Software Engineering in Java

Swing and MVC

fausto.spoto@univr.it

git@bitbucket.org:spoto/software-engineering-in-java-examples.git

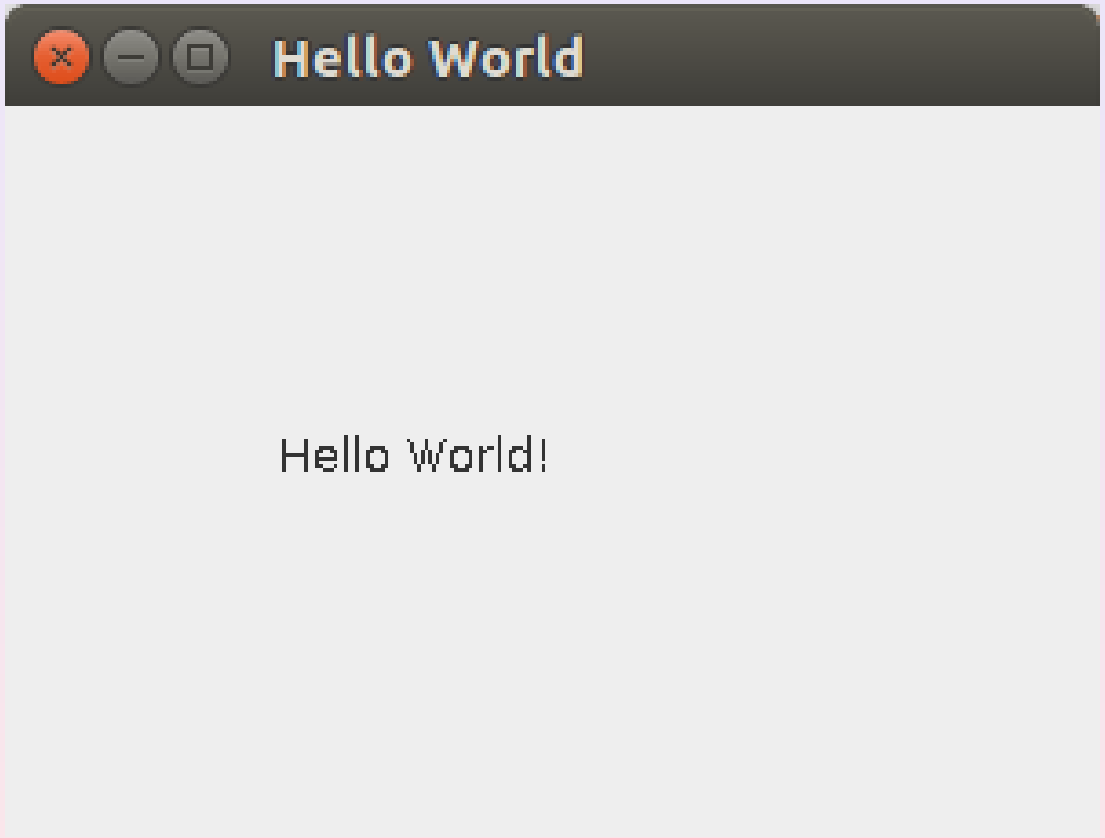
Graphical Applications

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);
            }
        });
    }
}
```

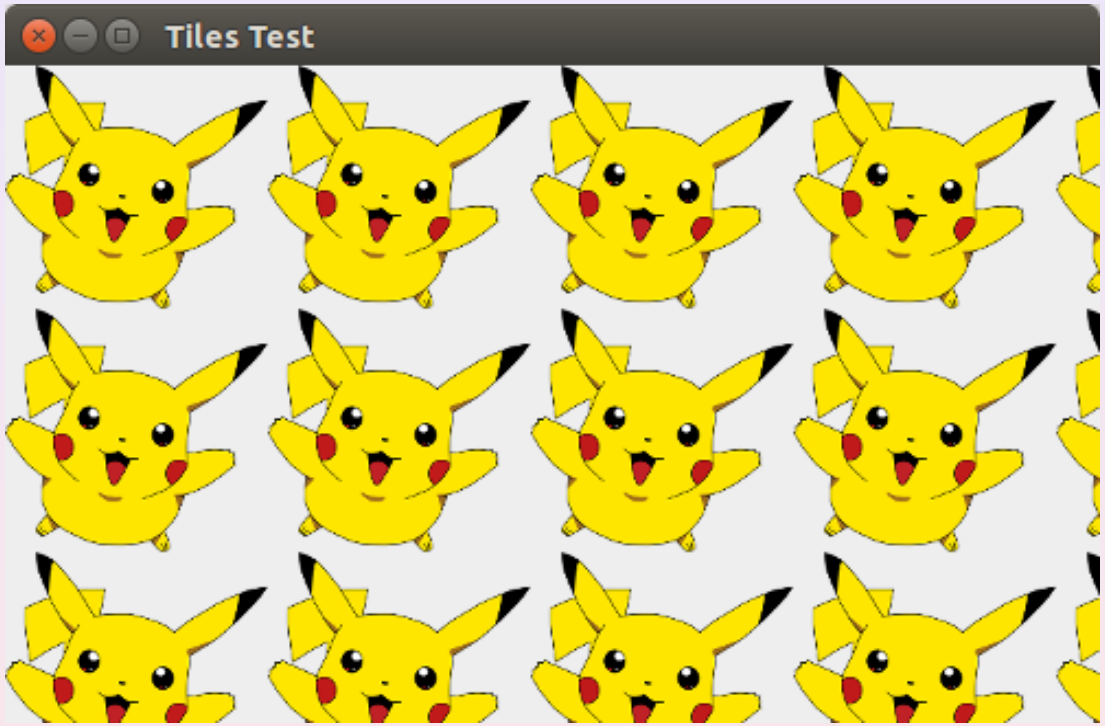
