Designing Interactive Systems 2

Lecture 8: Cross-Platform Applications

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

hci.rwth-aachen.de/dis2

RWTH AACHEN UNIVERSITY

CHAPTER 25

Java

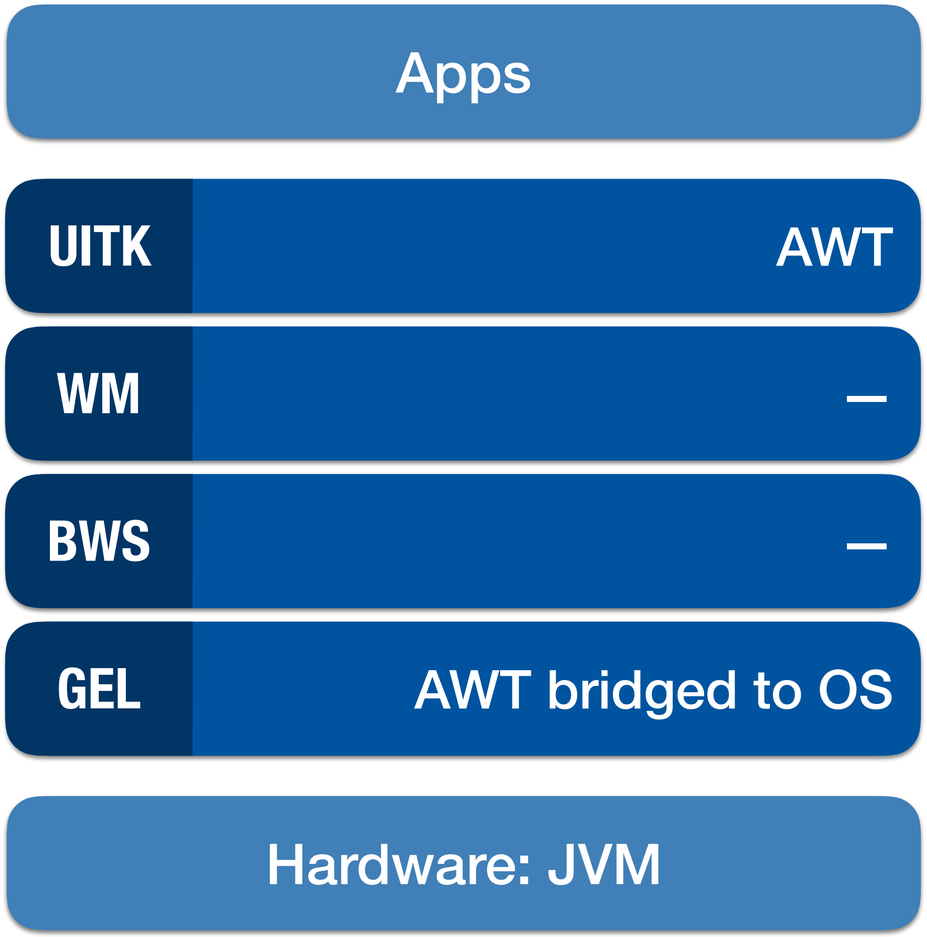
Java UITKs

- 1995: **AWT**
- 1998: **Swing**
- 2008: **JavaFX**

Java Abstract Window Toolkit (AWT)

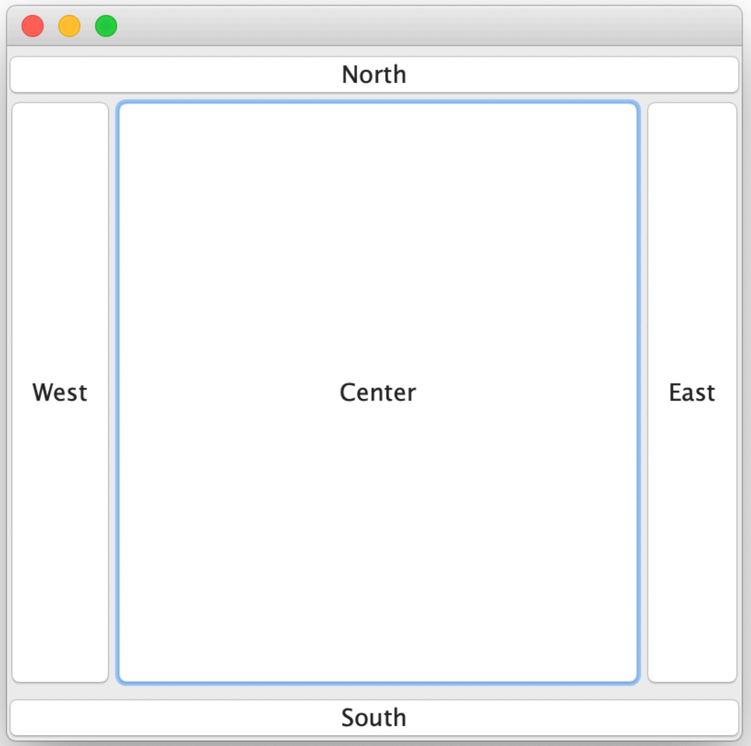
- Object-oriented UI toolkit for the Java platform introduced in 1995
- Maps to native widgets of the host platform
  i.e., good rendering performance, and native look and feel
- **Component** as top level object
- **Containers** can contain multiple widgets
- Events are handled with **Listeners**
- **Layout Managers** can be used to handle positioning

Java AWT



Java AWT: Layout Managers

- Layout managers position widgets dynamically
- Possible to use containers with child widgets
- Hence, no absolute positioning
- Various types: GridBagLayout, BorderLayout, FlowLayout, …



GridBagLayout

Java AWT: Listeners

- Originally, every event occurring in a component was handled by its parent

- No need to specify a target when adding a button

```
Button button = new Button("Click me");
add(button);
```

- But more work to process an event

```
public boolean action (Event e, Object o) {
   String caption = (String)o;
   if (e.target instanceof Button)
      if (caption == "Click me")
         System.out.println("Button clicked");
  return true;
}
```

Java AWT: Listeners

- With event listeners developer can choose where the events are processed
- Different types of listeners,
  e.g., ActionListener, ComponentListener, MouseMotionListener, …
- No need to specify a target when adding a button

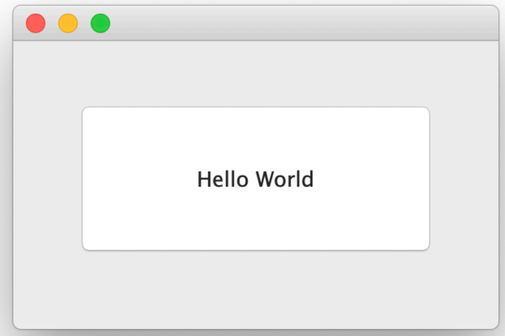
```
Button button = new Button("Click me");
button.addActionListener(this);
add(button);
```

- Reacting to a specific kind of event in own implementation

```
public void actionPerformed (ActionEvent e) {
   System.out.println("Button clicked”);
}
```

