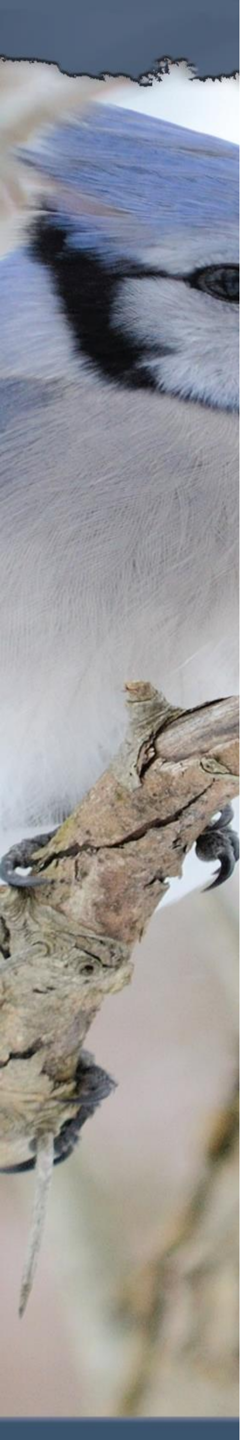
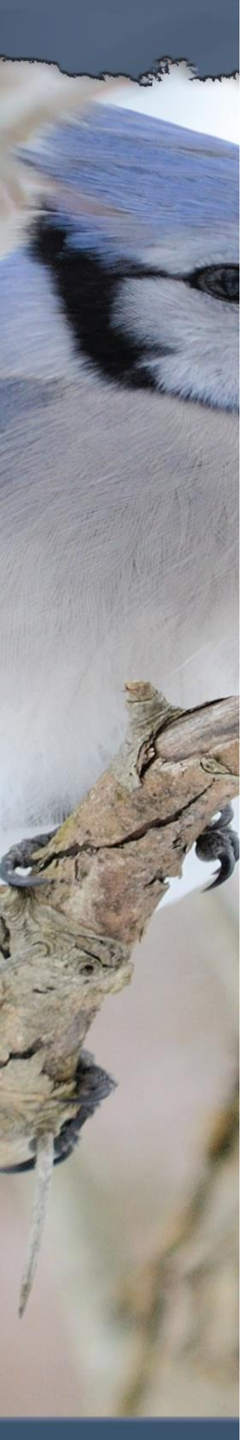


Building Graphical User Interfaces (2)

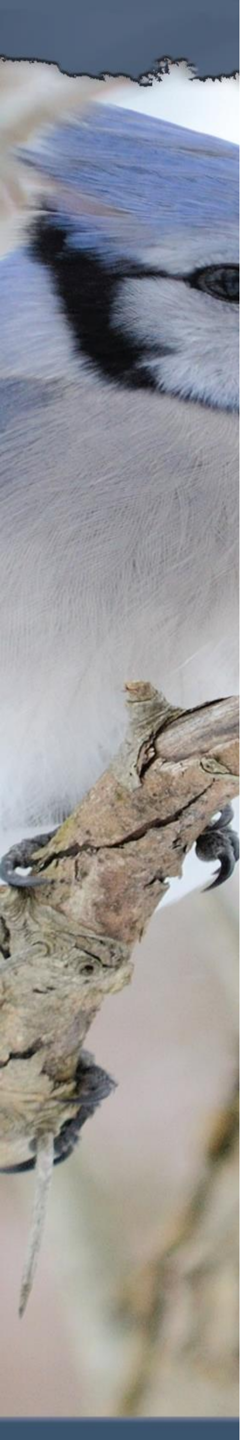
Overview

- Constructing GUIs
- GUI layout
- Event handling
- Styling



GUI Principles

- Components: GUI building blocks.
  - Buttons, menus, sliders, etc.
- Layout: arranging components to form a usable GUI.
  - Using layout *managers*.
- Events: reacting to user input.
  - Button presses, menu selections, etc.



