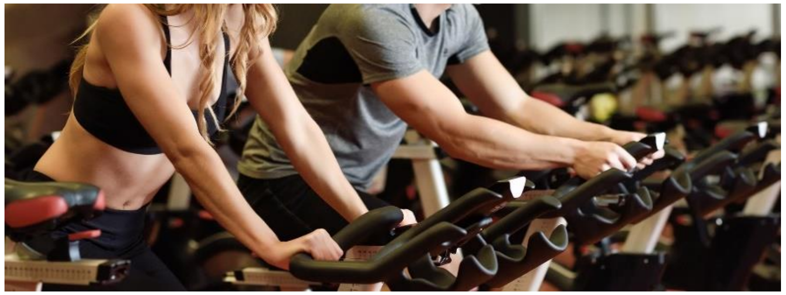
IECS273/274 [1111 3670/3671]
物件導向設計與實習

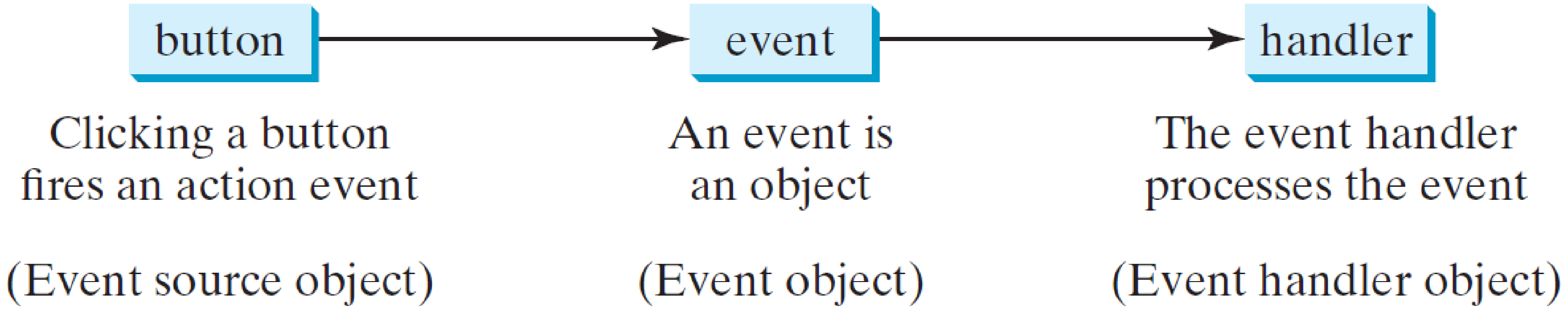
11#1 : GUI Layout



01 ▸ Event-Driven Programming

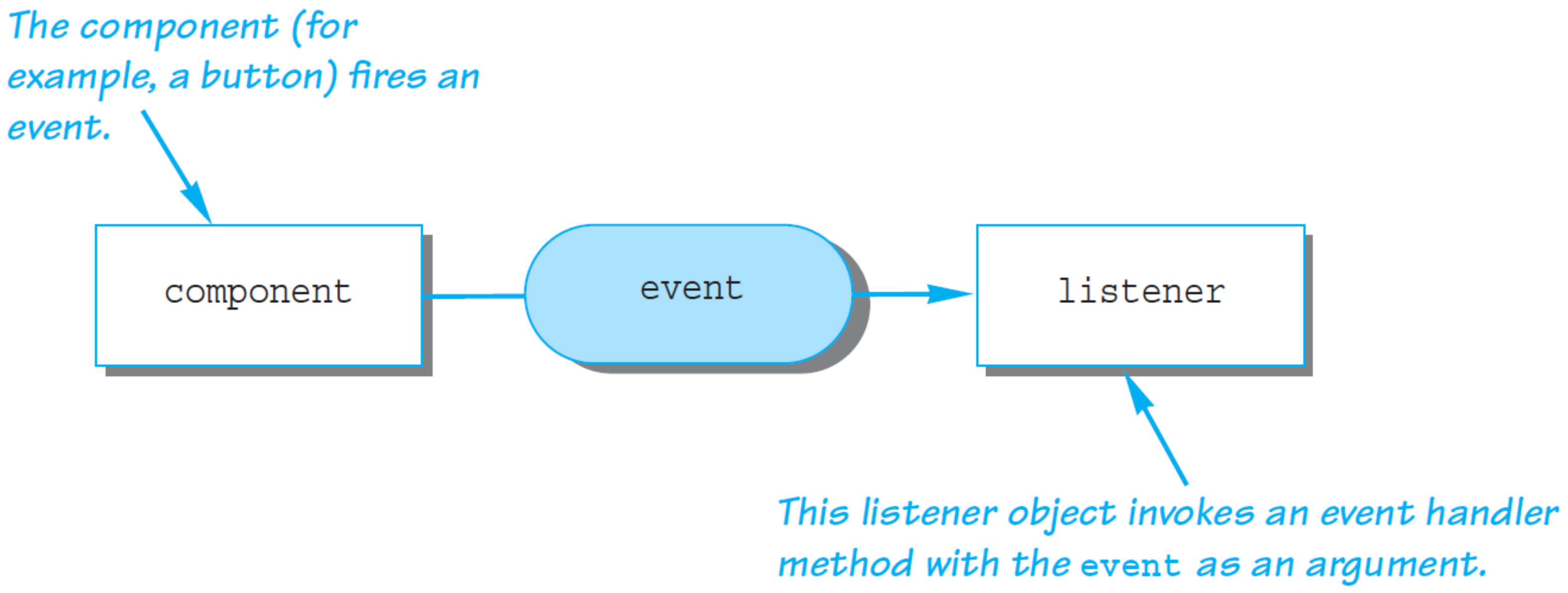
Event-Driven Programming

- **Event-driven programming** is a programming style that uses a signal-and-response approach to programming.
- Signals to objects are things called *events.* An **event** is an object that acts as a signal to another object known as a **listener**. The sending of the event is called **firing the event**.
- A listener object has methods that specify what will happen when events of various kinds are received by the listener. These methods that handle events are called **event handlers**.
- You can write code to process events such as a button click, mouse movement, and keystrokes.



Event-Driven Programming

- In event-driven programming, you create objects that can fire events, and you create listener objects to react to the events.


The component (for example, a button) fires an event.
component
event
listener
This listener object invokes an event handler method with the event as an argument.

Event-Driven Programming

- Applications classified by user interface
  - ✓ No UI Applications
  - ✓ Console Applications
  - ✓ GUI Applications
    - Windows_Based
    - Web_Based

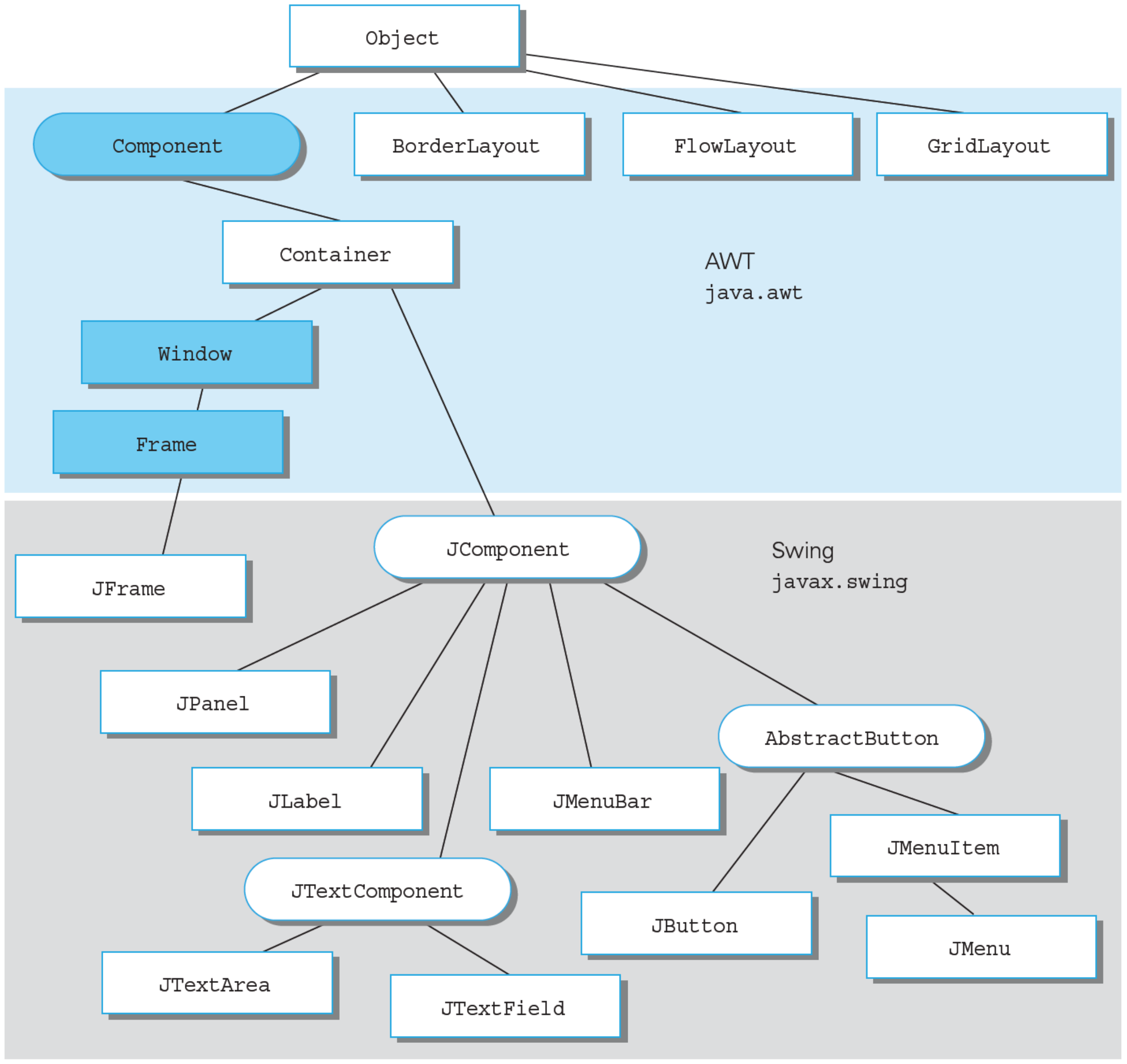
02

Concept of AWT & Swing & JavaFX

AWT & Swing

- A GUI (graphical user interface) is a windowing system that interacts with the user.
- The Java AWT (Abstract Window Toolkit) package is the original Java package for creating GUIs.
- The Swing pattckage is an improved version of the AWT.
  - ✓ It does not completely replace the AWT.
  - ✓ Some AWT classes are replaced by Swing classes, but other AWT classes are needed when using Swing.
- Swing GUIs are designed using a form of object-oriented programming known as **event-driven programming.**