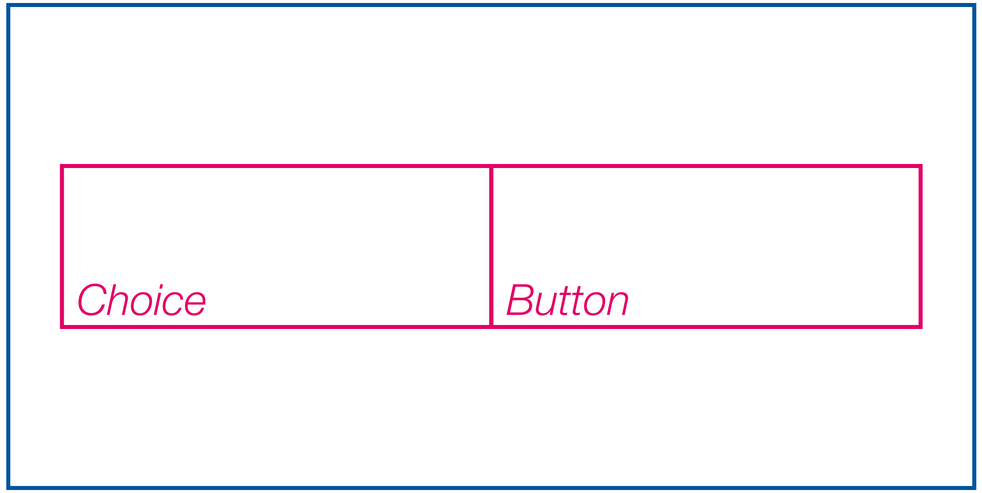
Hello, Java AWT

```
class HelloAWT extends Frame implements ActionListener {

  HelloAWT() {
    Button button = new Button("Hello World");
    button.setBounds(40, 60, 220, 95);
    button.addActionListener(this);
    add(button);

    setLayout(null);
    setSize(300, 200);
    setVisible(true);
  }

  public void actionPerformed (ActionEvent e) {
       System.exit(0);
  }

  public static void main(String argv[]) {
    new HelloAWT();
  }
}
```

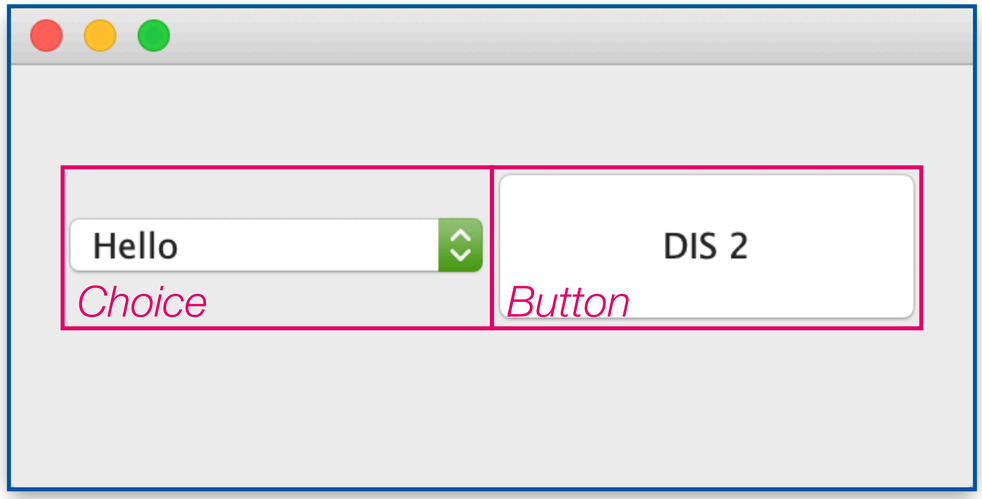


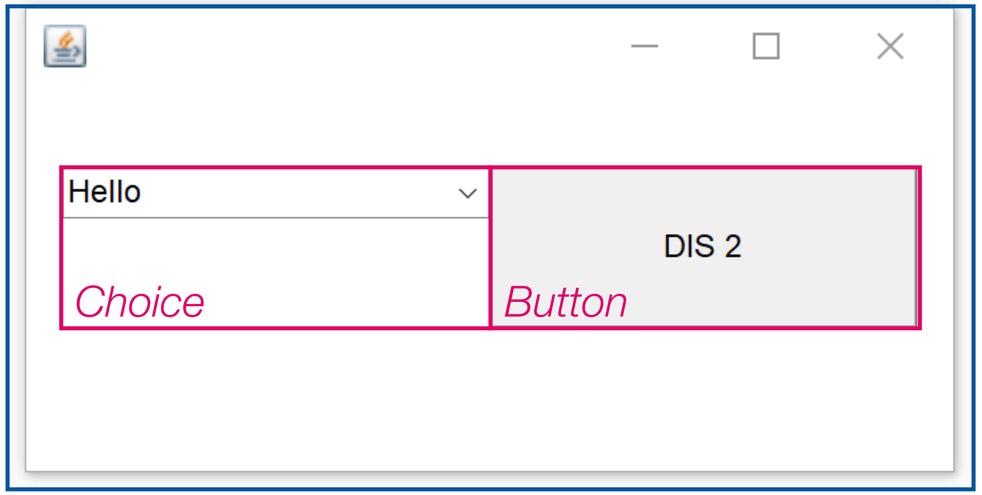
Java AWT: Layout

```
public Window() {
    Choice choice = new Choice();
    choice.add("Hello");
    choice.setBounds(20, 60, 160, 60);
    add(choice);

    Button button = new Button("DIS 2");
    button.setBounds(180, 60, 160, 60);
    add(button);

    setSize(360 , 180);
    setLayout(null);
    setVisible(true);
}
```



