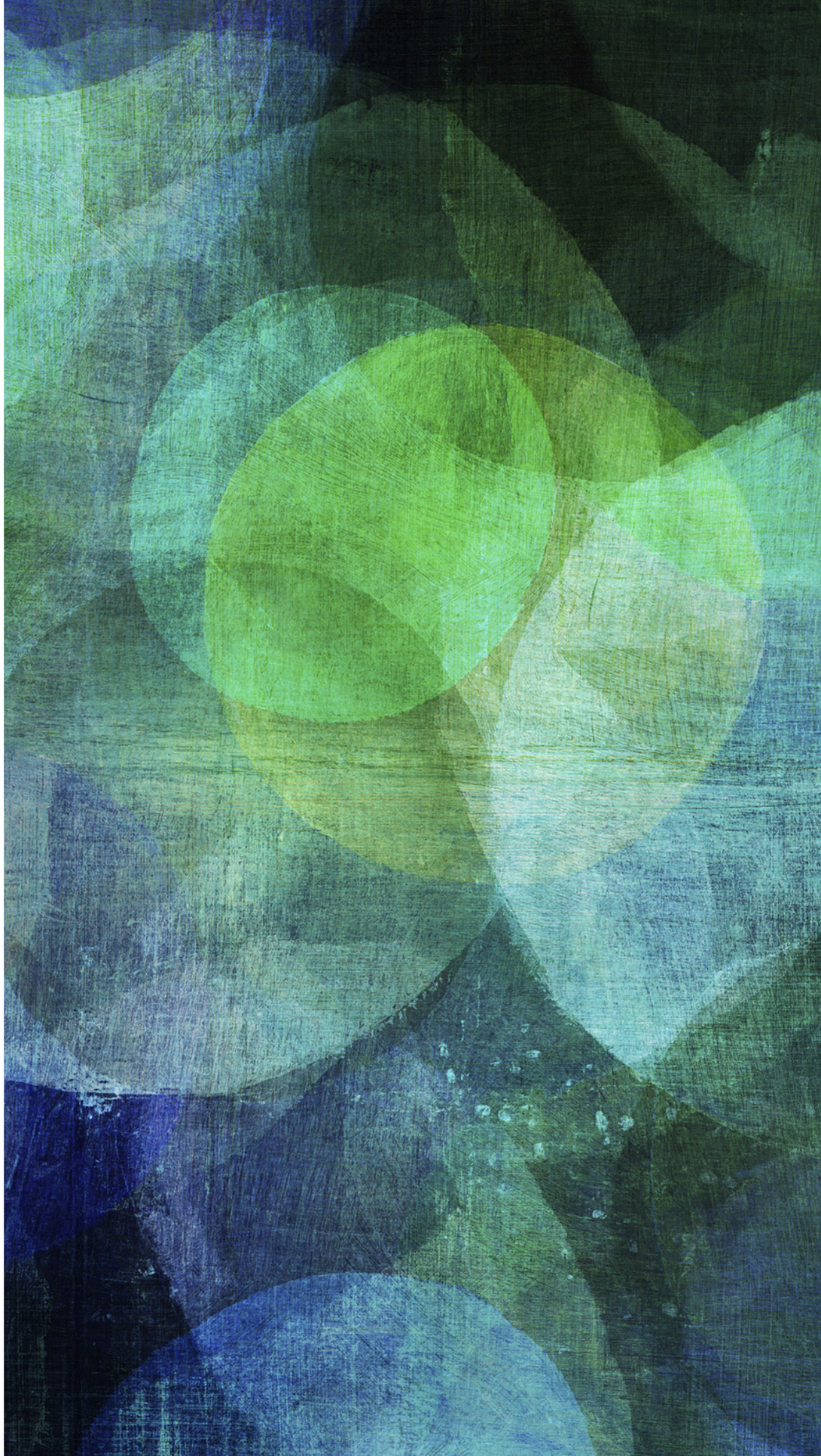
JAVA GUI SWING AND JAVAFX

Laborator 09
22-April-2019



GUI

- AWT (cea mai veche varianta, nu o vom trata astazi)
- Swing (a venit pentru a simplifica abordarea AWT)
  - Components
  - Layouts
  - Listeners
  - Dialogs
  - Swing Threads
  - Application example
- JavaFX

SWING FEATURES

- Platform Independent
- Customizable
- Extensible
- Configurable
- Lightweight
- Rich Controls
- Pluggable Look and Feel

COMPONENTS

- În JFC (Java Foundation Classes) / Swing găsim o multitudine de componente
- JLabel
- JButton
- JChekBox
- Choice
- JRadioButton
- JTextField
- JTextArea
- JList
- JScrollbar
- … (and more)
- https://docs.oracle.com/javase/tutorial/uiswing/components/index.html

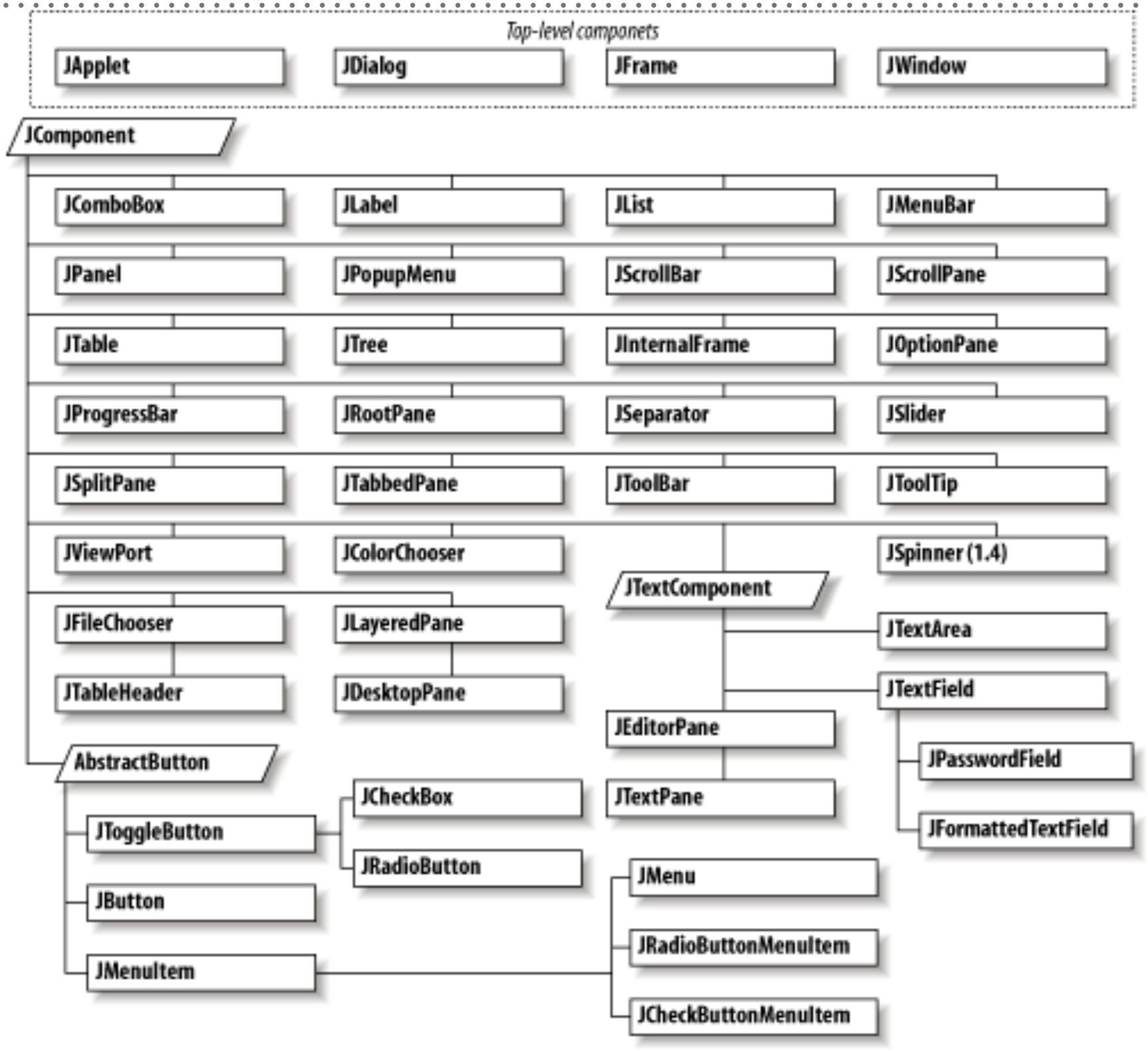
COMPONENTS

A component is an independent visual control. Swing Framework contains a large set of components which provide rich functionalities and allow high level of customization. They all are derived from JComponent class. All these components are lightweight components. This class provides some common functionality like pluggable look and feel, support for accessibility, drag and drop, layout, etc.

