Designing Interactive Systems 2

Lecture 8: Cross-Platform Toolkits

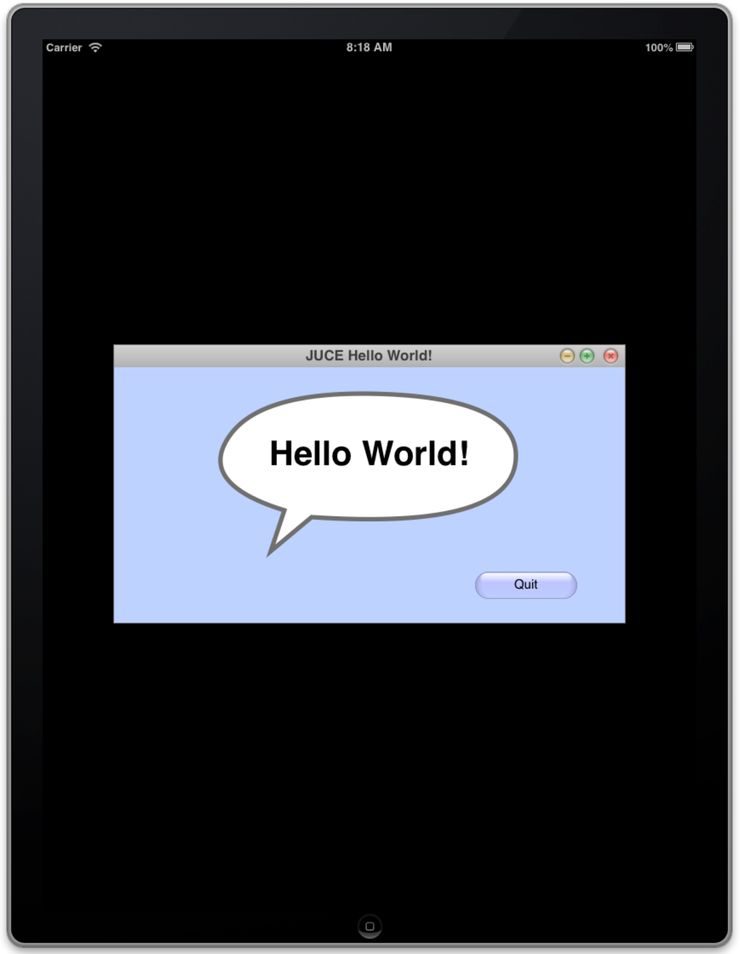
Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

hci.rwth-aachen.de/dis2

RWTH AACHEN UNIVERSITY

Why Cross-Platform Toolkits (Often) Suck

- Platform consistency vs. application consistency
- Keeping widget sets up-to-date with platform evolution
- Drawing in toolkit vs. native code
- Look & Feel is more than widgets!



CHAPTER 25

Java

Java UITKs: Three Generations

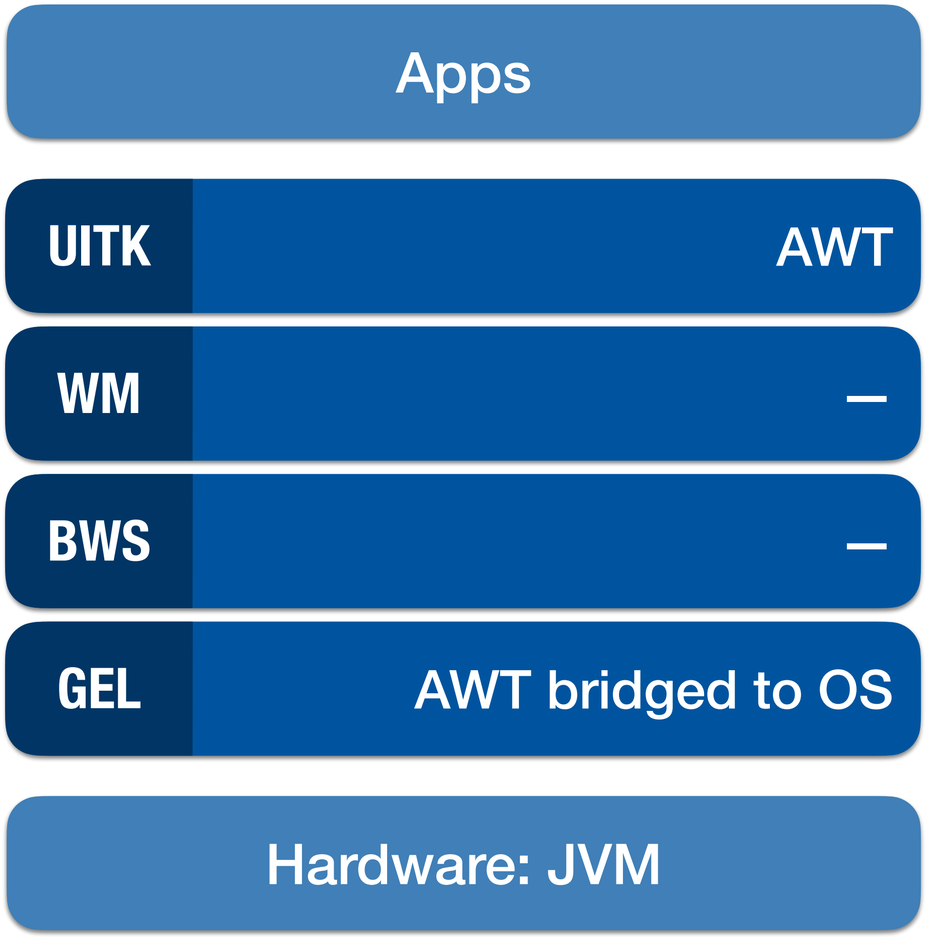
- 1995: **AWT**
- 1998: **Swing**
- 2008: **JavaFX**

Java Abstract Window Toolkit (AWT)

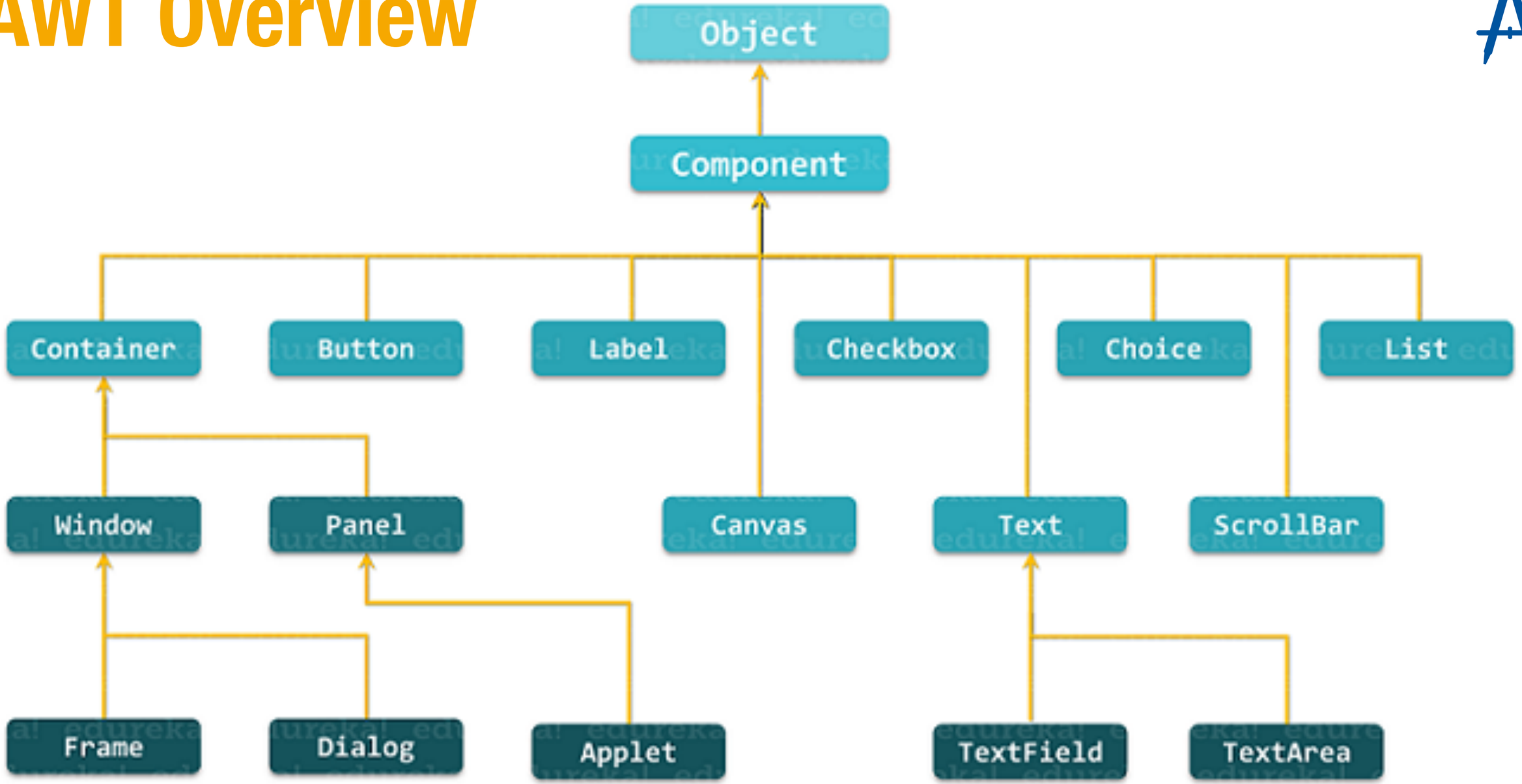
- Object-oriented UI toolkit for the Java platform
- Introduced with Java 1.0 in 1995
  - First version of AWT was developed in only 6 weeks(!)
- Maps to **native widgets** of the host platform