Java AWT: Layout



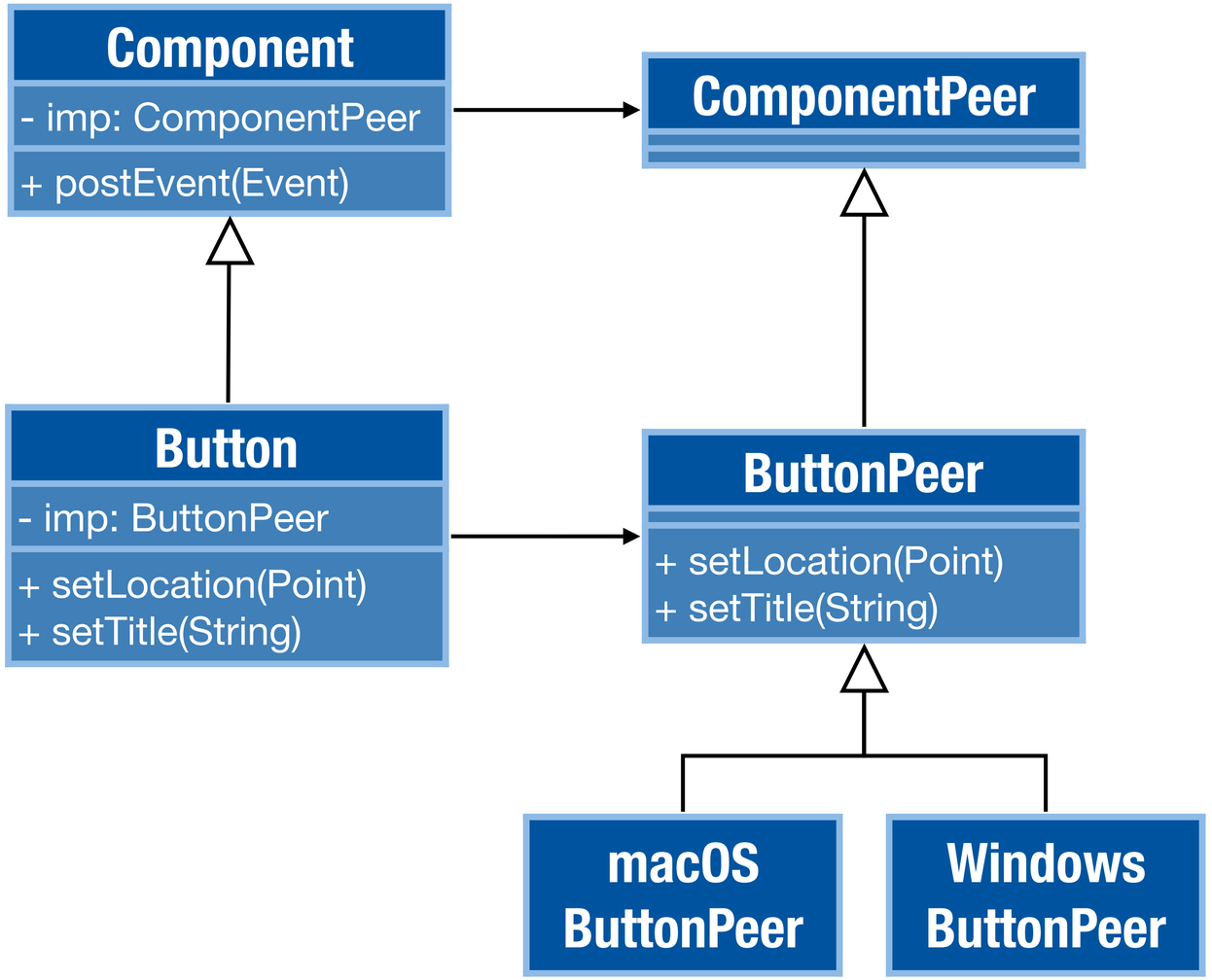


Java AWT: Bridge Pattern

- **Components**
  are the abstraction of widgets that are independent of the implementation

- **ComponentPeers**
  are the abstract implementors for the device-specific UITK

- Each peer comes with a concrete platform-specific implementation



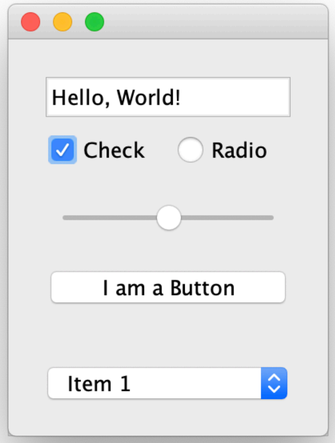
Java AWT: Drawbacks

- Layout inconsistencies
  - Not all widgets are equally resizable on every platform
- Two separate threads
  - One for Java, one for the native UI
- Small number of widgets
  - Only common denominator of all platforms possible
- Low extensibility
  - Only a bridge to native UI: adding a new OS might be a lot of work

Java Swing (JFC)

- Is a “lightweight” UI toolkit for Java, i.e. rendered by Java
- Uses AWT only for root-level widgets
- Four times as many widgets as AWT, including advanced widgets on all platforms
- Pluggable look and feel
- System style ist imitated, not native
- Introduced in 1998 but still frequently used

Java Swing: Pluggable Look and Feel

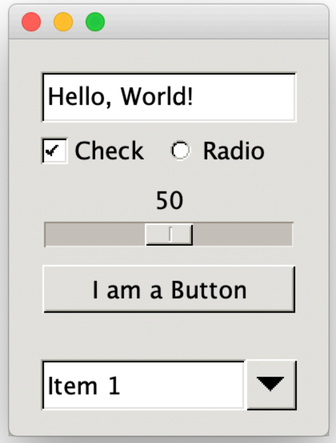


System

Ocean

Metal

Nimbus



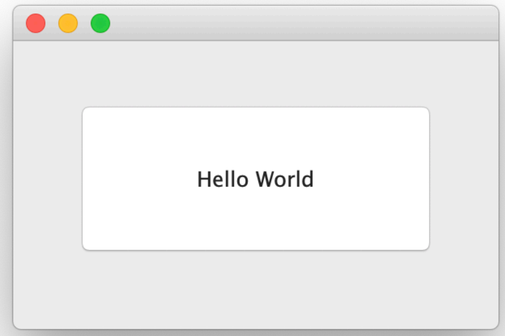
GTK

Motif

```
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

MetalLookAndFeel.setCurrentTheme(new OceanTheme());
UIManager.setLookAndFeel(new MetalLookAndFeel());

UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
```

Hello, Java Swing

```
class HelloSwing extends JFrame implements ActionListener {

  HelloSwing() {
    JButton button = new JButton("Hello World");
    button.setBounds(40, 40, 220, 95);
    button.addActionListener(this);
    add(button);

    setLayout(null);
    setSize(300, 200);
    setVisible(true);
  }

  public void actionPerformed (ActionEvent e) {
       System.exit(0);
  }

  public static void main(String argv[]) {
    new HelloSwing();
  }
}
```



Java Swing: Concurrency

- **Main thread**
  - Executes initial application code: The main method
  - Creates a `Runnable` object that initializes the GUI
- **Event dispatch threads**
  - Create or interact with Swing components
  - e.g. ActionListener implementation
- **Worker threads**
  - Time-consuming background tasks