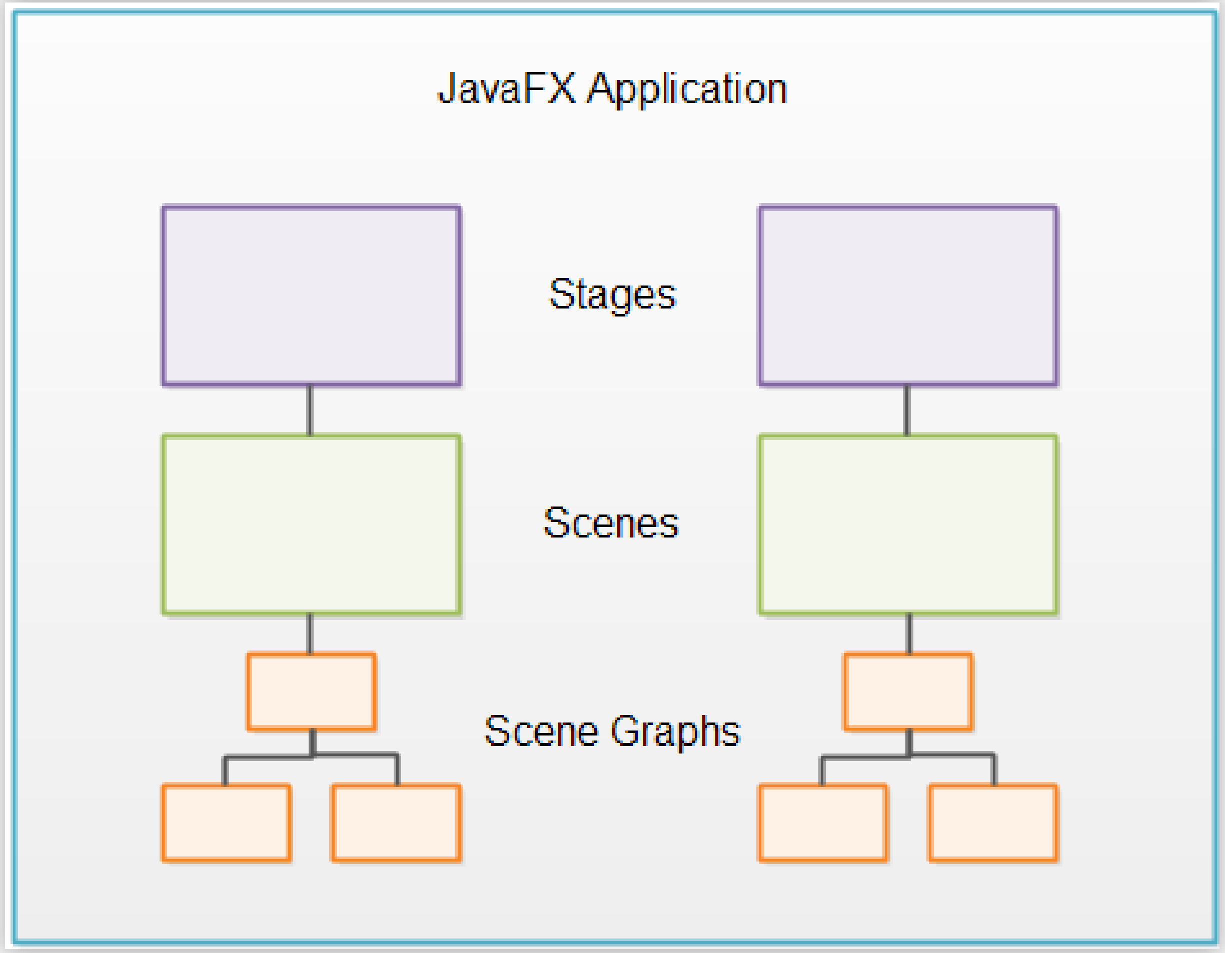
Tutorials

- Oracle
- Tutorialspoint
- Jenkov

Hello World!

- We create an application displaying window named “Hello world”.
  - 2 imports.
  - Inherit from javafx.application.Application
  - Override start(Stage primaryStage) throws Exception

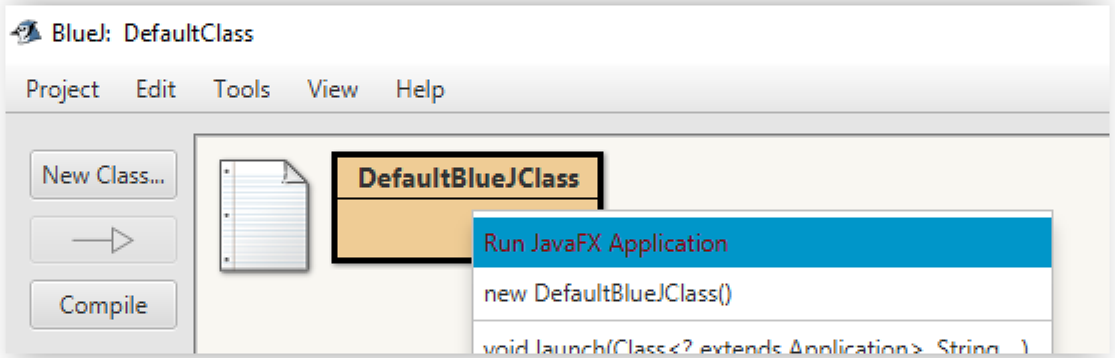
Stage, Scenes and Scene Graphs



jenkov.com

JavaFX-class in BlueJ

- We create a default JavaFX-class in BlueJ.
- Non-BlueJ users may find the code in ITL…



Task!

- Investigate what “MnemonicParsing” is.
- Try to change the code so that you may press the button using your keyboard with the hotkey “c”.

Layouts

- *JavaFX layouts* are components which contains other components inside them.
- The layout component manages the layout of the components nested inside it.
- A layout component must be attached to the scene graph of some Scene object to be visible.

jenkov.com

Layout examples

- Pane
- Hbox
- Vbox
- GridPane

Nested layouts

- It is possible to nest layout components inside other layout components.
- This can be useful to achieve a specific layout.
- For instance, to get horizontal rows of components which are not layed out in a grid, but differently for each row, you can nest multiple HBox layout components inside a VBox component.

jenkov.com

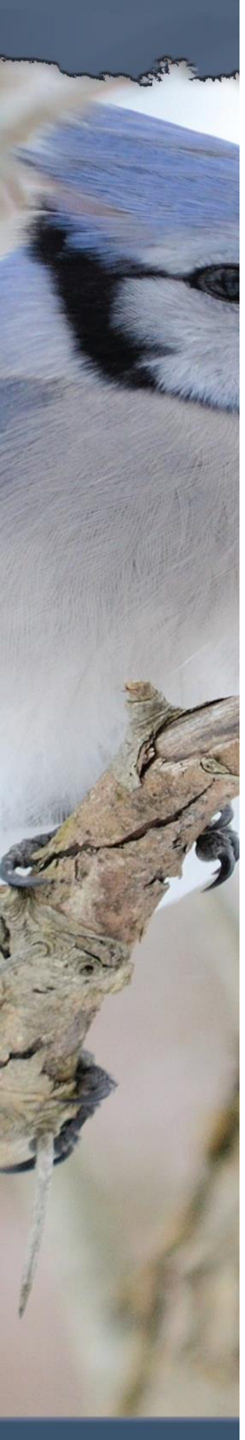
Styling

- Default CSS
- Scene CSS
- Parent CSS
- Component `style`

Default CSS

- The default style sheet for JavaFX is called Modena, and is located in the JavaFX JAR file.