James Gosling

AWT in the Reference Model



AWT Overview

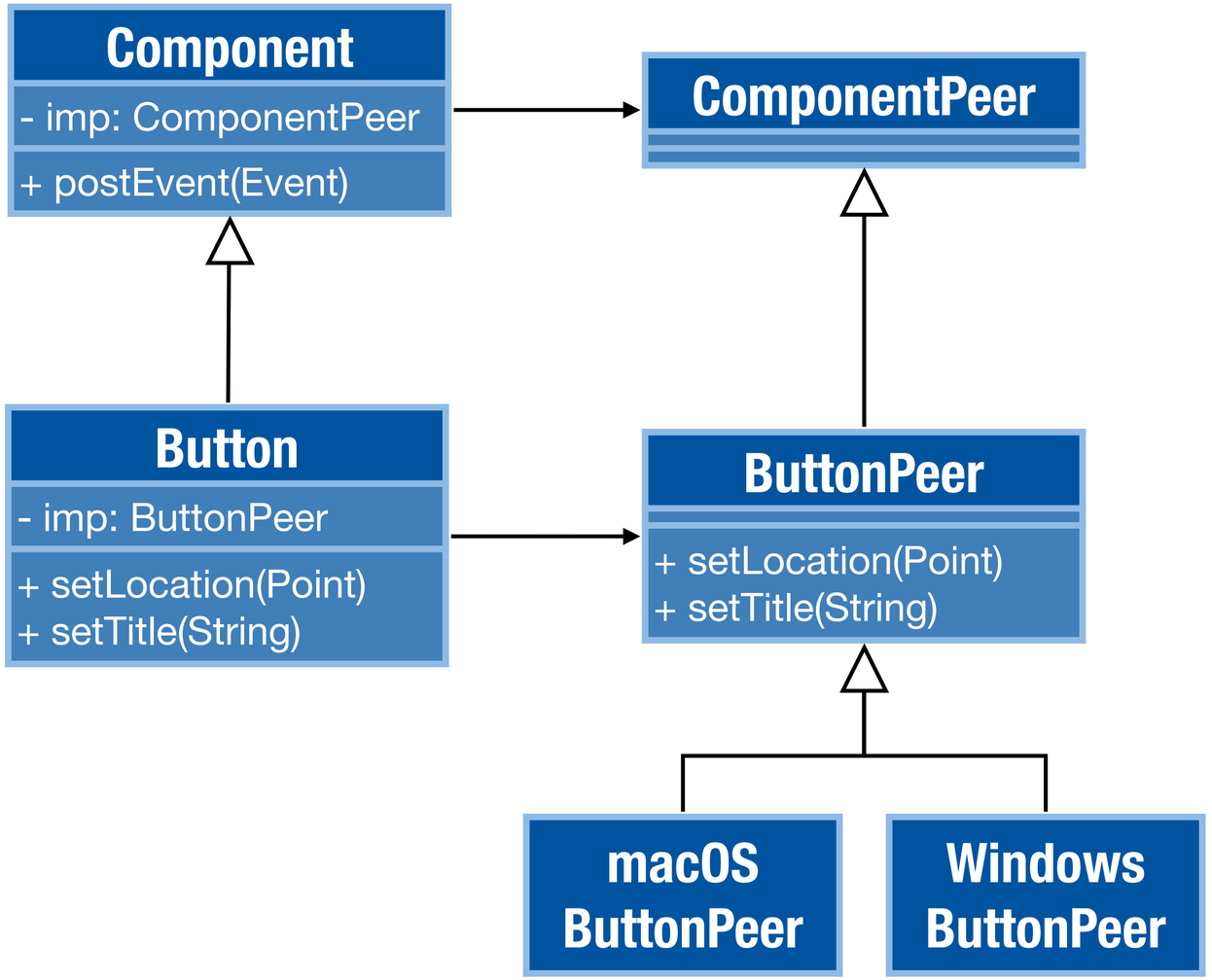


AWT: Bridge Pattern

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

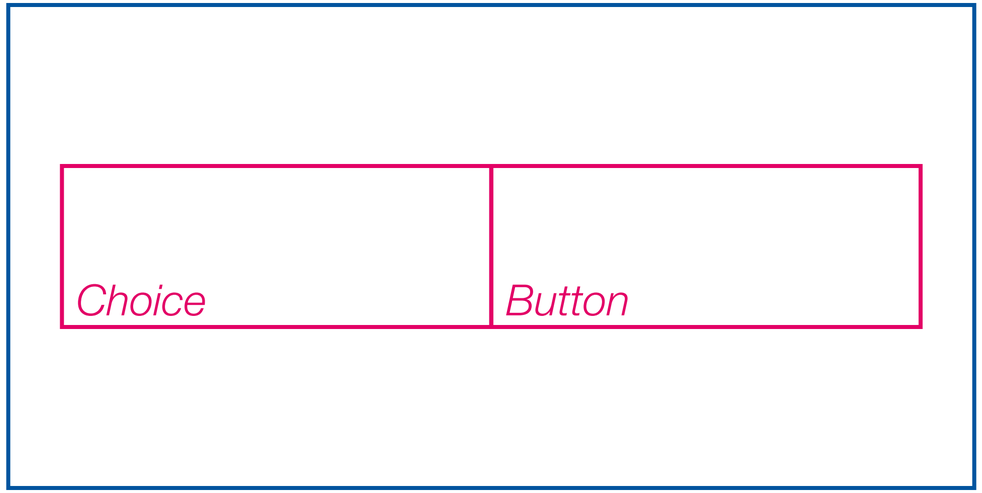
- **ComponentPeers**
  are the abstract implementors for the device-specific UITK