A Frame implements a Window and Contains Components

```
public class HelloWorldFrame extends JFrame {

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

A Component Knows how to Draw Itself

```java
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



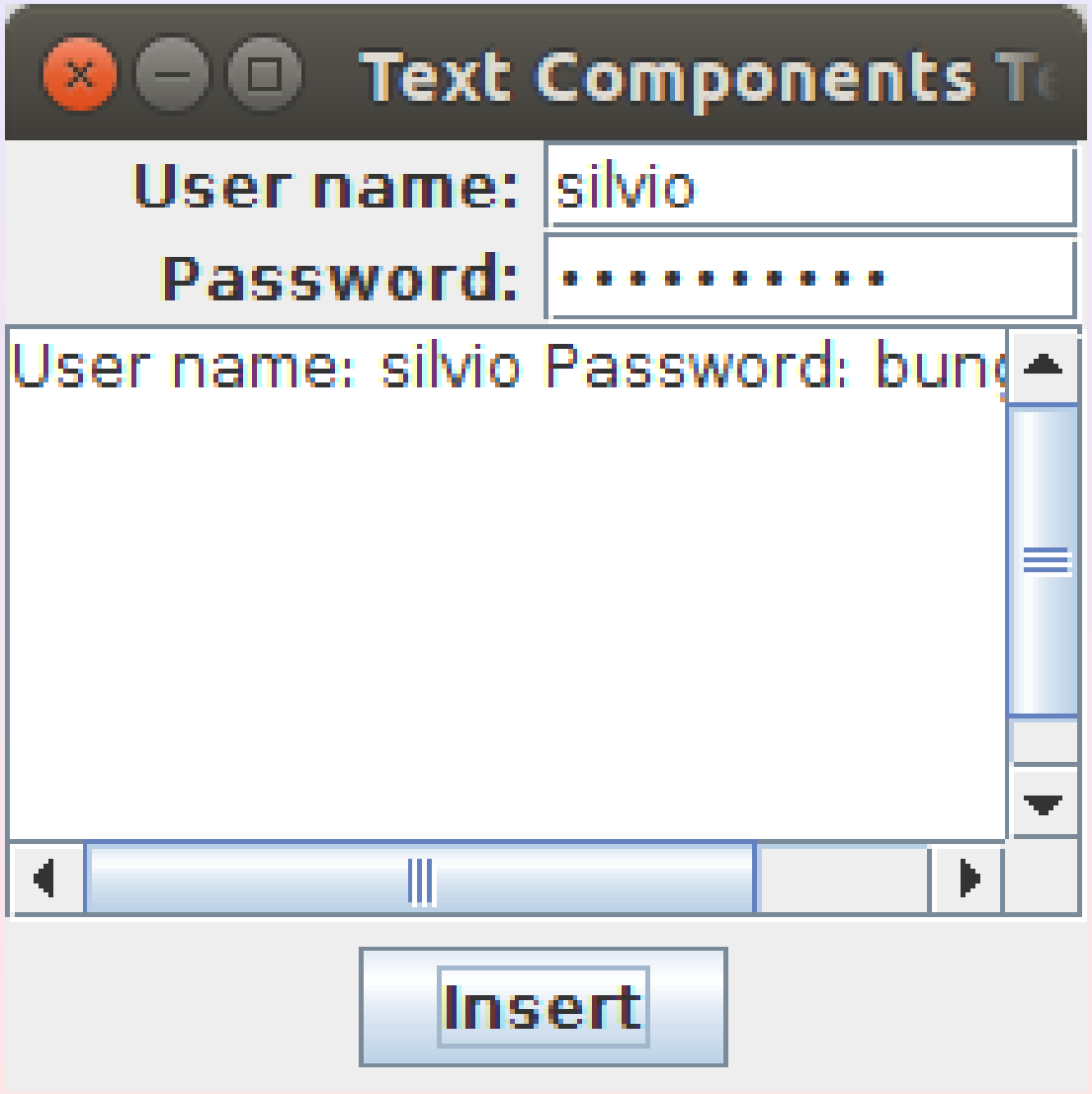
A Second Example of the Use of Swing