- Task: Find the default CSS and locate the default color for a button.

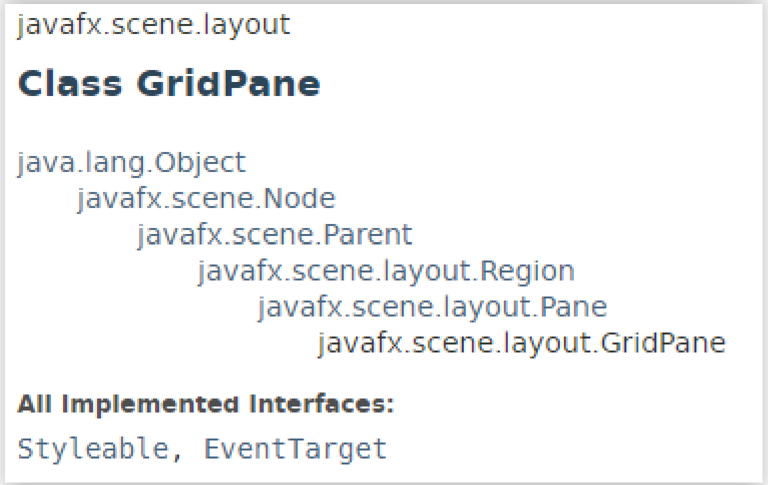


Scene specific CSS

- You may define a specific CSS for a scene. This will override the default CSS.
- Task: Create a CSS with a `root`-element (`.root{}`) where you define:
  - -fx-font-size : <font size>pt;
  - -fx-font-family: "Some font ";
  - -fx-base: <some color>;
  - -fx-background: <some color>;
- Style the scene: `scene.getStylesheets().add("<css file name>");`

Parent CSS

- Style all children of a specific parent.
- Layout components…



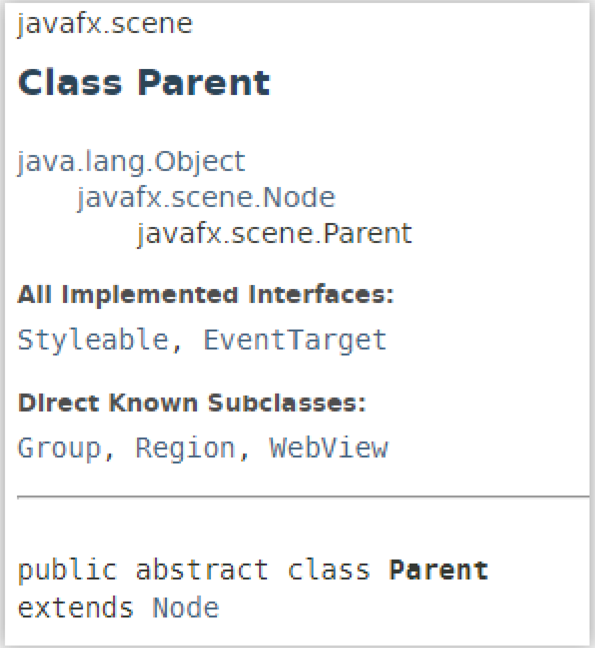
javafx.scene.layout

Class GridPane

```
java.lang.Object
    javafx.scene.Node
        javafx.scene.Parent
            javafx.scene.layout.Region
                javafx.scene.layout.Pane
                    javafx.scene.layout.GridPane
```

All Implemented Interfaces:

`Styleable, EventTarget`



javafx.scene

Class Parent

```
java.lang.Object
    javafx.scene.Node
        javafx.scene.Parent
```

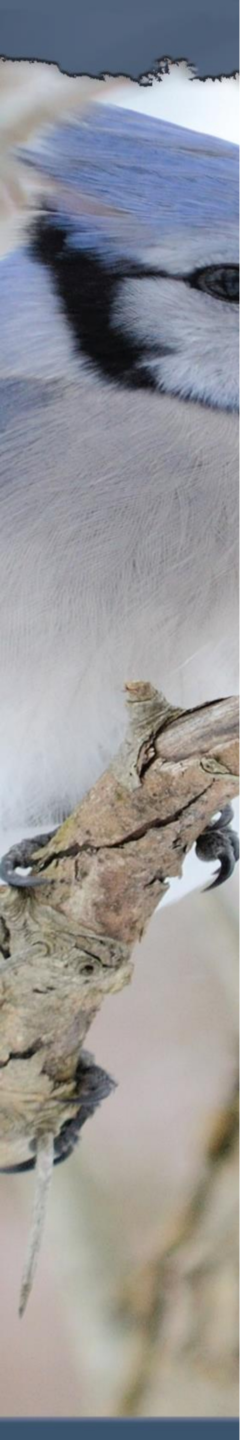
All Implemented Interfaces:

`Styleable, EventTarget`

Direct Known Subclasses:

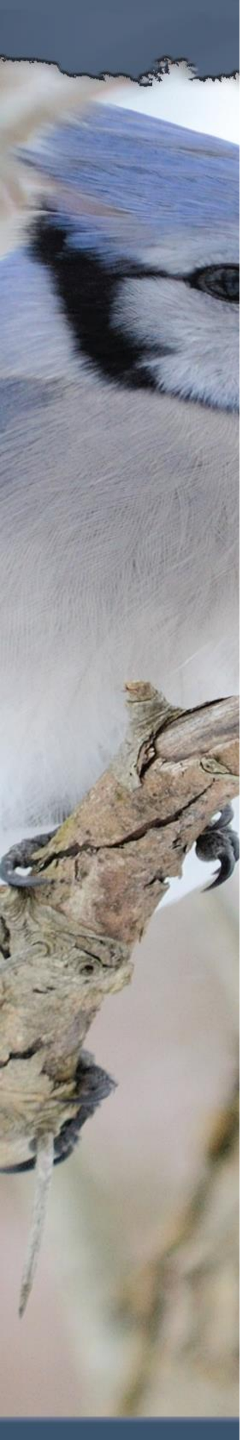
`Group, Region, WebView`

```
public abstract class Parent
extends Node
```



Task!