- Each peer comes with a concrete platform-specific implementation

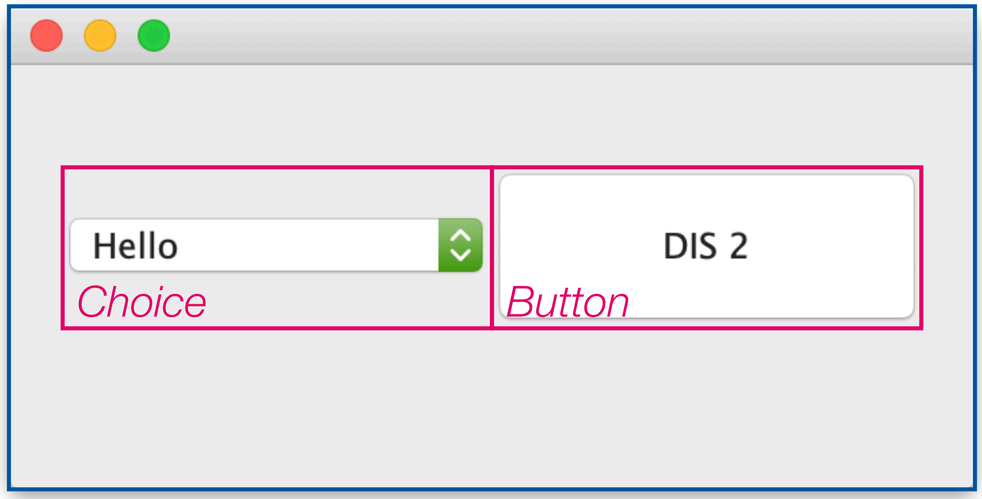


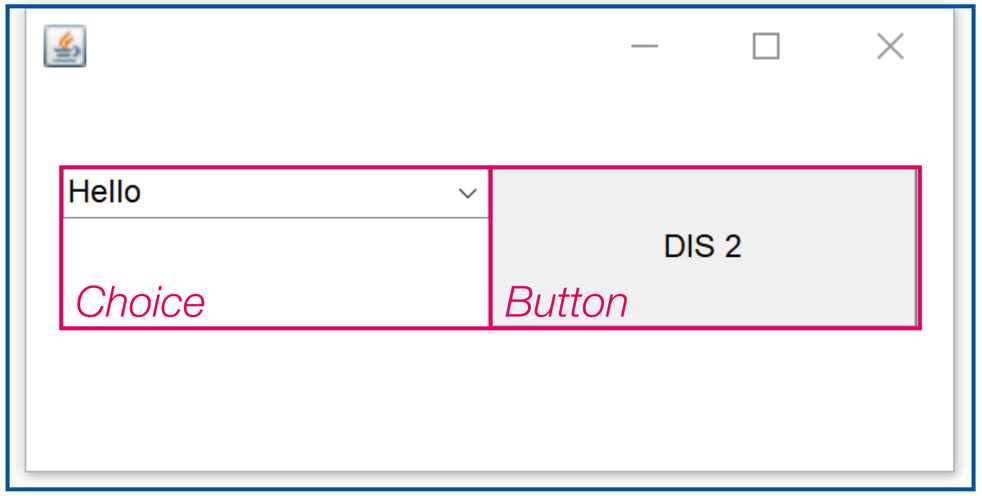
AWT: Cross-Platform Layout in Practice

```
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);
}
```



AWT: Cross-Platform Layout in Practice





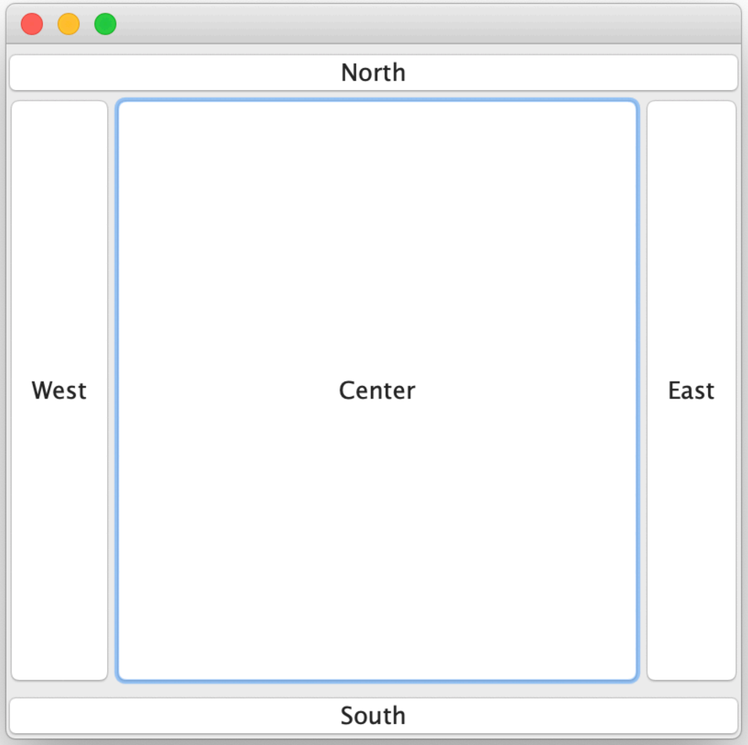
AWT: Layout Managers

- In AWT, widgets are grouped by putting them inside (i.e., making them children of) a **Container** widget

- But the actual layout of these children is managed by a **Layout Manager** attached to the Container

- Different Layout Managers define layout policy: **GridBagLayout, BorderLayout, FlowLayout, …**

- No (pixel-) absolute, only relative positioning



BorderLayout

Events in AWT 1.0

- Originally, every event occurring in Components (e.g., a Button) was handled by their parent **Container** (e.g., a Dialog) in its **action()** method
- No need to specify a target when adding a button
- Problem: Long **action()** methods with lots of **if** statements

Events in AWT 1.0

```
import java.awt.*;

public class HelloWorld extends Frame {
  public static void main(String argv[]) {
    new HelloWorld();
  }
  HelloWorld() {
    Button button = new Button("Click me");
    add(button, "Center");
    setSize(200, 200);
    setVisible(true);
  }
  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;
  }
}
```

AWT 1.1: Introducing Listeners

- Listeners let developer choose where events are processed
- Listener types for different kinds of events:
  e.g., ActionListener, ComponentListener, MouseMotionListener, …
- 1 widget can have multiple listeners, and 1 listener be connected to multiple widgets
- The developer adds a listener to the button's list of listeners for when it gets clicked:

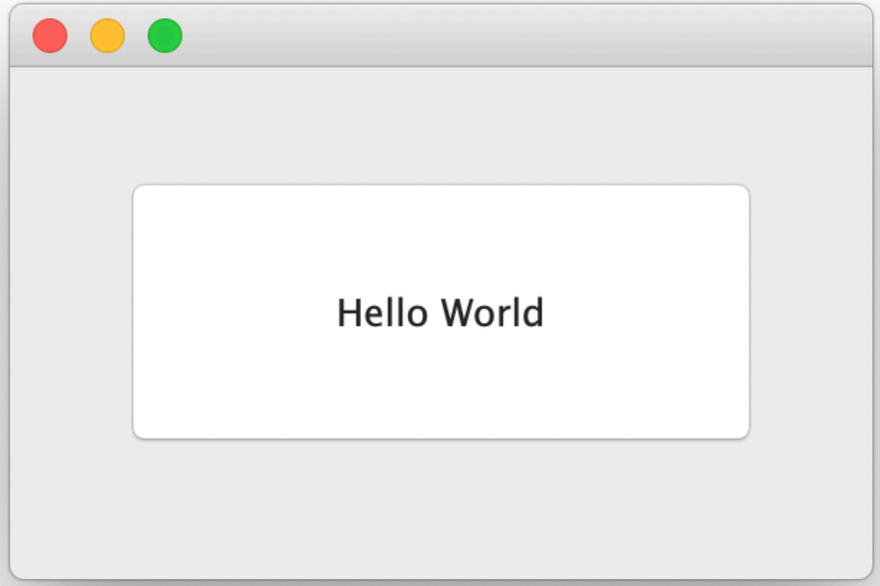
```
Button button = new Button("Click me");
button.addActionListener(this);
add(button);
```

- The listener object must implement a matching method for the event type:

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

Listeners

```
import java.awt.*

public class HelloWorld extends Frame implements ActionListener {

  HelloWorld() {
    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 HelloWorld();
  }
}
```



Java AWT: Consequences of Using Native Widgets

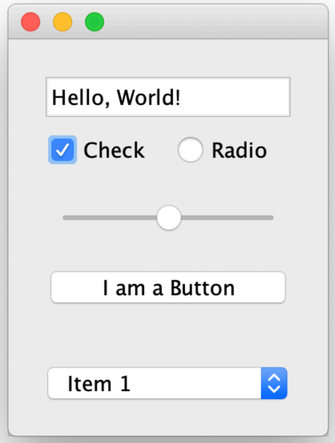
- Good rendering performance
- **But there's more to native look and feel than widgets!**

- Also creates many BWS windows —> **heavyweight** toolkit
- Small number of widgets, limited by what's available on **all** supported platforms
- Two separate threads (one for Java, one for the native UI) — race conditions
- Keeping up with host OS widget evolution, or adding a new host OS, is a lot of work

