Lecture 14

Java SE – Multithreading 3 + Reactive Streams + Sockets applied in Java FX RIA GUI

presentation

Java Programming – Software App Development

Cristian Toma

D.I.C.E/D.E.I.C – Department of Economic Informatics & Cybernetics
www.dice.ase.ro

Cristian Toma – Business Card

Agenda for Lecture 13 – JSE GUI

java.util.concurrent - Executor & ExecutorService, Callable, Future

Advanced Multi-Threading

1. Summary of Multi-threading in C vs. Java

Multi-threading vs. Multi-process "quite good/imperfect" analogies & terms:

USE MULTI-THREADING for

- PARALLELISM (e.g. adding 2 matrixes in parallel) and/or;
- CONCURENCY (e.g. handling GUI on one thread & business processing on other thread / even on mono-processor PC, handling multiple HTTP requests from the network / cooperation on the same data structure between at least 2 threads – producer and consumer)

Mutexes are used to prevent data inconsistencies due to ***race conditions***.
A ***race condition*** often occurs when two or more threads need to perform operations on the same memory area, but the results of computations depends on the order in which these operations are performed.
Mutexes are used for serializing shared resources. Anytime a global resource is accessed by more than one thread the resource should have a Mutex associated with it.
One can apply a mutex to protect a segment of memory ("***critical region***") from other threads.
Mutexes can be applied only to threads in a single process and do not work between processes as do semaphores in Linux IPC.

In Java Mutex is quite ⇔ synchronized (also please see java.util.concurrent.*)

GUI Event Dispatcher Thread

GUI event handling and painting code executes in a single thread, called the *event dispatcher thread*. This ensures that each event handler finishes executing before the next one executes and the painting isn't interrupted by events.

GUI Event Dispatcher Thread

invokeLater and invokeAndWait

In certain situations, you need to run the code in the event dispatcher thread to avoid possible deadlock. You can use the static methods, invokeLater and invokeAndWait, in the javax.swing.SwingUtilities class to run the code in the event dispatcher thread. You must put this code in the run method of a Runnable object and specify the Runnable object as the argument to invokeLater and invokeAndWait. The invokeLater method returns immediately, without waiting for the event dispatcher thread to execute the code. The invokeAndWait method is just like invokeLater, except that invokeAndWait doesn't return until the event-dispatching thread has executed the specified code.

GUI Event Dispatcher Thread

Launch Application from Main Method

So far, you have launched your GUI application from the main method by creating a frame and making it visible. This works fine for most applications. In certain situations, however, it could cause problems. To avoid possible thread deadlock, you should launch GUI creation from the event dispatcher thread as follows:

```
public static void main(String[] args) {
  SwingUtilities.invokeLater(new Runnable() {
    public void run() {
      // Place the code for creating a frame and setting it properties
    }
  });
}
```

Section Conclusion

Fact: **Multi-threading vs. Multi-process**

In few **samples** it is easy to understand:

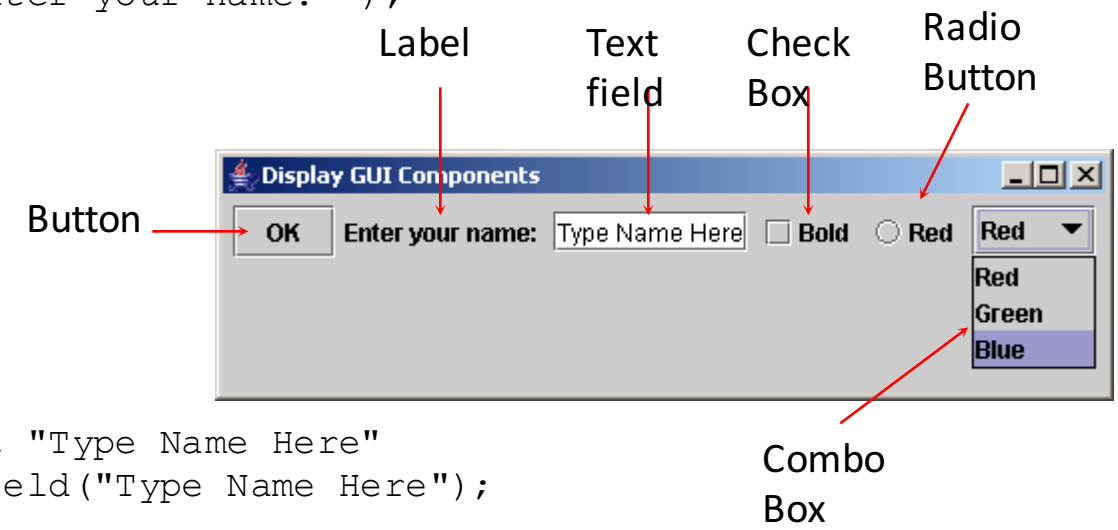
- Necessity: Parallelism vs. Concurrency
- Multi-Process vs. Multi-Threading
- Atomic operations (counter++)
- Multi-threading model mapping