```
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 {
    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 Inner 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 Inner 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");
    }
});
```

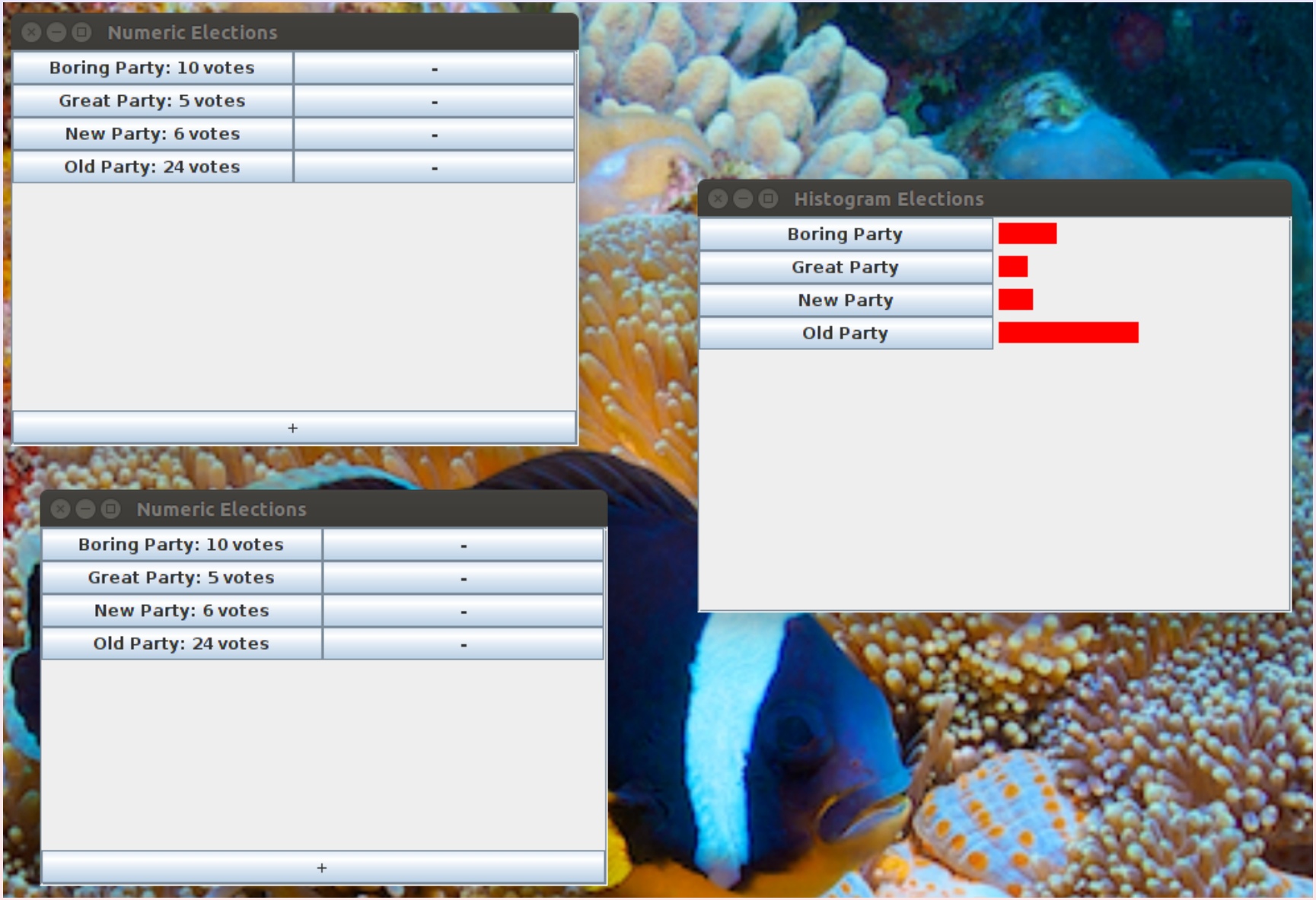
Listeners as Lambdas (only Java 8) (great!)

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

Buttons