Java Swing (JFC)

- Introduced in 1998 but still used frequently
- Uses its **own** widgets implemented in Java
  –> “**lightweight**” UI toolkit, i.e. rendered in Java
  - Uses AWT only for root-level widgets
  - 4x as many widgets as AWT
- **Pluggable look and feel**
  - Can mimic host platform, or be a custom theme

Java Swing: Pluggable Look and Feel

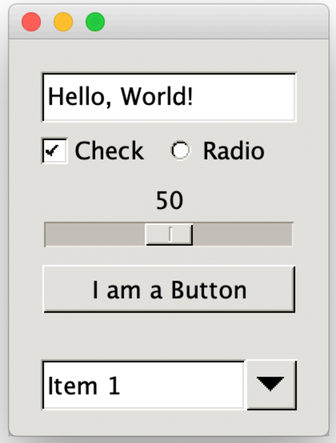


System

Ocean

Metal

Nimbus



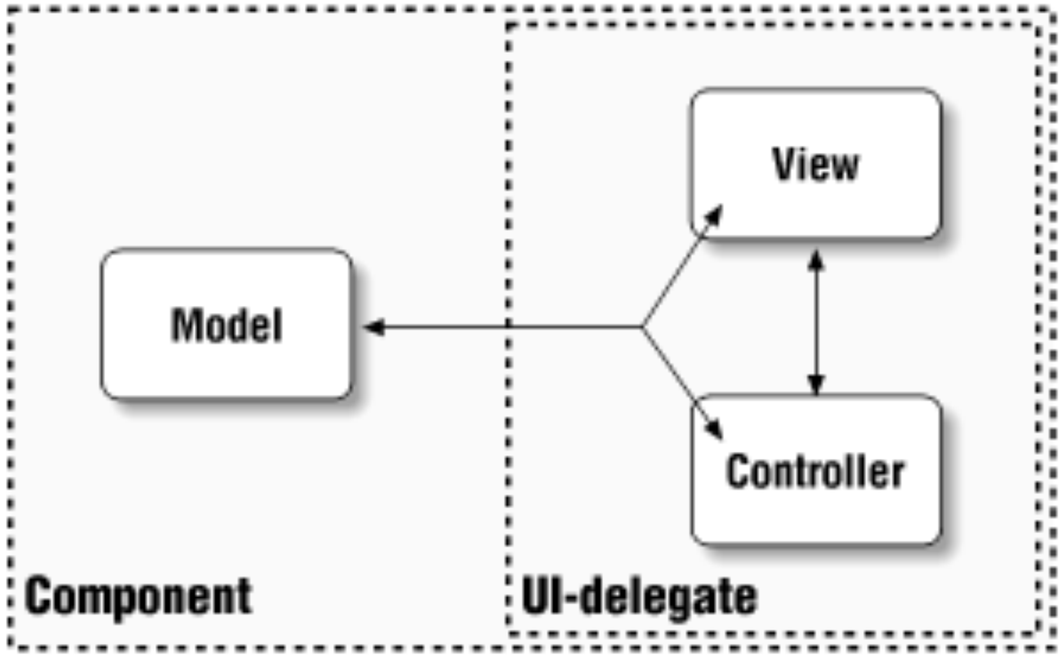
GTK

Motif

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

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

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

MVC in Swing

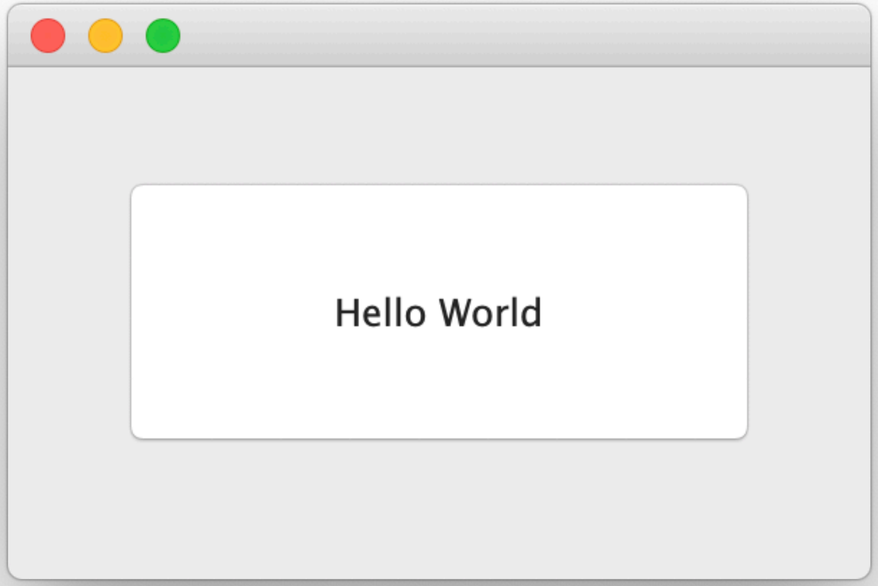
- View and controller combined into delegate
- Interfaces for Model and View (e.g. ButtonModel, ButtonUI)
- Delegates implement ComponentUI
- Allows customization of UIs and pluggable Look & Feel



[Eckstein et al.: Java Swing, O'Reilly]