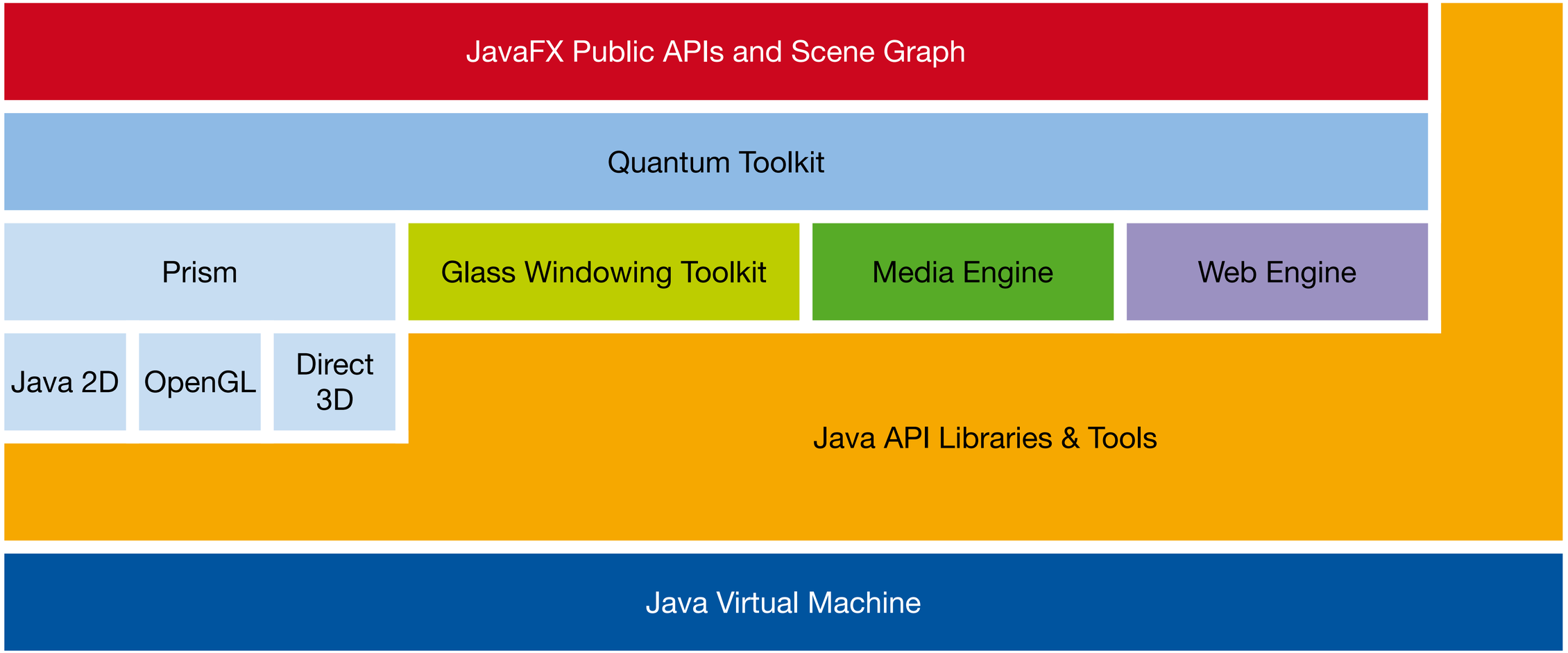
Java Swing: Rendering

- How to repaint a *JFrame*?
- `repaint(`*`Rectangle`* `r)`
  - Java puts a repaint in the event queue
  - To increase performance, multiple requests might be aggregated
  - Choppy animations possible
- `paintImmediately(`*`int`* `x,` *`int`* `y,` *`int`* `w,` *`int`* `h)`
  - Due to overhead less time for program execution

JavaFX

- JavaFX modernizes UI capabilities
  - New accelerated UI rendering
  - Visual effects
  - Defining UI style with CSS files
  - FXML as UIDL
- JavaFX brings some new constructs to the Java language
  - Observable class properties and collection classes
  - Bindings

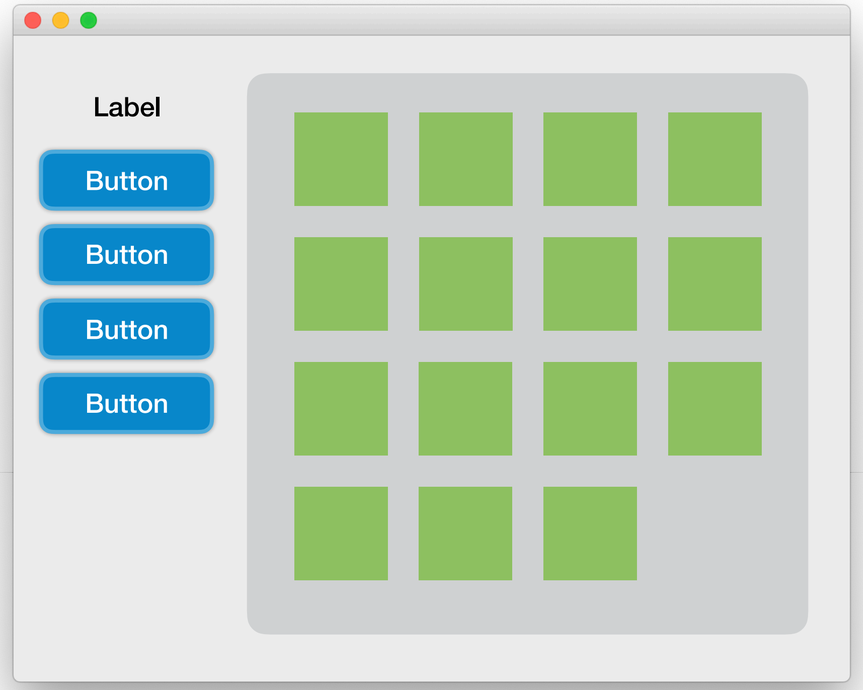
JavaFX: Architecture

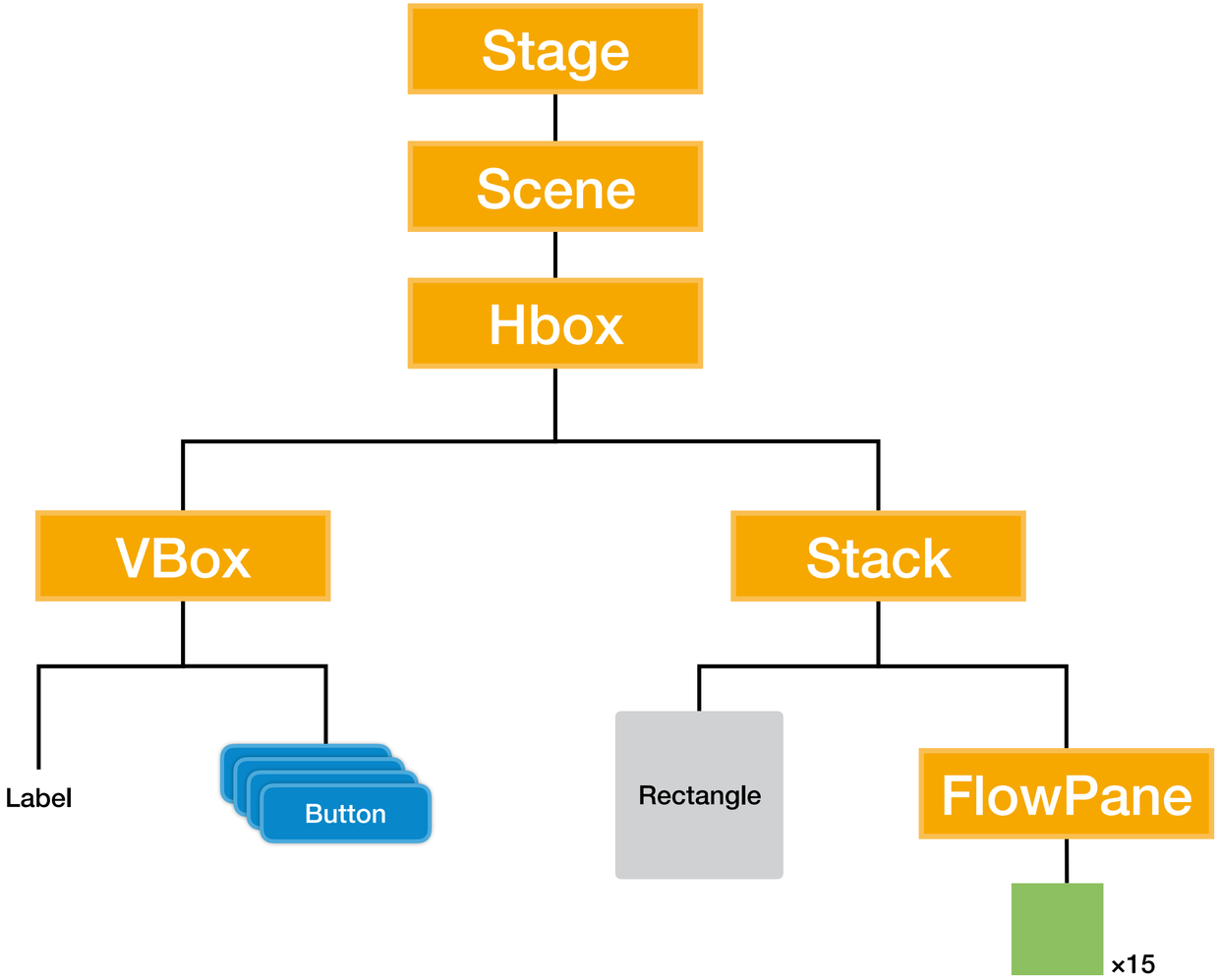


JavaFX: SceneGraph

- **Stage**
  - Top level JavaFX container
  - Equivalent to a Frame
  - Mapped to a native window in the host system
  - Displays one or multiple scenes
- **Scene**
  - Container for all content in a scene graph
  - A scene represents what is visible on screen
- **Node**
  - Widgets, shapes, views, layout containers,…
  - Elements are arranged in a tree