A container holds a group of components. It provides a space where a component can be managed and displayed. Containers are of two types:

1. Top level Containers
    - It inherits Component and Container of AWT.
    - It cannot be contained within other containers.
    - Heavyweight.
    - Example: JFrame, JDialog, JApplet
2. Lightweight Containers
    - It inherits JComponent class.
    - It is a general purpose container.
    - It can be used to organize related components together.
    - Example: JPanel

COMPONENTS

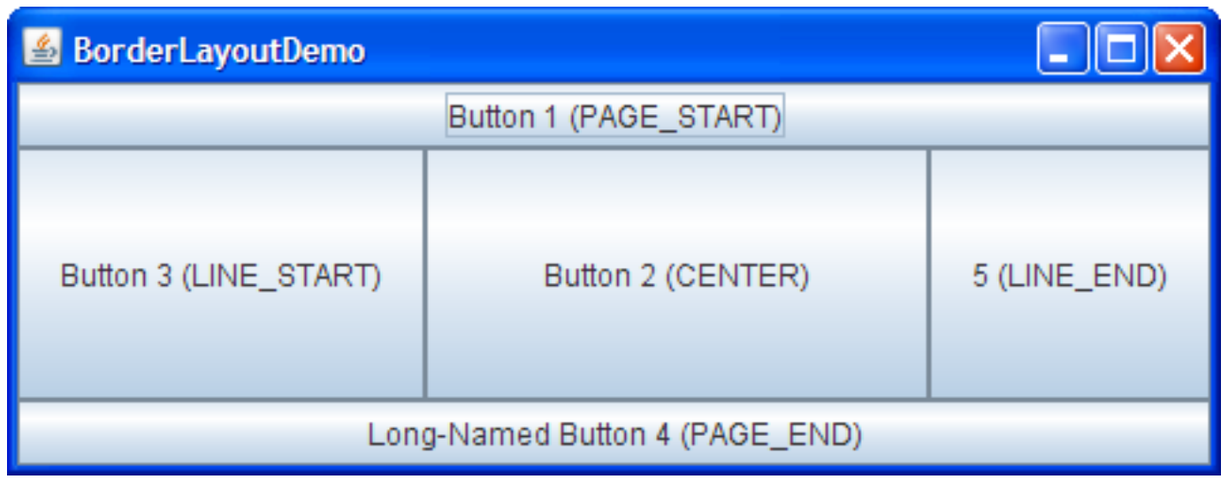

Top-level componets
JApplet
JDialog
JFrame
JWindow
JComponent
JComboBox
JLabel
JList
JMenuBar
JPanel
JPopupMenu
JScrollBar
JScrollPane
JTable
JTree
JInternalFrame
JOptionPane
JProgressBar
JRootPane
JSeparator
JSlider
JSplitPane
JTabbedPane
JToolBar
JToolTip
JViewPort
JColorChooser
JSpinner (1.4)
JTextComponent
JFileChooser
JLayeredPane
JTextArea
JTextField
JTableHeader
JDesktopPane
JEditorPane
JPasswordField
AbstractButton
JCheckBox
JTextPane
JFormattedTextField
JToggleButton
JRadioButton
JMenu
JButton
JRadioButtonMenuItem
JMenuItem
JCheckButtonMenuItem

LAYOUT OPTIONS

- BorderLayout
- BoxLayout
- CardLayout
- FlowLayout
- GridBagLayout
- GridLayout
- SpringLayout
- https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

LAYOUT OPTIONS

- BorderLayout
- https://docs.oracle.com/javase/tutorial/uiswing/layout/border.html



EVENT HANDLING

Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven. Event describes the change in state of any object. **For Example :** Pressing a button, Entering a character in Textbox, Clicking or Dragging a mouse, etc.

EVENT HANDLING

➤ Components of Event Handling

Event handling has three main components,

- ➤ **Events :** An event is a change in state of an object.
- ➤ **Events Source :** Event source is an object that generates an event.
- ➤ **Listeners :** A listener is an object that listens to the event. A listener gets notified when an event occurs.

EVENT HANDLING

➤ How events are handled

A source generates an Event and send it to one or more listeners registered with the source. Once event is received by the listener, they process the event and then return. Events are supported by a number of Java packages, like **java.util**, **java.awt** and **java.awt.event**.

EVENT LISTENERS

- ActionListener
- AdjustmentListener
- ComponentListener
- FocusListener
- ItemListener
- KeyListener
- MouseListener
- MouseWheelListener
- TreeExpansionListener
- Text Listener
- WindowListener
- … (many more)

EVENT LISTENERS

Listener Interface	Adapter Class	Listener Methods
ActionListener	none	actionPerformed(ActionEvent)
AncestorListener	none	ancestorAdded(AncestorEvent) ancestorMoved(AncestorEvent) ancestorRemoved(AncestorEvent)
CaretListener	none	caretUpdate(CaretEvent)
CellEditorListener	none	editingStopped(ChangeEvent) editingCanceled(ChangeEvent)
ChangeListener	none	stateChanged(ChangeEvent)
ComponentListener	ComponentAdapter	componentHidden(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
ContainerListener	ContainerAdapter	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
DocumentListener	none	changedUpdate(DocumentEvent) insertUpdate(DocumentEvent) removeUpdate(DocumentEvent)
ExceptionListener (introduced in 1.4)	none	exceptionThrown(Exception)
FocusListener	FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)
HierarchyBoundsListener (introduced in 1.3)	HierarchyBoundsAdapter	ancestorMoved(HierarchyEvent) ancestorResized(HierarchyEvent)
HierarchyListener (introduced in 1.3)	none	hierarchyChanged(HierarchyEvent)
HyperlinkListener	none	hyperlinkUpdate(HyperlinkEvent)
InputMethodListener	none	caretPositionChanged(InputMethodEvent) inputMethodTextChanged(InputMethodEvent)
InternalFrameListener	InternalFrameAdapter	internalFrameActivated(InternalFrameEvent) internalFrameClosed(InternalFrameEvent) internalFrameClosing(InternalFrameEvent) internalFrameDeactivated(InternalFrameEvent) internalFrameDeiconified(InternalFrameEvent) internalFrameIconified(InternalFrameEvent) internalFrameOpened(InternalFrameEvent)

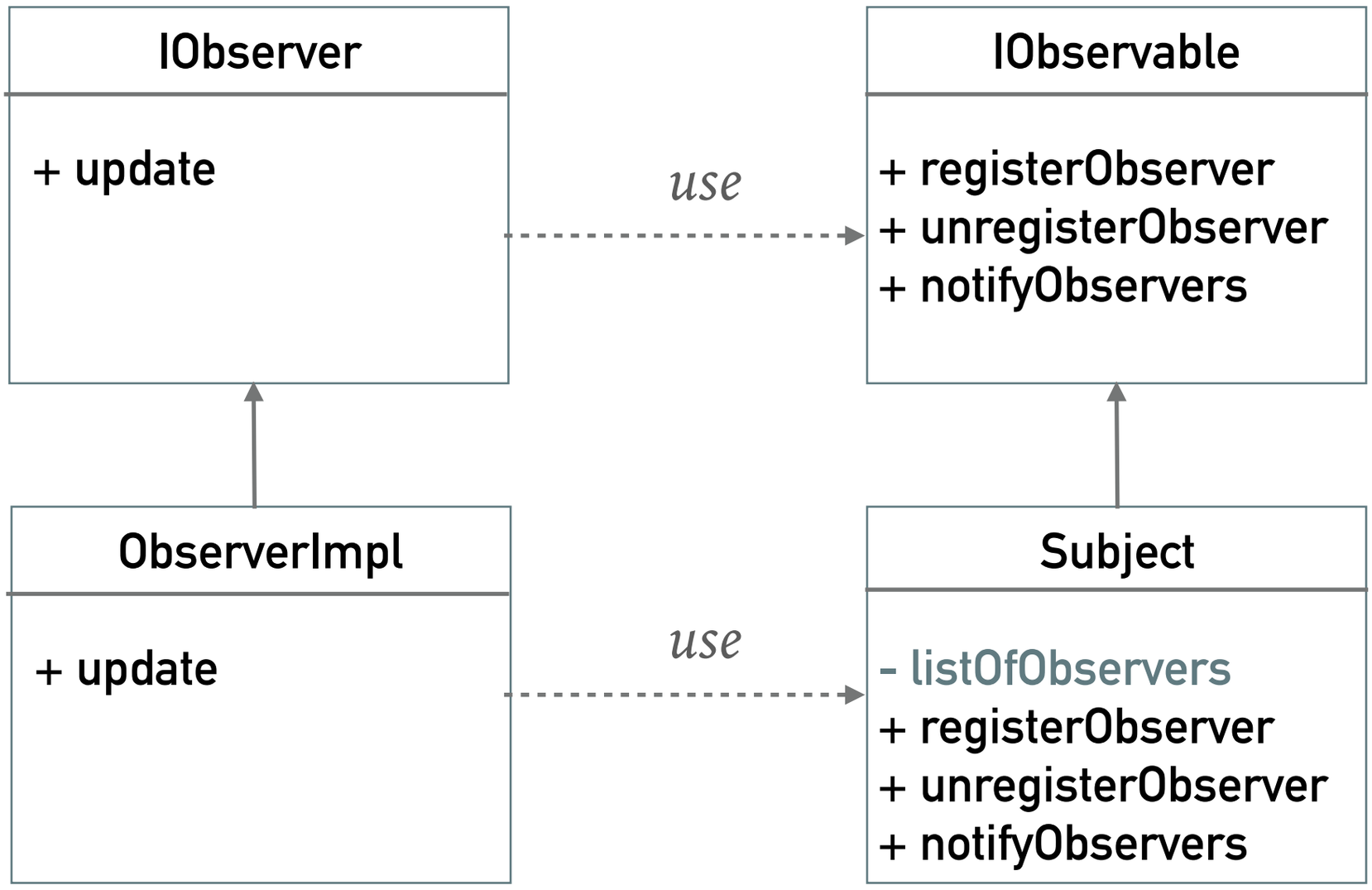
EVENT LISTENERS

Listener Interface	Adapter Class	Listener Methods
ItemListener	none	itemStateChanged(ItemEvent)
KeyListener	KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
ListDataListener	none	contentsChanged(ListDataEvent) intervalAdded(ListDataEvent) intervalRemoved(ListDataEvent)
ListSelectionListener	none	valueChanged(ListSelectionEvent)
MenuDragMouseListener	none	menuDragMouseDragged(MenuDragMouseEvent) menuDragMouseEntered(MenuDragMouseEvent) menuDragMouseExited(MenuDragMouseEvent) menuDragMouseReleased(MenuDragMouseEvent)
MenuKeyListener	none	menuKeyPressed(MenuKeyEvent) menuKeyReleased(MenuKeyEvent) menuKeyTyped(MenuKeyEvent)
MenuListener	none	menuCanceled(MenuEvent) menuDeselected(MenuEvent) menuSelected(MenuEvent)
MouseInputListener (extends MouseListener and MouseMotionListener)	MouseInputAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseDragged(MouseEvent) mouseMoved(MouseEvent)
MouseListener	MouseAdapter, MouseInputAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener	MouseMotionAdapter, MouseInputAdapter	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
MouseWheelListener (introduced in 1.4)	none	mouseWheelMoved(MouseWheelEvent)
PopupMenuListener	none	popupMenuCanceled(PopupMenuEvent) popupMenuWillBecomeInvisible(PopupMenuEvent) popupMenuWillBecomeVisible(PopupMenuEvent)