AWT & Swing



以下取材自：Y. D. Liang, *Introduction to Java Programming and Data Structures (11th Edition),* Pearson

03 ▶ Swing Basic

A Simple Window

- A simple window can consist of an object of the **Jframe** class
  - ✓ A JFrame object includes a border and the usual three buttons for minimizing, changing the size of, and closing the window
  - ✓ The JFrame class is found in the javax.swing package

    ```
    JFrame firstWindow = new JFrame();
    ```

- A **JFrame** can have components added to it, such as buttons, menus, and text labels
  - ✓ These components can be programmed for action

    ```
    firstWindow.add(endButton);
    ```

  - ✓ It can be made visible using the *setVisible* method

    ```
    firstWindow.setVisible(true);
    ```

Buttons

- A button object is created from the class **JButton** and can be added to a JFrame
- The argument to the JButton constructor is the string that appears on the button when it is displayed

```
JButton endButton = new JButton("Click to end program.");

firstWindow.add(endButton);
```

Action Listeners and Action Events

- Clicking a button fires an event
  - ✓ The event object is "sent" to another object called a listener:
    - A method in the listener object is invoked automatically.
    - It is invoked with the event object as its argument.
  - ✓ In order to set up this relationship, a GUI program must do two things:
    - It must specify, for each button, what objects are its listeners.
    - It must define the methods that will be invoked automatically.
- When the event is sent to the listener

```
EndingListener buttonEar = new EndingListener();

endButton.addActionListener(buttonEar);
```

  - ✓ A listener object named buttonEar is created and registered as a listener for the button named endButton
  - ✓ A button fires events known as *action events*, which are handled by listeners known as *action listeners*

Action Listeners and Action Events

- Different kinds of components require different kinds of listener classes to handle the events they fire
  - ✓ An action listener is an object whose class implements the *ActionListener* interface

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

  - ✓ The EndingListener class defines its actionPerformed method
    - When the user clicks the endButton, an action event is sent to the action listener for that button.
    - The EndingListener object buttonEar is the action listener for endButton.
    - The action listener buttonEar receives the action event as the parameter e to its actionPerformed method, which is automatically invoked. That e must be received, even if it is not used.

Containers and Layout Managers

- Multiple components can be added to the content pane of a **JFrame** using the *add* method
  - ✓ The add method does not specify how these components are to be arranged
  - ✓ To describe how multiple components are to be arranged, a **layout manager** is used.
    - There are a number of layout manager classes such as *BorderLayout, FlowLayout, and GridLayout.*
    - If a layout manager is not specified, a default layout manager is used.

Border Layout Managers

- A **BorderLayout manager** places the components that are added to a JFrame object into five regions
  - ✓ These regions are: BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST, and BorderLayout.Center
- A BorderLayout manager is added to a JFrame using the setLayout method

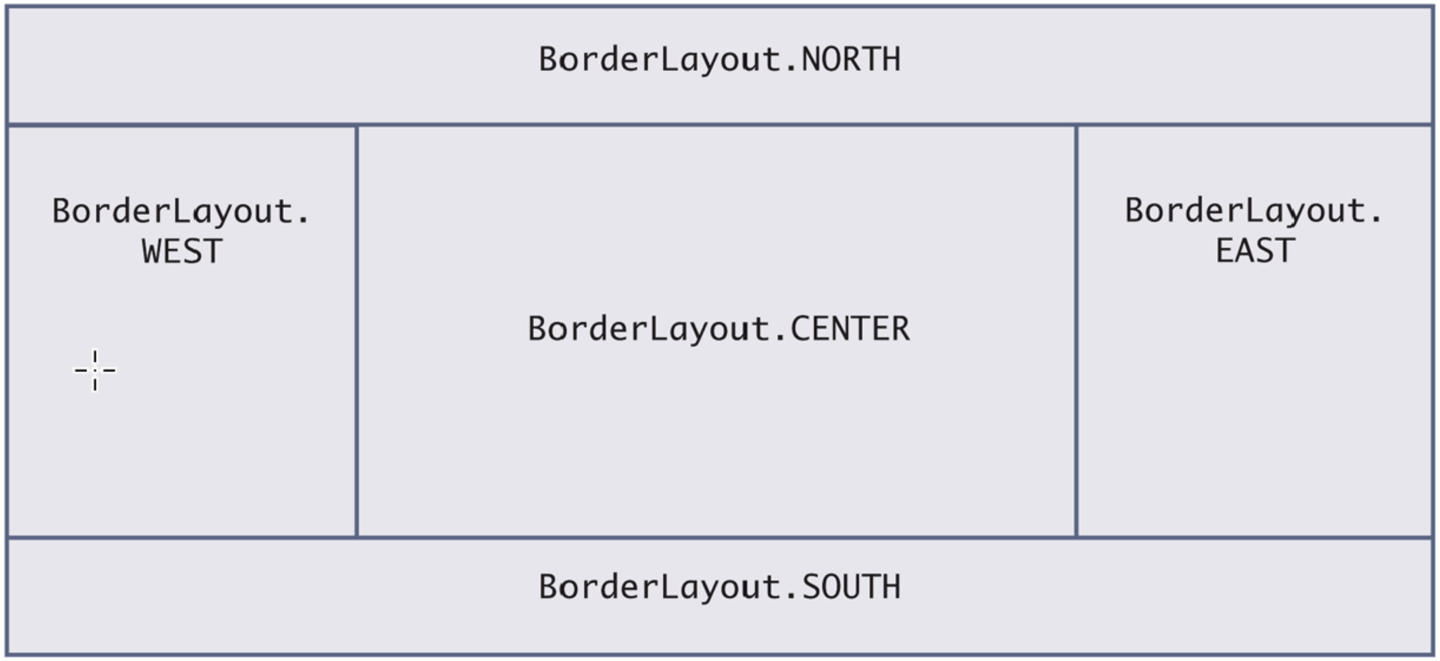
```
setLayout(new BorderLayout());
```

- When using a BorderLayout manager, the location of the component being added is given as a second argument to the add method

```
add(label1, BorderLayout.NORTH);
```

Components can be added in any order since their location is specified.

Border Layout Managers



Labels

- A label is an object of the class **JLabel**
  - ✓ Text can be added to a JFrame using a label
  - ✓ The text for the label is given as an argument when the JLabel is created
  - ✓ The label can then be added to a JFrame

```
JLabel greeting = new JLabel("Hello");
add(greeting);
```

Color

- In Java, a color is an object of the class **Color**
  - ✓ The class Color is found in the java.awt package
  - ✓ There are constants in the Color class that represent a number of basic colors
- A JFrame can not be colored directly
  - ✓ A program must color something called the content pane of the JFrame
  - ✓ Since the content pane is the "inside" of a JFrame, coloring the content pane has the effect of coloring the inside of the JFrame
  - ✓ The background color of a JFrame can be set using the following code:

```
getContentPane().setBackground(Color);
```

Panel

- A GUI is often organized in a hierarchical fashion, with containers called panels inside other containers
- A panel is an object of the **JPanel** class that serves as a simple container
  - ✓ It is used to group smaller objects into a larger component (the panel)
  - ✓ One of the main functions of a JPanel object is to subdivide a JFrame or other container

Menu Bars, Menus, and MenuItems

- A menu is an object of the class **JMenu**
  - ✓ A choice on a menu is called a *menu item*, and is an object of the class **JMenuItem**
    - A menu can contain any number of menu items
    - A menu item is identified by the string that labels it and is displayed in the order to which it was added to the menu
  - ✓ The *add* method is used to add a menu item to a menu in the same way that a component is added to a container object

```
JMenu diner = new
JMenu("Daily Specials");
JMenuItem lunch = new
JMenuItem("Lunch Specials");
lunch.addActionListener(this);
diner.add(lunch);
```

Note that the this parameter has been registered as an action listener for the menu item

Menu Bars and JFrame

- A *menu bar* is a container for menus, typically placed near the top of a windowing interface
  - ✓ The add method is used to add a menu to a menu bar in the same way that menu items are added to a menu

    ```
    JMenuBar bar = new JMenuBar();
    bar.add(diner);
    ```

  - ✓ The menu bar can be added to a JFrame in two different ways
    - Using the setJMenuBar method

      ```
      setJMenuBar(bar);
      ```

    - Using the add method – which can be used to add a menu bar to a JFrame or any other container

Text Fields

- There is also a constructor with one additional **String** parameter for displaying an initial String in the text field

```
JTextField name = new JTextField(
                                  "Enter name here.", 30);
```

  - ✓ A Swing GUI can read the text in a text field using the *getText* method

```
String inputString = name.getText();
```

  - ✓ The method *setText* can be used to display a new text string in a text field

```
name.setText("This is some output");
```

Text Areas

- A *text area* is an object of the class **JTextArea**
  - ✓ It is the same as a text field, except that it allows multiple lines
  - ✓ Two parameters to the JTextArea constructor specify the minimum number of lines, and the minimum number of characters per line that are guaranteed to be visible

```
JTextArea theText = new JTextArea(5,20);
```

  - ✓ Another constructor has one addition **String** parameter for the string initially displayed in the text area

```
JTextArea theText = new JTextArea(
                                  "Enter\ntext here." 5, 20);
```

- The line-wrapping policy for a JTextArea can be set using the method *setLineWrap*
  - ✓ The method takes one boolean type argument
  - ✓ If the argument is true, then any additional characters at the end of a line will appear on the following line of the text area
  - ✓ If the argument is false, the extra characters will remain on the same line and not be visible

```
theText.setLineWrap(true);
```