JavaFX: SceneGraph





JavaFX: Quantum Toolkit

- **Prism**
  - Processes render jobs
  - Hardware render path if possible
- **Glass Windowing Toolkit**
  - Thin platform-dependent layer
  - Provides windows, timers
  - Uses hosts event queues
  - Supports multi-touch events
- **Media Engine** renders photos, plays audio and video
- **Pulse**
  - Event that indicates the Scene Graph to render
  - Event driven
  - 60 times per second during animation
- Own threads for Prism and for media

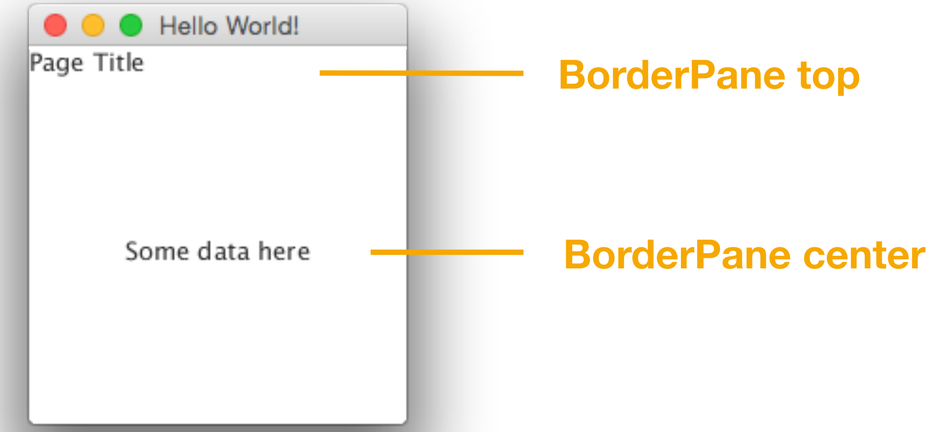
Hello, FXML

Java code

```
BorderPane border = new BorderPane();
Label toppanetext = new Label("Page Title");
border.setTop(toppanetext);
Label centerpanetext = new Label ("Some data here");
border.setCenter(centerpanetext);
```

FXML

```
<BorderPane>
    <top>
        <Label text="Page Title"/>
    </top>
    <center>
        <Label text="Some data here"/>
    </center>
</BorderPane>
```



JavaFX: Verdict

- Open source since JDK 11
- JavaFX comes in platform-specific modules
- Module path is needed for execution
- No native look
  - But possible to create custom CSS Files
- “Fun” bundling all required files for distribution across platforms

CHAPTER 26

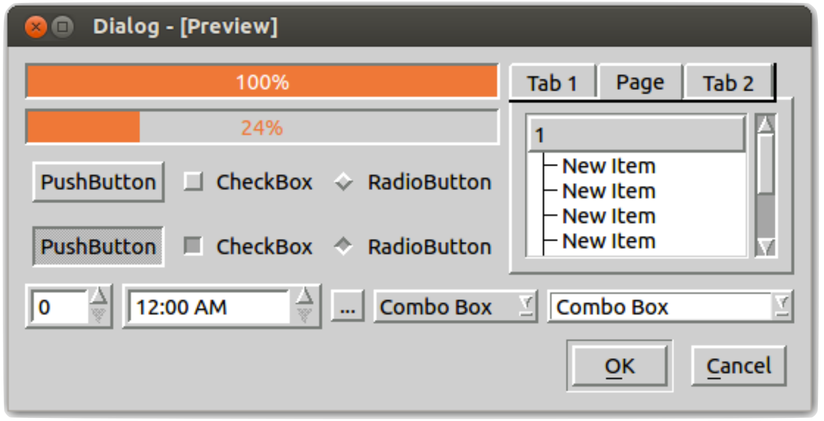
Qt

Qt

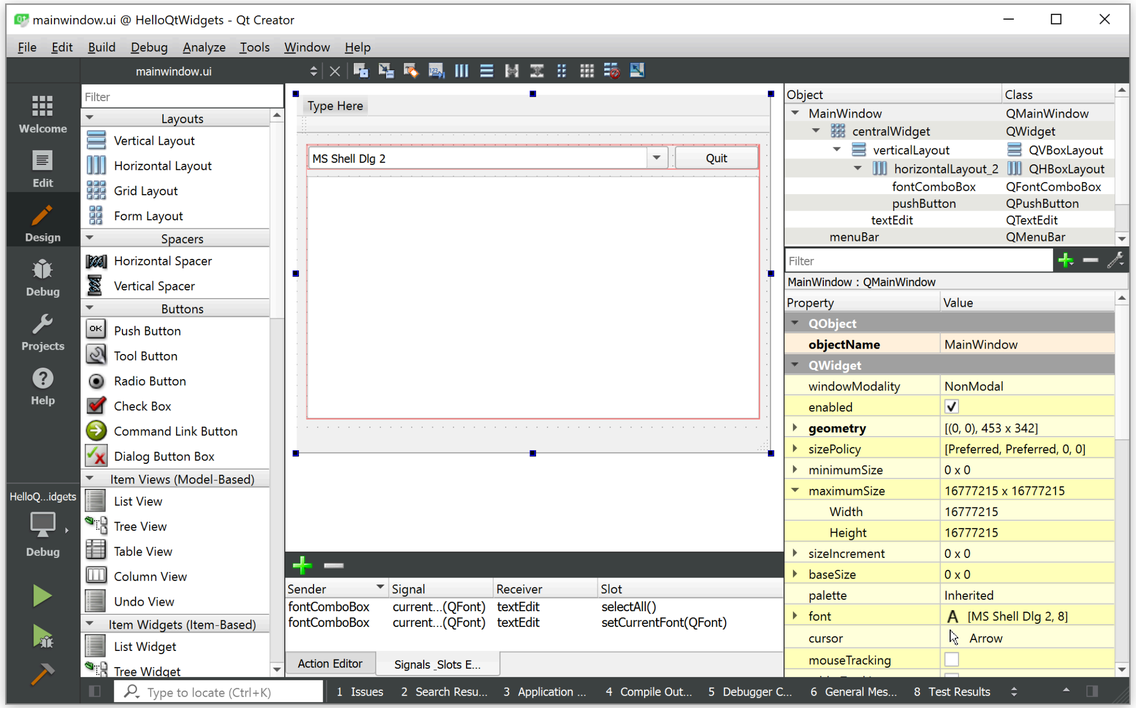
- Cross platform GUI toolkit
- **Qt Widgets**
  - Designed for the desktop
  - Standard widgets designed for WIMP interfaces
- **Qt Quick**
  - Focus on mobile devices and graphical effects
  - New UIDL

