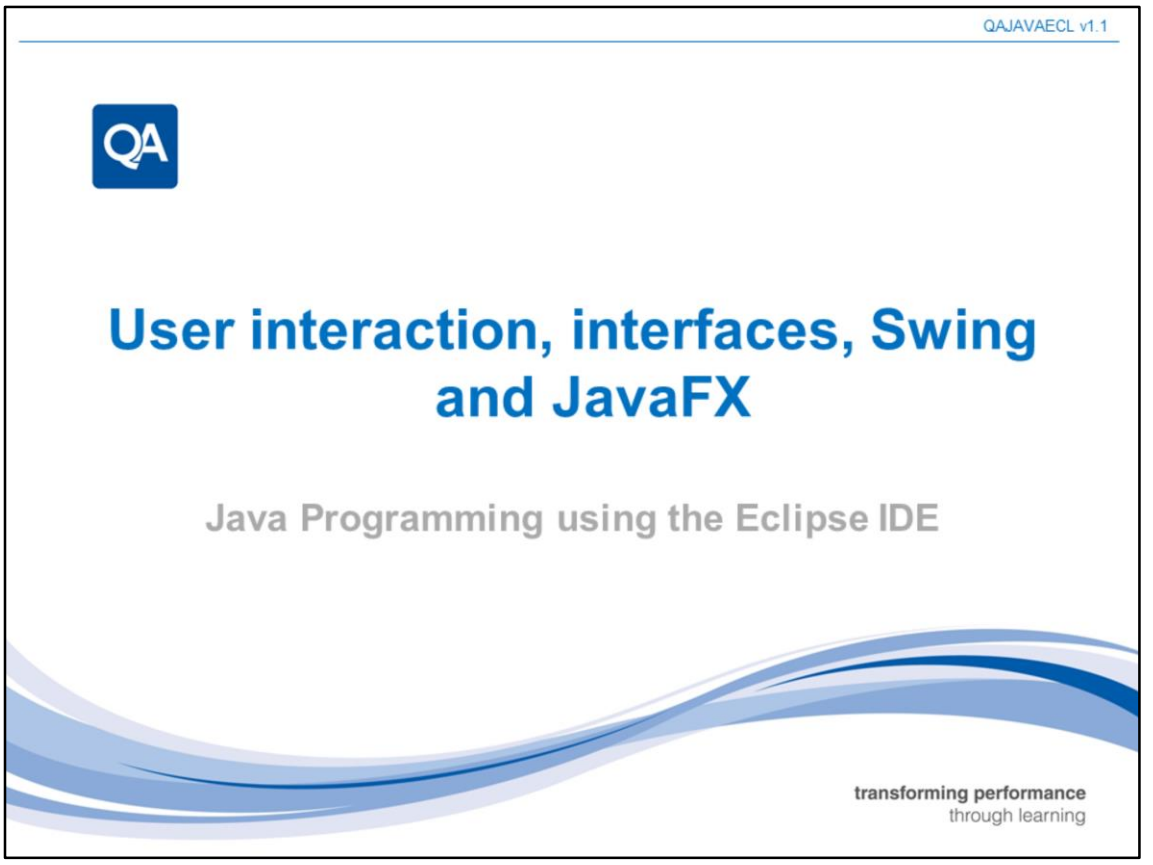
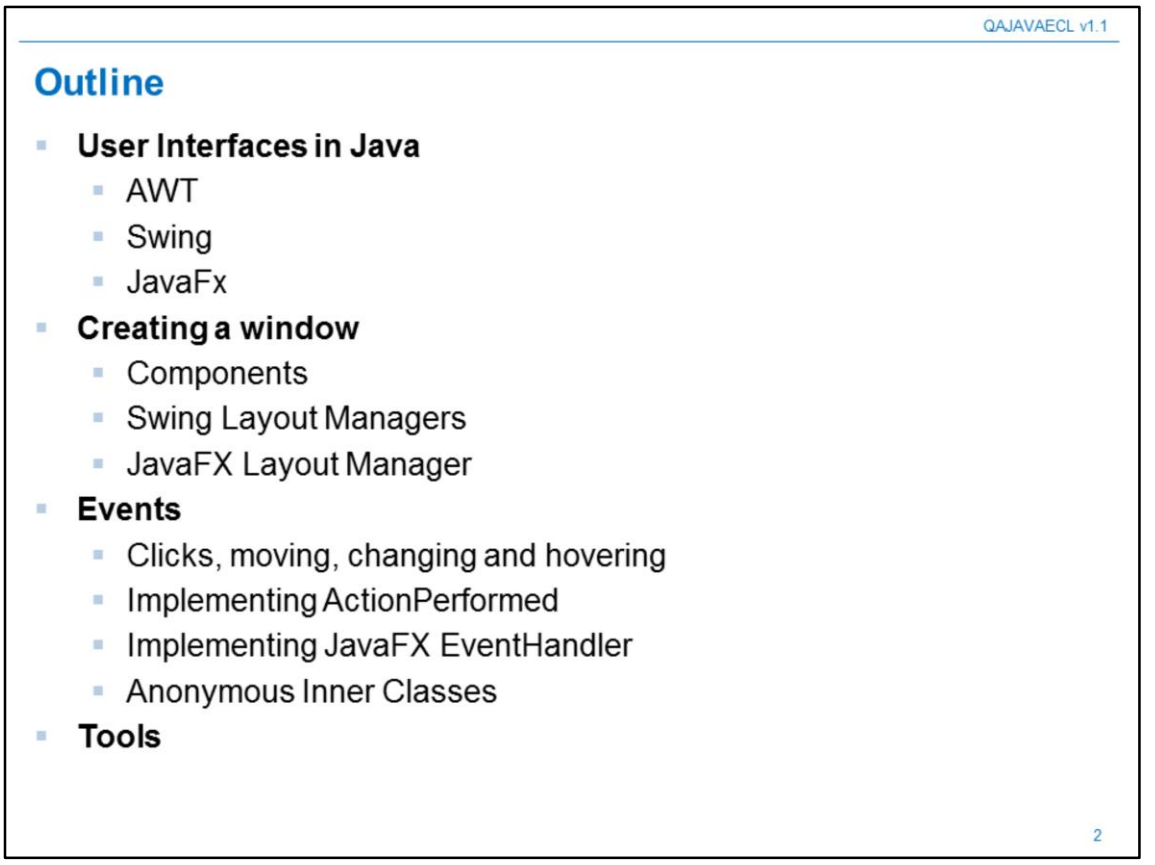

QAJAVAECL v1.1
QA
User interaction, interfaces, Swing and JavaFX
Java Programming using the Eclipse IDE
transforming performance
through learning



This covers just a small number of different options in the swing and JavaFX libraries, see