- Task: Create a CSS with a `root`-element (`.root{}`) where you define:
  - -fx-base: <different color than last time>;
  - -fx-background: < different color than last time >;
- Style the `GridPane`: `pane.getStylesheets().add("<css file name>");`
- Keep the former CSS, so that you now use two CSS-files. One for the scene, and one for the gridpane.

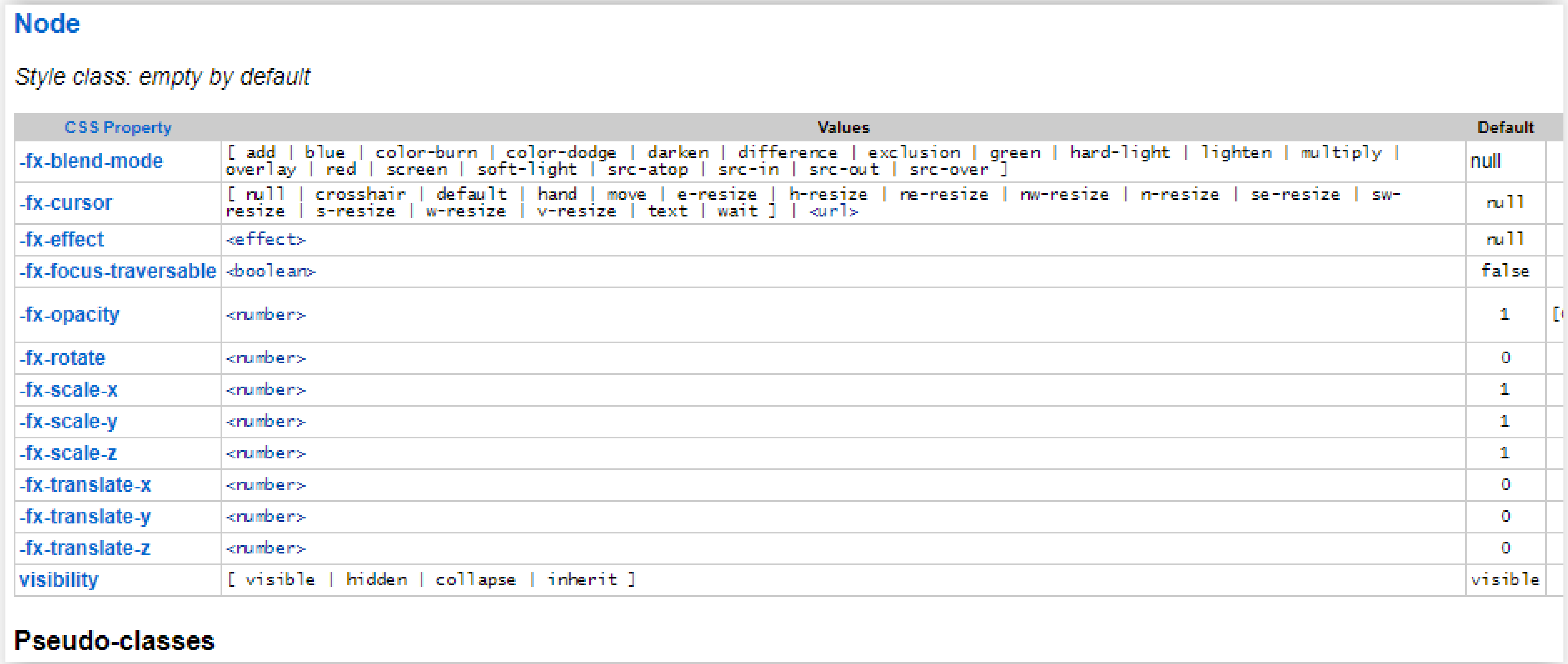


Component style

- You may style a component directly.
- Task: Style the button like this:
  - `button.setStyle("-fx-background-color: #ff0000");`
- Keep using the two CSS for earlier tasks. There is now four style-resources in use:
  - Default
  - Scene
  - Parent (GridPane)
  - Component (Button)

Css reference

- oracle



Node

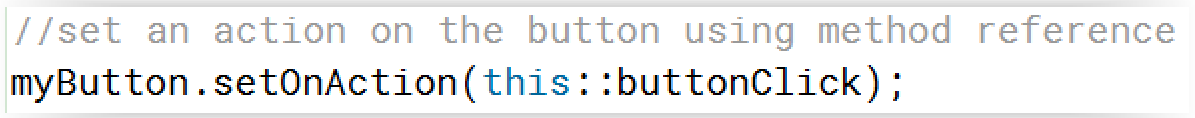
Style class: empty by default

CSS Property	Values	Default	
-fx-blend-mode	[add | blue | color-burn | color-dodge | darken | difference | exclusion | green | hard-light | lighten | multiply | overlay | red | screen | soft-light | src-atop | src-in | src-out | src-over]	null	
-fx-cursor	[null | crosshair | default | hand | move | e-resize | h-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize | v-resize | text | wait] | <url>	null	
-fx-effect	<effect>	null	
-fx-focus-traversable	<boolean>	false	
-fx-opacity	<number>	1	[
-fx-rotate	<number>	0	
-fx-scale-x	<number>	1	
-fx-scale-y	<number>	1	
-fx-scale-z	<number>	1	
-fx-translate-x	<number>	0	
-fx-translate-y	<number>	0	
-fx-translate-z	<number>	0	
visibility	[visible | hidden | collapse | inherit]	visible	