Qt Widgets

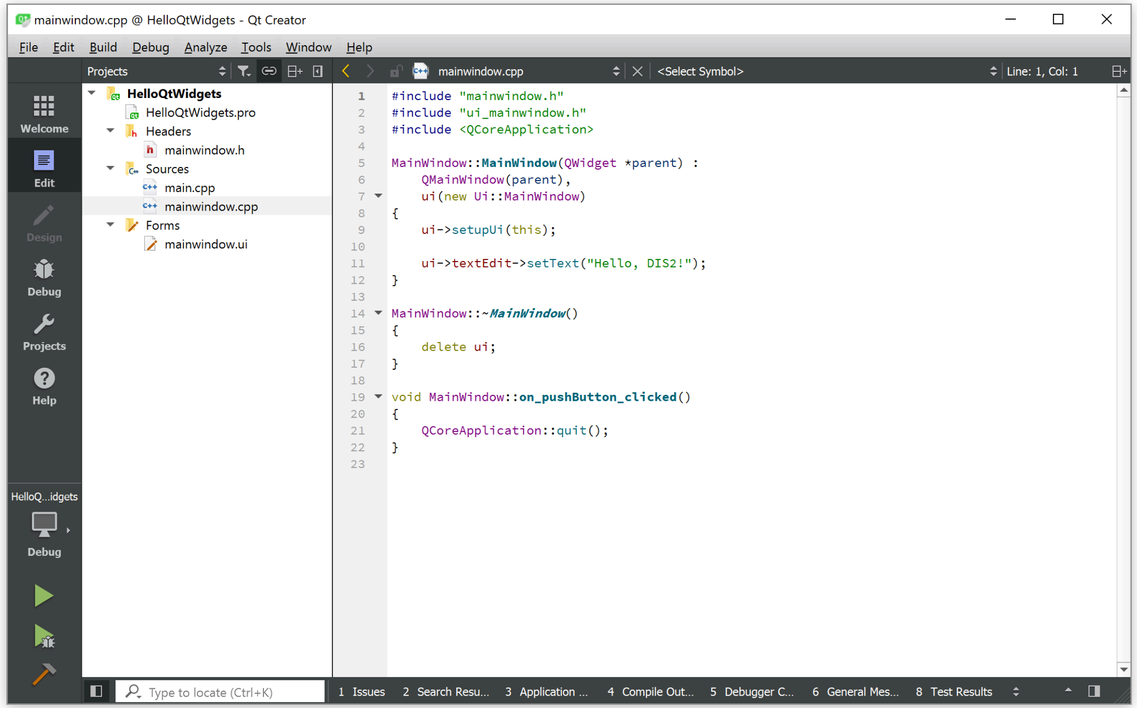
- Original version of Qt
- Designed for WIMP interfaces
- Runs on Mac, Windows, Linux
- UIDS with XML based files that are compiled into C++
- Emulates native look on every platform
- Rich library of widgets



Qt Widgets: UIDS



UI Layout



Source Code

Qt Widgets: Signals & Slots

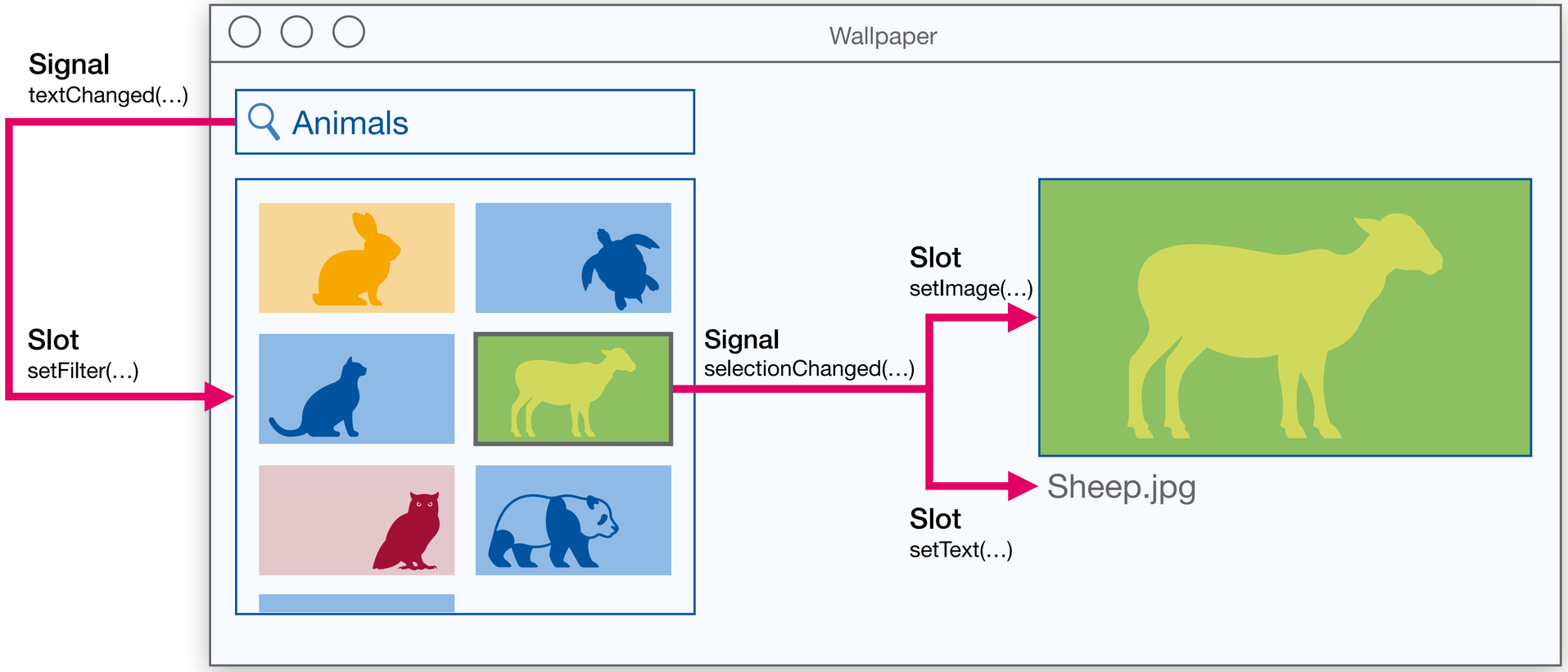
.h File

```
class Counter : public QObject {
    Q_OBJECT

public:
    Counter();
    int value() const { return m_value; }

public slots:
    void increment();
    void setValue(int value);

signals:
    void valueChanged(int newValue);

private:
    int m_value;
};
```

.cpp File

```
Counter::Counter() {
    m_value = 0;
}


void Counter::setValue(int value) {
    if (value != m_value) {
        m_value = value;
        emit valueChanged(value);
    }
}

void Counter::increment() {
    m_value++;
    emit valueChanged(m_value);
}
```

Qt Widgets: Signals & Slots



Qt Widgets: Signals & Slots

- **Signals**
  are emitted by objects when they change their state in a way that might be interesting for other objects