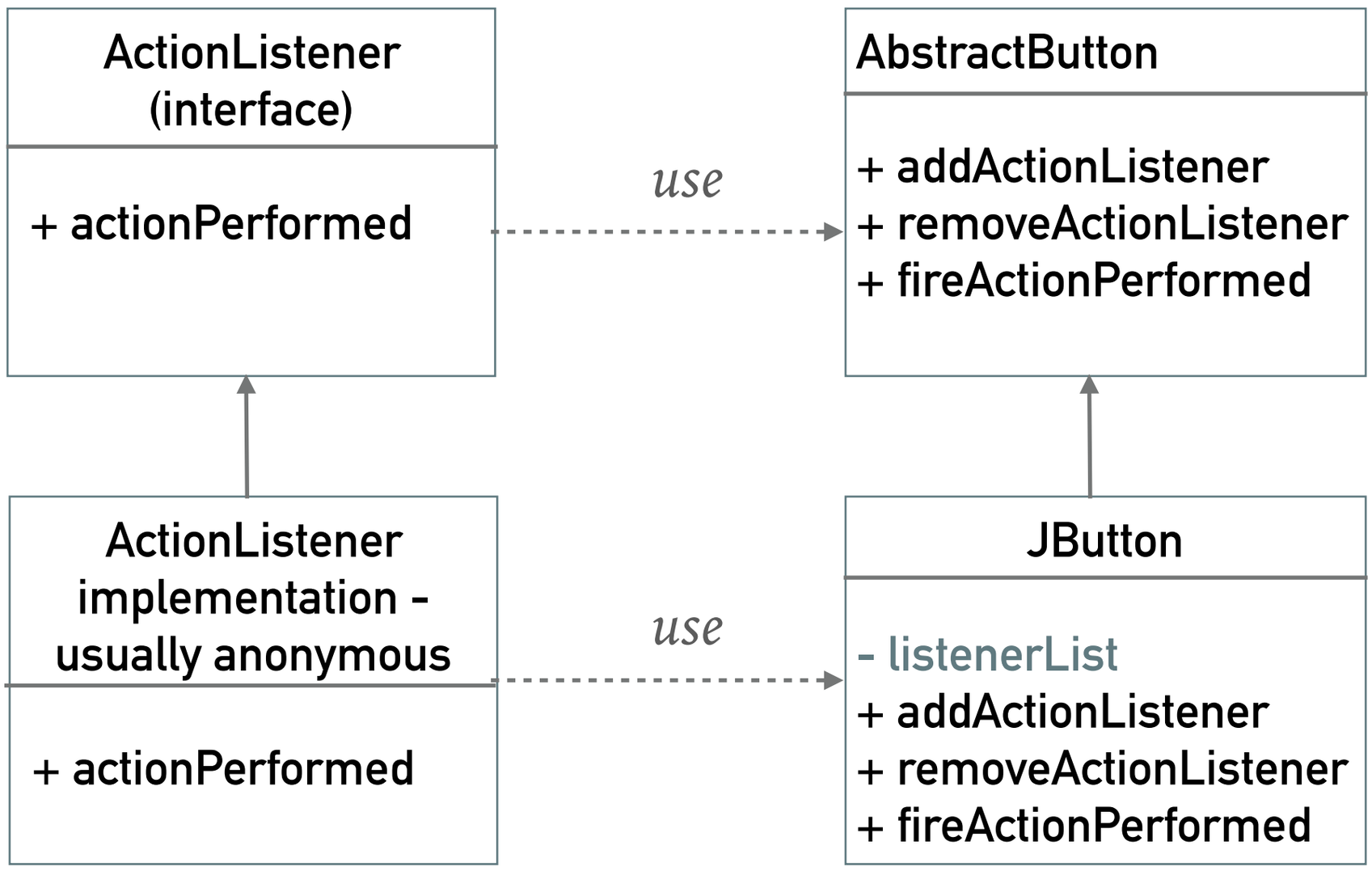
EVENT LISTENERS

Listener Interface	Adapter Class	Listener Methods
PropertyChangeListener	none	propertyChange(PropertyChangeEvent)
TableColumnModelListener	none	columnAdded(TableColumnModelEvent) columnMoved(TableColumnModelEvent) columnRemoved(TableColumnModelEvent) columnMarginChanged(ChangeEvent) columnSelectionChanged(ListSelectionEvent)
TableModelListener	none	tableChanged(TableModelEvent)
TreeExpansionListener	none	treeCollapsed(TreeExpansionEvent) treeExpanded(TreeExpansionEvent)
TreeModelListener	none	treeNodesChanged(TreeModelEvent) treeNodesInserted(TreeModelEvent) treeNodesRemoved(TreeModelEvent) treeStructureChanged(TreeModelEvent)
TreeSelectionListener	none	valueChanged(TreeSelectionEvent)
TreeWillExpandListener	none	treeWillCollapse(TreeExpansionEvent) treeWillExpand(TreeExpansionEvent)
UndoableEditListener	none	undoableEditHappened(UndoableEditEvent)
VetoableChangeListener	none	vetoableChange(PropertyChangeEvent)
WindowFocusListener (introduced in 1.4)	WindowAdapter	windowGainedFocus(WindowEvent) windowLostFocus(WindowEvent)
WindowListener	WindowAdapter	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)
WindowStateListener (introduced in 1.4)	WindowAdapter	windowStateChanged(WindowEvent)

DP OBSERVER


IObserver
+ update
use
IObservable
+ registerObserver
+ unregisterObserver
+ notifyObservers
ObserverImpl
+ update
use
Subject
- listOfObservers
+ registerObserver
+ unregisterObserver
+ notifyObservers

ACTION LISTENER



DIALOGS

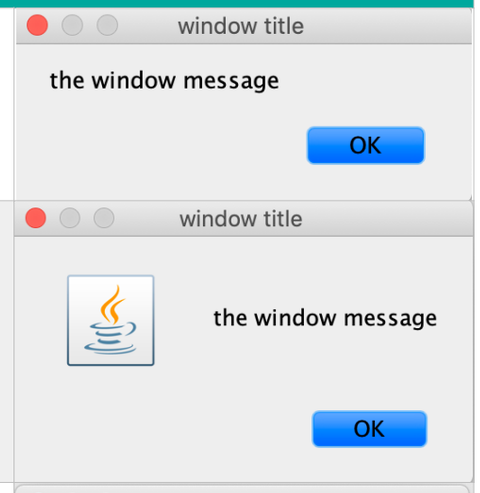
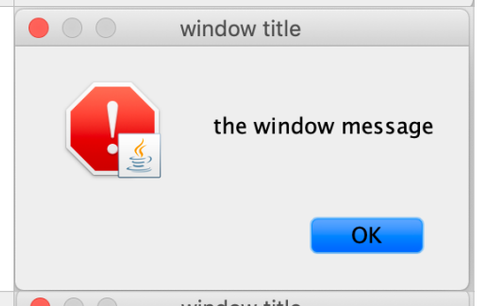
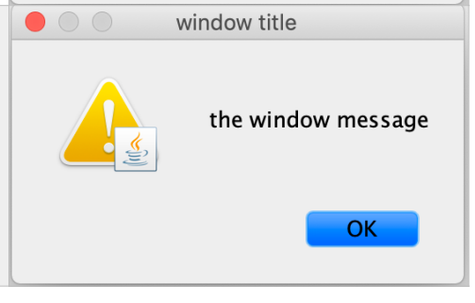
- JDialog - este folosită pentru a crea dialoguri customizate
- JOptionPane - pentru a crea dialoguri INFO, WARNING etc
- ProgressMonitor - pentru a monitoriza progresul unui proces
- JColorChooser - alegerea unei culori
- JFileChooser - alegerea unui fișier

DIALOGS

- JOptionPane - pentru a crea dialoguri INFO, WARNING etc

```
JOptionPane.showMessageDialog(jFrame, "the window message", "window title",
        JOptionPane.WARNING_MESSAGE);
```