Pseudo-classes

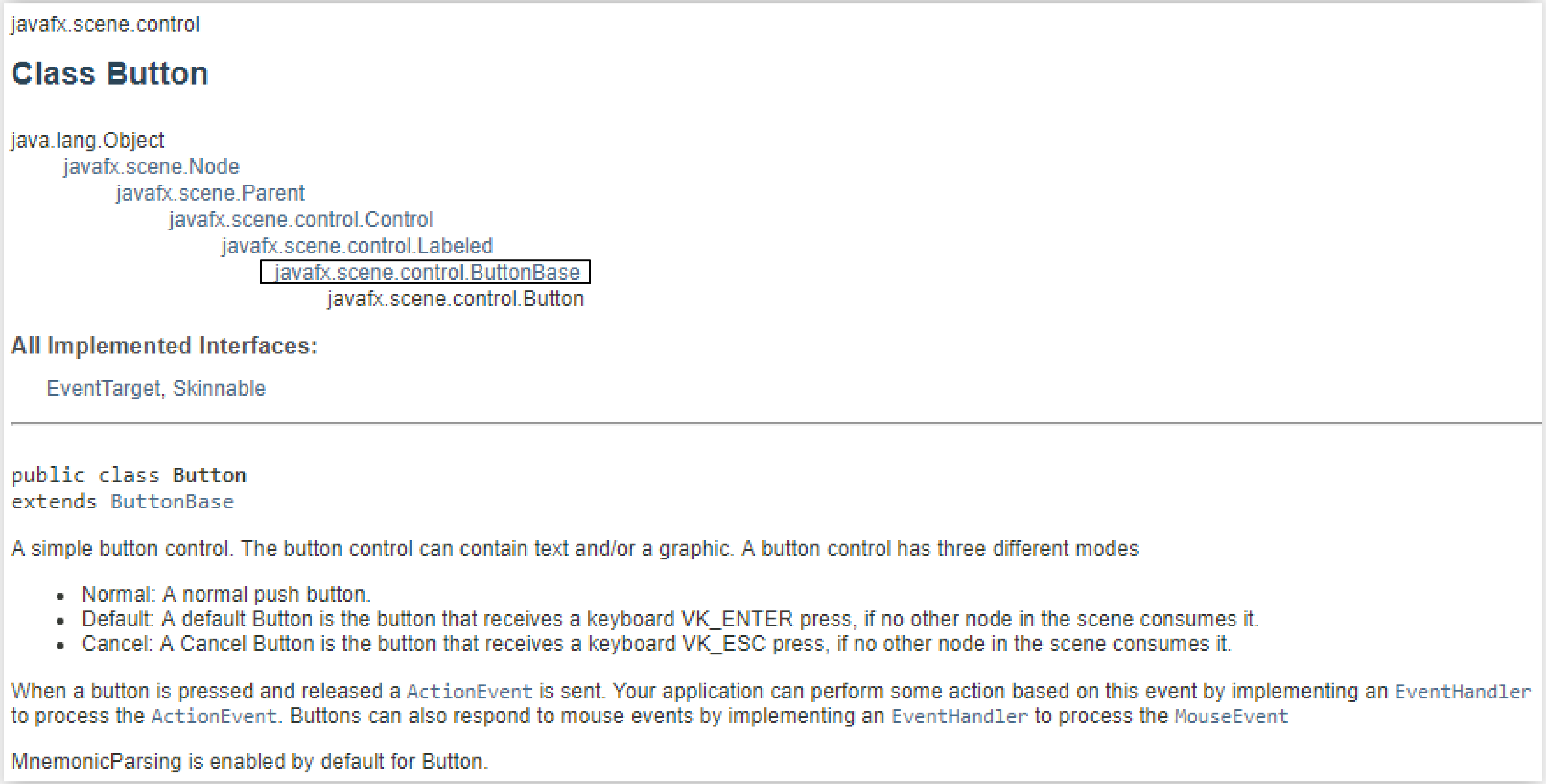
ActionEvent

- Topic next week☺



```
//set an action on the button using method reference
myButton.setOnAction(this::buttonClick);
```

Button



javafx.scene.control

Class Button

```
java.lang.Object
    javafx.scene.Node
        javafx.scene.Parent
            javafx.scene.control.Control
                javafx.scene.control.Labeled
                    javafx.scene.control.ButtonBase
                        javafx.scene.control.Button
```

All Implemented Interfaces:

EventTarget, Skinnable

```
public class Button
extends ButtonBase
```

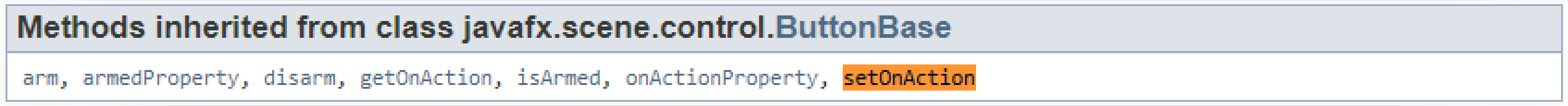
A simple button control. The button control can contain text and/or a graphic. A button control has three different modes

- Normal: A normal push button.
- Default: A default Button is the button that receives a keyboard VK_ENTER press, if no other node in the scene consumes it.
- Cancel: A Cancel Button is the button that receives a keyboard VK_ESC press, if no other node in the scene consumes it.