04

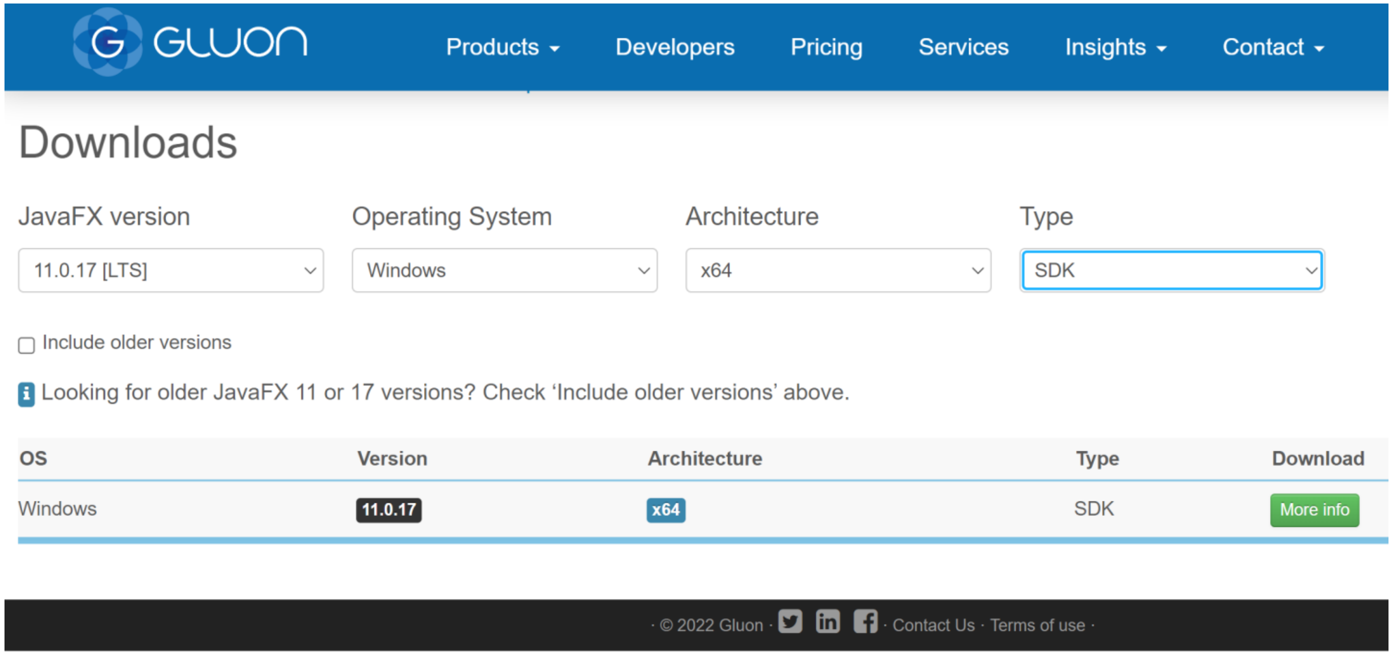
JavaFX Developing Environment

JavaFX

- JavaFX is a new framework for developing Java GUI programs.
  - ✓ JavaFX is much simpler to learn and use for new Java programmers.
  - ✓ JavaFX is a better pedagogical tool for demonstrating object-oriented programming than Swing.
  - ✓ Swing is essentially dead because it will not receive any further enhancement.
- JavaFX is the new GUI tool for developing cross-platform rich GUI applications on desktop computers and on handheld devices. It can develop simple GUI programs using layout panes, groups, buttons, labels, text fields, colors, fonts, images, image views, and shapes.
- The **javafx.application.Application** class defines the essential framework for writing JavaFX programs.

Developing JavaFX by Eclipse

1, Download JavaFX SDK
https://gluonhq.com/products/javafx/



Downloads

JavaFX version: 11.0.17 [LTS]
Operating System: Windows
Architecture: x64
Type: SDK

☐ Include older versions

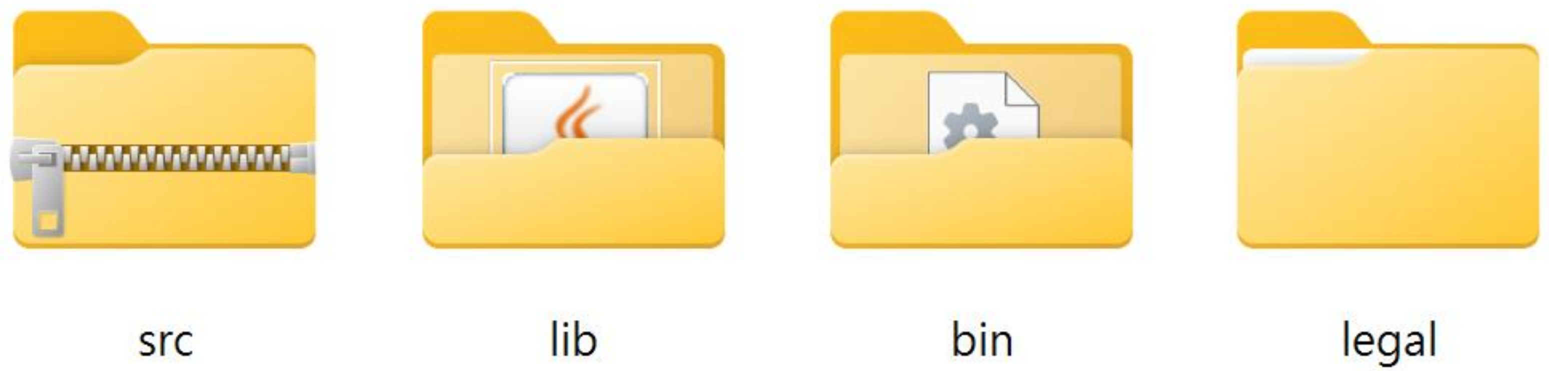
Looking for older JavaFX 11 or 17 versions? Check 'Include older versions' above.

OS	Version	Architecture	Type	Download
Windows	11.0.17	x64	SDK	More info

· © 2022 Gluon · · Contact Us · Terms of use ·

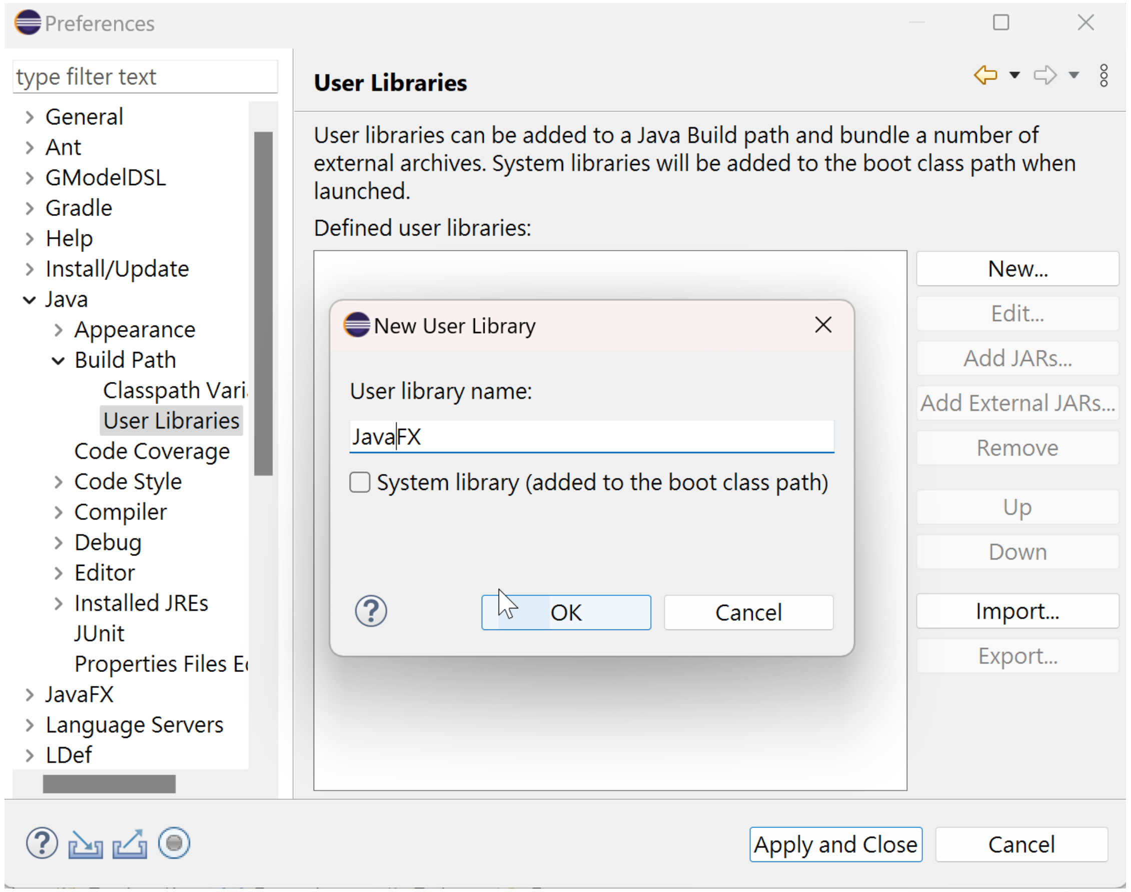
Developing JavaFX by Eclipse

2. Decompress openjfx-19_windows-x64_bin-sdk.zip



Developing JavaFX by Eclipse

3. Eclipse IDE
   Window -> Preferences -> Java -> Build Path ->User Libraries，New...
   User Library name = JavaFX
   System Library(added to the boot class path) dont check



Preferences

type filter text

- General
- Ant
- GModelDSL
- Gradle
- Help
- Install/Update
- Java
  - Appearance
  - Build Path
    - Classpath Vari
    - User Libraries
  - Code Coverage
  - Code Style
  - Compiler
  - Debug
  - Editor
  - Installed JREs
  - JUnit
  - Properties Files E
- JavaFX
- Language Servers
- LDef

User Libraries

User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

Defined user libraries:

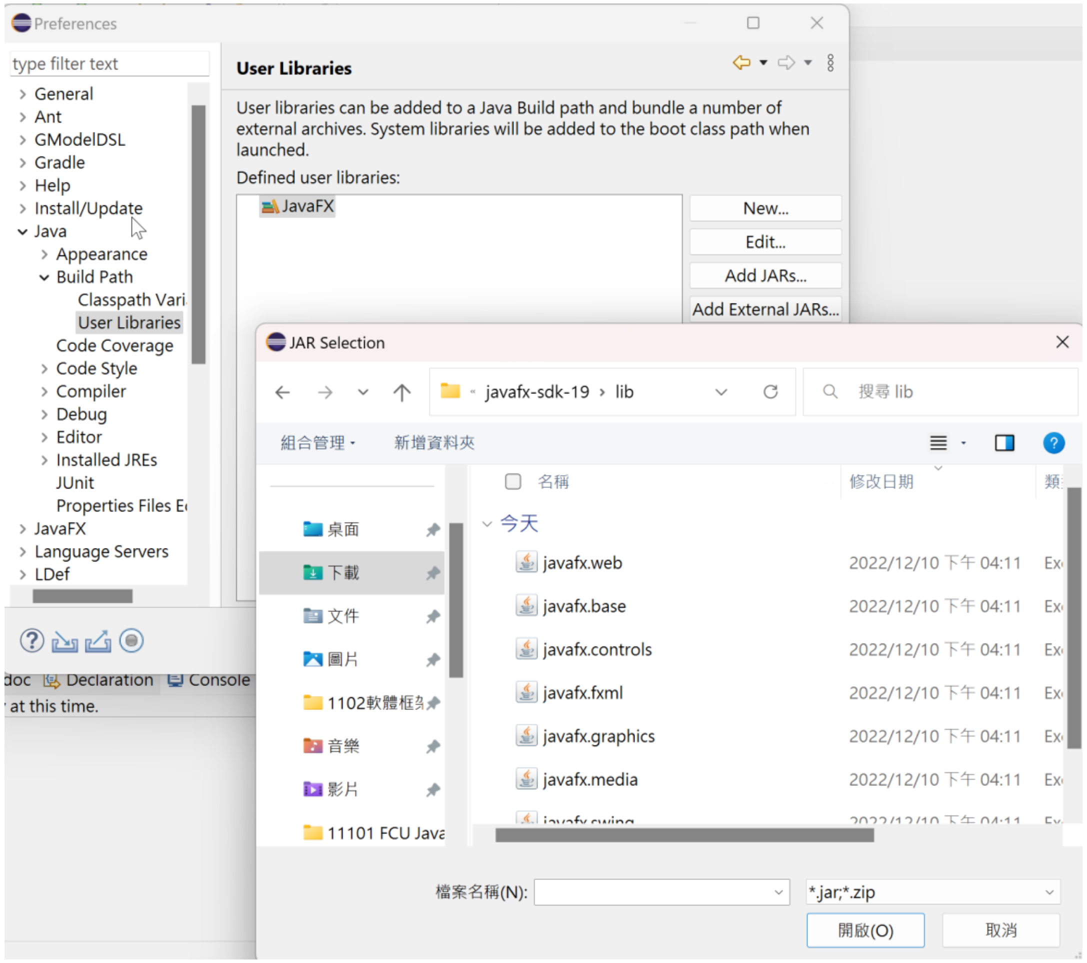
New User Library

User library name: JavaFX

☐ System library (added to the boot class path)

Developing JavaFX by Eclipse

4 、Defined user libraries = JavaFX
Add External JARs...
javafx-sdk-11.0.2 lib 、select all .jar and .zip file > Open



User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

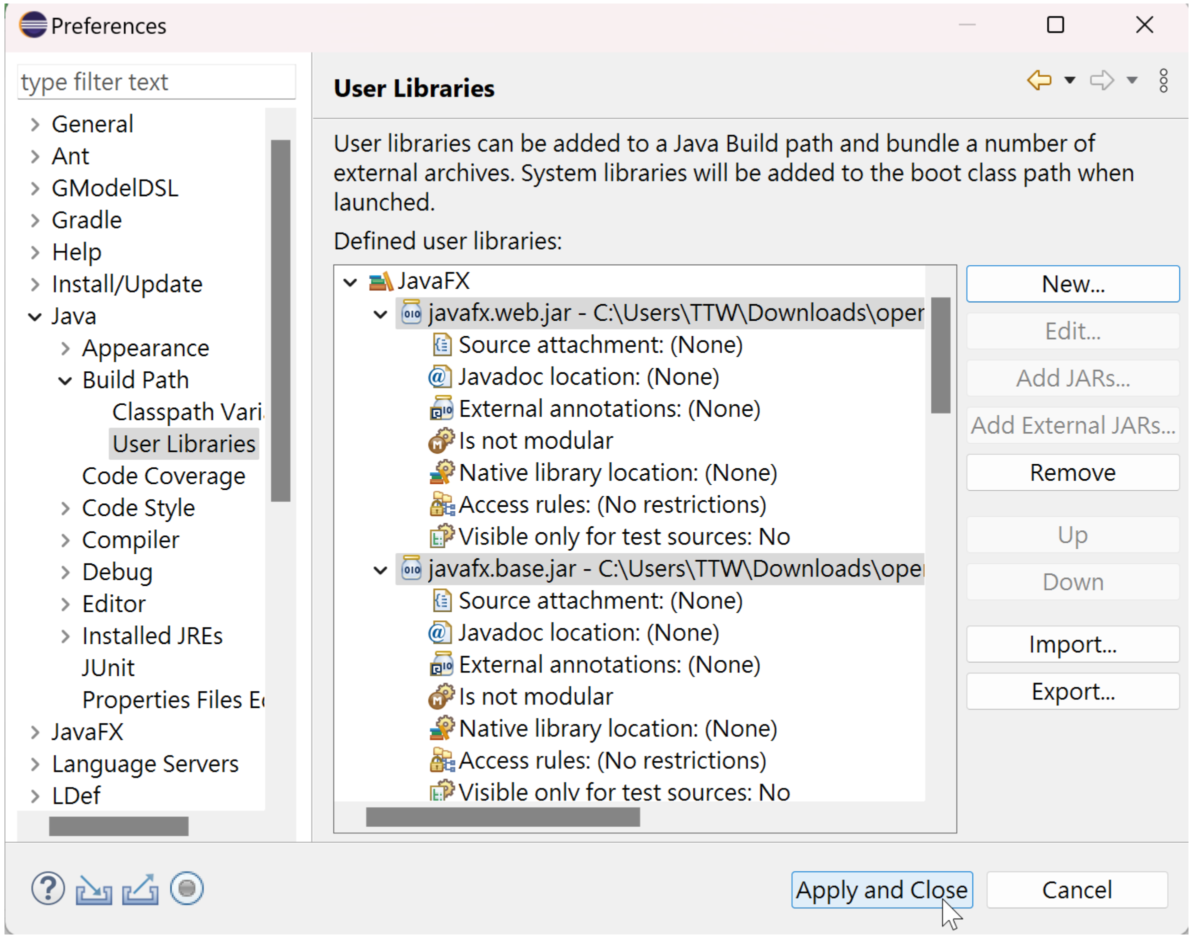
Defined user libraries:

JavaFX

名稱	修改日期
javafx.web	2022/12/10 下午 04:11
javafx.base	2022/12/10 下午 04:11
javafx.controls	2022/12/10 下午 04:11
javafx.fxml	2022/12/10 下午 04:11
javafx.graphics	2022/12/10 下午 04:11
javafx.media	2022/12/10 下午 04:11

Developing JavaFX by Eclipse

5. Apply and Close



Preferences

type filter text

- General
- Ant
- GModelDSL
- Gradle
- Help
- Install/Update
- Java
  - Appearance
  - Build Path
    - Classpath Vari
    - User Libraries
  - Code Coverage
  - Code Style
  - Compiler
  - Debug
  - Editor
  - Installed JREs
  - JUnit
  - Properties Files E
- JavaFX
- Language Servers
- LDef

User Libraries

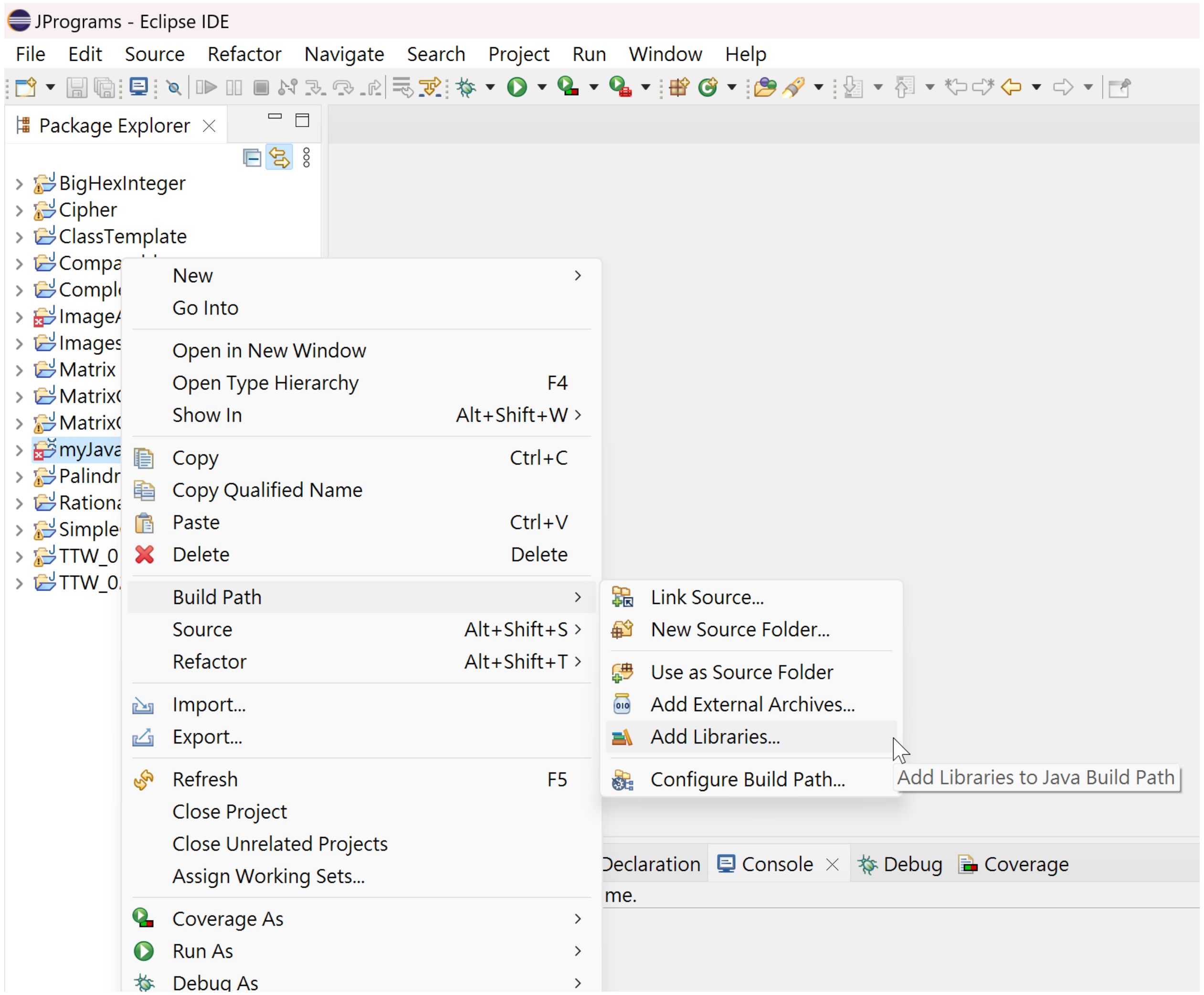
User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

Defined user libraries:

```
JavaFX
  javafx.web.jar - C:\Users\TTW\Downloads\oper
    Source attachment: (None)
    Javadoc location: (None)
    External annotations: (None)
    Is not modular
    Native library location: (None)
    Access rules: (No restrictions)
    Visible only for test sources: No
  javafx.base.jar - C:\Users\TTW\Downloads\ope
    Source attachment: (None)
    Javadoc location: (None)
    External annotations: (None)
    Is not modular
    Native library location: (None)
    Access rules: (No restrictions)
    Visible only for test sources: No
```