For swing: http://docs.oracle.com/javase/tutorial/uiswing/components/

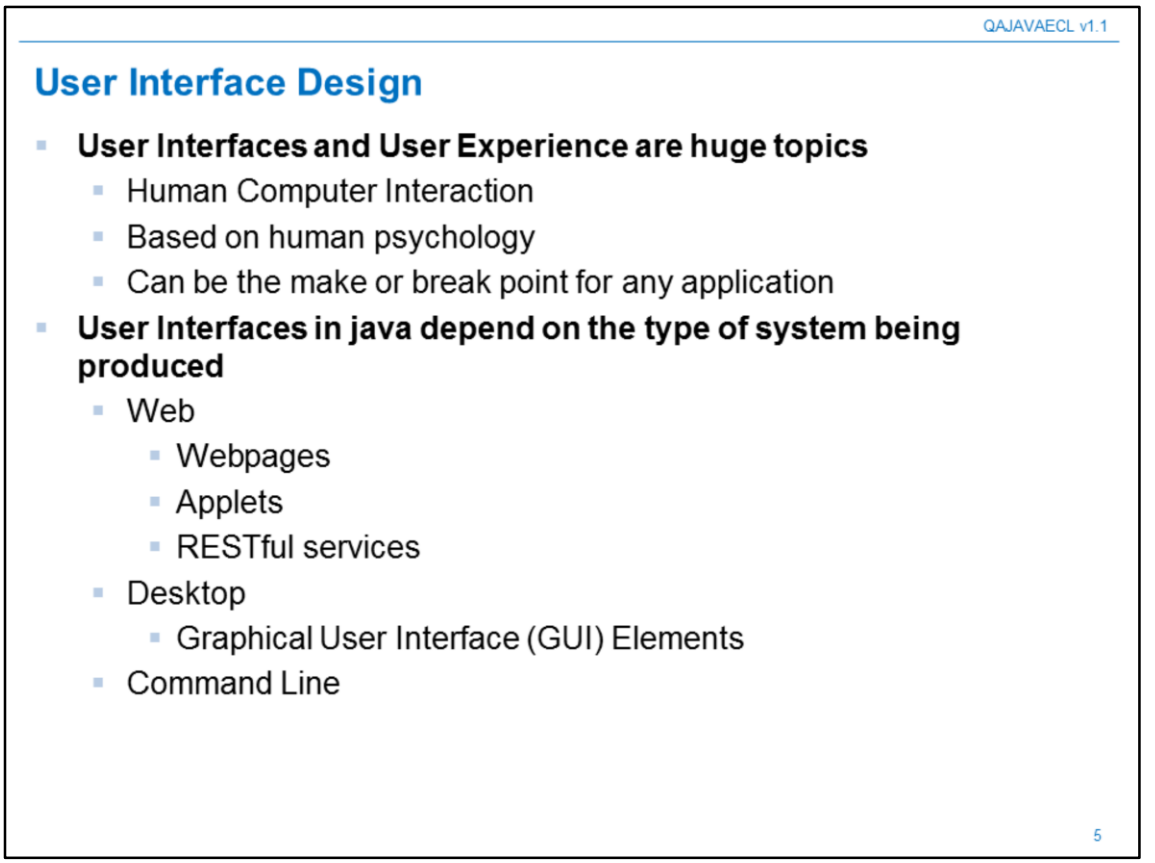
For JavaFX: https://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm

Objectives

- **By the end of this session we should be able to:**
  - Create basic Java swing interfaces
  - Create basic JavaFX interfaces
  - Create actions that happen after events
  - Install JavaFX in Eclipse IDE
  - Know where to look for more information and tools that can help

Outline

- **User Interfaces in Java**
  - AWT
  - Swing
  - JavaFX
- **Creating a window**
  - Components
  - Swing Layout Managers
  - JavaFX layout Managers
- **Events**
  - Clicks, moving, changing and hovering
  - Implementing ActionPerformed
  - Implementing JavaFX EventHandler
  - Anonymous Inner Classes
- **Tools**



We're not going to have time to look at the entire sphere of user interface design in this course, if you're interested in more about that side of things then there is a four day User Experience Course we offer!

http://www.qa.com/training-courses/technical-it-training/application-and-web-development/user-experience/user-experience-fundamentals/

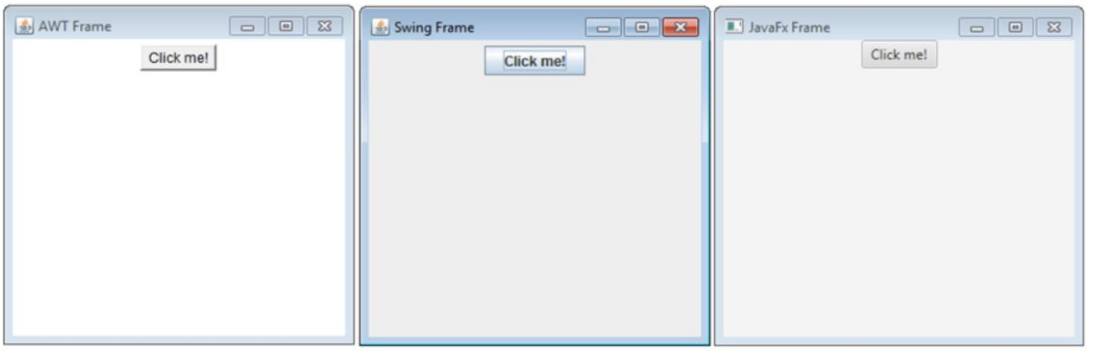
GUIs in Java

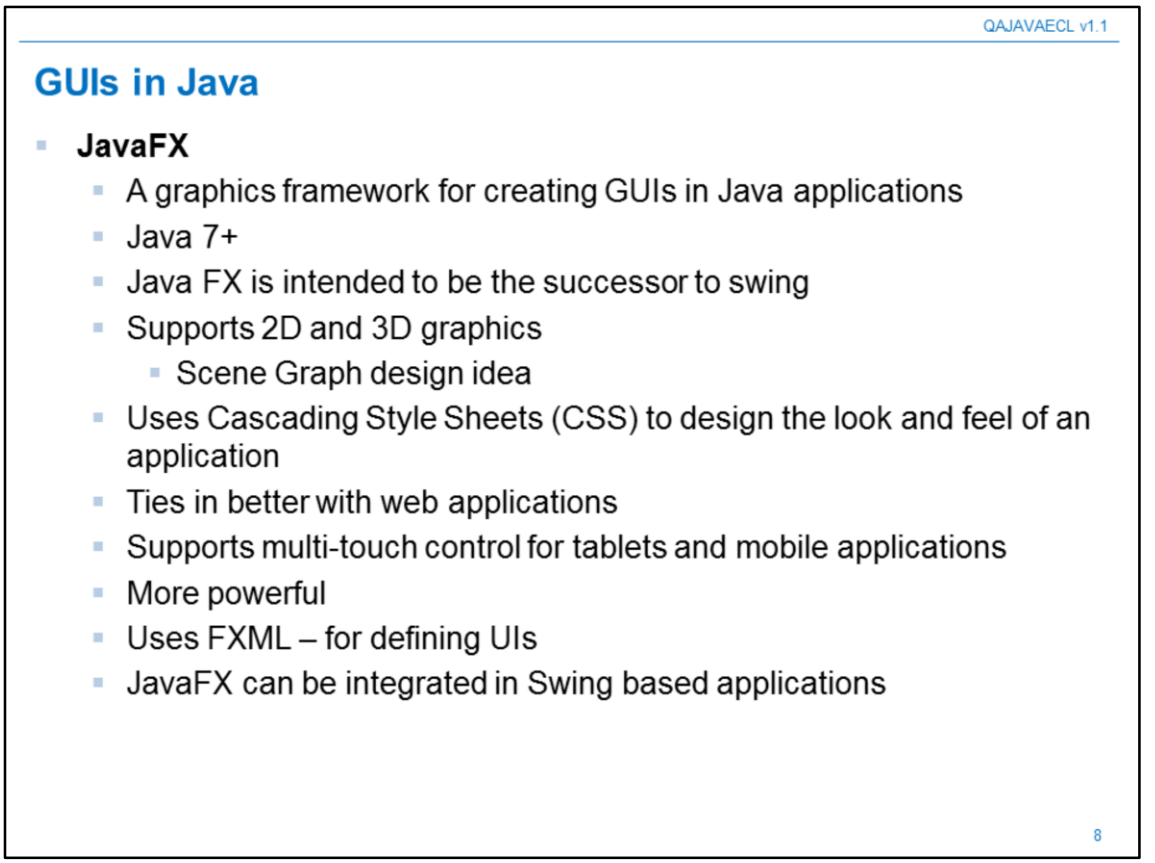
- **Java Abstract Windows Toolkit**
  - Original tool for creating GUI applications in java
  - Calls the underlying operating system to create elements on the screen
  - Caused the program to look like a native Windows or Apple program when run on those operating systems

- **Problems for the write once, run anywhere philosophy as each operating system treated their window elements differently**

GUIs in Java

- **Swing**
  - GUI Widget toolkit in Java
  - Uses its own implementation of elements on the screen
  - Programs will look and function the same on any operating system
  - Can change the look and feel of the UI to suit the application
  - More advanced components available
    - JTable, JTabbedPane etc.