GUI: Swing, Inner Class, Controls, Event – OPTIONAL | Java FX - Mandatory

GUI Intro

Creating GUI Objects

```
// Create a button with text OK
JButton jbtOK = new JButton("OK");

// Create a label with text "Enter your name: "
JLabel jlblName = new JLabel("Enter your name: ");
```



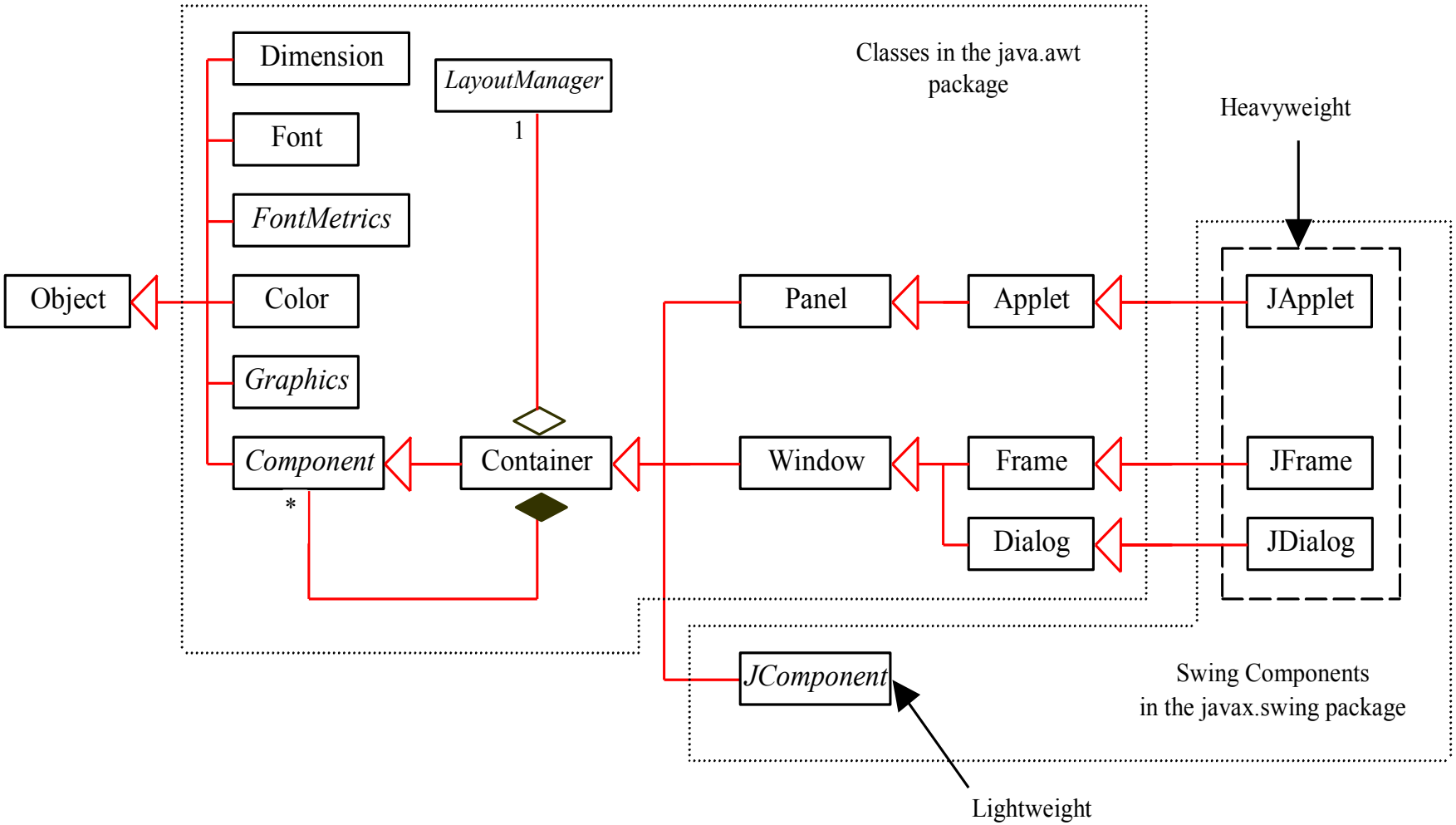

```
// Create a text field with text "Type Name Here"
JTextField jtfName = new JTextField("Type Name Here");

// Create a check box with text bold
JCheckBox jchkBold = new JCheckBox("Bold");

// Create a radio button with text red
JRadioButton jrbRed = new JRadioButton("Red");

// Create a combo box with choices red, green, and blue
JComboBox jcboColor = new JComboBox(new String[]{"Red",
  "Green", "Blue"});
```