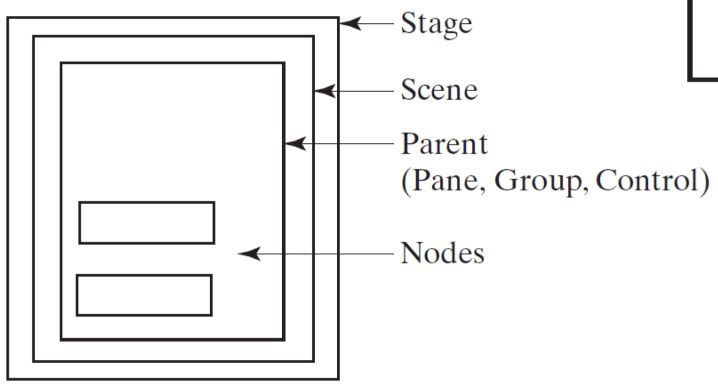
Developing JavaFX by Eclipse

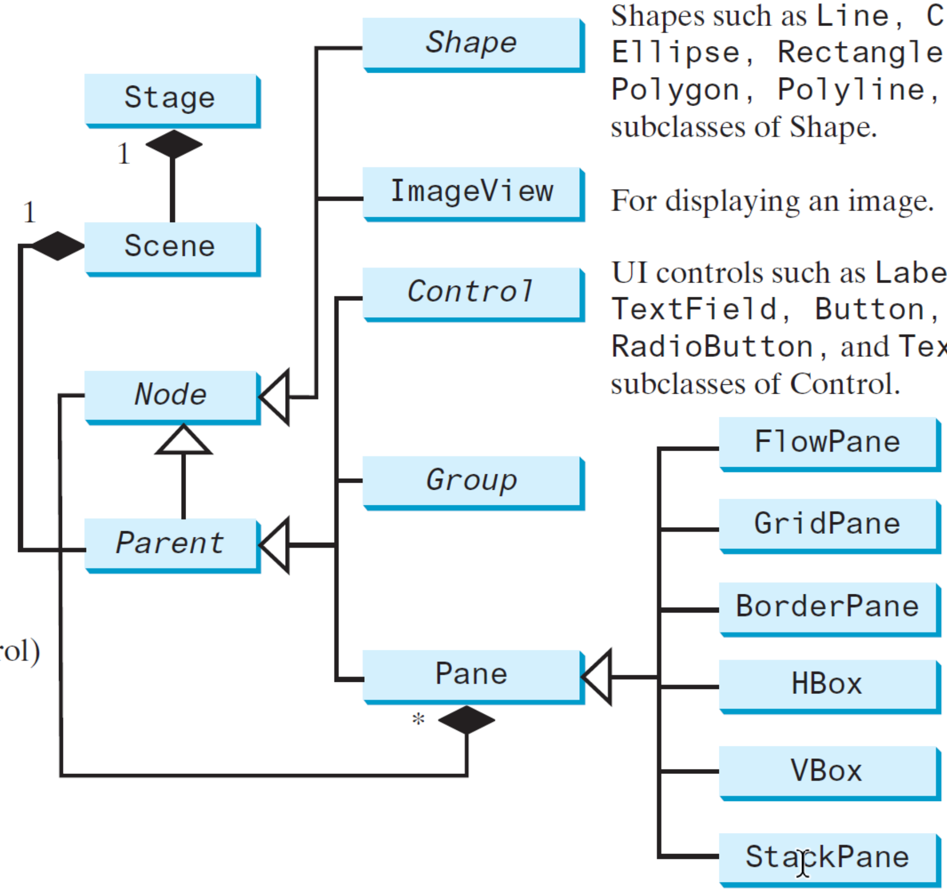
6. Eclipse IDE Package Explorer -> JavaFX
   Build Path -> Add Libraries... 。



05 ▸ Java FX (OpenJFX)

JavaFX





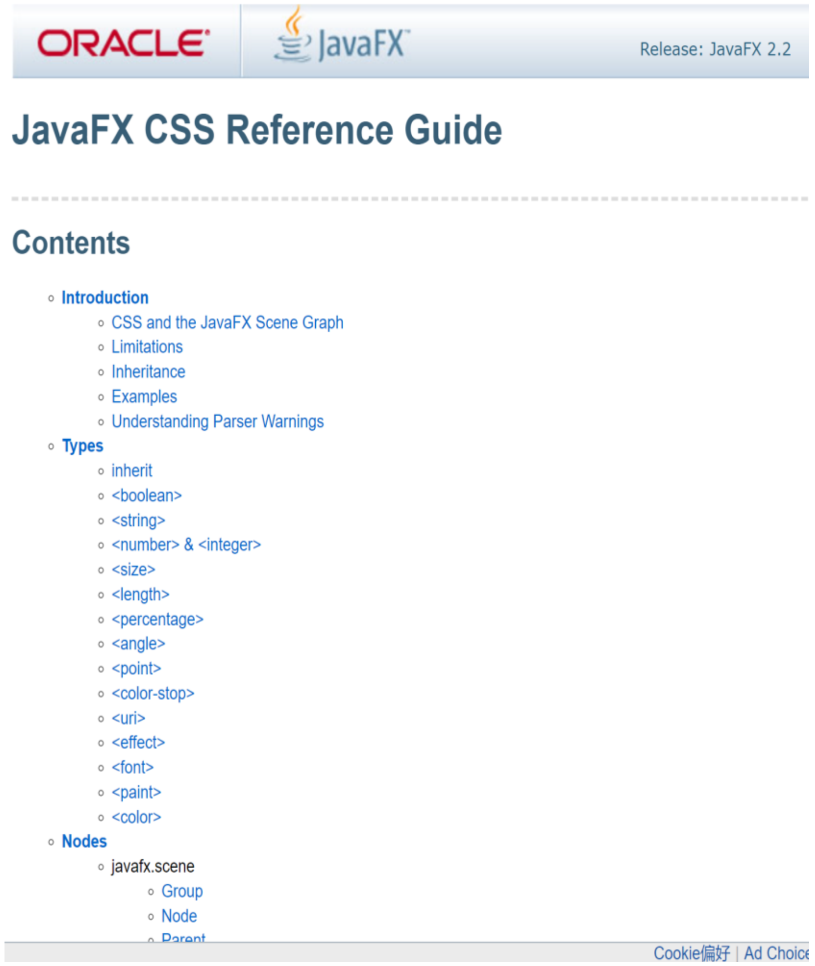
Shapes such as Line, Circle, Ellipse, Rectangle, Arc, Polygon, Polyline, and Text are subclasses of Shape.

For displaying an image.

UI controls such as Label, TextField, Button, CheckBox, RadioButton, and TextArea are subclasses of Control.

Common Properties and Methods for Nodes

- The **Node** class defines many properties and methods that are common to all nodes.
- JavaFX style properties are similar to cascading style sheets (CSS) used to specify the styles for HTML elements in a Web page. Therefore, the style properties in JavaFX are called JavaFX CSS.
- In JavaFX, a style property is defined with a prefix –fx–. Each node has its own style properties. You can find these properties at *docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html*


Release: JavaFX 2.2

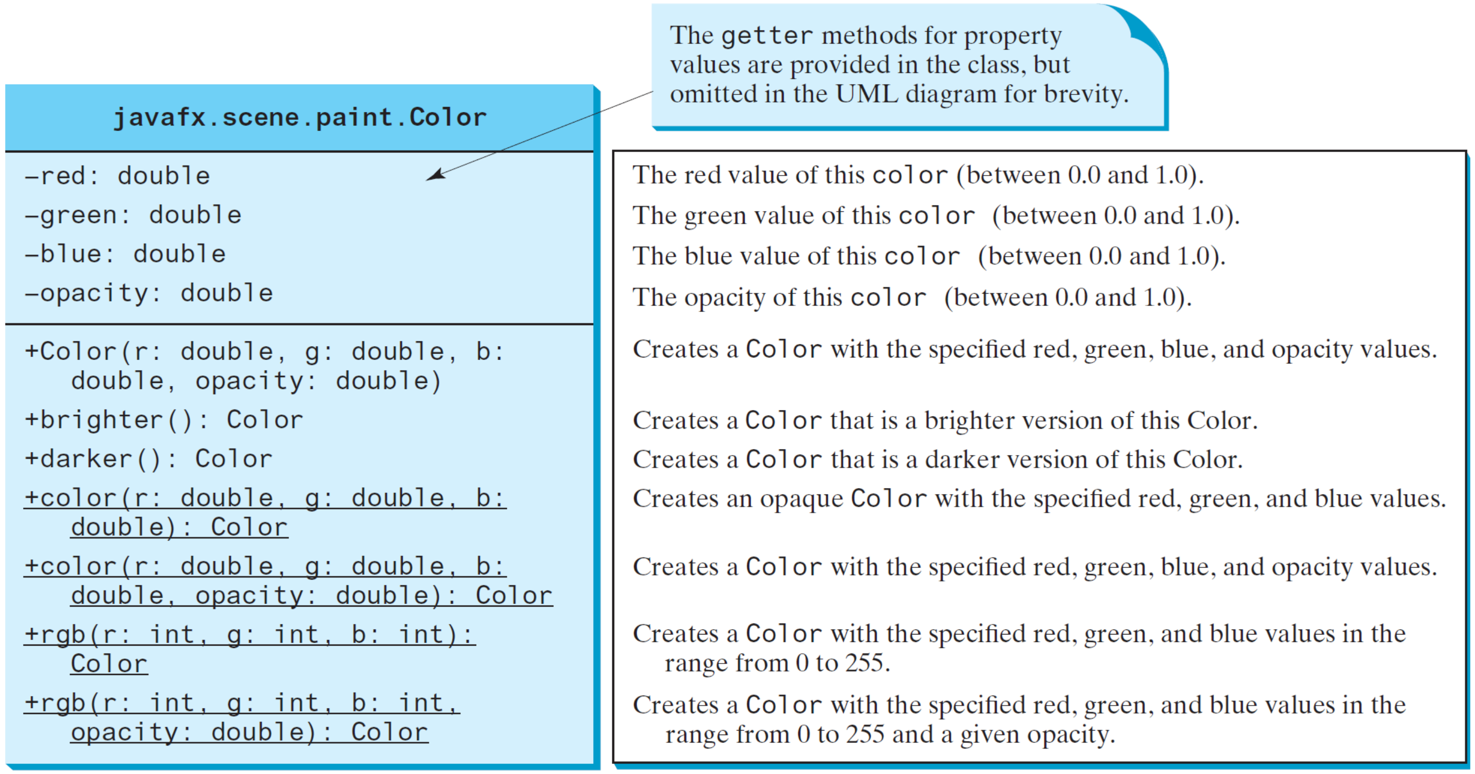
JavaFX CSS Reference Guide

Contents

Color Class

- The **Color** class can be used to create colors.
- JavaFX defines the abstract **Paint** class for painting a node. The **javafx.scene.paint.Color** is a concrete subclass of Paint, which is used to encapsulate colors,

  public Color(double r, double g, double b, double opacity);
  Color color = **new** Color(0.25, 0.14, 0.333, 0.51);