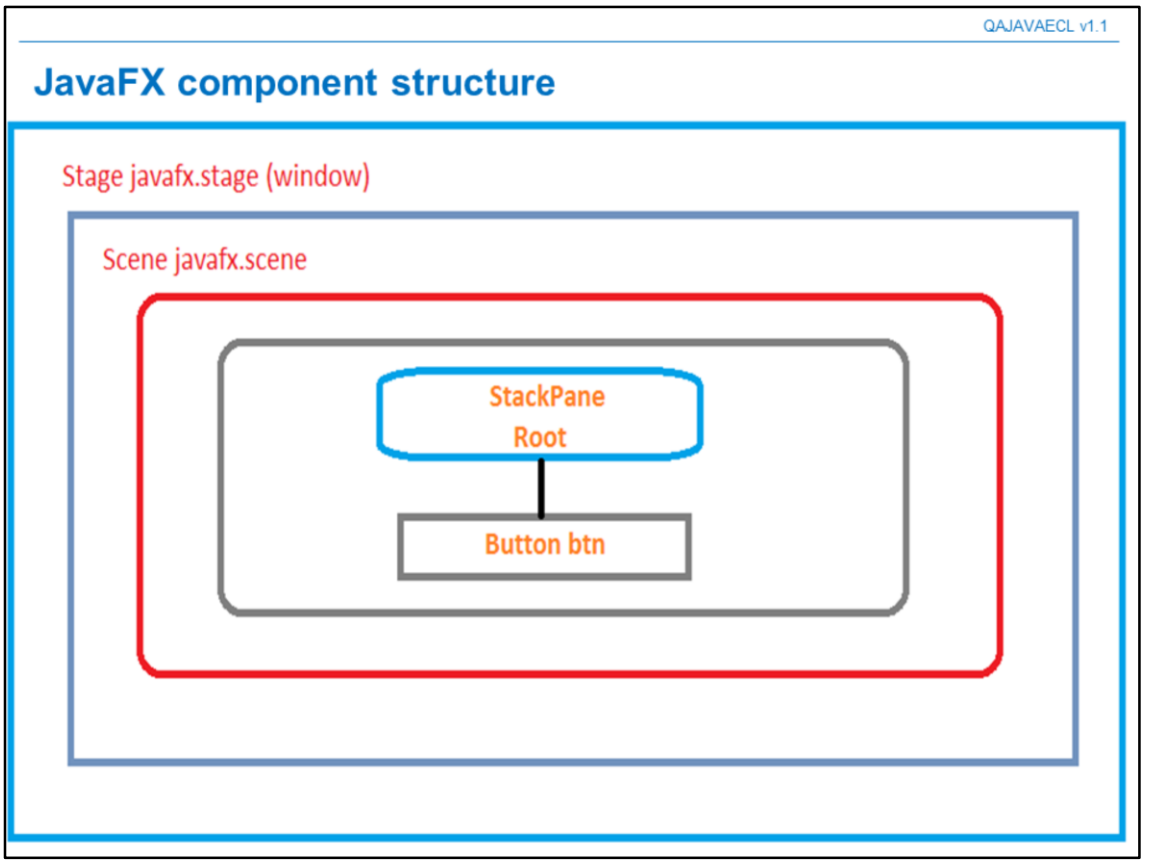




More information can be found about it here:

http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm

http://docs.oracle.com/javafx/2/css_tutorial/jfxpub-css_tutorial.htm



JavaFX terminology is a little bit different from swing, the main window of JavaFX on which every component resides is called “Stage (window)”. On top of the stage there is a framework called “Scene” on which we can add components (Buttons, TextFields etc)

Outline

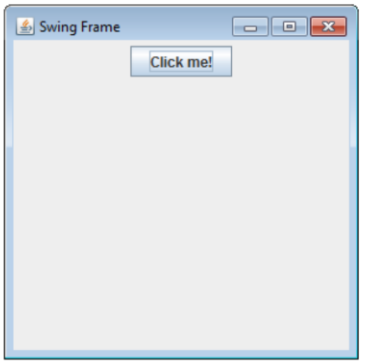
- **User Interfaces in Java**
  - AWT
  - Swing
  - JavaFX
- **Creating a window**
  - Components
  - Swing Layout Managers
  - JavaFX Layout Managers
- **Events**
  - Clicks, moving, changing and hovering
  - Implementing ActionPerformed
  - Implementing JavaFX EventHandler
  - Anonymous Inner Classes
- **Tools**

The JFrame

- **The base component of any GUI is the frame**

```
JFrame f  = new JFrame();
Container container = f.getContentPane();
f.setSize(300,300);
f.setTitle("Swing Frame");
f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
container.setLayout(new FlowLayout());
container.add(new JButton("Click me!"));
```

- **The Default Close Operation states what java should do when the X is clicked on. It will not close the program automatically!**
- **The Swing components are all imported from the javax.swing.* library**

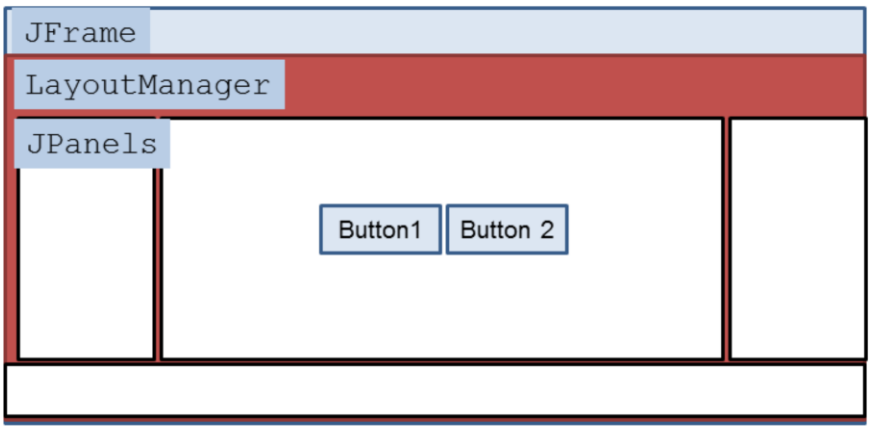


11

Remember, you can automatically import classes in eclipse using the Ctrl+O shortcut key. Most the swing components will automatically import using this. It's generally not good practice to just import everything using the * wildcard as this bloats the final packaged Jar file (and potentially can cause clashes with name spaces).

Adding Components

- **The JFrame provides the option to add components directly to the frame**
  - This isn't the recommended way of doing things
  - Can lead to unpredictable layouts!
  - It is better to use LayoutManagers and JPanels



This type of layering can create very complex looking layouts, but allow for each component to be reused. Each panel can also have a layout manager and more panels added to them.

JPanels

- **JPanels allow grouping of components together**
  - We want to add components to the JPanel rather than the frame directly
  - Allows for code reuse
  - Extending the JPanel is one way of handling what happens when elements on the screen are interacted with

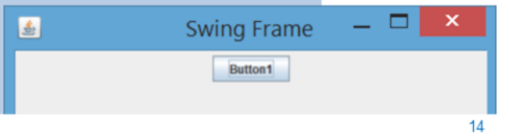
```
JFrame f2 = new JFrame("Swing Frame");
//additional options

JPanel panel = new JPanel();

panel.add(button1);

f2.add(panel)
```

  - An empty panel doesn't look like anything interesting, but when we start adding layouts and components later it will make a difference

JButtons

- **JButtons draw clickable buttons on the screen**
  - There are numerous constructors that can be used
  - The text on the button is either set in the constructor or by using the setText() method
  - Buttons can have icons, text, backgrounds, actions associated with them and many more options
    - The Java API goes over all the different methods that can be used!

```
JFrame f2 = new JFrame("Swing Frame");
//additional options

JPanel panel = new JPanel();

JButton button1 = new JButton("Button1");
panel.add(button1);

f2.add(panel)
```



14

Layouts

- **Layouts tell Java how to organise and display components added to a JFrame or JPanel**
  - BorderLayout
  - BoxLayout
  - CardLayout
  - FlowLayout
  - GridBagLayout
  - GridLayout
  - GroupLayout
  - SpringLayout
- **In general people find that they will end up using one or two of these and adapting them to each situation**
- **You can add the layout when you create the component, or add it afterwards**