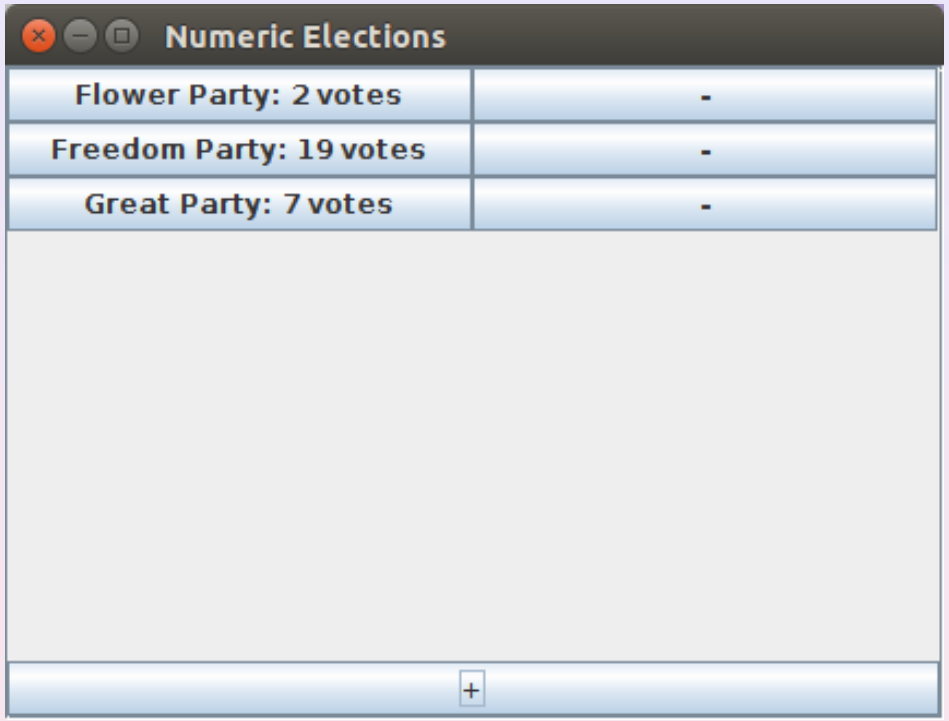
```
JPanel southPanel = new JPanel();
JButton insertButton = new JButton("Insert");
// a panel has by default the flow layout
southPanel.add(insertButton);
insertButton.addActionListener(
    event -> textArea.append("User name: " + textField.getText() +
        " Password: " + new String(passwordField.getPassword())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