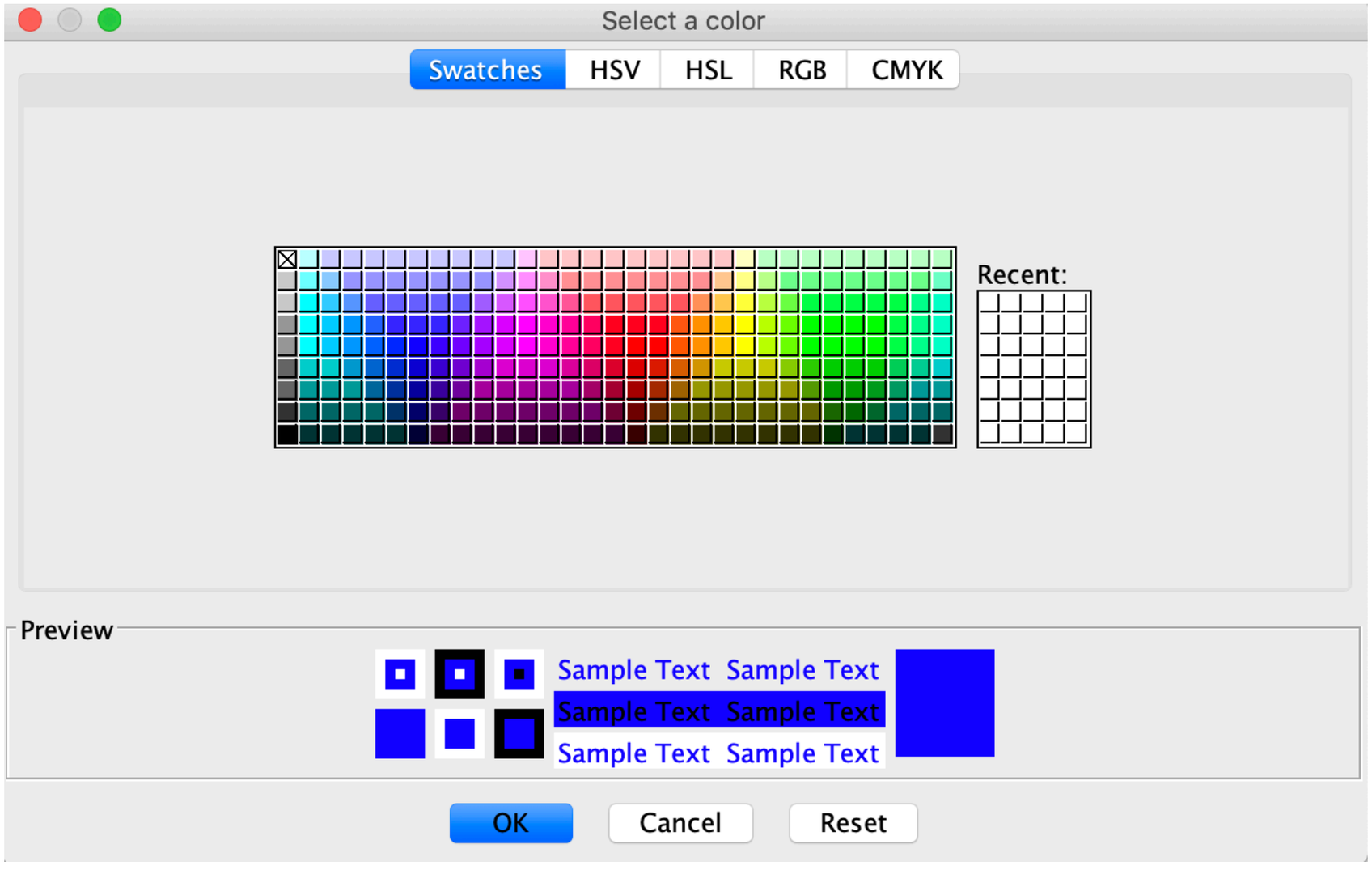
- showConfirmationDialog
- showOptionDialog

Option showMessageDialog	Result
`JOptionPane.PLAIN_MESSAGE`	
`JOptionPane.INFORMATION_MESSAGE`	
`JOptionPane.ERROR_MESSAGE`	
`JOptionPane.WARNING_MESSAGE`	

DIALOGS

- JColorChooser - alegerea unei culori

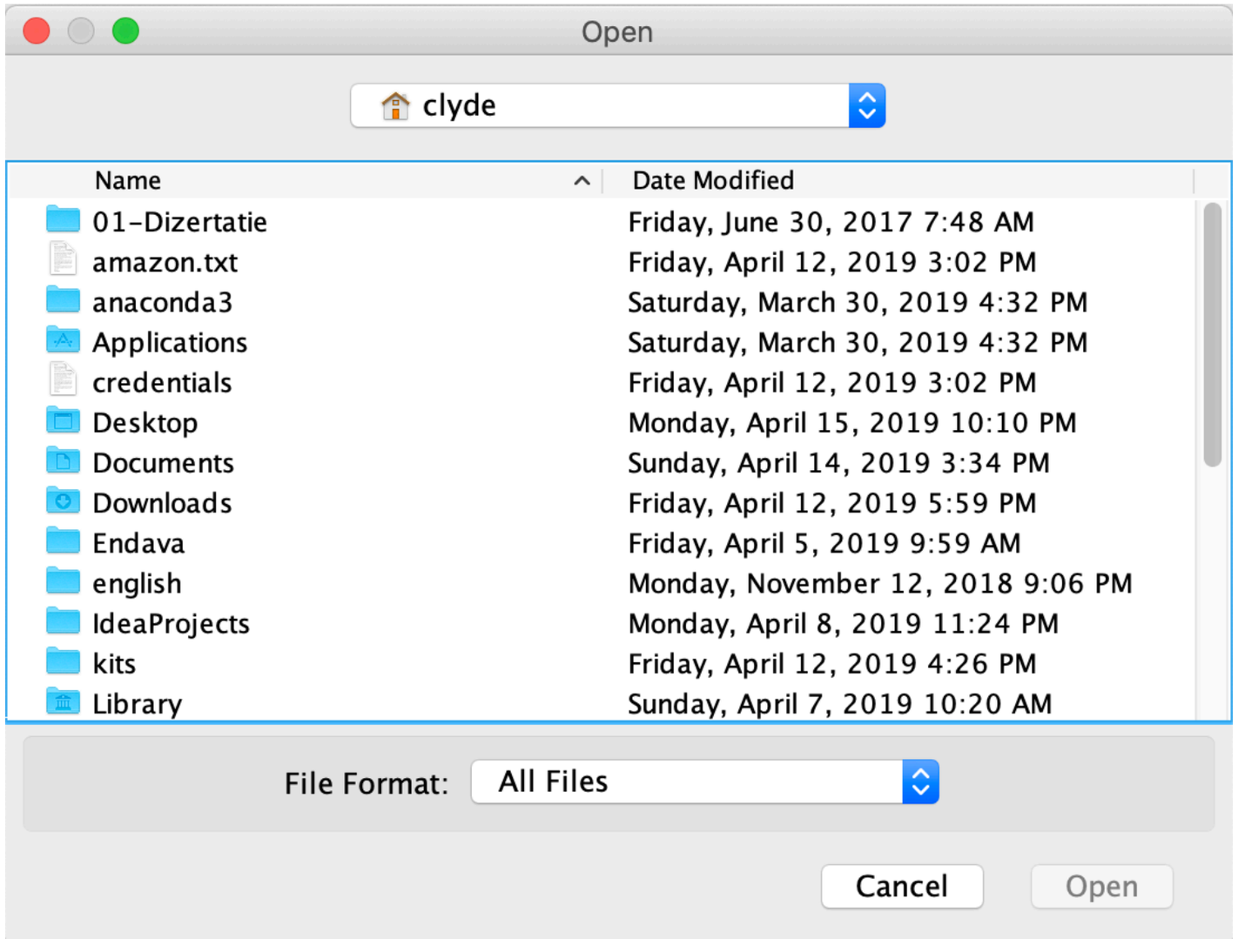
```
Color color = JColorChooser.showDialog(jFrame,
  "Select a color”,Color.BLUE);
```



DIALOGS

- JFileChooser - alegerea unui fișier

```
final JFileChooser fc = new JFileChooser();
int returnVal = fc.showOpenDialog(jFrame);
```



AWT / SWING CODE EXAMPLE

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class DialogTestMain {

    public static void main(String[] args) {
        // create a frame with a title
        JFrame jFrame = new JFrame("My first Swing application");
        // set preferred size to be used in pack
        jFrame.setPreferredSize( new Dimension(800, 600));
        // set the default close action to be system exit
        jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // create a panel to be added later to the JFrame
        JPanel jPanel = new JPanel();
        // add a button in the panel
        JButton testButton = new JButton("Button");
        jPanel.add(testButton);
        // add an action for clicking on the button,
        // ActionListener is an awt interface
        testButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
//                JOptionPane.showConfirmDialog(jFrame, "the window message", "window title",
//                        JOptionPane.OK_OPTION, JOptionPane.WARNING_MESSAGE);
//                JOptionPane.showMessageDialog(jFrame, "the window message", "window title",
//                        JOptionPane.WARNING_MESSAGE);
//                Color color = JColorChooser.showDialog(jFrame,"Select a color",Color.BLUE);

                //Create a file chooser
                final JFileChooser fc = new JFileChooser();
                int returnVal = fc.showOpenDialog(jFrame);

            }
        });
        // add the panel to the frame
        jFrame.setContentPane(jPanel);
        // perform configuration of window
        jFrame.pack();
        // make it visible
        jFrame.setVisible(true);