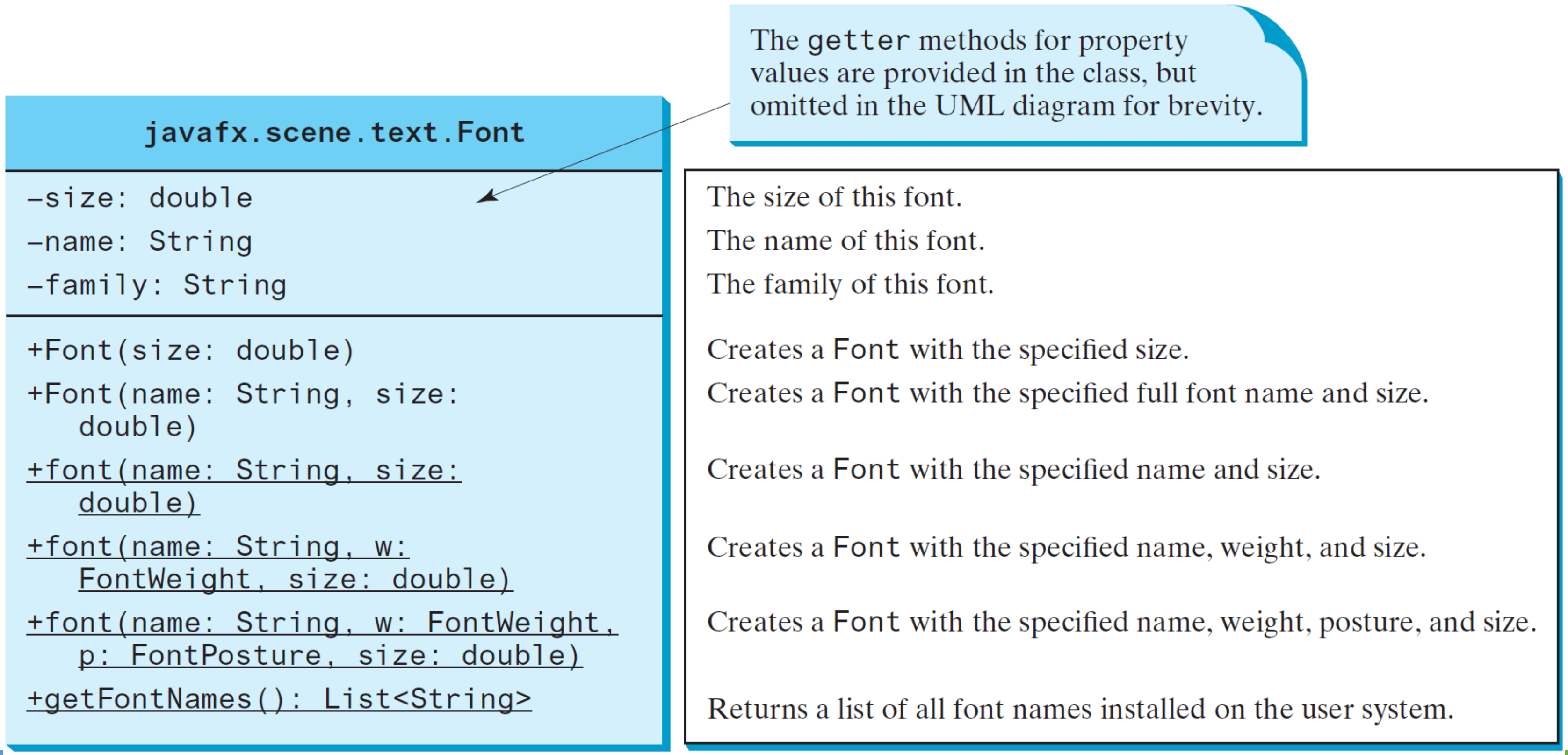
The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.paint.Color	
-red: double	The red value of this color (between 0.0 and 1.0).
-green: double	The green value of this color (between 0.0 and 1.0).
-blue: double	The blue value of this color (between 0.0 and 1.0).
-opacity: double	The opacity of this color (between 0.0 and 1.0).
+Color(r: double, g: double, b: double, opacity: double)	Creates a Color with the specified red, green, blue, and opacity values.
+brighter(): Color	Creates a Color that is a brighter version of this Color.
+darker(): Color	Creates a Color that is a darker version of this Color.
+color(r: double, g: double, b: double): Color	Creates an opaque Color with the specified red, green, and blue values.
+color(r: double, g: double, b: double, opacity: double): Color	Creates a Color with the specified red, green, blue, and opacity values.
+rgb(r: int, g: int, b: int): Color	Creates a Color with the specified red, green, and blue values in the range from 0 to 255.
+rgb(r: int, g: int, b: int, opacity: double): Color	Creates a Color with the specified red, green, and blue values in the range from 0 to 255 and a given opacity.

Font Class

- A **Font** describes font name, weight, and size.
- You can set fonts for rendering the text. The **javafx.scene.text.Font** class is used to create fonts.

```
Font font1 = new Font("SansSerif", 16);
Font font2 = Font.font("Times New Roman", FontWeight.BOLD,
FontPosture.ITALIC, 12);
```

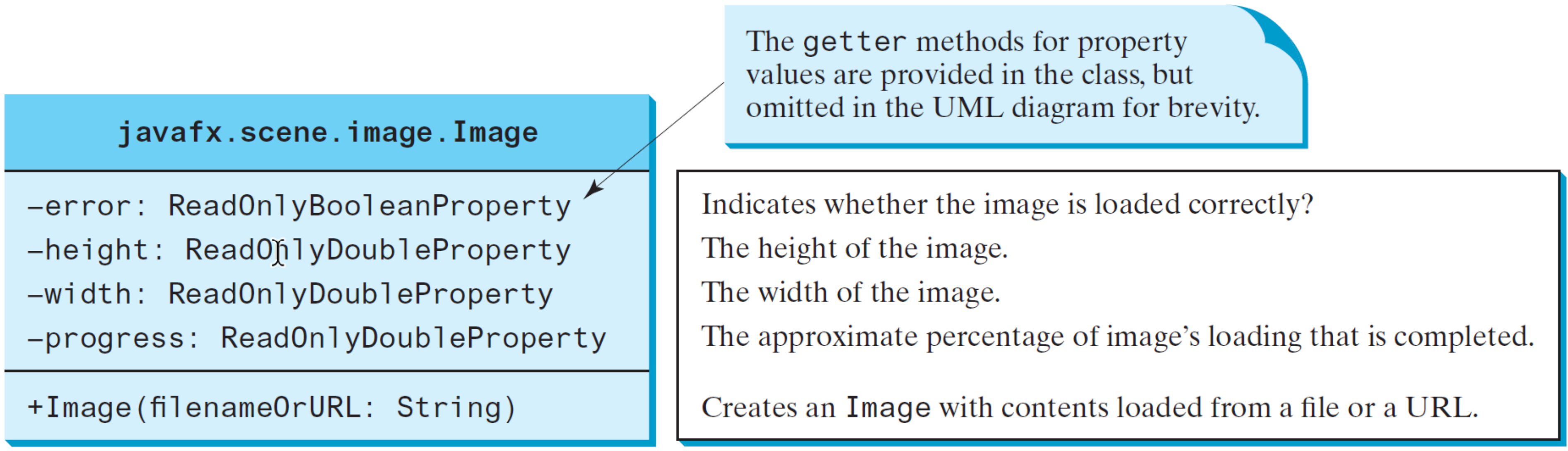


The `getter` methods for property values are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.text.Font	
-size: double	The size of this font.
-name: String	The name of this font.
-family: String	The family of this font.
+Font(size: double)	Creates a Font with the specified size.
+Font(name: String, size: double)	Creates a Font with the specified full font name and size.
+font(name: String, size: double)	Creates a Font with the specified name and size.
+font(name: String, w: FontWeight, size: double)	Creates a Font with the specified name, weight, and size.
+font(name: String, w: FontWeight, p: FontPosture, size: double)	Creates a Font with the specified name, weight, posture, and size.
+getFontNames(): List<String>	Returns a list of all font names installed on the user system.

The Image and ImageView Classes

- The **Image** class represents a graphical image, and the **ImageView** class can be used to display an image.
- The **javafx.scene.image.Image** class represents a graphical image and is used for loading an image from a specified filename or a URL


The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.

```
javafx.scene.image.Image

-error: ReadOnlyBooleanProperty
-height: ReadOnlyDoubleProperty
-width: ReadOnlyDoubleProperty
-progress: ReadOnlyDoubleProperty

+Image(filenameOrURL: String)
```

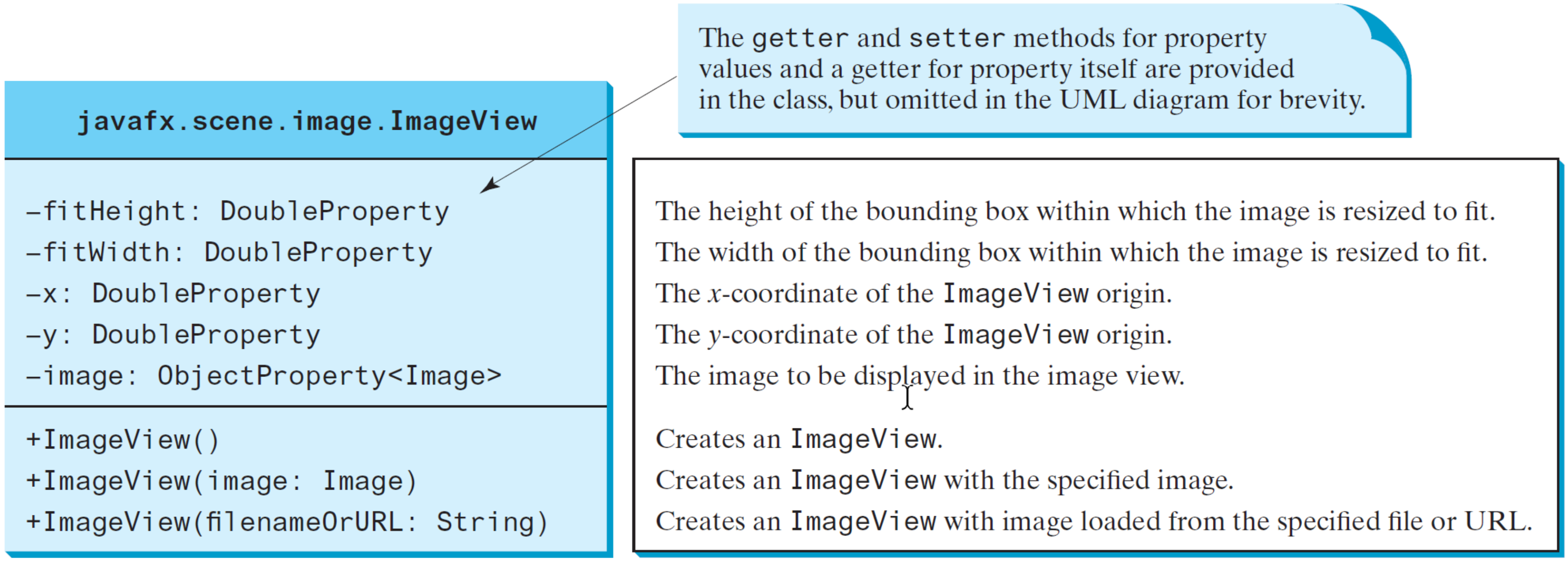
Indicates whether the image is loaded correctly?
The height of the image.
The width of the image.
The approximate percentage of image's loading that is completed.