add(southPanel, BorderLayout.SOUTH);

pack();
```

A Serious Swing Application!



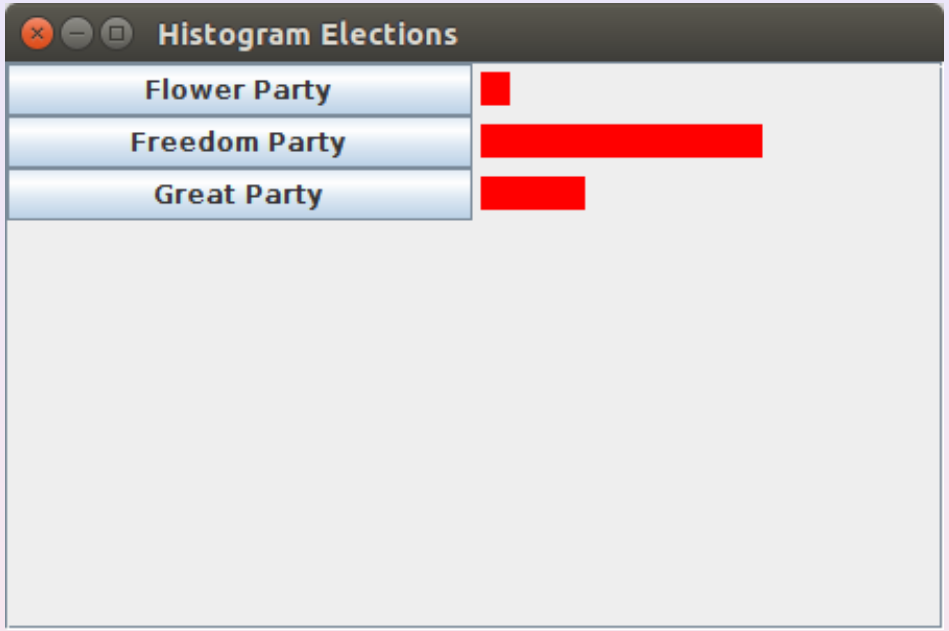
Elections: First View

We will develop a system with two interfaces, for handling election charts



- by clicking on a party name one can increase its votes
- by clicking on the minus sign, one can remove a party
- by clicking on the plus sign, one can add a new party

Elections: Second View



- by clicking on a party name one can increase its votes
- no provision for adding/removing parties

Separation of Concerns

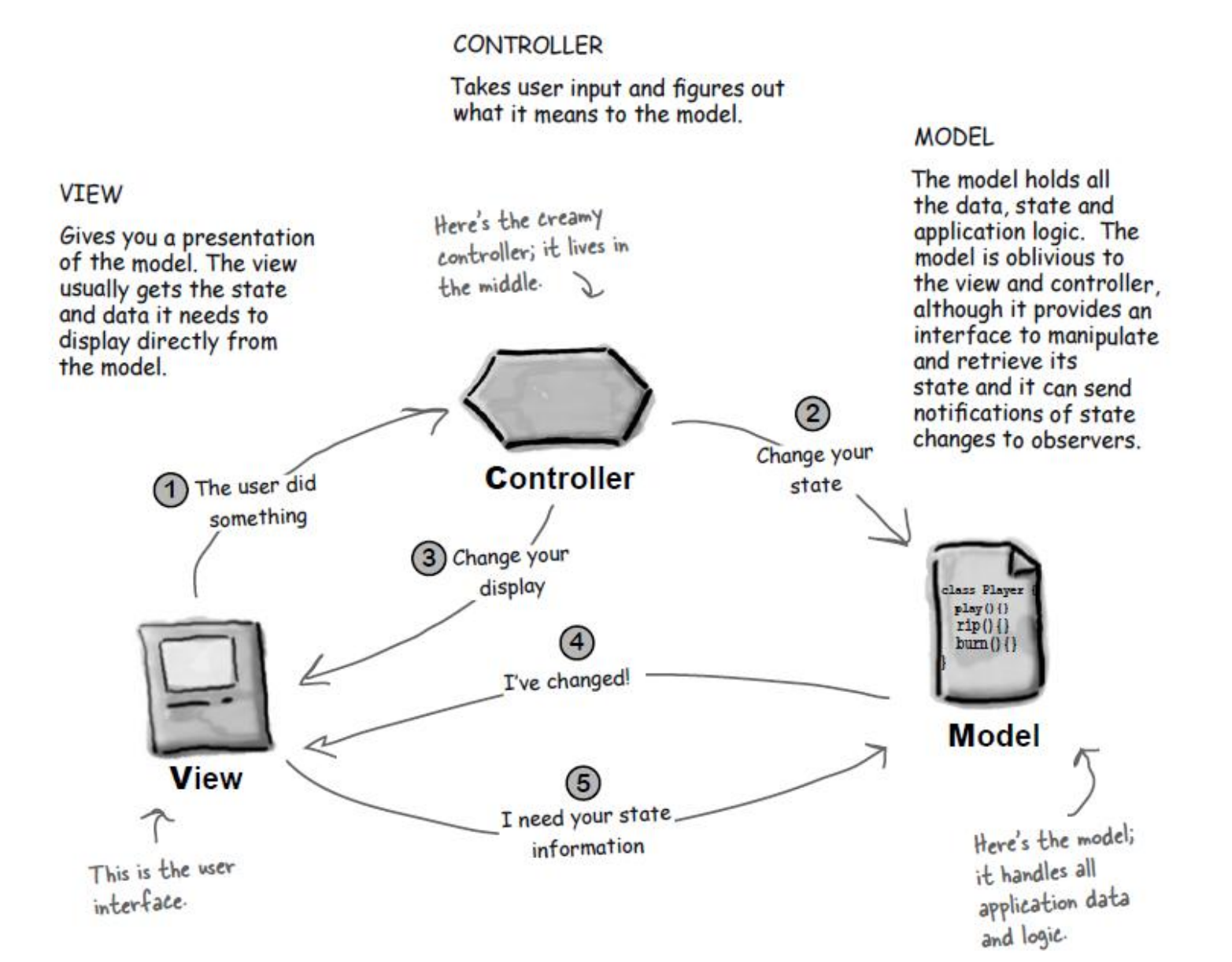
The graphical interface should be kept separate from the logic:

- for distinct versions
- 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

The MVC Triple 1/2