    }
}
```

SWING THREADS

În cadrul unei aplicații swing găsim un număr de thread-uri

- Thread-ul initial, cel care executa codul aplicației
- Even dispatch thread, unde sunt executate toate portiunile de cod din event-handling. Majoritatea codului ce interacționează cu framework-ul swing trebuie să se execute pe acest thread.
- Worker threads, cunoscute sub numele de thread-uri background, unde se execută task-urile time-consuming.

Aceste thread-uri sunt create automat de framework, datoria dezvoltatorului este să le utilizeze astfel încât să obțină o aplicație responsove.

SWING THREADS

javax.swing.SwingWorker<T, V>

- abstract T doInBackground()

este metoda ce execută task-ul in background si returnează rezultatul

- void process(List<V> data)

metoda ce proceseaza rezultatele intermediare, in event dispatch thread. Aici se poate face updatarea interfeței grafice cu rezultatele intermediare pentru a notifica utilizatorul că se petrece ceva.

- void publish(V... data)

face forward la date către event dispatch thread, către meted process. A se chema din metoda doInBackground

- void execute()

programeaza acest worker pentru executie de catre un worker thread

- SwingWorker.StateValue getState()

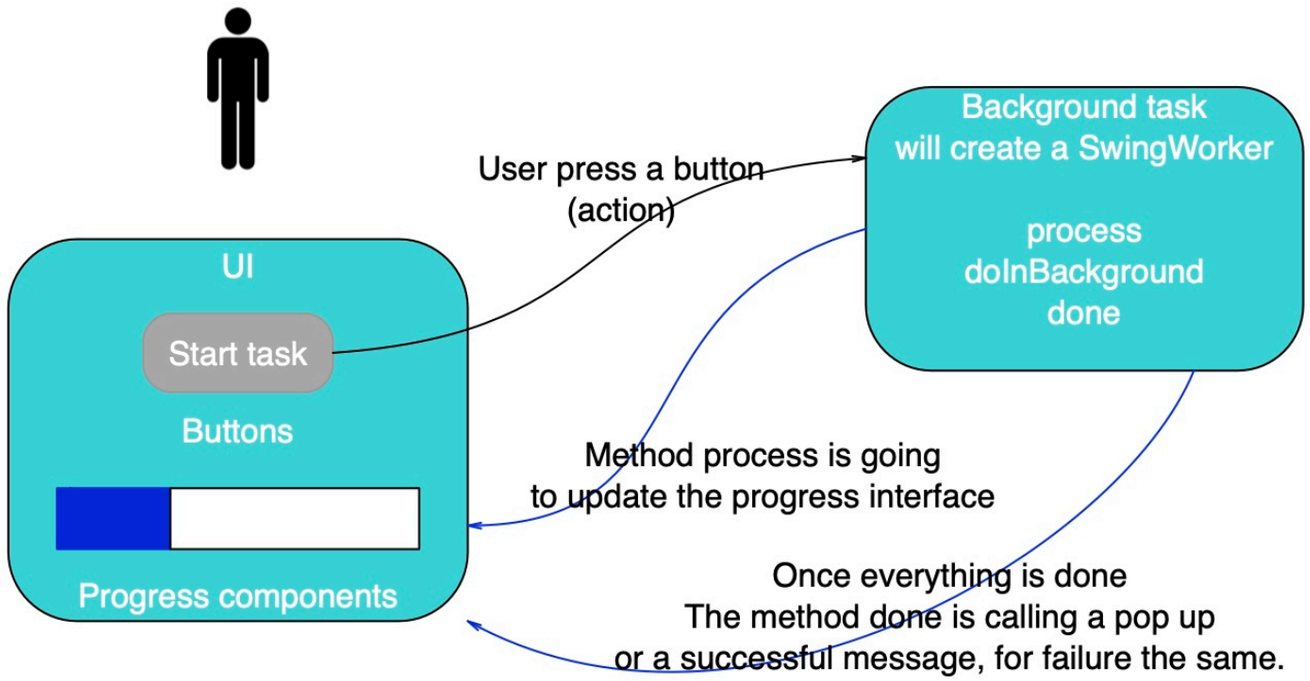
obtine starea a acestui worker, valorile pot fi din PENDING, STARTED, or DONE.

SWING

For long tasks is better to have a separate thread.

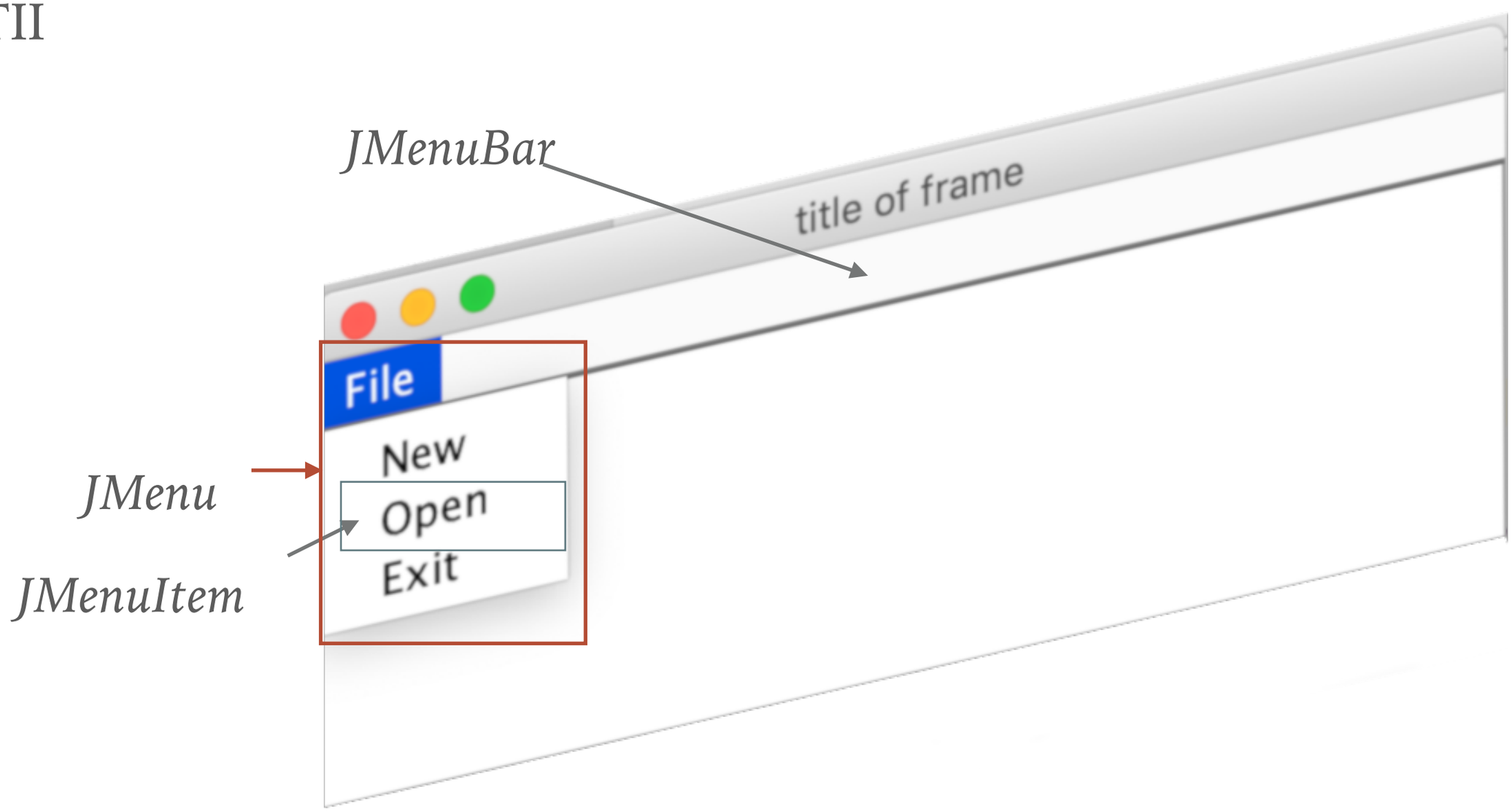
Also from time to time the UI needs to be updated in order to offer a seamless experience to the user. (e.g. 30% is done, we are at processing user X)

Once the task is done a pop-up or an area on the GUI needs to show that. The user needs to be aware about success or failure of his/her actions.



SWING

APLICAȚII



https://docs.oracle.com/javase/8/docs/api/javax/swing/JMenuBar.html

https://docs.oracle.com/javase/8/docs/api/javax/swing/JMenu.html

https://docs.oracle.com/javase/8/docs/api/javax/swing/JMenuItem.html

SWING

Swing ce am abordat:

- Components
  - JButtons, JList, JTable, Dialogs
- Layouts
  - BoxLayout
- Listeners
  - How to get an event from a Component (JButton)
- Swing Threads
  - How to keep or application responsive
- Application example

JAVAFX

- JavaFX ce vom aborda
  - Features
  - Controls
  - JavaFX Threads
  - Application example

JAVAFX

- JavaFX is Java's next-generation **Graphical User Interface (GUI)** toolkit. It's a platform that makes it easy to rapidly build high-performance Java client-side applications.
- JavaFX 8 is written totally from scratch in Java language, it makes you feel at home. Therefore, applications written in JavaFX can be deployed on desktops, laptops, the Web, embedded systems, mobiles, and tablets.
- In addition, JavaFX's flexible FXML support allows you to build **MVC** (**Model-View-Controller**) architectural pattern applications easily, and use the WYSIWYG approach using the Scene Builder tool.

JAVAFX FEATURES

- **Java APIs**: JavaFX is a Java library that consists of classes and interfaces that are written in Java code.
- **FXML and Scene Builder**: This is an XML-based declarative markup language for constructing a JavaFX application user interface. You can code in FXML or use JavaFX Scene Builder to interactively design the GUI. Scene Builder generates FXML markup that can be ported to an IDE like NetBeans, where you can add the business logic. Moreover, the FXML file that is generated can be used directly inside the JavaFX application.
- **WebView**: This is a web component that uses `WebKit`, an HTML render engine technology, to make it possible to embed web pages within a JavaFX application. JavaScript running in `WebView` can call Java APIs and vice-versa.
- **Swing/SWT interoperability**: The existing Swing and SWT applications can benefit from JavaFX features such as rich graphics, media playback, and embedded web content.
- **Built-in UI controls and CSS**: JavaFX provides all the major UI controls, and some extra uncommon controls like charts, pagination, and accordion that are required to develop a full-featured application. Components can be skinned with standard web technologies such as CSS.
- **3D graphics features**: Support for the 3D graphics library is included.
- **Canvas API**: You can draw directly inside a JavaFX scene area using the Canvas API, which consists of one graphical element (node).
- **Multitouch support**: Multitouch operations are supported based on the capabilities of the underlying platform.
- **Hardware-accelerated graphics pipeline**: JavaFX graphics are based on the graphics-rendering pipeline, *Prism*. The Prism engine smoothly and quickly renders JavaFX graphics when used with a supported graphics card or **graphics processing unit** (**GPU**). If a system does not feature one of them, then Prism defaults to the software-rendering stack.
- **High-performance media engine**: This engine provides a stable, low-latency media framework that is based on the `GStreamer` multimedia framework. The playback of web multimedia content is supported with the media pipeline.
- **Self-contained deployment model**: Self-contained application packages have all of the application resources and a private copy of the Java and JavaFX runtimes. They are distributed as native installable packages and provide the same installation and launch experience as native applications for that operating system.

JAVAFX PACKAGES

Module Name	Description
javafx.base	Defines the base APIs for the JavaFX UI toolkit, including APIs for bindings, properties, collections, and events.
javafx.controls	Defines the UI controls, charts, and skins that are available for the JavaFX UI toolkit.
javafx.fxml	Defines the FXML APIs for the JavaFX UI toolkit.
javafx.graphics	Defines the core scene graph APIs for the JavaFX UI toolkit such as layout containers, application lifecycle, shapes, transformations, canvas, input, painting, image handling, and effects. It also defines APIs for animation, CSS, concurrency, geometry, printing, and windowing.
javafx. media	Defines APIs for playback of media and audio content.
javafx.swing	Defines APIs for the JavaFX/Swing interoperability, which allows mixing Swing and JavaFX components in the same application.
javafx.web	Defines APIs for the loading and displaying web contents.

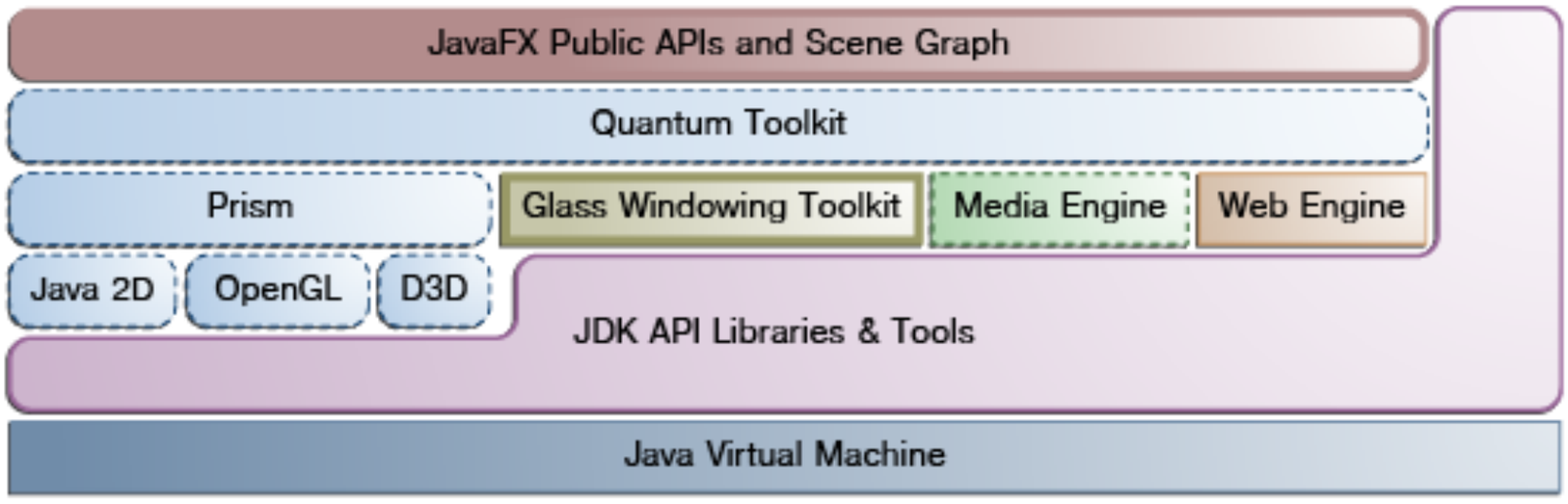
JAVAFX

- Controls
- JavaFX is having a big number of controls and also the possibilities to customise them using CSS

JAVAFX

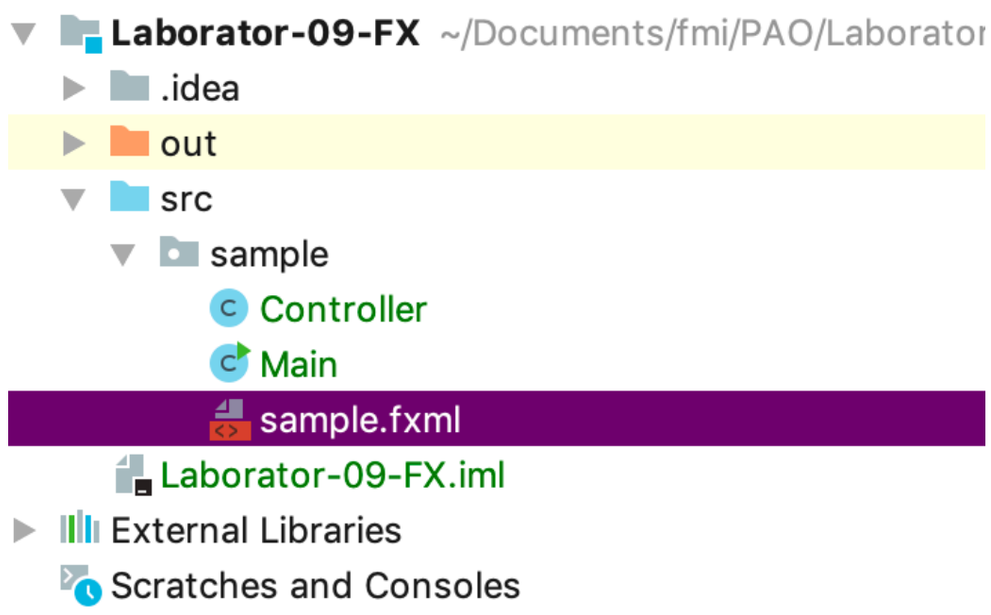
JavaFX architecture

- Scene Graph
- Java Public APIs for JavaFX Features
- Graphics System
- Glass Windowing Toolkit
- Media and Images
- Web Component
- CSS
- UI Controls
- Layout
- 2-D and 3-D Transformations
- Visual Effects



JAVAFX

- IntelliJ is having an application type JavaFX
- Once you build an application using JavaFX prototype, you are getting a project like below.

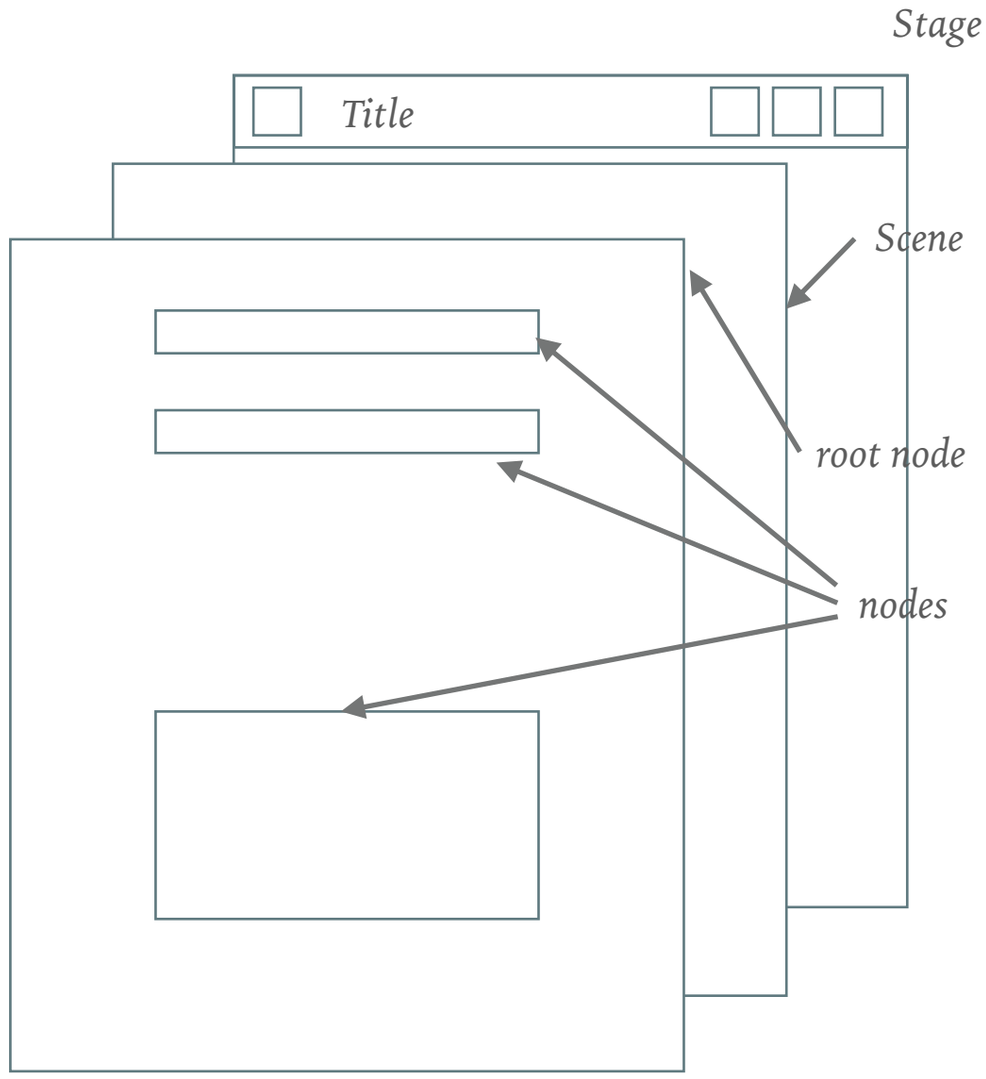


- The project is build using a MVC pattern (a controller, a view - the dxml file and a Main class to start the program
- For editing the fxml file it is good to have a Scene builder application. You can download the new version from here: https://gluonhq.com/products/scene-builder/#download

JAVAFX

Window structure

- The main class for a JavaFX application should extend the `javafx.application.Application` class. The `start()` method is the *main entry point* for all JavaFX applications.
- A JavaFX application defines the user interface container by means of a *stage* and a *scene*. The JavaFX `Stage` class is the top-level JavaFX container. The JavaFX `Scene` class is the container for all content.
- In JavaFX, the content of the scene is represented as a hierarchical scene graph of nodes.