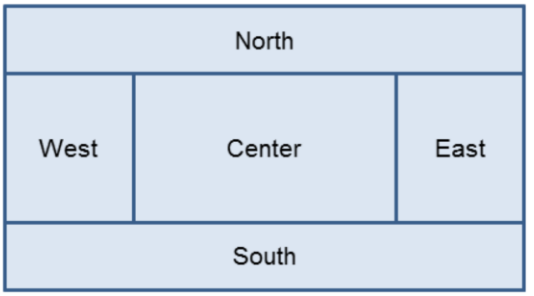
```
f.setLayout(new BorderLayout());
f2.setLayout(new GridLayout(10,1));

JPanel panel = new JPanel(new BorderLayout());
panel.setLayout(new FlowLayout());
```

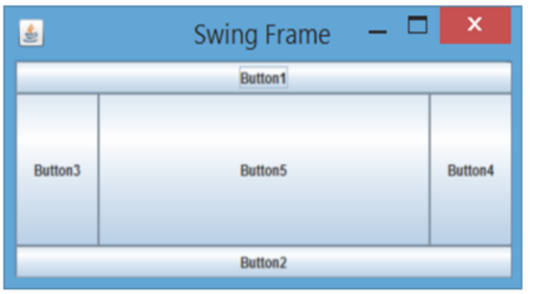

There are many different layout managers available, we're going to focus on three specific ones, feel free to explore the others yourself!

http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

Border Layout

- **This splits a panel into five parts**






```
JPanel panel = new JPanel(new BorderLayout());
panel.add(button1, "North"); //not recommended
panel.add(button2, BorderLayout.SOUTH);
panel.add(button3, BorderLayout.WEST);
panel.add(button4, BorderLayout.EAST);
panel.add(button5, BorderLayout.CENTER);
```

16

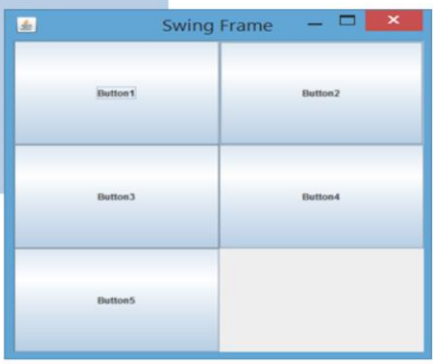
Border layouts can be very useful if you want to have a panel to the left hand side which has buttons in and a text box in the middle, or if you want a status bar at the bottom. The layout adapts as you change the frame size both when the frame is setup and when you are running the program so the components always stay in the same place in relation to the frame.

While you can just give the string location of where you want to place the component this is not recommended, the BorderLayout class provides some static variables, NORTH, SOUTH, EAST, WEST and CENTER which will ensure the layout is setup correctly.

Grid Layout

- **Grid layout is more flexible than Border Layout and allows components to be added in a grid of a specified size**
  - Constructor initialises size of the grid
  - new GridLayout(vertical, horizontal)
  - Components are not given a location when added, but fill up the grid automatically going horizontally, first then vertically

```
panel.setLayout(new GridLayout(3,2));
panel.add(button1);
panel.add(button2);
panel.add(button3);
panel.add(button4);
panel.add(button5);
```

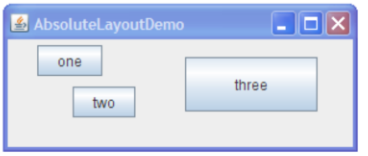


17

Using no layouts – Absolute Positioning

- **It is possible to avoid using layouts at all within your swing applications, but this isn't recommended**
  - Default layout managers are included when you create frames and panels so you need to use setLayout(null) which disables the default layout manager
  - This can cause problems when the window is resized
  - Give components a position based on the inset of the window

```
JButton b1 = new JButton("one");

Insets insets = pane.getInsets();
Dimension size = b1.getPreferredSize();
b1.setBounds(25 + insets.left, 5 + insets.top,
             size.width, size.height)
```



Code and image adapted from:
http://docs.oracle.com/javase/tutorial/uiswing/layout/none.html

JLabels and JTextFields

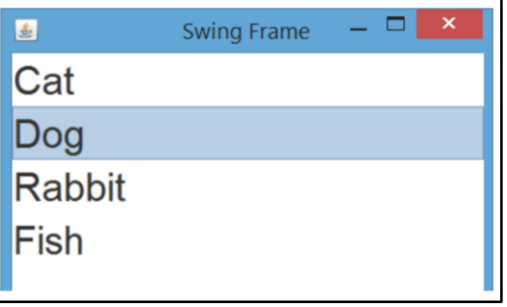
- **A JLabel contains text which is written to the screen**
  - Different methods available for changing how they look and behave
  - Usually used with JTextFields to describe what the field is for
- **A JTextField allows the user to enter text**
  - They can display text by default
  - Enable/Disable editing the text
  - Get the text for using in the program later

```
JLabel nameLabel = new JLabel("Name");
centerPanel.add(nameLabel);
JTextField nameBox = new JTextField();
centerPanel.add(nameBox);
```

JList

- **A list component that allows for values to be set and highlighted according to each row**
  - Initialise with an array of values
  - getSelectedIndex() and getSelectedValue() return the currently selected item in the list
  - Uses generics to say what type of data is being stored in the list

```
String[] animals = {"Cat", "Dog", "Rabbit", "Fish"};
JList<String> list = new JList<String>(animals);
list.setFont(new Font("", 0, 40));
panel.add(list, BorderLayout.CENTER);
```

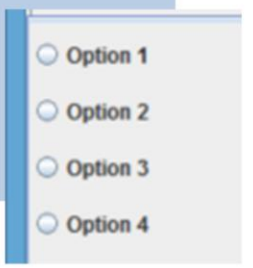


You can change the font style and size of most components by using the setFont() method. Here I have set the font to size 40 for ease of reading.

Radio Buttons and Grouping Items

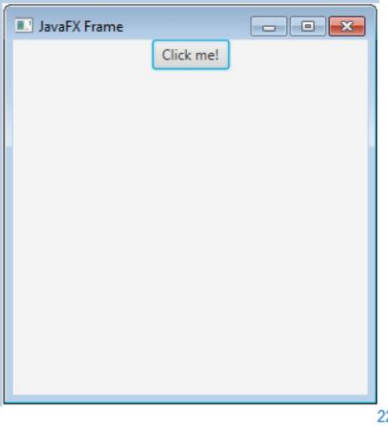
- **Radio buttons and checkboxes need to be added to a ButtonGroup to be associated with one another**
  - If you don't do this then each radio button is considered as a stand alone button, allowing you to select all the items rather than just one

```
JRadioButton b1 = new JRadioButton("Option 1");
JRadioButton b2 = new JRadioButton("Option 2");

ButtonGroup group = new ButtonGroup();
group.add(b1);
group.add(b2);

radioPanel.add(b1);
radioPanel.add(b2);
```



The JavaFX stage

```
public class JavaFXFrame extends Application{

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

@Override
public void start(Stage primaryStage) throws Exception {
Button btn = new Button("Click me!");
StackPane root = new StackPane();
root.setAlignment(Pos.TOP_CENTER);
root.getChildren().add(btn);
Scene scene = new Scene(root,300,300);
primaryStage.setTitle("JavaFX Frame");
primaryStage.setScene(scene);
primaryStage.show();
}

}
```



