6, \mathbf{c_7}, \mathbf{c_8}, \mathbf{c_9}, \mathbf{c_{10}}, \mathbf{c_{11}}, \mathbf{c_{12}}$
c_3 has. . .

This algorithm terminates if an already seen configuration is not put back in the working set again. **Eventually**, the solved configuration must appear at the head of the working set queue and the algorithm stops

Every time a new configuration is pushed into the working set, we store somewhere the parent configuration that originated it, so that at the end we can reconstruct the sequence of configurations towards the solved one

There is an exponential number of configurations!

Implementing the Working Set Algorithm

```
public Solution(Configuration configuration) {
    Map<Configuration, Originator> seen = new HashMap<>();
    Queue<Configuration> workingSet = new LinkedBlockingQueue<>();
    workingSet.offer(configuration);
    seen.put(configuration, new Originator(configuration, 0));

    do {
        Configuration current = workingSet.poll();
        Rules rules = new Rules(current);
        if (rules.isSolved()) {
            steps = mkSteps(current, seen, null);
            return;
        }
        for (Configuration next: rules.nextConfigurations()) {
            Originator oldWay = seen.get(next);
            int newSteps = seen.get(current).steps + 1;
            if (oldWay == null || oldWay.steps > newSteps) {
                workingSet.offer(next);
                seen.put(next, new Originator(current, newSteps));
            }
        }
    }
    while (!workingSet.isEmpty());
}
```

A Comparator as Strategy Specification

```
workingSet = new LinkedBlockingQueue<>();
```

The algorithm runs into out of memory on a 8 gigabytes computer

A Comparator as Strategy Specification

```
workingSet = new PriorityQueue<>(comparator);
```

With a comparator that embeds *all the following three* strategies, the algorithms terminates in less than a second on a slow machine:

1. lowest tiles first
2. ordered tiles first
3. if the topmost two rows are solved, it is better

Memory Considerations

Despite a clever comparator, the cost in memory is around 4 gigabytes to find a solution, because configurations are expensive bidimensional arrays:

```
public class ArrayBackedConfiguration extends Abstrac
    private final int[][] tiles;

    private ArrayBackedConfiguration(int[][] tiles) {
        this.tiles = new int[4][];
        for (int y = 0; y < 4; y++)
            this.tiles[y] = tiles[y].clone();
    }
```

A much better solution packs the information into a long and requires around 200K to find a solution:

```
public class LongBackedConfiguration extends Abst
    private long tiles;

    private LongBackedConfiguration(long tiles) {
        this.tiles = tiles;
    }
```