```
@Override
public void start(Stage primaryStage) throws Exception{
    Parent root =
FXMLLoader.load(getClass().getResource("sample.fxml"));
    primaryStage.setTitle("Hello World");
    primaryStage.setScene(new Scene(root, 800, 480));
    primaryStage.show();
}
```

JAVAFX

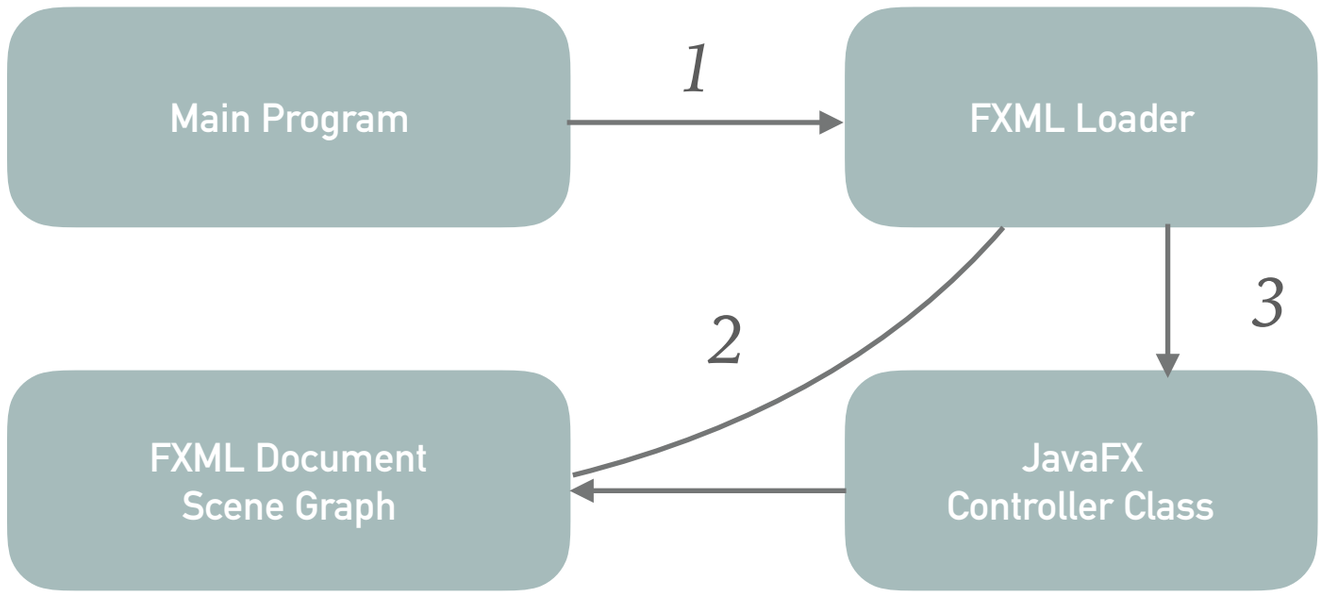
➤FXML is a markup language for defining layouts in JavaFX