22

To find out more: http://docs.oracle.com/javafx/2/ui_controls/overview.htm

JavaFX Labels

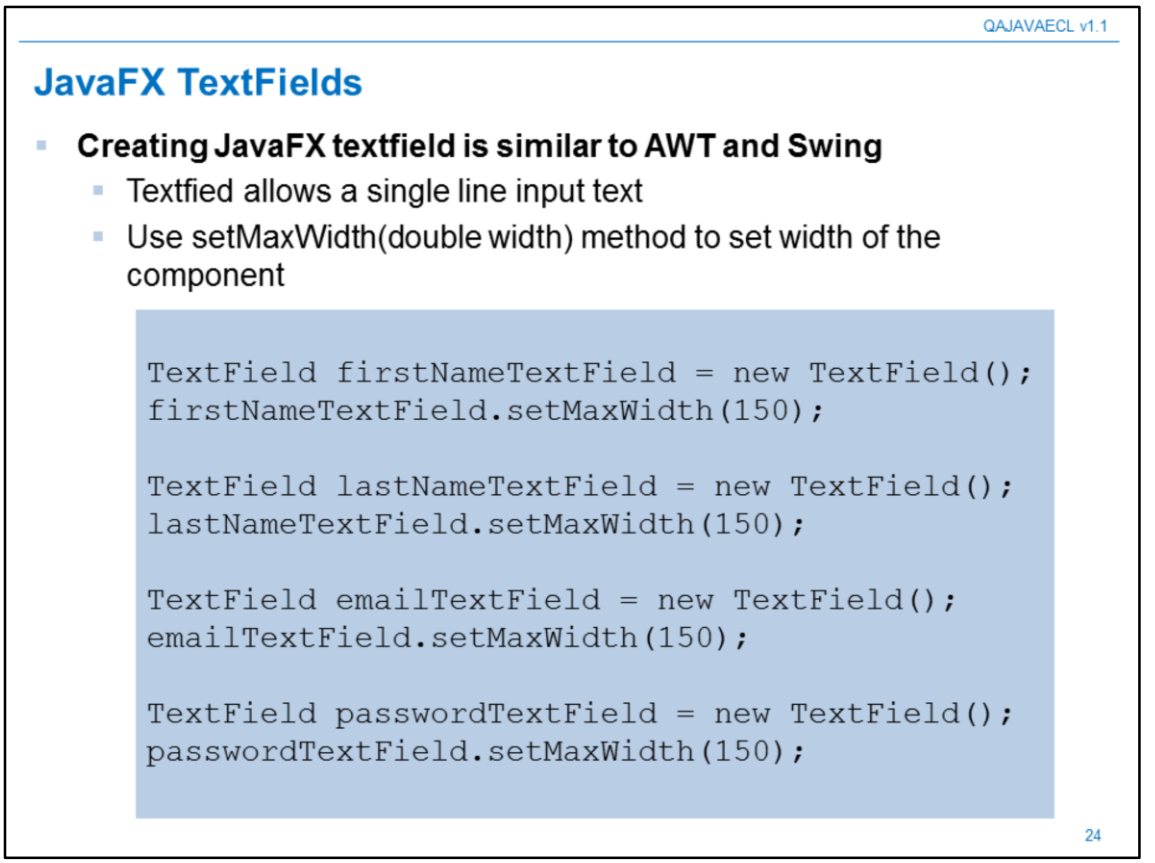
- **JavaFX labels are similar to AWT and Swing. Labels are represented by means of classes**
  - Creating labels means is the process of creating objects from Label class
  - Labels can be customised

```
Label firstNameLabel = new Label("First Name");
firstNameLabel.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));

Label lastNameLabel = new Label("Last Name");
lastNameLabel.setFont(Font.font("Times New Roman",FontWeight.BOLD,30));

Label emailLabel = new Label("Email ID");
emailLabel.setFont(Font.font("Verdana", FontWeight.EXTRA_BOLD, 20));

Label passwordLabel = new Label("Email ID");
emailLabel.setFont(Font.font("Verdana", FontWeight.EXTRA_BOLD, 20));
```

  - The above code creates four Label objects and they are customised with different Font formats

23

You can change the font style and size of most components by using the setFont() method. Here lastNameLabel has been customised with font size 30 and Times New Roman font type just to differentiate.



JavaFX TextField class has two constructors

TextField()

Creates an empty textfield

TextField(String text)

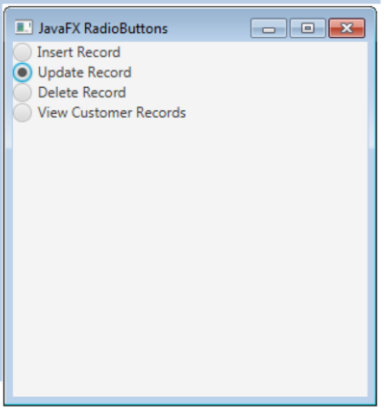
Creates a textfield with initial text content

Please use the following link to find out more details of the method summary

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TextField.html

JavaFX RadioButton and ToggleGroup

```
RadioButton insert = new RadioButton("Insert Record");
RadioButton update = new RadioButton("Update Record");
RadioButton delete = new RadioButton("Delete Record");
RadioButton viewRecords = new RadioButton("View Customer
Records");

ToggleGroup transaction = new ToggleGroup(); // To create a
button group

insert.setToggleGroup(transaction);
update.setToggleGroup(transaction);
delete.setToggleGroup(transaction);
viewRecords.setToggleGroup(transaction);

// sets the selection
update.setSelected(true);
```



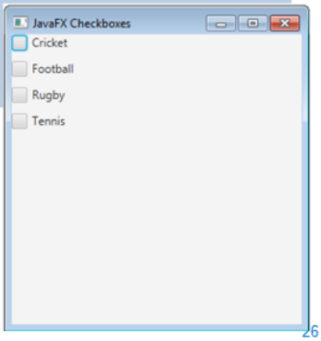
25

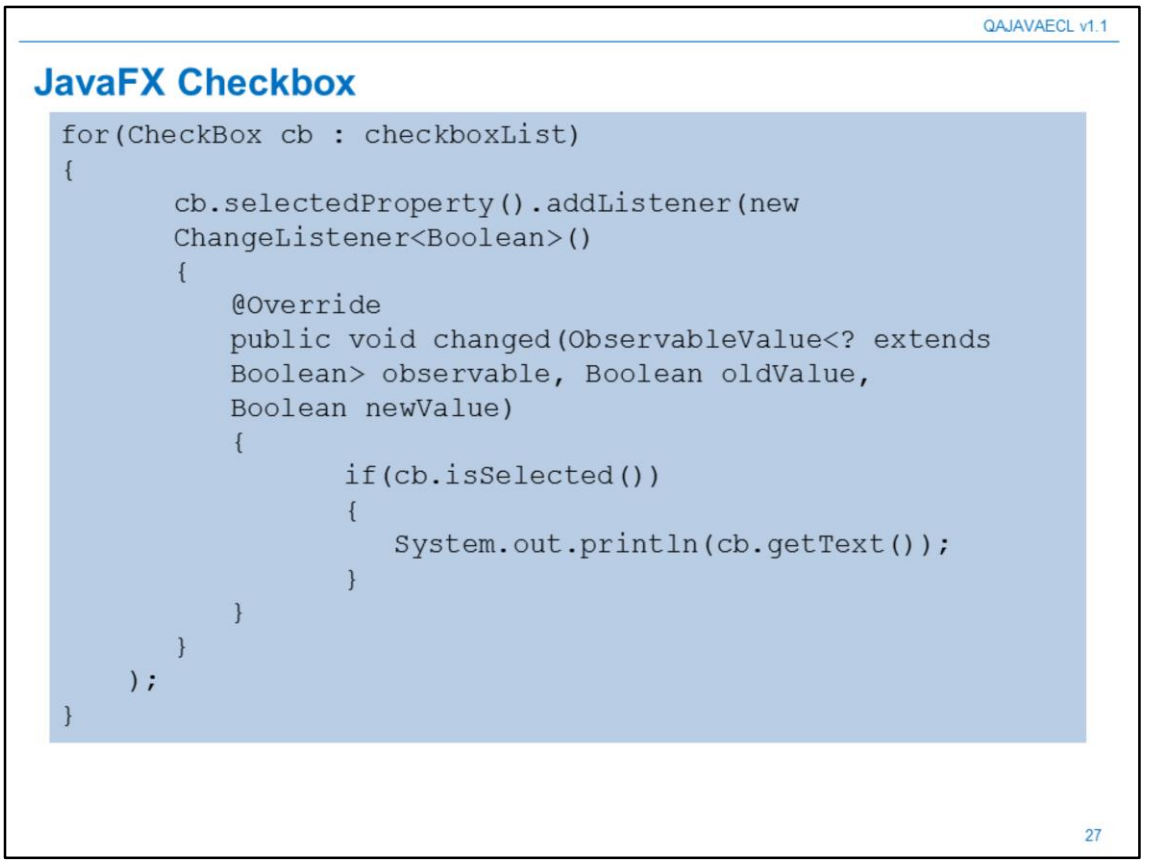
To create a button group we use ToggleGroup class, here the radio buttons are grouped into one category named as “transaction”. So that user can select only one radio button. In the button group, the first button has the focus by default but I have used setSelected(boolean value) method on update