When a button is pressed and released a `ActionEvent` is sent. Your application can perform some action based on this event by implementing an `EventHandler` to process the `ActionEvent`. Buttons can also respond to mouse events by implementing an `EventHandler` to process the `MouseEvent`

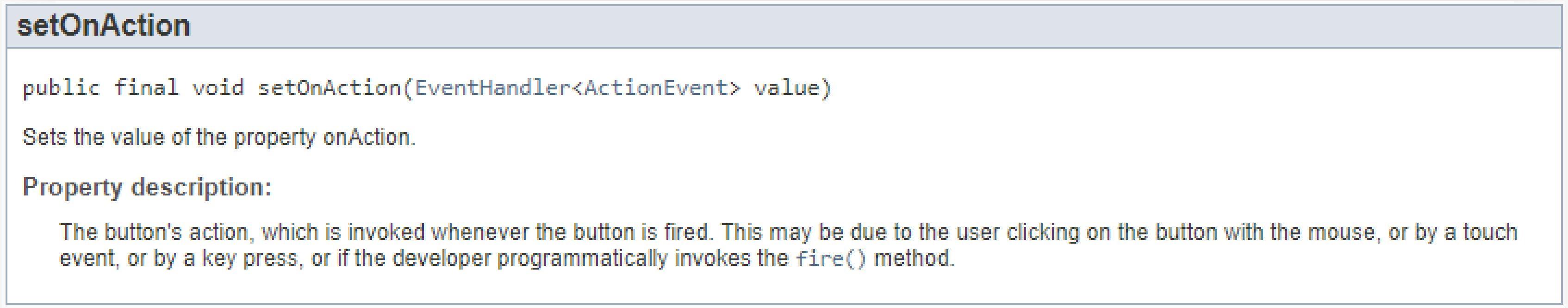
MnemonicParsing is enabled by default for Button.

Button, cont.



Methods inherited from class javafx.scene.control.ButtonBase

```
arm, armedProperty, disarm, getOnAction, isArmed, onActionProperty, setOnAction
```



setOnAction

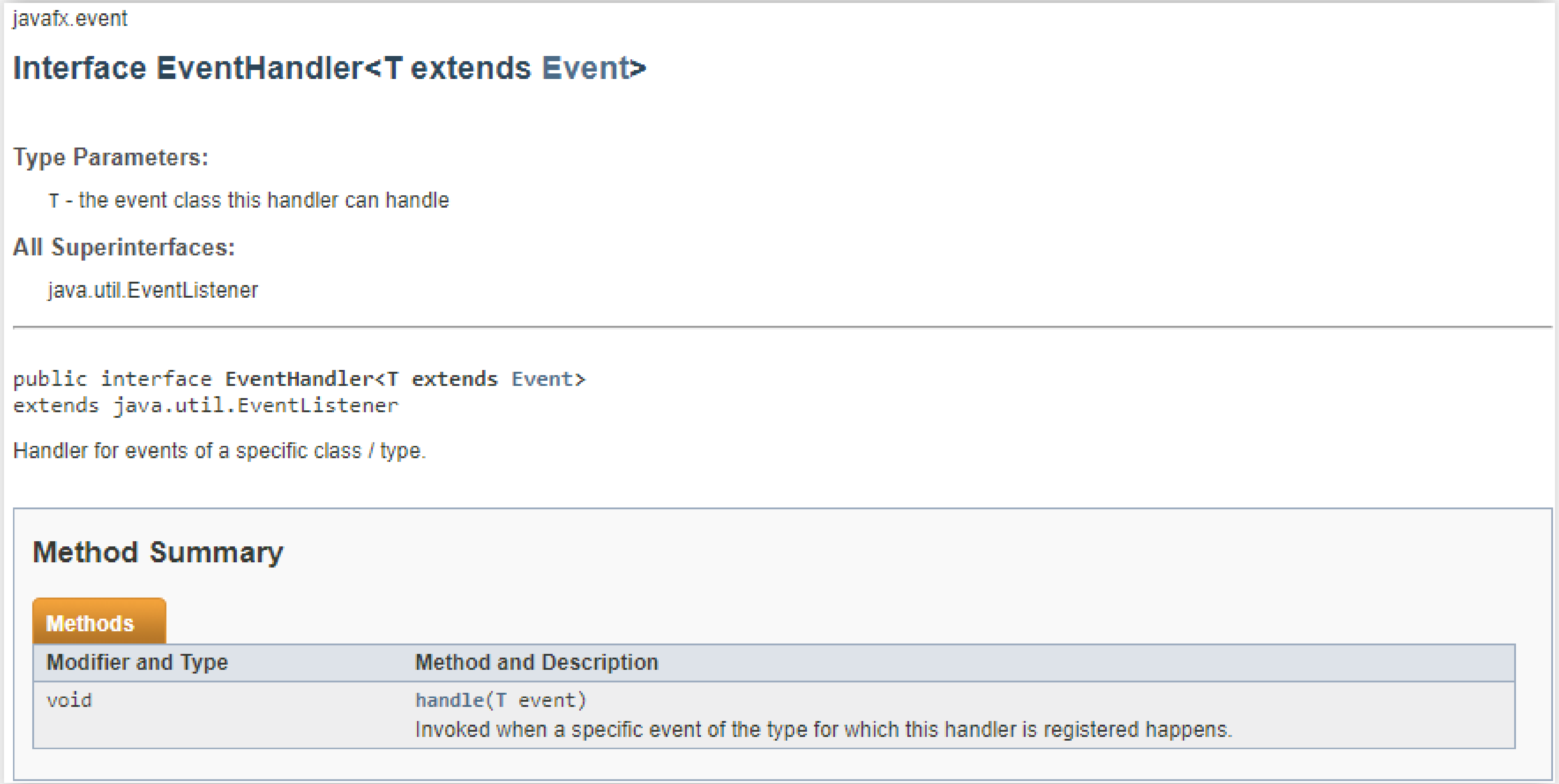
```
public final void setOnAction(EventHandler<ActionEvent> value)
```

Sets the value of the property onAction.