```
<?xml version="1.0" encoding="UTF-8"?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.text.*?>
<?import javafx.scene.layout.*?>
<GridPane hgap="10" vgap="10">
   <padding>
      <Insets top="10" right="10" bottom="10" left="10"/>
   </padding>
   <children>
      <Label text="User name:" GridPane.columnIndex="0" GridPane.rowIndex="0"
            GridPane.halignment="RIGHT"/>
      <Label text="Password:" GridPane.columnIndex="0" GridPane.rowIndex="1"
            GridPane.halignment="RIGHT"/>
      <TextField GridPane.columnIndex="1" GridPane.rowIndex="0"/>
      <PasswordField GridPane.columnIndex="1" GridPane.rowIndex="1"/>
      <HBox GridPane.columnIndex="0" GridPane.rowIndex="2"
            GridPane.columnSpan="2" alignment="CENTER" spacing="10">
         <children>
            <Button text="Ok"/>
            <Button text="Cancel"/>
         </children>
      </HBox>
   </children>
</GridPane>
```

➤Main.java

```java
package sample;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        // load the view from fxml file
        Parent root = FXMLLoader.load(getClass().getResource("sample.fxml"));
        // configuring the Stage and show it
        primaryStage.setTitle("Hello World");
        primaryStage.setScene(new Scene(root, 300, 275));
        primaryStage.show();
    }


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

JAVAFX



1. *Main program invokes FXML Loader.*
2. *FXML Loader parses FXML Document and builds scene graph.*
3. *FXML Loader instantiates Controller and invokes Controller's initialize() method.*

JAVAFX

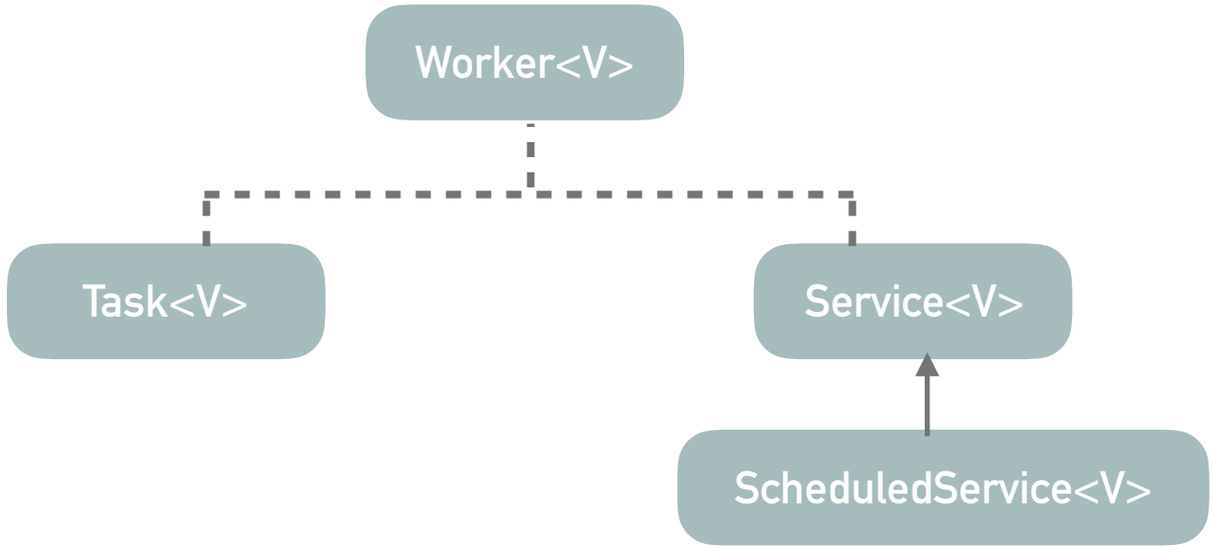
Worker, Service and Task (javafx.concurrent)

Both the Task and Service classes implement the Worker interface with common JavaFX properties and a well-defined life cycle.