Creates an Image with contents loaded from a file or a URL.

The Image and ImageView Classes

- The **javafx.scene.image.ImageView** is a node for displaying an image. An ImageView can be created from an Image object.

```
Image image = new Image("image/us.gif");
ImageView imageView = new ImageView(image);
```

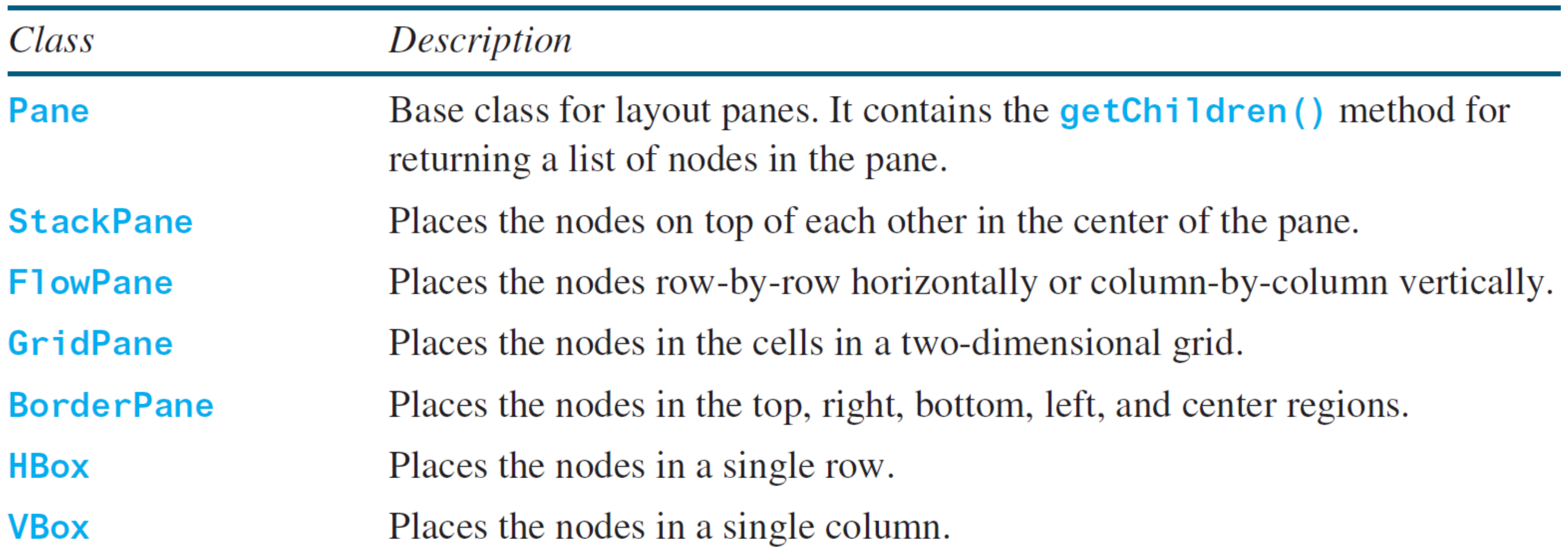


The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.image.ImageView	
-fitHeight: DoubleProperty	The height of the bounding box within which the image is resized to fit.
-fitWidth: DoubleProperty	The width of the bounding box within which the image is resized to fit.
-x: DoubleProperty	The *x*-coordinate of the ImageView origin.
-y: DoubleProperty	The *y*-coordinate of the ImageView origin.
-image: ObjectProperty<Image>	The image to be displayed in the image view.
+ImageView()	Creates an ImageView.
+ImageView(image: Image)	Creates an ImageView with the specified image.
+ImageView(filenameOrURL: String)	Creates an ImageView with image loaded from the specified file or URL.

Layout Panes and Groups

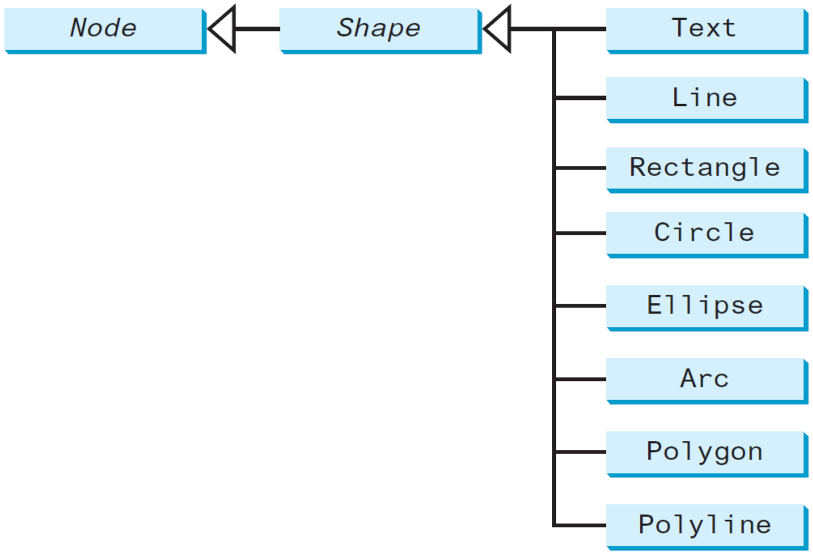
- JavaFX provides many types of **panes** for automatically laying out nodes in a desired location and size.
- Panes and groups are the containers for holding nodes. The **Group** class is often used to group nodes and to perform transformation and scale as a group. *Panes and UI control objects are resizable, but group, shape, and text objects are not resizable.* JavaFX provides many types of panes for organizing nodes in a container,



Class	*Description*
`Pane`	Base class for layout panes. It contains the `getChildren()` method for returning a list of nodes in the pane.
`StackPane`	Places the nodes on top of each other in the center of the pane.
`FlowPane`	Places the nodes row-by-row horizontally or column-by-column vertically.
`GridPane`	Places the nodes in the cells in a two-dimensional grid.
`BorderPane`	Places the nodes in the top, right, bottom, left, and center regions.
`HBox`	Places the nodes in a single row.
`VBox`	Places the nodes in a single column.

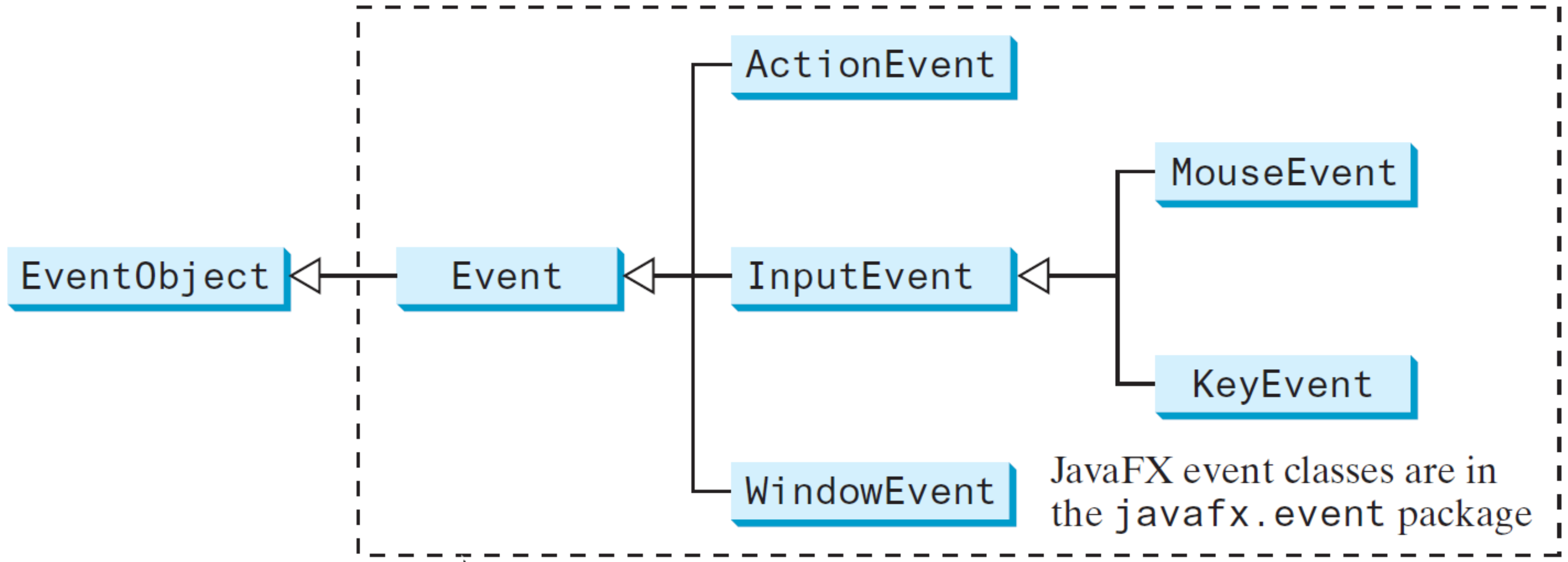
Shapes

- JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.
- The **Shape** class is the abstract base class that defines the common properties for all shapes. Among them are the *fill, stroke, and strokeWidth* properties.
  - ✓ The fill property specifies a color that fills the interior of a shape.
  - ✓ The stroke property specifies a color that is used to draw the outline of a shape.
  - ✓ The strokeWidth property specifies the width of the outline of a shape.
- The classes **Text, Line, Rectangle, Circle, Ellipse, Arc, Polygon, and Polyline** for drawing texts and simple shapes.
  All these are subclasses of Shape.