JavaFX Checkbox

- **Checkbox allows the users to select multiple options**

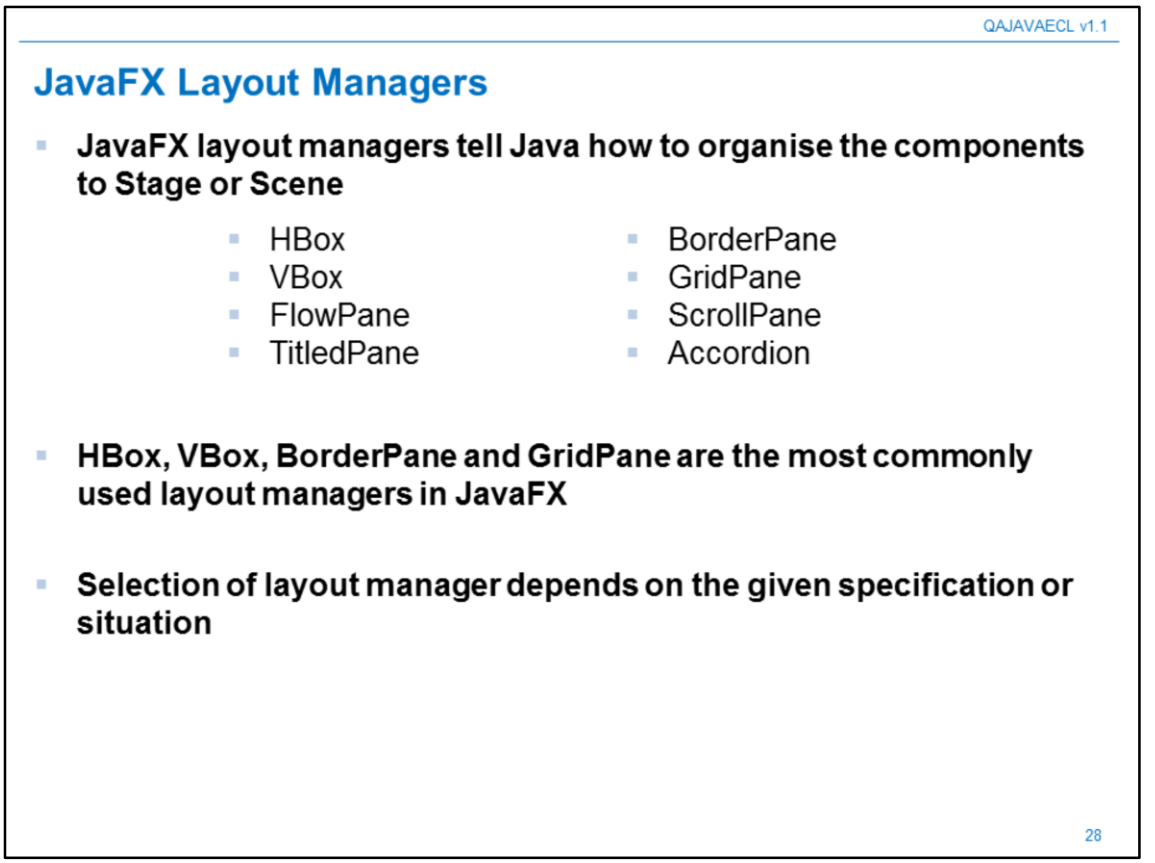
```
CheckBox cb1 = new CheckBox("Cricket");

CheckBox cb2 = new CheckBox("Football");

CheckBox cb3 = new CheckBox("Rugby");

CheckBox cb4 = new CheckBox("Tennis");

CheckBox checkboxList [] = {cb1,cb2,cb3,cb4};
```





This slide shows a loop which iterates through an array of CheckBox objects.

ChangeListener is one of the event handlers which is associated with CheckBox component.



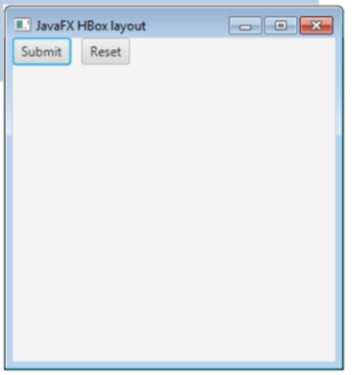
There are many different JavaFX layout managers available, we're going to focus on some specific ones, feel free to explore the others yourself in the exercise!

http://docs.oracle.com/javase/8/javafx/layout-tutorial/index.html

HBox layout

- **This layout is used to organize the JavaFX nodes or components in a horizontal row**

```
Button btn1 = new Button("Submit");

Button btn2 = new Button("Reset");

HBox h = new HBox(10); // Spacing = 10

h.getChildren().addAll(btn1,bnt2);
```

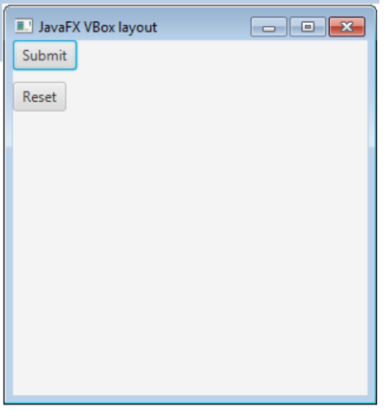


29

VBox layout

- **VBox places JavaFX nodes stacked in a vertical column**

```
Button btn1 = new Button("Submit");

Button btn2 = new Button("Reset");

VBox v = new VBox(10); // Spacing = 10

v.getChildren().addAll(btn1,bnt2);
```

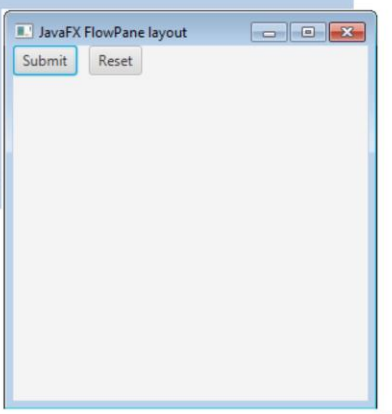


30

FlowPane

- **This layout organizes JavaFX nodes in a row just like the flow of text in a text editor. It wraps JavaFX nodes to the next line when horizontal space is less than the total of all the nodes' width**

```
Button btn1 = new Button("Submit");

Button btn2 = new Button("Reset");

FlowPane flow = new FlowPane();

flow.setHgap(10);

flow.getChildren().addAll(btn1,bnt2);
```

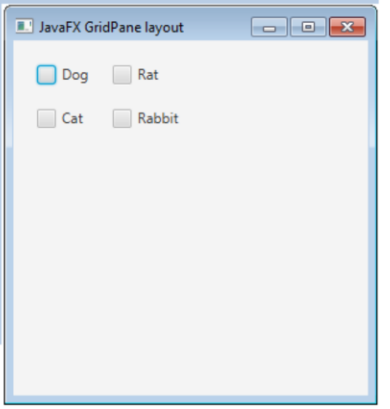


31

GridPane

- **This layout represents a table structure. It can specify horizontal and vertical spaces**

```
CheckBox cb1 = new CheckBox("Dog");
CheckBox cb2 = new CheckBox("Cat");
CheckBox cb3 = new CheckBox("Rat");
CheckBox cb4 = new CheckBox("Rabbit");

GridPane grid = new GridPane();
grid.setPadding(new Insets(20));
grid.setHgap(20);
grid.setVgap(20);
grid.add(cb1, 0, 0);
grid.add(cb2, 0, 1);
grid.add(cb3, 1, 0);
grid.add(cb4, 1, 1);
```