Hello, Swing

```
import javax.swing.*;

public 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 Removes Race Conditions

- **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 (handling events *and* drawing)
  - E.g., ActionListener implementation
- **Worker threads**
  - Time-consuming background tasks

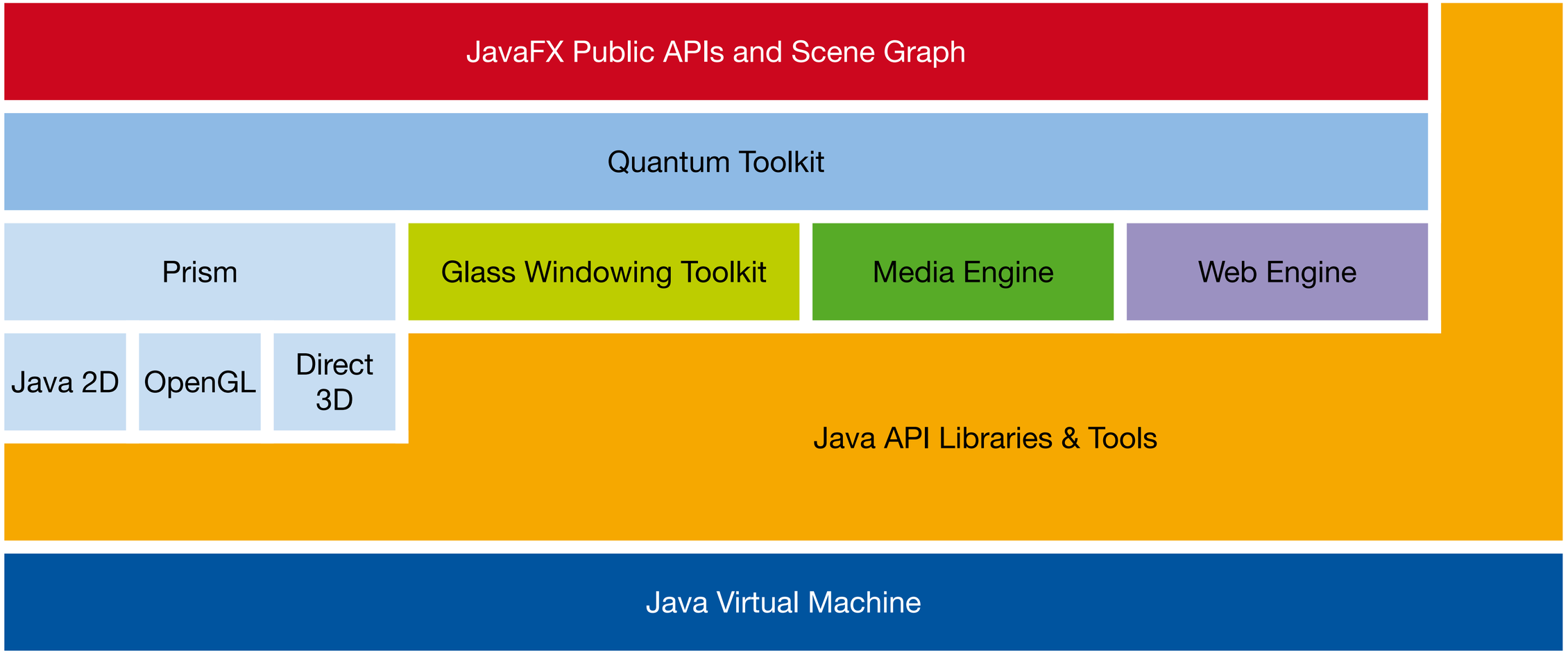
Java Swing: Rendering

- 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 (since 2008)

- 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

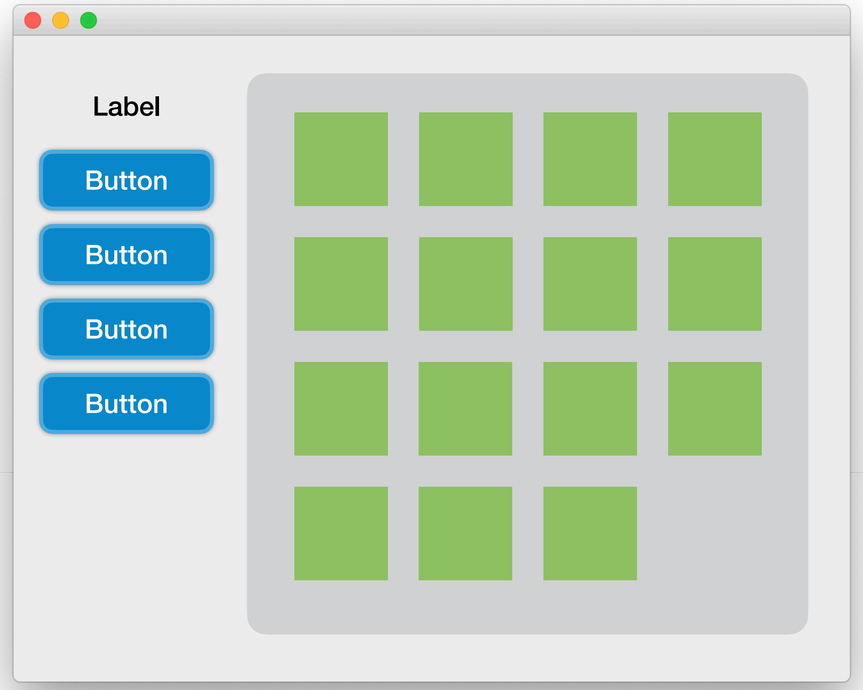
JavaFX: Architecture

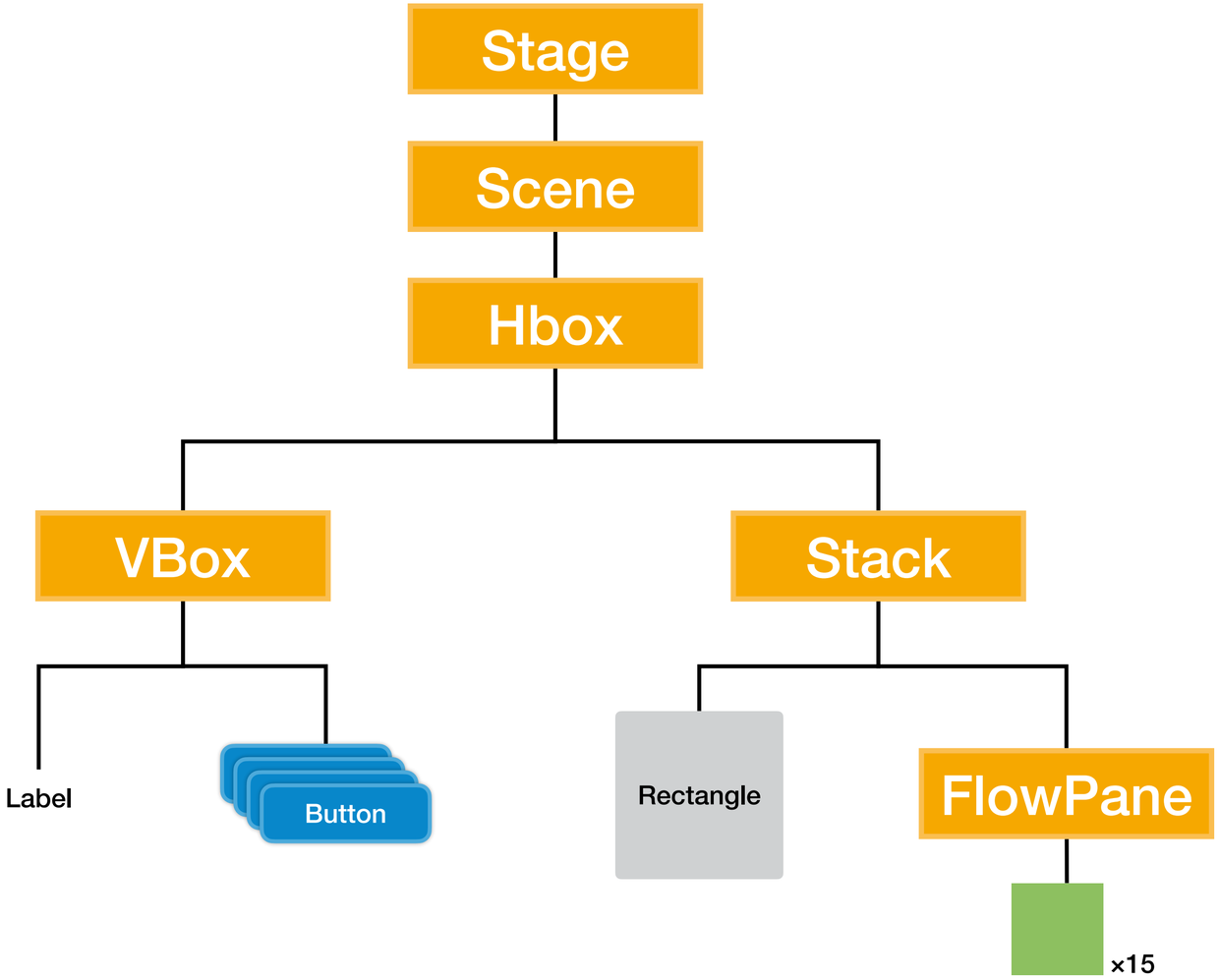


JavaFX: SceneGraph

- **Stage**
  - Top-level JavaFX container
  - Equivalent to a **Frame/JFrame**
  - Mapped to native host window
  - Displays one or multiple scenes
- **Scene**
  - Container for all content in a scene graph
  - Represents what is visible on screen; style with CSS
- **Node**
  - Widgets, shapes, views, layout containers,…
  - Elements arranged in a tree

JavaFX: SceneGraph



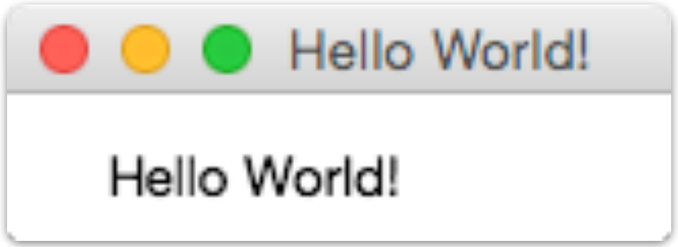


JavaFX: Quantum Toolkit

- **Prism**
  - Processes render jobs
  - Hardware render path if possible
- **Glass Windowing Toolkit**
  - Thin platform-dependent layer
  - Provides windows, timers
  - Uses host's event queues & threading mechanisms
  - 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, JavaFX

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class HelloWorld extends Application {

    @Override public void start(Stage stage) {
        Scene scene =
             new Scene(new Group(new Text(25, 25, "Hello World!")));

        stage.setTitle("Hello World!");
        stage.setScene(scene);
        stage.sizeToScene();
        stage.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
    }
}
```



FXML

- XML-based language to construct object graphs
  - Document structure parallels scene graph structure
  - UI structure is easier to read
- UI independent of program code
- Does not require recompilation
- Easy localization