Property description:

The button's action, which is invoked whenever the button is fired. This may be due to the user clicking on the button with the mouse, or by a touch event, or by a key press, or if the developer programmatically invokes the `fire()` method.

EventHandler



javafx.event

Interface EventHandler<T extends Event>

Type Parameters:

T - the event class this handler can handle

All Superinterfaces:

java.util.EventListener

```
public interface EventHandler<T extends Event>
extends java.util.EventListener
```

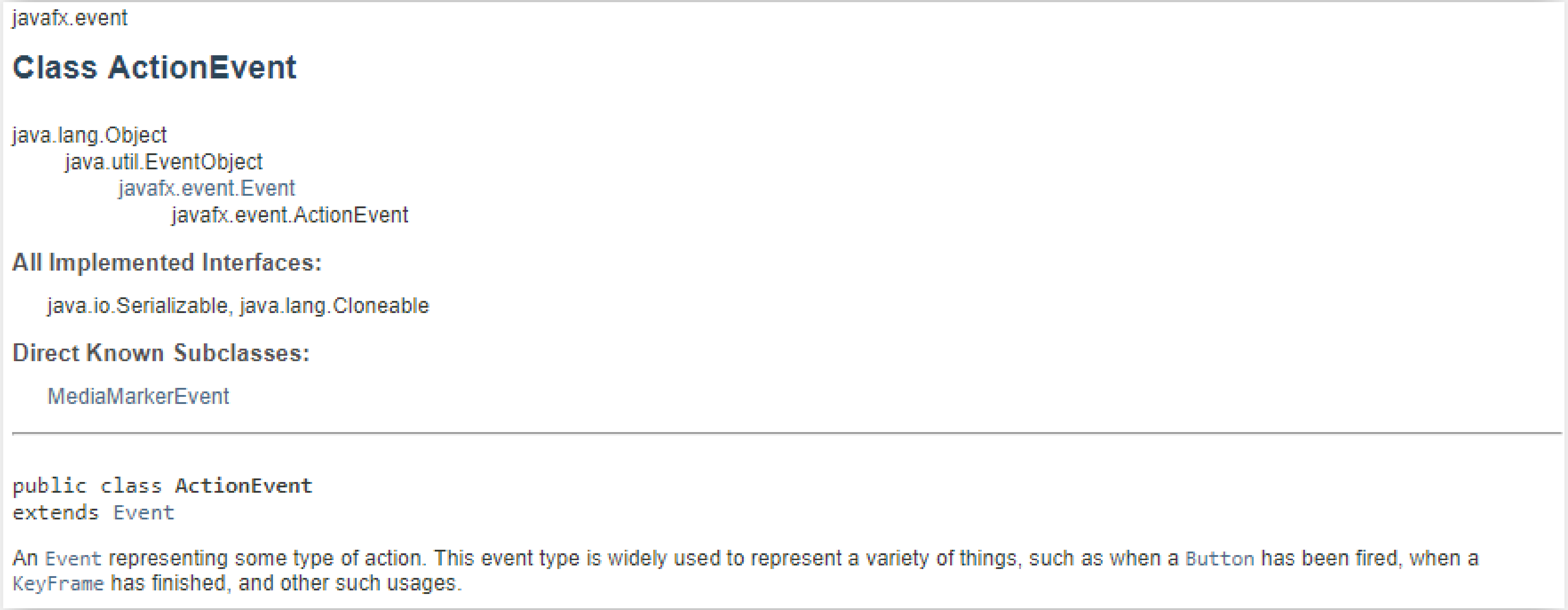
Handler for events of a specific class / type.

Method Summary

Methods

Modifier and Type	Method and Description
void	`handle(T event)` Invoked when a specific event of the type for which this handler is registered happens.

ActionEvent



javafx.event

Class ActionEvent

java.lang.Object
    java.util.EventObject
        javafx.event.Event
            javafx.event.ActionEvent

All Implemented Interfaces:

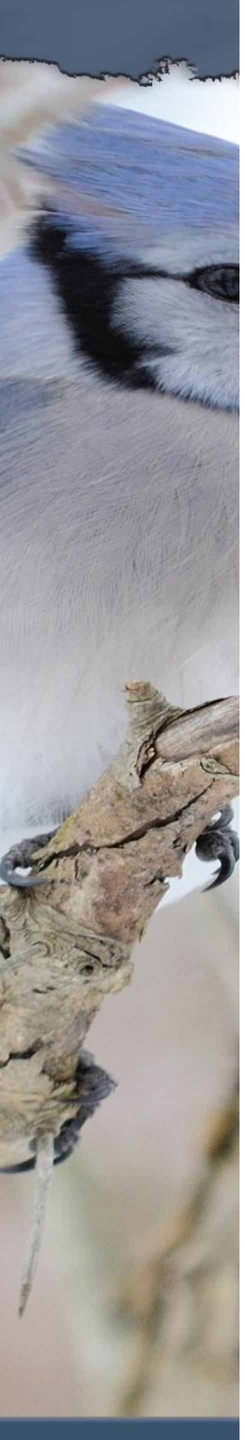
java.io.Serializable, java.lang.Cloneable

Direct Known Subclasses:

MediaMarkerEvent

```
public class ActionEvent
extends Event
```

An `Event` representing some type of action. This event type is widely used to represent a variety of things, such as when a `Button` has been fired, when a `KeyFrame` has finished, and other such usages.