```
public class MVC {
    public final Model model;
    public final Controller controller;
    private final Set<View> views = new HashSet<>();

    public MVC(Model model,Controller controller) {
        this.model = model;
        this.controller = controller;

        model.setMVC(this);
        controller.setMVC(this);
    }

    public synchronized void register(View view) {
        this.views.add(view);
    }

    public synchronized void unregister(View view) {
        this.views.remove(view);
    }
```

The MVC Triple 2/2

```
public interface ViewTask {

    /**
     * Applies the task to the given view.
     *
     * @param view
     */
    void process(View view);
}

/**
 * Applies the given task to all views currently registered.
 *
 * @param task
 */
public synchronized void forEachView(ViewTask task) {
    // Internal iteration, preferred since we do not need
    // to expose the modifiable set of views
    for (View view: views)
        task.process(view);
}
```

The Starting Point of the Program

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

            @Override
            public void run() {
                MVC mvc = new MVC(new Model(), new Controller());

                new NumericElectionsFrame(mvc).setVisible(true);
                new NumericElectionsFrame(mvc).setVisible(true);
                new HistogramElectionsFrame(mvc).setVisible(true);
            }
        });
    }
}
```

The Model

```
public class Model {
    private MVC mvc;
    private final Map<String, Integer> votes = new HashMap<>();

    public void setMVC(MVC mvc) {
        this.mvc = mvc;
    }

    // 5: I need your state information
    public Iterable<String> getParties() {

    public int getVotesFor(String party) {

    // 2: change your state
    public void addParty(String party) {

    public void removeParty(String party) {

    public void setVoteFor(String party, int votes) {
}
```

The Model: Implementation

```java
// 5: I need your state information
public Iterable<String> getParties() {
    // in alphabetical order
    return new TreeSet<>(votes.keySet());
}

public int getVotesFor(String party) {
    return votes.get(party);
}

// 2: change your state
public void addParty(String party) {
    votes.put(party, 0);
    mvc.forEachView(View::onModelChanged);
}

public void removeParty(String party) {
    votes.remove(party);
    mvc.forEachView(View::onModelChanged);
}

public void setVoteFor(String party, int votes) {
    this.votes.put(party, votes);
    mvc.forEachView(View::onModelChanged);
}
```

The Controller

```
public class Controller {
    private MVC mvc;

    public void setMVC(MVC mvc) {
        this.mvc = mvc;
    }

    // 1: the user did something
    public void insertParty(View view) {

    public void addParty(String party) {

    public void removeParty(String party) {

    public void registerVoteFor(String party) {
}
```

The Controller: Implementation

```
// 1: the user did something
public void insertParty(View view) {
    view.askForNewPartyName();
}

public void addParty(String party) {
    mvc.model.addParty(party);
}

public void removeParty(String party) {
    mvc.model.removeParty(party);
}

public void registerVoteFor(String party) {
    mvc.model.setVoteFor(party, mvc.model.getVotesFor(party) + 1);
}
```

The View

```
public interface View {
    // 3: change your display
    void askForNewPartyName();

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

The View: First Implementation 1/4