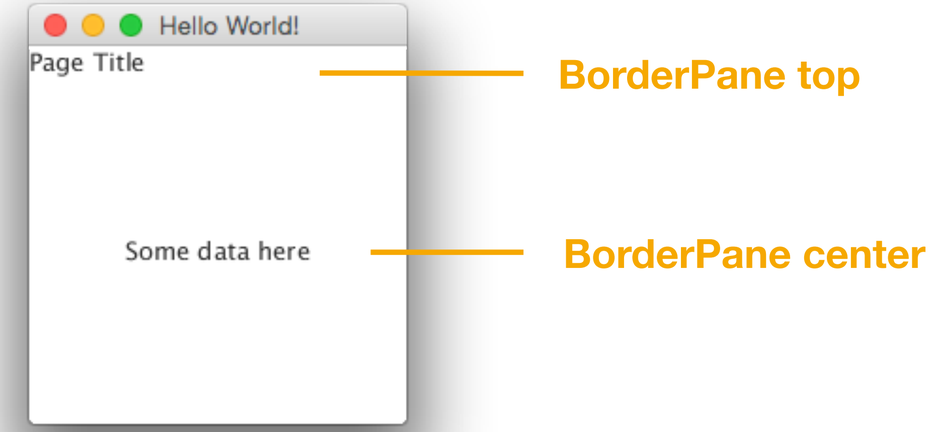
Hello, FXML

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: Pros & Cons

- Java's modern official UITK
- Open source since 2018
- Comes in platform-specific modules
- Module path is needed for execution
- No system-native look, but you can create custom CSS files
- "Fun" bundling all required files for distribution across platforms

CHAPTER 26

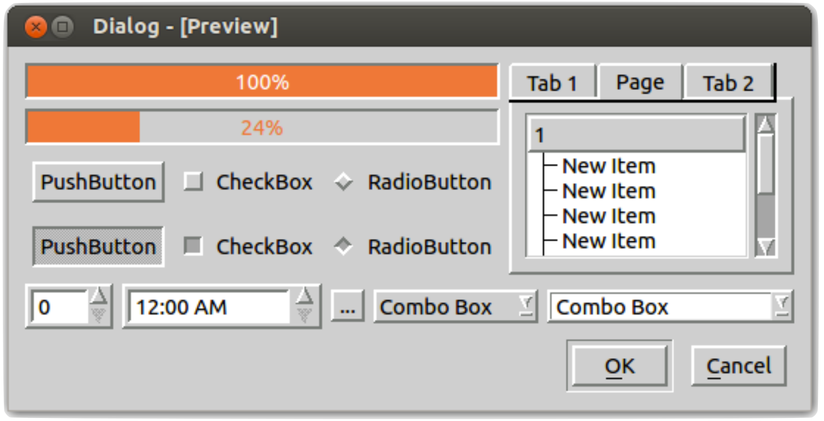
Qt

Qt

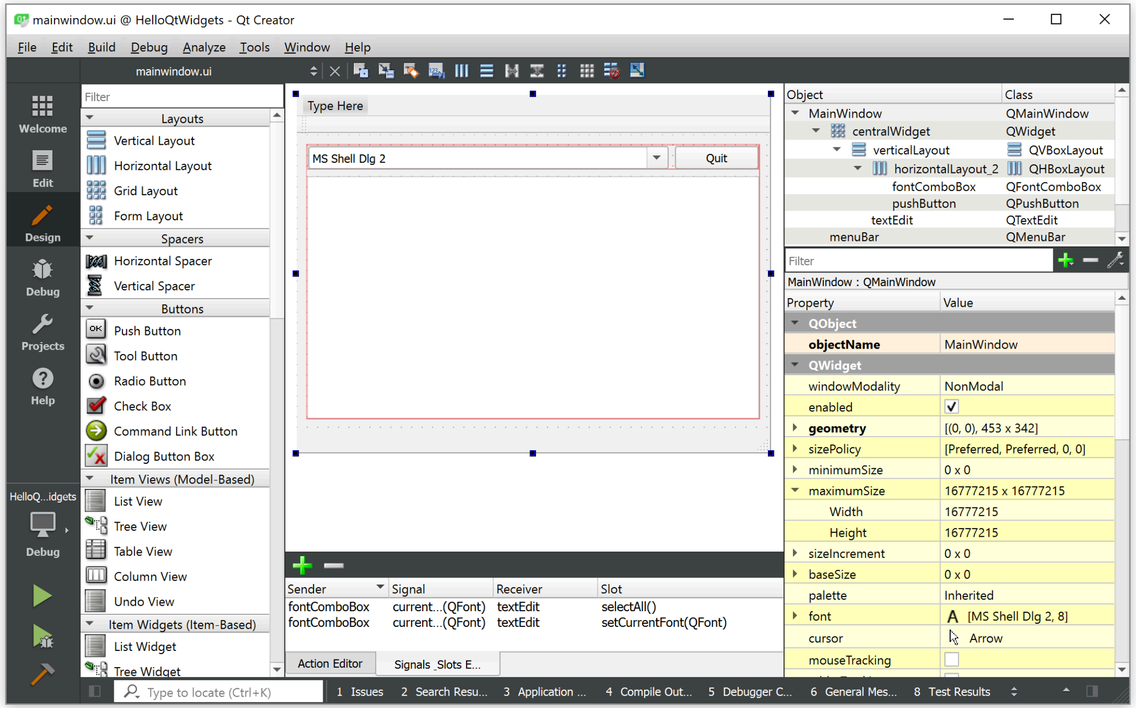
- Cross platform GUI toolkit
- **Qt Widgets**
  - Designed for the desktop
  - Standard widgets designed for WIMP interfaces
- **Qt Quick**
  - Focus on mobile devices and graphical effects
  - New UIDL

Qt Widgets

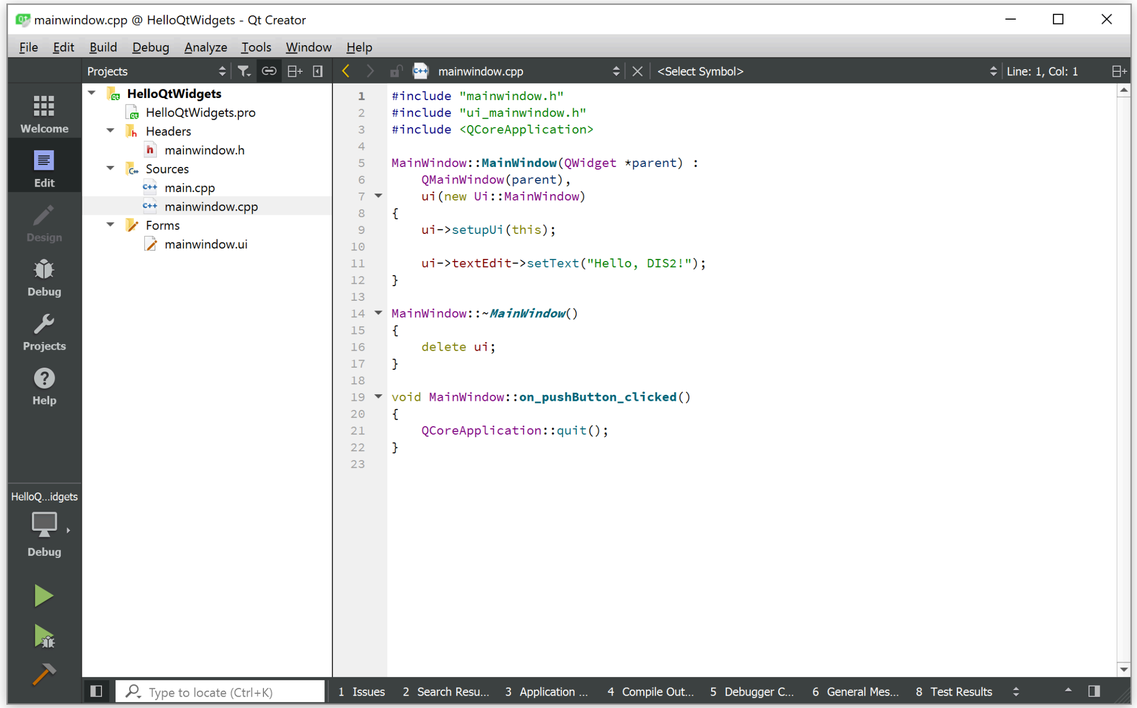
- Original version of Qt
- Designed for WIMP interfaces
- Runs on Mac, Windows, Linux
- UIDS with XML based files that are compiled into C++
- Emulates native look on every platform
- Rich library of widgets