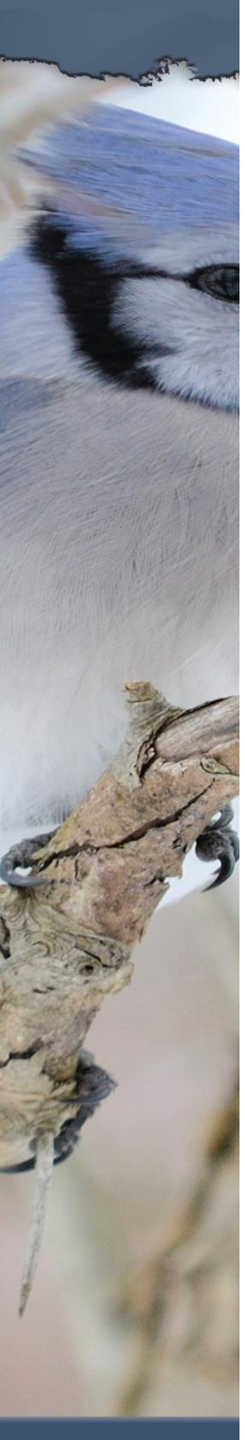


Task!

- Create a class that implements `EventHandler<ActionEvent>` and calls the method `buttonClick` in `DefaultBlueJClass`.

Recap

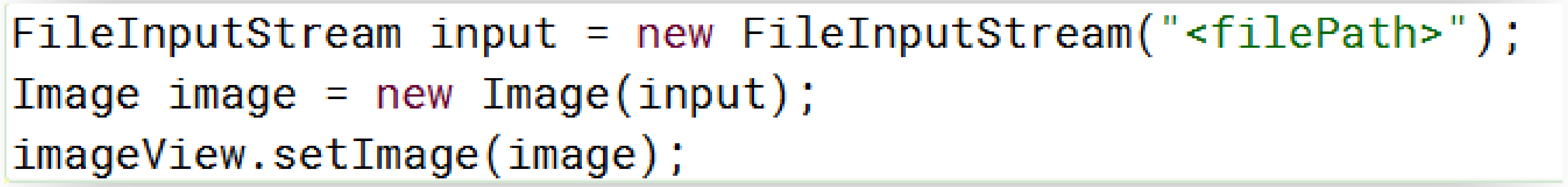
- We register an event handler for the button click event. In our example, we want to call a specific method (`buttonClick`) when the button is clicked.
- Java 8 (lambda expressions) makes it easier than before. We can easily explain what to do with the `ActionEvent`. We no longer need to create a class to place the handler method.



HowTo display images

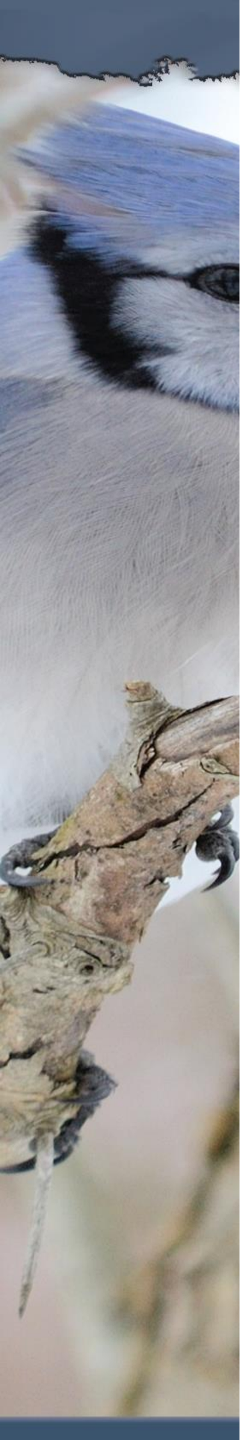
1. Create a file input stream referencing your image file.
2. Create the image by passing the input stream.
3. Create the image view by passing the image.

Example



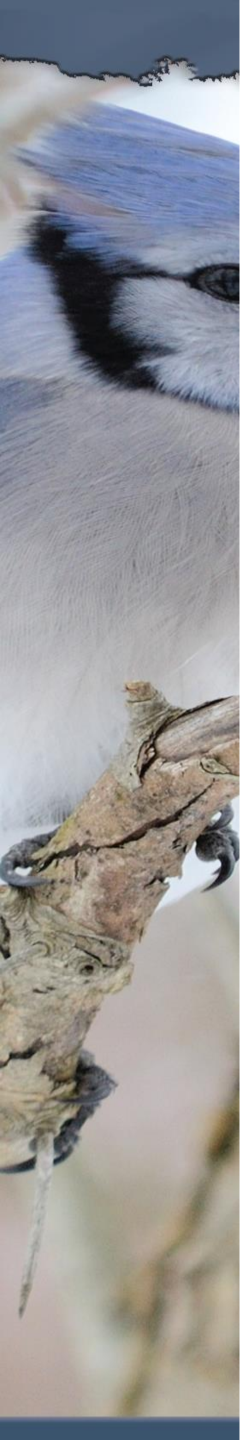
```
FileInputStream input = new FileInputStream("<filePath>");
Image image = new Image(input);
imageView.setImage(image);
```

- Task! Add an image to the application.



TextFields

- Task!
  - Create a textfield.
  - Add it to the application.
  - When the button is clicked, write the content of the textfield to System.out.



Arbeidskrav 2

- Walk through.
- Questions?

Nå

- Kahoot☺
- Deretter øving her på Fjerdingen (sjekk TimEdit for rom)

- Neste uke: Exception handling