Events and Event Sources

- An event is an object created from an event source. Firing an event means to create an event and delegate the handler to handle the event. When you run a GUI program, the program interacts with the user and the events drive its execution. This is called event-driven programming. An event can be defined as a signal to the program that something has happened. Events are triggered by external user actions, such as mouse movements, mouse clicks, and keystrokes. The program can choose to respond to or ignore an event.
- An event in JavaFX is an object of the **javafx.event.Event** class.

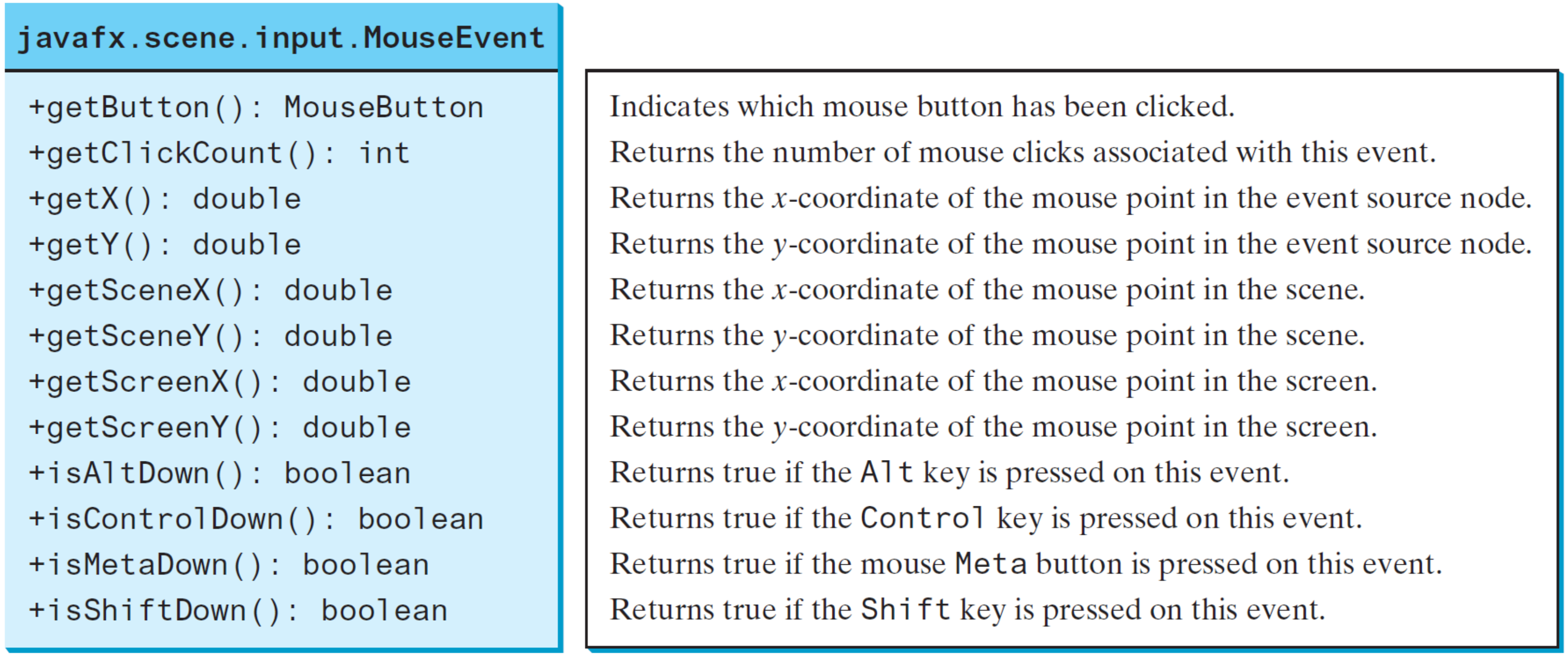


Registering Handlers and Handling Events

- A handler is an object that must be registered with an event source object and it must be an instance of an appropriate event-handling interface.
- Java uses a delegation-based model for event handling: A source object fires an event, and an object interested in the event handles it. The latter object is called an event handler or an event listener. For an object to be a handler for an event on a source object, two things are needed:
  - ✓ The handler object must be an instance of the corresponding event–handler interface to ensure the handler has the correct method for processing the event. JavaFX defines a unified handler interface EventHandler<T extends Event> for an event T. The handler interface contains the handle(T e) method for processing the event.
  - ✓ The handler object must be registered by the source object. Registration methods depend on the event type.

Mouse Events

- A **MouseEvent** is fired whenever a mouse button is pressed, released, clicked, moved, or dragged on a node or a scene.
- The MouseEvent object captures the event, such as the number of clicks associated with it, the location (the x- and y-coordinates) of the mouse, or which mouse button was pressed.
- The MouseEvent class encapsulates information for mouse events.



javafx.scene.input.MouseEvent	
+getButton(): MouseButton	Indicates which mouse button has been clicked.
+getClickCount(): int	Returns the number of mouse clicks associated with this event.
+getX(): double	Returns the *x*-coordinate of the mouse point in the event source node.
+getY(): double	Returns the *y*-coordinate of the mouse point in the event source node.
+getSceneX(): double	Returns the *x*-coordinate of the mouse point in the scene.
+getSceneY(): double	Returns the *y*-coordinate of the mouse point in the scene.
+getScreenX(): double	Returns the *x*-coordinate of the mouse point in the screen.
+getScreenY(): double	Returns the *y*-coordinate of the mouse point in the screen.
+isAltDown(): boolean	Returns true if the `Alt` key is pressed on this event.
+isControlDown(): boolean	Returns true if the `Control` key is pressed on this event.
+isMetaDown(): boolean	Returns true if the mouse `Meta` button is pressed on this event.
+isShiftDown(): boolean	Returns true if the `Shift` key is pressed on this event.

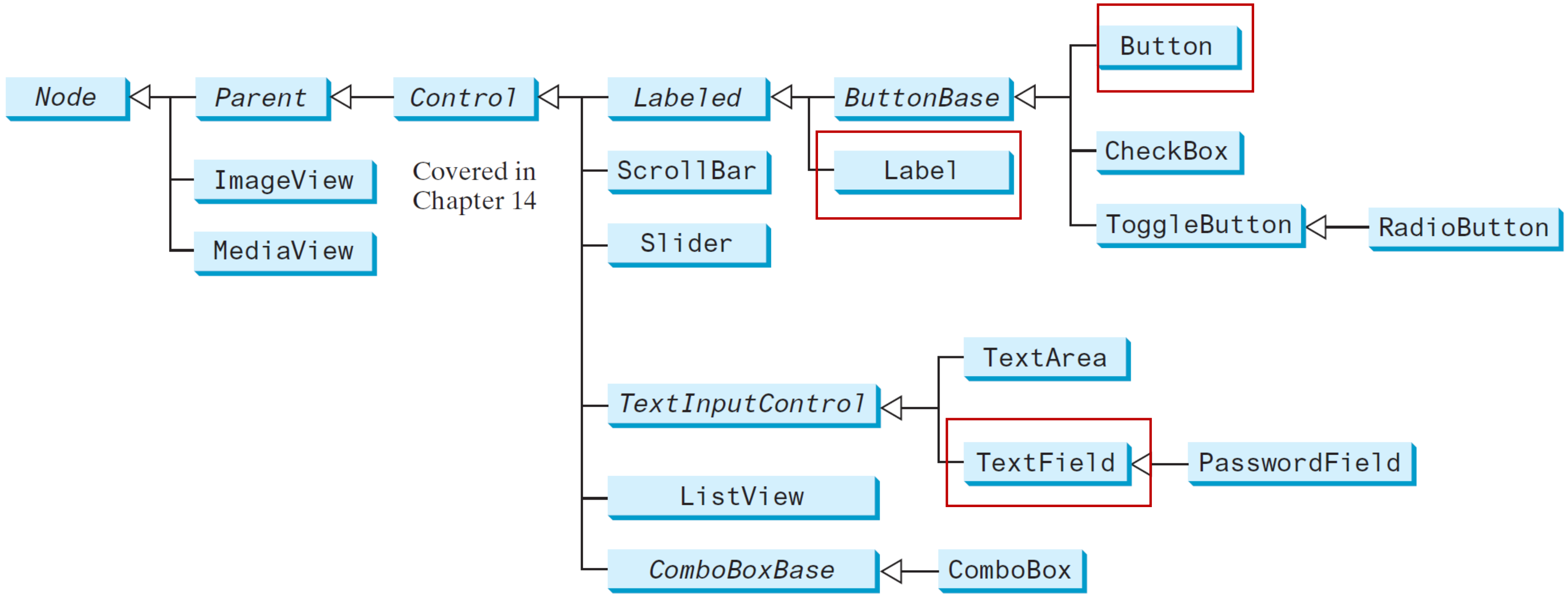
Key Events

- A **KeyEvent** is fired whenever a key is pressed, released, or typed on a node or a scene.
- Key events enable the use of the keys to control and perform actions, or get input from the keyboard. The KeyEvent object describes the nature of the event (namely, that a key has been pressed, released, or typed) and the value of the key.
- The KeyEvent class encapsulates information for keyboard events.

`javafx.scene.input.KeyEvent`	
`+getCharacter(): String`	Returns the character associated with the key in this event.
`+getCode(): KeyCode`	Returns the key code associated with the key in this event.
`+getText(): String`	Returns a string describing the key code.
`+isAltDown(): boolean`	Returns true if the `Alt` key is pressed on this event.
`+isControlDown(): boolean`	Returns true if the `Control` key is pressed on this event.
`+isMetaDown(): boolean`	Returns true if the mouse `Meta` button is pressed on this event.
`+isShiftDown(): boolean`	Returns true if the `Shift` key is pressed on this event.

JavaFX UI Controls

- JavaFX provides many UI controls for developing a comprehensive user interface.
- A graphical user interface (GUI) makes a program user-friendly and easy to use. Creating a GUI requires creativity and knowledge of how UI controls work. Since the UI controls in JavaFX are very flexible and versatile, you can create a wide assortment of useful user interfaces for rich GUI applications.


Node
Parent
Control
Labeled
ButtonBase
Button
CheckBox
ToggleButton
RadioButton
Label
ImageView
MediaView
Covered in Chapter 14
ScrollBar
Slider
TextInputControl
TextArea
TextField
PasswordField
ListView
ComboBoxBase
ComboBox