Qt Widgets: UIDS



UI Layout



Source Code

Qt Widgets: Signals & Slots

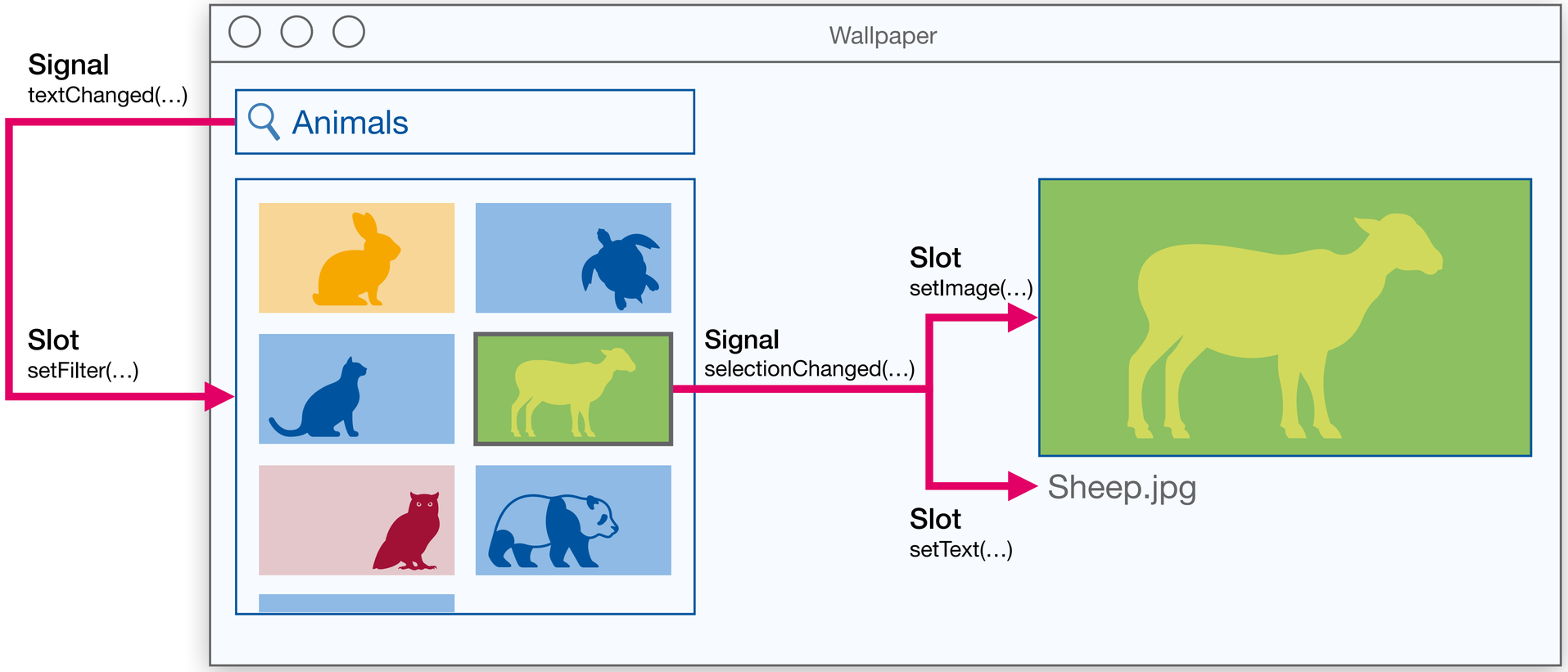
.h File

```
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);
```