BorderPane

- It layouts JavaFX nodes in a top, bottom, left, right or center region
- Each region can only have one node


```
Button top = new Button("Top");
Button right = new Button("Right");
Button bottom = new Button("Bottom");
Button left = new Button("Left");
Button center = new Button("Center");

BorderPane border = new BorderPane();
border.setPadding(new Insets(10, 20, 10, 20));

border.setTop(top);
border.setRight(right);
border.setBottom(bottom);
border.setLeft(left);
border.setCenter(center);
```

33

The default alignment of JavaFX nodes in BorderPane is based on the following setup

Top: Pos.TOP_LEFT

Bottom: Pos.BOTTOM_LEFT

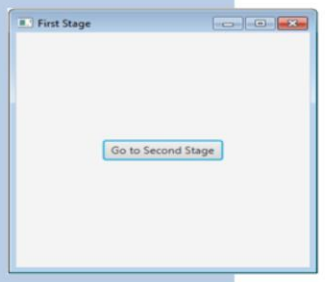
Left: Pos.TOP_LEFT

Right: Pos.TOP_RIGHT

Center: Pos.CENTER

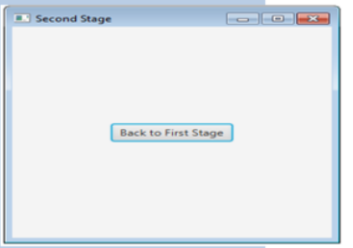
Navigation between JavaFX stages

```
public class FirstJavaFXStage extends Application {
private Button secondStage;
@Override
public void start(Stage primaryStage) {
    secondStage = new Button("Go to Second Stage");
    StackPane root = new StackPane();
    root.getChildren().add(secondStage);
    Scene scene = new Scene(root,300,300);
    primaryStage.setScene(scene);
    primaryStage.setTitle("First Stage");
        firstStage.setOnAction(new
EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
    primaryStage.close();//Close the current stage
     new SecondJavaFXStage().start(new Stage());
    }
    });
      primaryStage.show();
}
```



Navigation between JavaFX stages

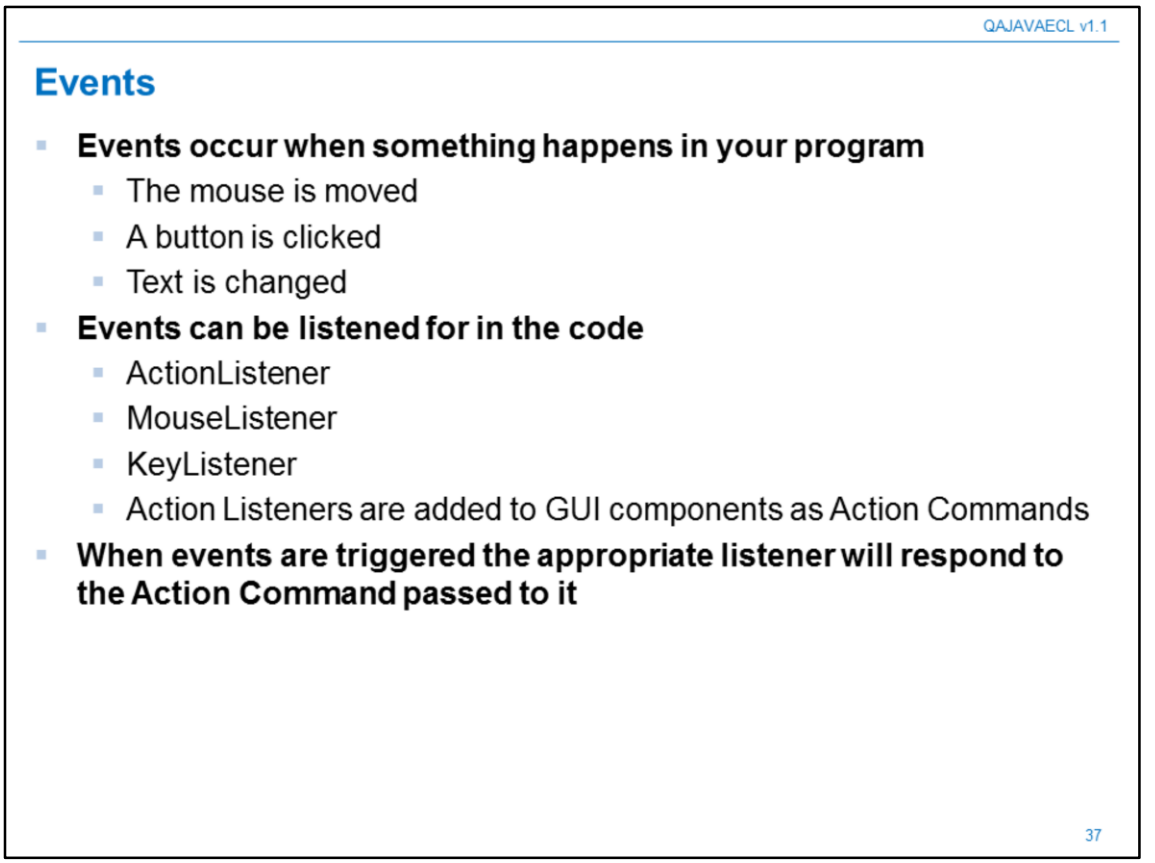
```
public class SecondJavaFXStage extends Application {
private Button firstStage;
@Override
public void start(Stage primaryStage) {
    Button firstStage = new Button("Back to First
    Stage");
    StackPane root = new StackPane();
    root.getChildren().add(firstStage);
    Scene scene = new Scene(root,300,300);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Second Stage");
        firstStage.setOnAction(new
EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
    primaryStage.close();//Close the current stage
     new FirstJavaFXStage().start(new Stage());
    }
    });
      primaryStage.show();
}
```



35

Outline

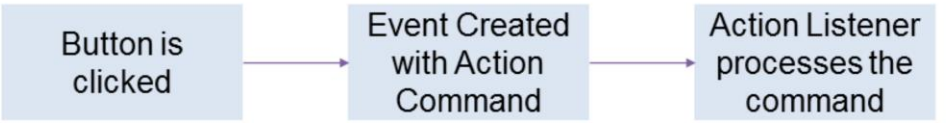
- **User Interfaces in Java**
  - AWT
  - Swing
  - JavaFx
- **Creating a window**
  - Components
  - Swing Layout Managers
  - JavaFX Layout Managers
- **Events**
  - Clicks, moving, changing and hovering
  - Implementing actionPerformed
  - Implementing JavaFX EventHandler
  - Anonymous Inner Classes
- **Tools**