```
public class NumericElectionsFrame extends JFrame implements View {
    private final MVC mvc;
    private final JPanel scores;

    public NumericElectionsFrame(MVC mvc) {
        this.mvc = mvc;
        mvc.register(this);

        setPreferredSize(new Dimension(430, 300));
        setTitle("Numeric Elections");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        this.scores = buildWidgets();

        onModelChanged();
    }
```

The View: First Implementation 2/4

```
private JPanel buildWidgets() {
    JPanel panel = new JPanel();
    panel.setLayout(new BorderLayout());

    JPanel scores = new JPanel();
    scores.setLayout(new GridLayout(0, 2));
    panel.add(scores, BorderLayout.NORTH);

    JButton addParty = new JButton("+");
    addParty.addActionListener(e -> mvc.controller.insertParty(this));
    panel.add(addParty, BorderLayout.SOUTH);

    add(new JScrollPane(panel));

    return scores;
}
```

The View: First Implementation 3/4

```
@Override
public void onModelChanged() {
    Model model = mvc.model;

    scores.removeAll();
    for (String party: model.getParties()) {
        JButton label = new JButton(party + ": " + model.getVotesFor(party) + " votes");
        label.addActionListener(e -> mvc.controller.registerVoteFor(party));
        scores.add(label);
        JButton remove = new JButton("-");
        remove.addActionListener(e -> mvc.controller.removeParty(party));
        scores.add(remove);
    }

    pack();
}

@Override
public void askForNewPartyName() {
    new InsertPartyNameDialog(mvc.controller);
}
```

The View: First Implementation 4/4

```
class InsertPartyNameDialog extends JDialog {

    public InsertPartyNameDialog(Controller controller) {
        super((Dialog) null);

        setTitle("Insert Party Name");
        setLayout(new FlowLayout());
        add(new JLabel("Insert new party name: "));

        JTextField textField = new JTextField("nome partito");
        textField.addActionListener(e -> {
            String party = textField.getText();
            if (!party.isEmpty())
                controller.addParty(party);

            setVisible(false);
            dispose();
        });
        add(textField);

        pack();
        setVisible(true);
    }
}
```

A Custom Component for Histograms

```java
public class Histogram extends JComponent {
    private final float percent;

    /**
     * Builds a component that represents a histogram.
     * The size is given as a percent (0..1) of the total width of 200 pixels.
     *
     * @param percent
     */

    public Histogram(float percent) {
        this.percent = percent;
    }

    @Override
    protected void paintComponent(Graphics g) {
        // questo è vero da Java 1.2 in poi
        Graphics2D g2 = (Graphics2D) g;
        g2.setColor(Color.RED);
        g2.fillRect(4, 4, (int) (200 * percent), 16);
    }

    @Override
    public Dimension getPreferredSize() {
        return new Dimension(208, 24);
    }
}
```

The View: Second Implementation 1/3

```
public class HistogramElectionsFrame extends JFrame implements View {
    private final MVC mvc;
    private final JPanel scores;

    public HistogramElectionsFrame(MVC mvc) {
        this.mvc = mvc;
        mvc.register(this);

        setPreferredSize(new Dimension(450, 300));
        setTitle("Histogram Elections");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        this.scores = buildWidgets();

        onModelChanged();
    }
```

The View: Second Implementation 2/3

```
private JPanel buildWidgets() {
    JPanel panel = new JPanel();
    panel.setLayout(new BorderLayout());

    JPanel scores = new JPanel();
    scores.setLayout(new GridLayout(0, 2));
    panel.add(scores, BorderLayout.NORTH);

    add(new JScrollPane(panel));

    return scores;
}
```

The View: Second Implementation 3/3

```java
@Override
public void onModelChanged() {
    Model model = mvc.model;

    int totalVotes = 0;
    for (String party: model.getParties())
        totalVotes += model.getVotesFor(party);

    scores.removeAll();
    for (String party: model.getParties()) {
        JButton label = new JButton(party);
        label.addActionListener(e -> mvc.controller.registerVoteFor(party));
        scores.add(label);
        scores.add(new Histogram((float) model.getVotesFor(party) / totalVotes));
    }

    pack();
}

@Override
public void askForNewPartyName() {
}
```

The View: Third Implementation

Exercise

Implement a third version of the view, where the names of the parties are reported in the top part of the frame, in distinct colors, and below is reported a pie chart, with colors corresponding to those of the parties. No provision for adding or removing parties.

http://blue-walrus.com/2012/09/simple-pie-chart-in-java-swing