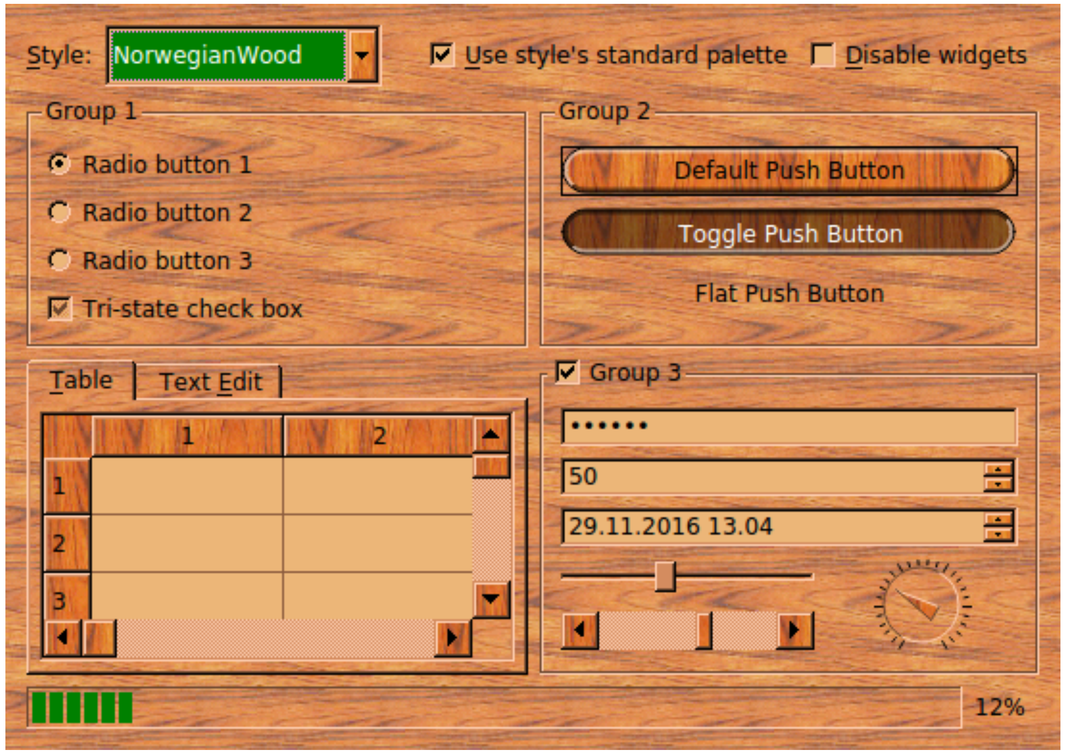
a: 3
b: 3

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

a: 3
b: 4

Qt Widgets: Styling the UI

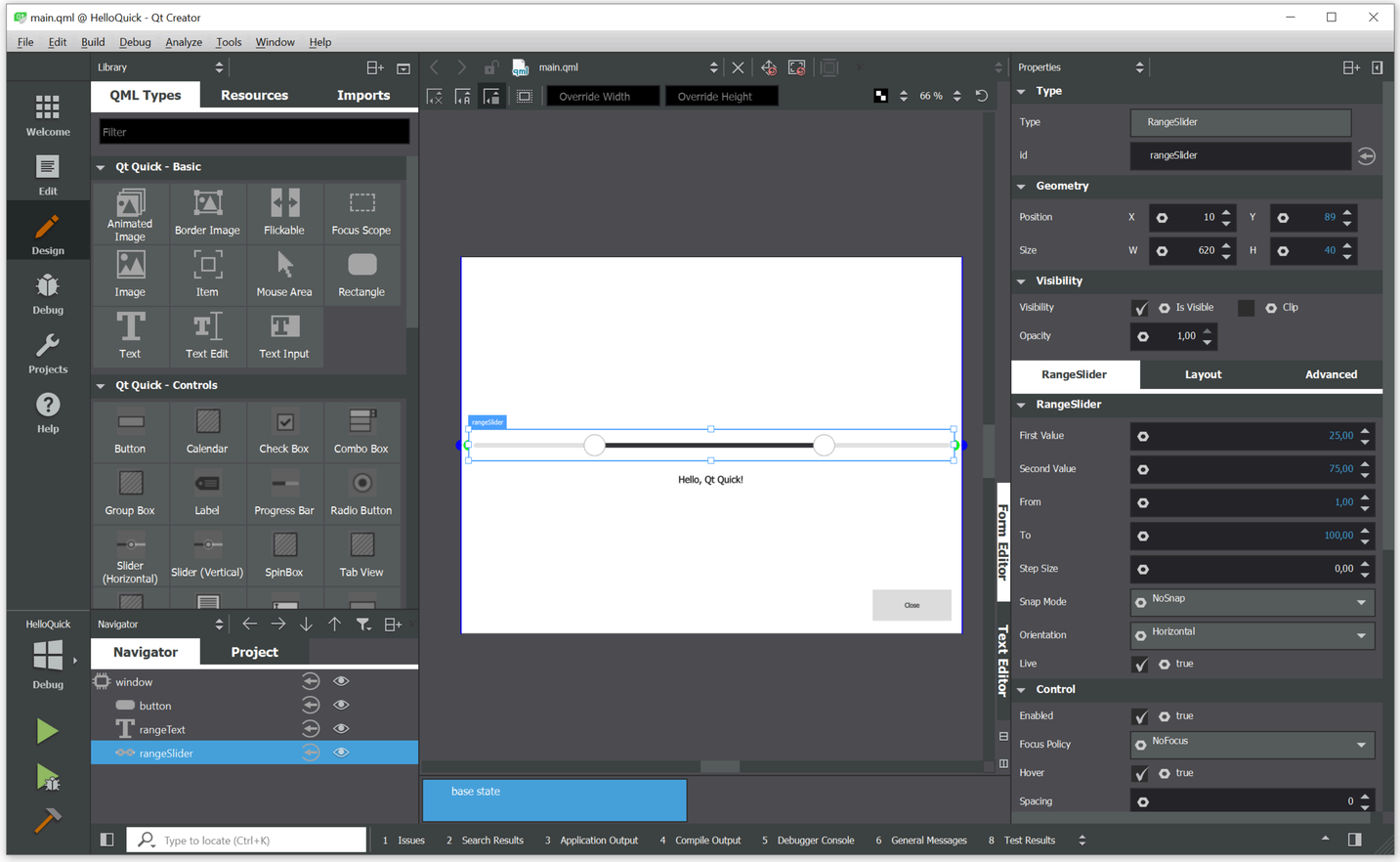
- The visual style of Qt Widgets applications is defined by the operating system
- QProxyStyle allows to make customizations across all system looks
- Not your first choice for a highly customized UI

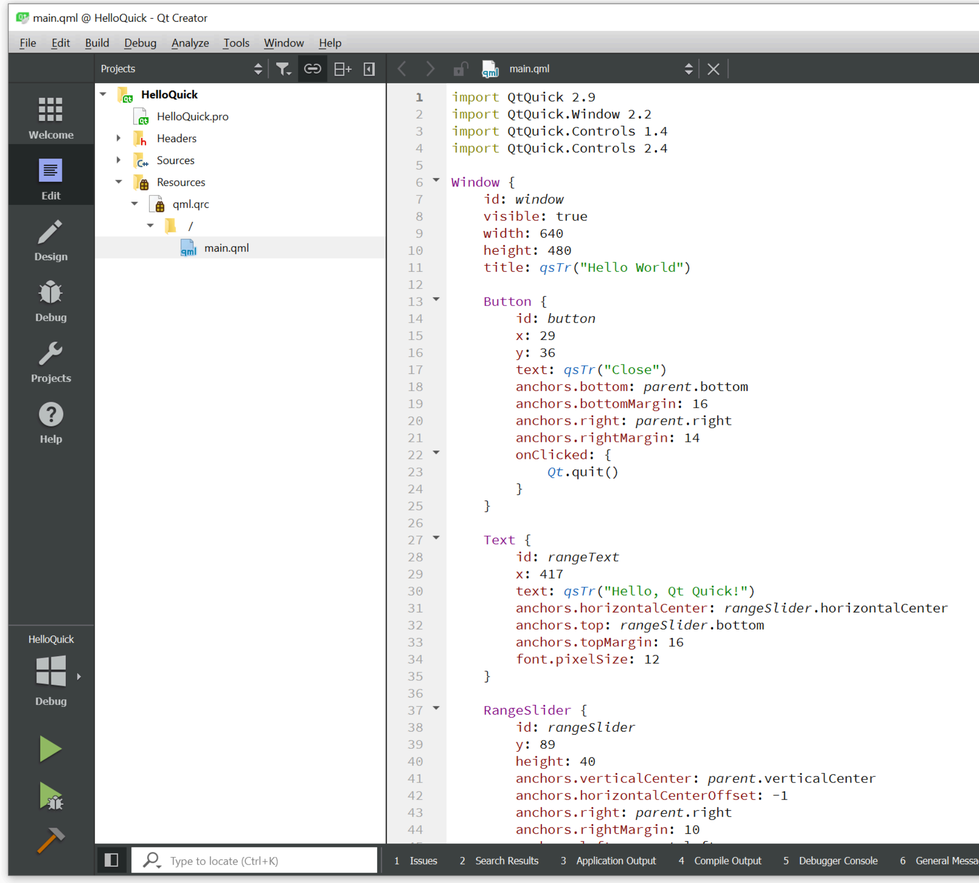


Qt User Interface Creation Kit (Quick)

- 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

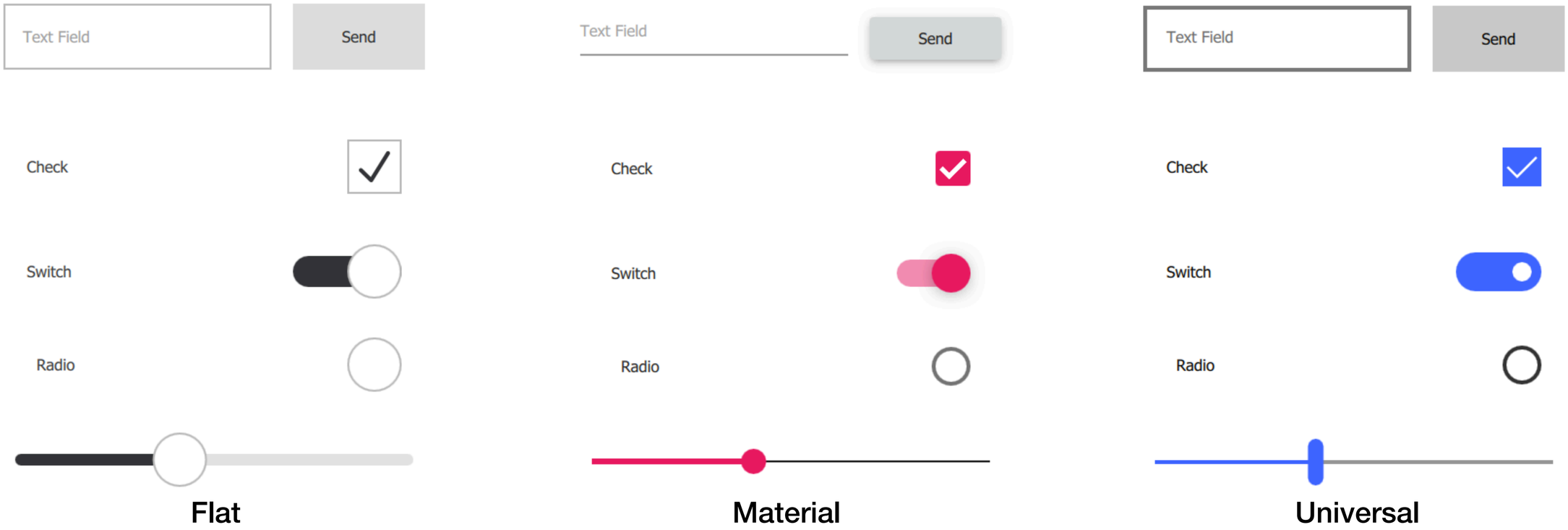
- A widget in the QML file can define **states**
- **Transitions** can be used to animate state changes

```
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()
    }
}
```

Summary: Cross-Platform Toolkits

- Challenges & Tradeoffs
  - Native vs. cross-platform style
  - VMs vs. Bridging
  - Widget complexity
- Trends
  - Decoupling of UI and code
  - Integrating web & mobile technologies
  - Animation as first class citizen
- Patterns
  - MVC
  - Delegation (LayoutManagers,…)
  - Listeners, Signals & Slots
  - Pluggable Look & Feel
  - UIDLs
  - UITK life cycles
  - Developer productivity is key!