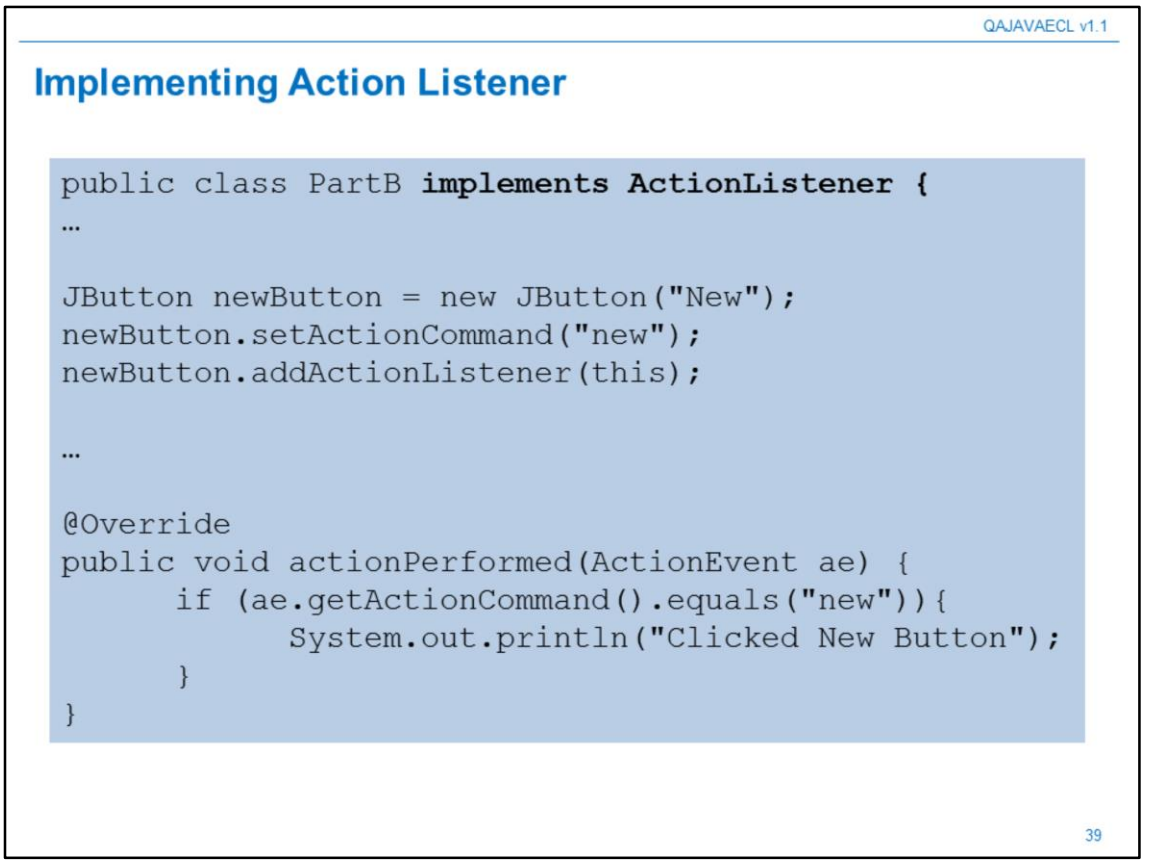
Events are not only for GUI or swing based programs, but this is a good way to explore them!

The ActionListener

- **When the button is created it is given two pieces of information**
  - What the action command is when the button is clicked
  - Where it should send the event to



- **There are two ways of implementing this in the code**
  - The class can implement action listener and have an actionPerformed method
  - Use an anonymous inner class to process the command



If you don't set the action command then the text of the button is used instead.

More than one component can have the same Action Command, for example, the "New" option in the menu and the "New" JButton can both have the same action command and so the program reacts in the same way no matter which component is used to generate the event.

Anonymous Inner Classes

- **Keep all the code for the action with the button in one place**
- **Don't need an action command as there is only one event this class is tied to**
- **Cannot share action event code between button and menu options**
- **Can lead to confusing code!**

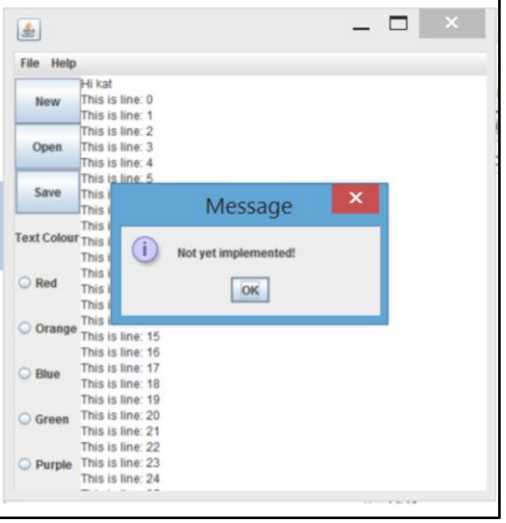
```
JButton openButton = new JButton("Open");
openButton.addActionListener(
      new ActionListener(){
            @Override
            public void actionPerformed(ActionEvent ae) {
                  System.out.println
                              ("Clicked Open Button");
            }
      }
);
```

Message Dialog

- **Message Dialogs are useful for informing the user about problems**
  - They are provided by the JOptionPane class as a static method showMessageDialog
  - The JFrame that the dialog belongs to is required
  - The message can be anything
  - The icon is based on the type of message dialog

```
JOptionPane.showMessageDialog
(frame, "Not yet implemented!");
```

- **Option dialogs can be used for queries with more than one answer**



JavaFX Events

- **JavaFX has its own libraries to support event handling**
- **JavaFX events are defined in javafx.event package**
- **JavaFX provides several events which are represented by classes, including**
  - ActionEvent
  - WindowEvent
  - KeyEvent
  - InputEvent

JavaFX Event Handler

- **JavaFX events are handled by an event handler**
- **An event handler is an implementation class of a specific event type. JavaFx provides an interface named EventHandler which can handle all sorts of JavaFX events**
- **setOn Event – event hatndler must register with events in order to receive event notifications**
  - We can use the setOnXXX methods to register event handlers

```
Button okButton = new Button("OK");
okButton.setOnAction(
      new EventHandler<ActionEvent>(){
            @Override
            public void handle(ActionEvent ae) {
                  System.out.println
                              ("Clicked OK Button");
            }
      }
);
```

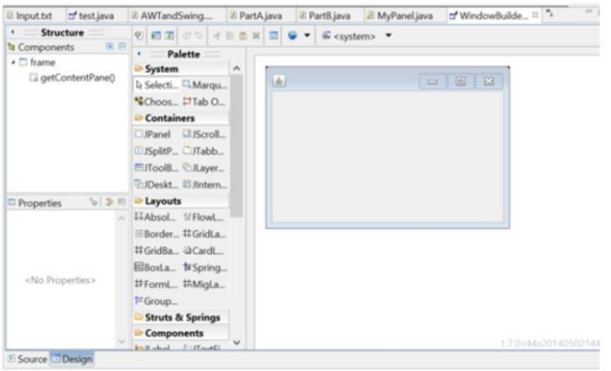
43

Outline

- **User Interfaces in Java**
  - AWT
  - Swing
  - JavaFX
- **Creating a window**
  - Components
  - Swing Layout Managers
  - JavaFX Layout Managers
- **Events**
  - Clicks, moving, changing and hovering
  - Implementing ActionPerformed
  - Implementing JavaFX EventHandler
  - Anonymous Inner Classes
- **Tools**

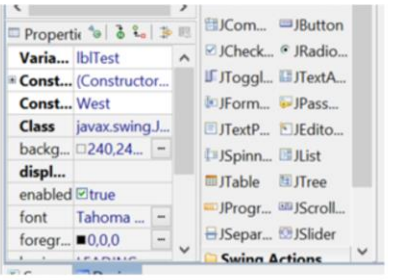
Design tools in eclipse

- **Designing everything by hand can be rather off-putting**
  - Complicated layering of components
- **The Window Builder is a WYSIWYG (what you see is what you get) design tool for swing GUIs in eclipse**
  - New → Other → Application Window
  - Allows you to manipulate the Source code directly
  - Also can see and manipulate the design without needing to run the program every time



Window Builder

- **The design view allows you to "drag and drop" components onto the JFrame**
  - Similar to visual studio, or other IDEs for different languages
  - Drag and drop any component you want to the location you want
  - Properties for each component can be edited using the properties window to the left of the screen
  - Double click on components to automatically generate the action listener code
    - Uses anonymous inner classes
  - Note: You should still use JPanels and layout managers when using this tool




```
JButton btnButton = new JButton("Button 1");
btnButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
    }
});
frame.getContentPane().add(btnButton, BorderLayout.CENTER);
```

46

Summary

- **User Interfaces in Java**
  - AWT
  - Swing
- **Creating a window**
  - Components
  - Swing Layout Managers
  - JavaFX Layout Managers
- **Events**
  - Clicks, moving, changing and hovering
  - Implementing ActionPerformed
  - Implementing JavaFX EventHandler
  - Anonymous Inner Classes
- **Tools**

Exercise

- Install JavaFX plugin
- Design and create Login and Registration GUIs
- Add events to respond to when you click on buttons
- Implement navigations between JavaFX stages
- Recall File Handling