Worker to be used only once

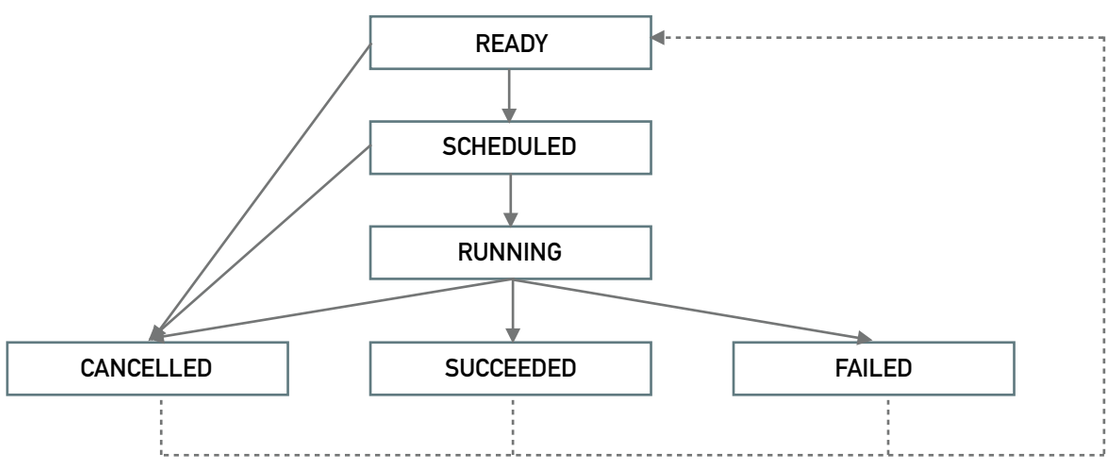
Service used many more times

V - the class type of returned value



Worker, Service and Task - States

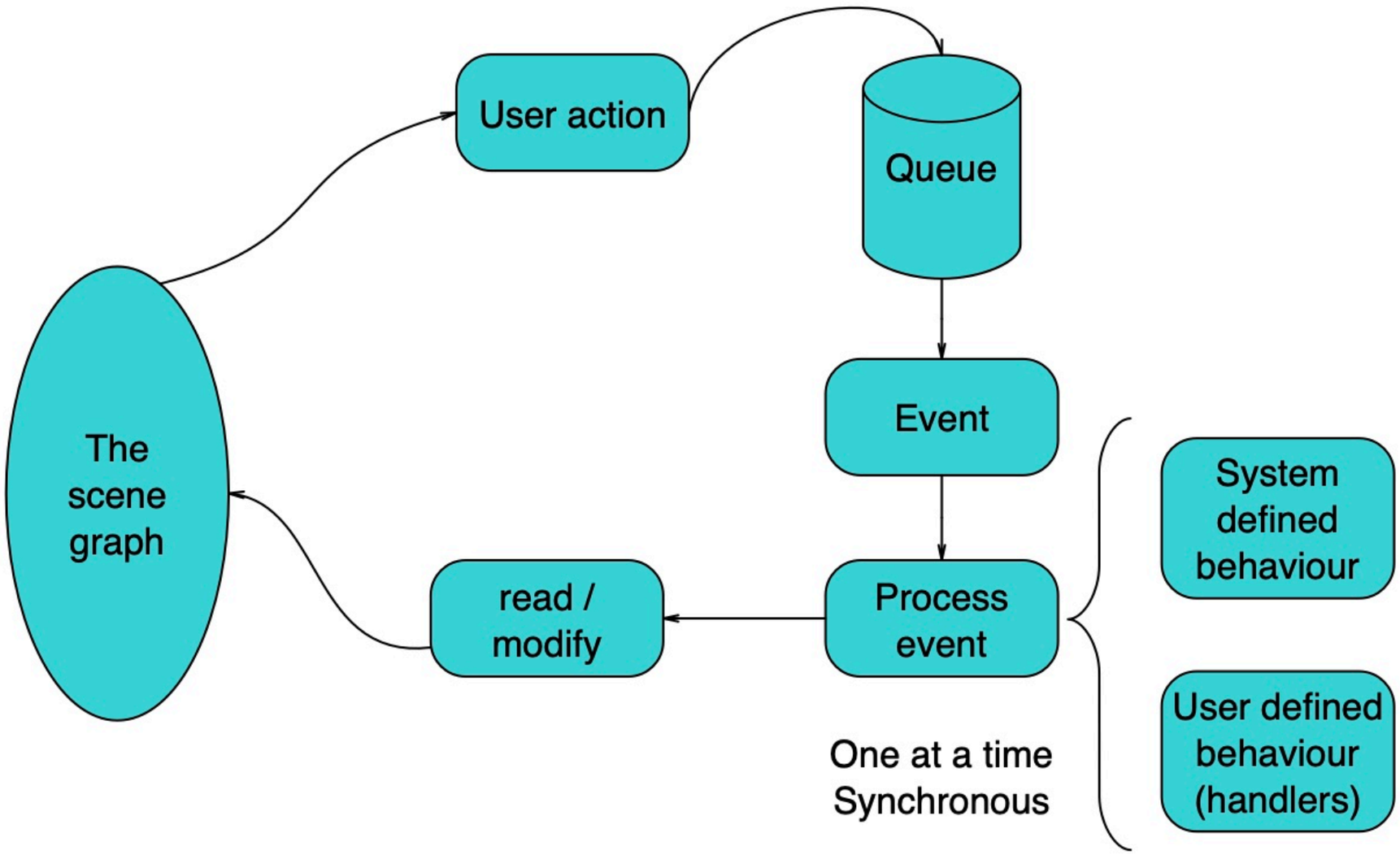
State	Description
READY	The only beginning valid state for a task.
SCHEDULED	After READY and before RUNNING (running == true).
RUNNING	During normal execution (running == true).
SUCCEEDED	The task completed normally. ReadOnlyObjectProperty<V> value is accessible.
FAILED	An exception occurred during execution. ReadOnlyObject-Property<Exception> exception contains the exception.
CANCELLED	The task was cancelled.



Only for Service

JAVAFX THREADS

JavaFX single thread behaviour

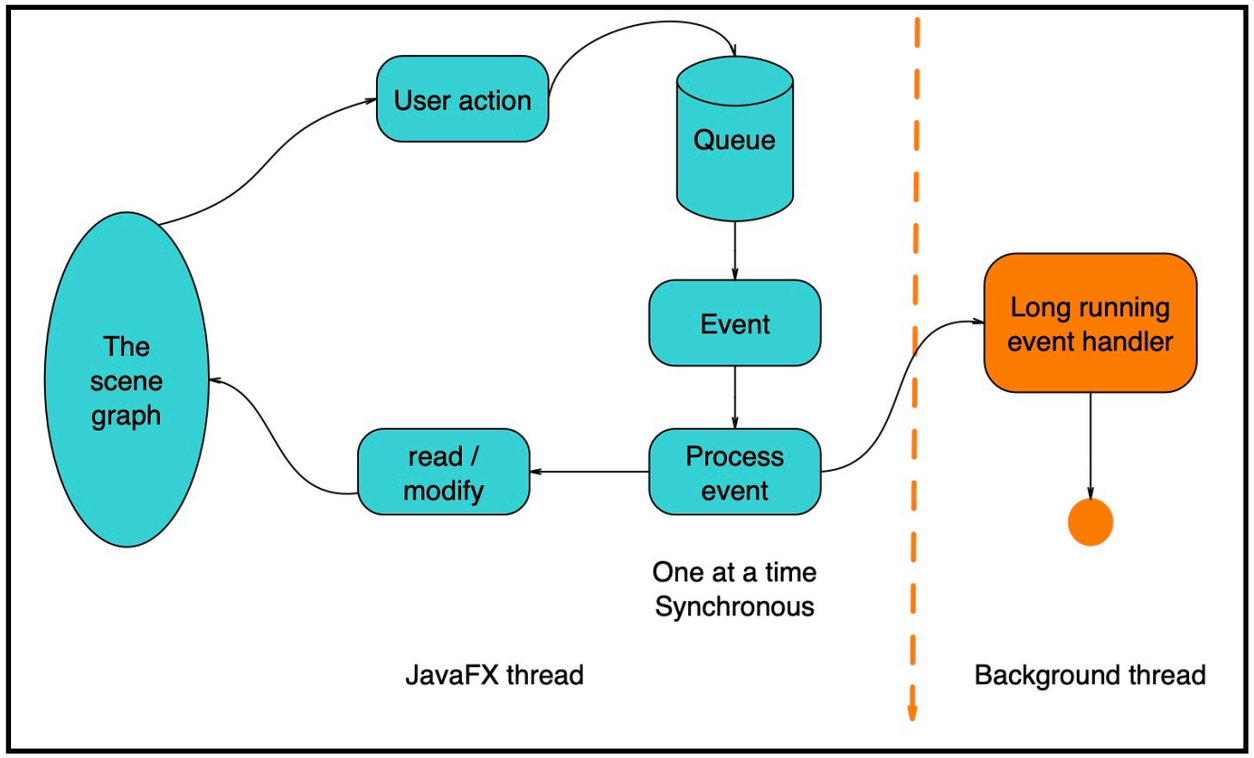


JAVAFX THREADS

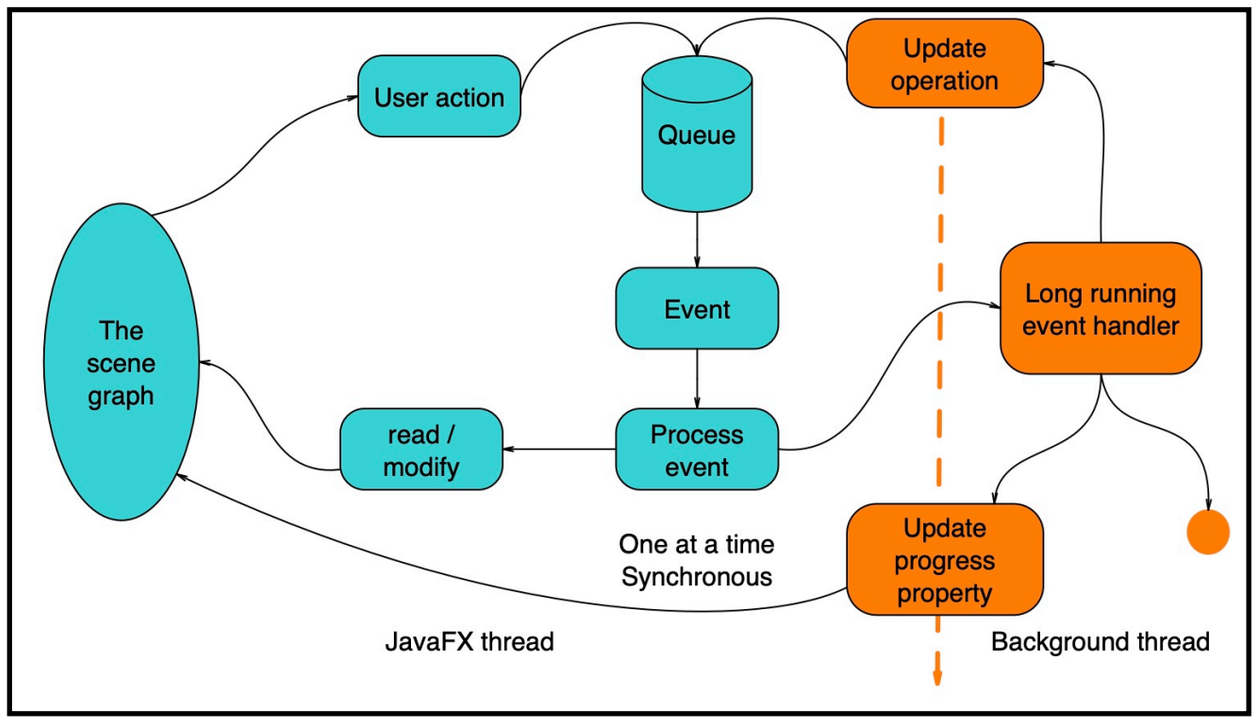
JavaFX multi thread

We have two options:

1. We can create a new Thread and process our event in background



2. We can have a Worker (Task or Service) which will process our task in a parallel thread but also it will provide updates on how the operations are going too.



REFERINȚE

- https://docs.oracle.com/javase/tutorial/uiswing/
- https://docs.oracle.com/javase/tutorial/uiswing/components/index.html
- https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html
- https://docs.oracle.com/javase/tutorial/uiswing/concurrency/index.html
- JavaFX, https://docs.oracle.com/javafx/2/index.html
- Scene builder new version, https://gluonhq.com/products/scene-builder/#download
- Event listener, https://docs.oracle.com/javase/tutorial/uiswing/events/handling.html
- https://www.studytonight.com/java/java-swing

JAVAFX

JavaFX ce am abordat

- Controls - SceneBuilder
  - Build interface easily with tools
- JavaFX Threads
  - How to keep our application responsive
- Application example