- **Slots**
  are normal member functions that are used for receiving signals

- Advantages
  - Loose coupling
  - Type safety

Qt Widgets: Signals & Slots

- The connect method binds slots to signals
- These connections are unidirectional
- Signals fill in the parameters of the slots from left to right
- All parameters of the slot have to be filled

```
Counter a, b;
```

a: 0
b: 0

```
a.setValue(1);
b.setValue(2);
```

a: 1
b: 2

```
QObject::connect(&a, SIGNAL(valueChanged(int)),
                 &b, SLOT(setValue(int)));
```

```
a.setValue(3);
```

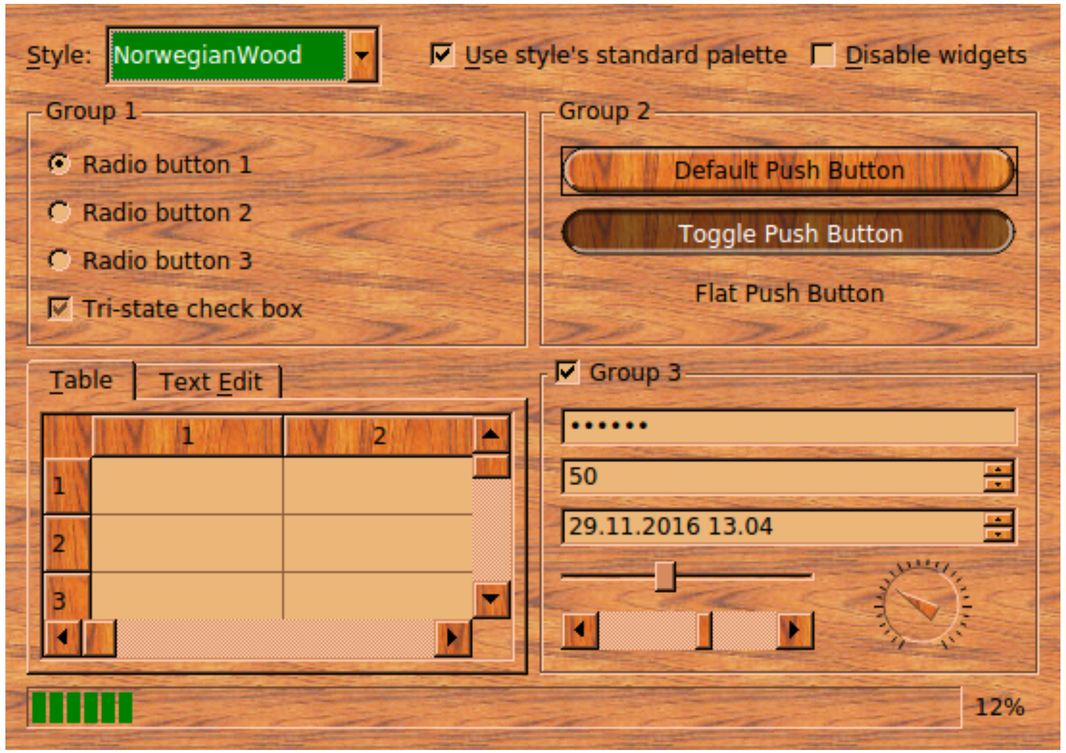
a: 3
b: 3

```
b.setValue(4);
```

a: 3
b: 4

Qt Widgets: Styling the UI

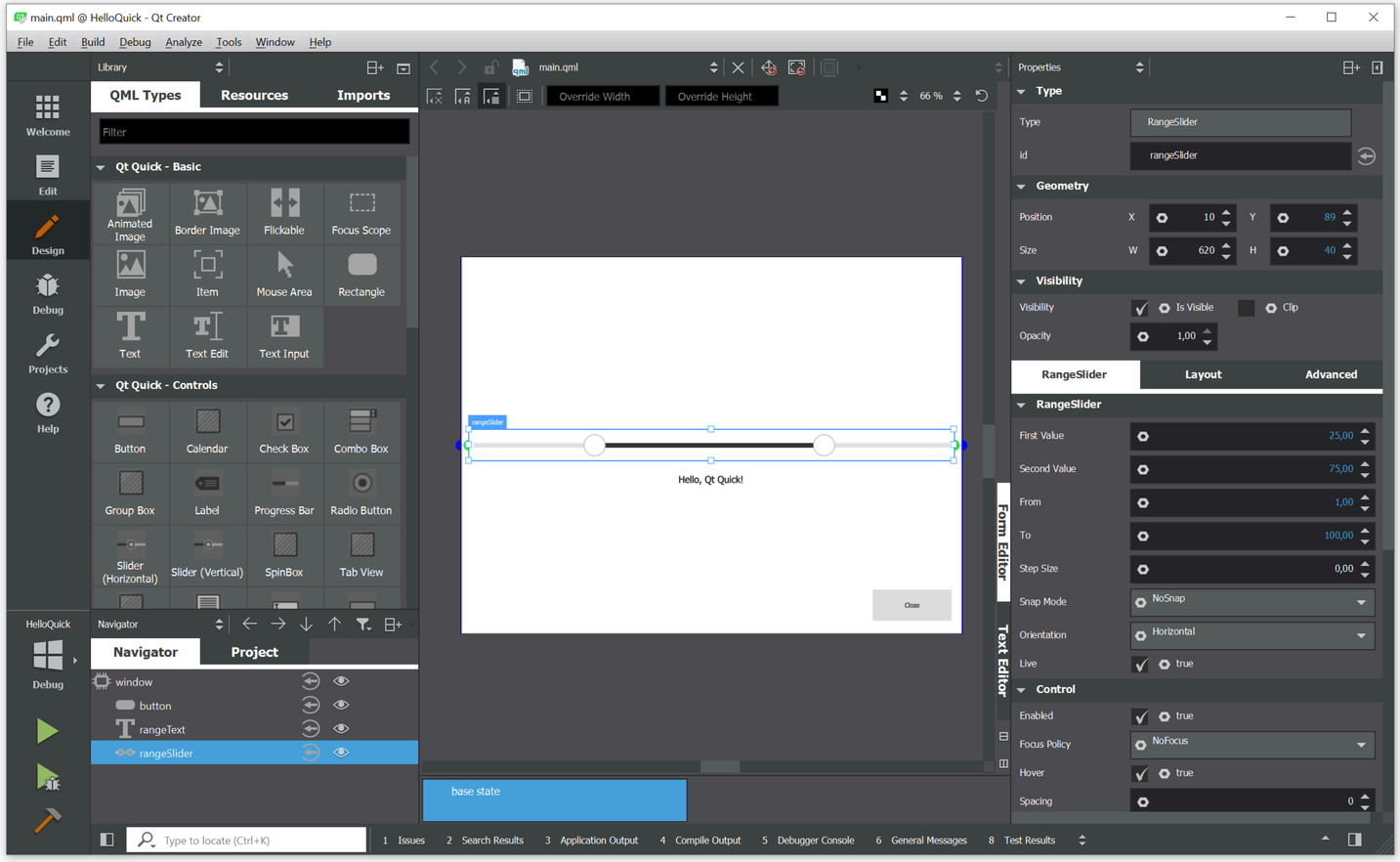
- The visual style of Qt Widgets applications is defined by the operating system
- QProxyStyle allows to make customizations across all system looks
- Not your first choice for a highly customized UI

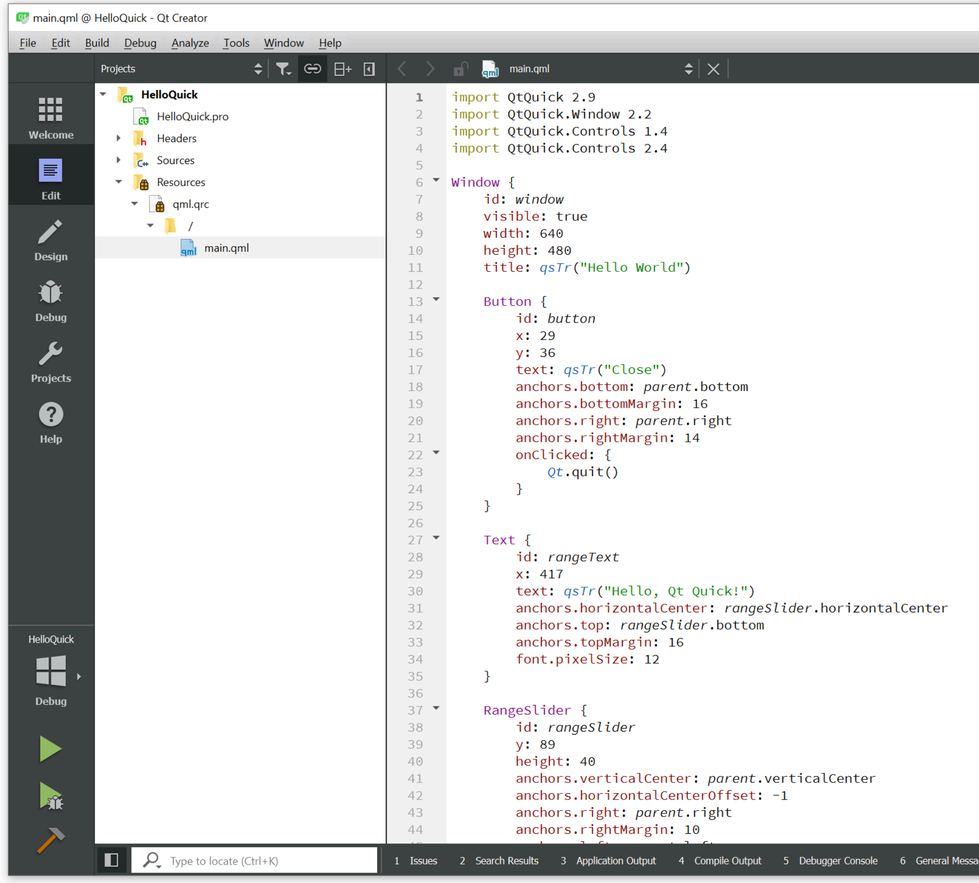


Qt User Interface Creation Kit (Quick)

- Bringing Qt to the “new” operating systems: Android, iOS
- Adds support for touch
- Easier to integrate graphical effects
- New UIDL that includes JavaScript
- Qt Quick Controls: new set of standard widgets

Qt Quick: UIDS





Qt Quick: QML

```
Rectangle {
    id: rect
    width: 250; height: 250

    Button {
        anchors.bottom: parent.bottom
        anchors.horizontalCenter: parent.horizontalCenter
        text: "Change color!"
        onClicked: {
            rect.color = Qt.rgba(Math.random(), Math.random(), Math.random(), 1);
        }
    }
}
```

Qt Quick: Animation

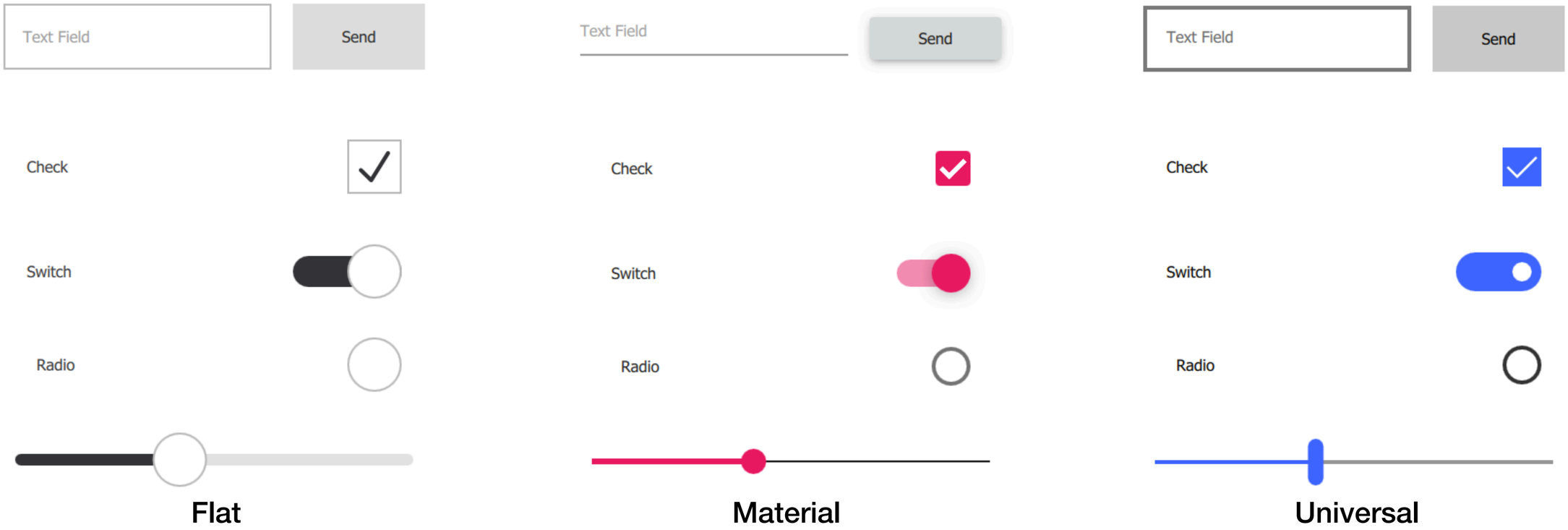
- A widget in the QML file can define **states**
- **Transitions** can be used to animate state changes

```
Rectangle {
    id: rectangle
    width: 200
    height: 200

    states: [
        State {
            name: "stateRed"
            PropertyChanges {
                target: rectangle; color: "red"
        }},
        State {
            name: "stateBlue"
            PropertyChanges {
                target: rectangle; color: "blue"
   }}]

    transitions: [
        Transition {
            from: "*"
            to: "*"
            ColorAnimation {
                duration: 2000
    }}]
}
```

Qt Quick: Styles

- Qt Quick apps can be themed to match the look of a native app

```
qputenv("QT_QUICK_CONTROLS_STYLE", "Material");
```



Qt Quick: Integrating QObjects

- In main function

```
QQmlContext* context = engine.rootContext();
Counter counter;
context->setContextProperty("counter", &counter);
```

- Reacting to a signal of the counter

```
Text {
    id: element
    Connections {
        target: counter
        onValueChanged: element.text = counter.value()
    }
}
```

- Accessing slots of the counter

```
Button {
    onClicked: {
        counter.increment()
    }
}
```