2. Java GUI – Swing vs. AWT

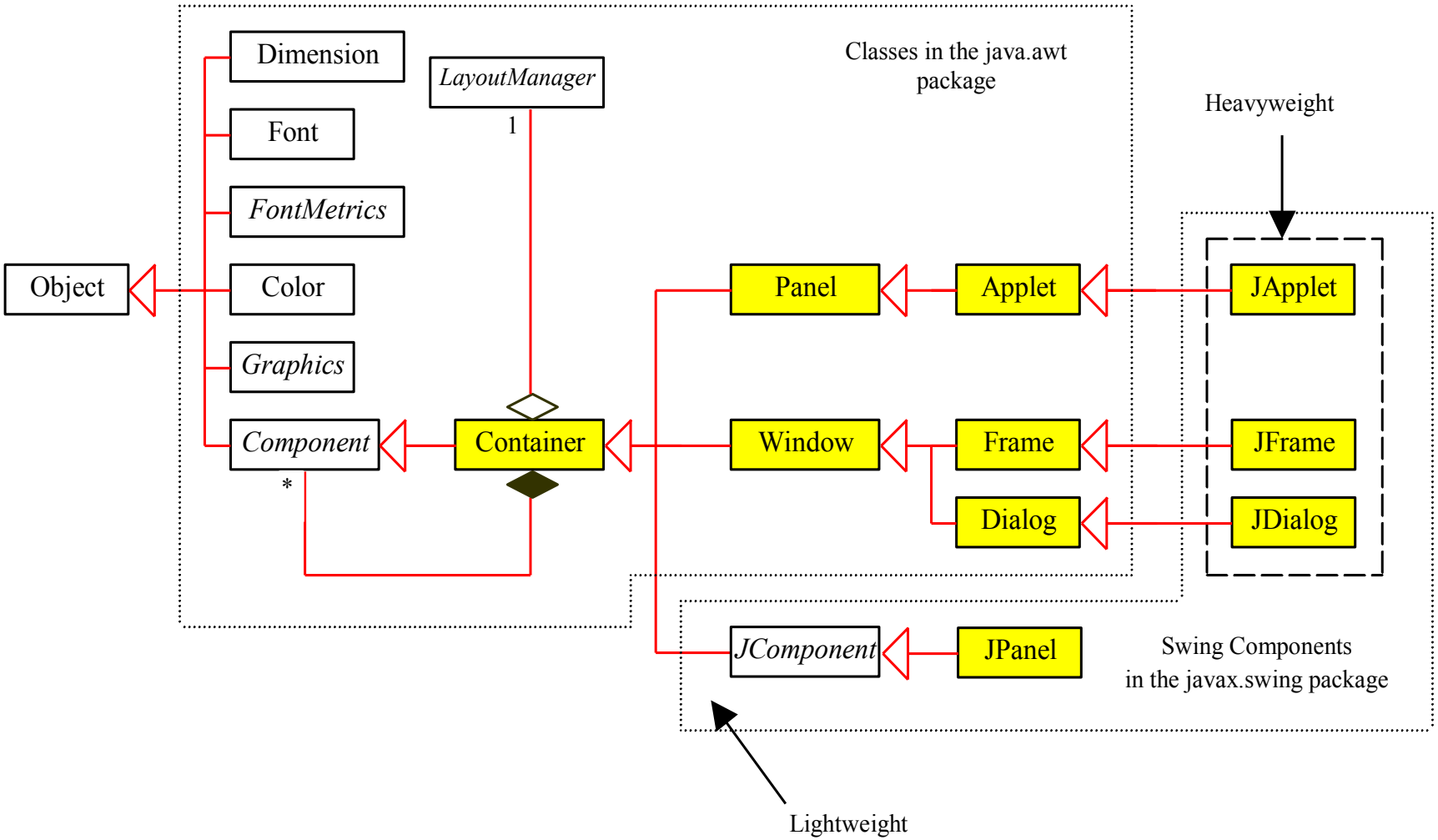
So why do the GUI component classes have a prefix *J*? Instead of JButton, why not name it simply Button? In fact, there is a class already named Button in the java.awt package.

When Java was introduced, the GUI classes were bundled in a library known as the Abstract Windows Toolkit (AWT). For every platform on which Java runs, the AWT components are automatically mapped to the platform-specific components through their respective agents, known as *peers*. AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects. Besides, AWT is prone to platform-specific bugs because its peer-based approach relies heavily on the underlying platform. With the release of Java 2, the AWT user-interface components were replaced by a more robust, versatile, and flexible library known as *Swing components*. Swing components are painted directly on canvases using Java code, except for components that are subclasses of java.awt.Window or java.awt.Panel, which must be drawn using native GUI on a specific platform. Swing components are less dependent on the target platform and use less of the native GUI resource. For this reason, Swing components that don't rely on native GUI are referred to as *lightweight components*, and AWT components are referred to as *heavyweight components*.

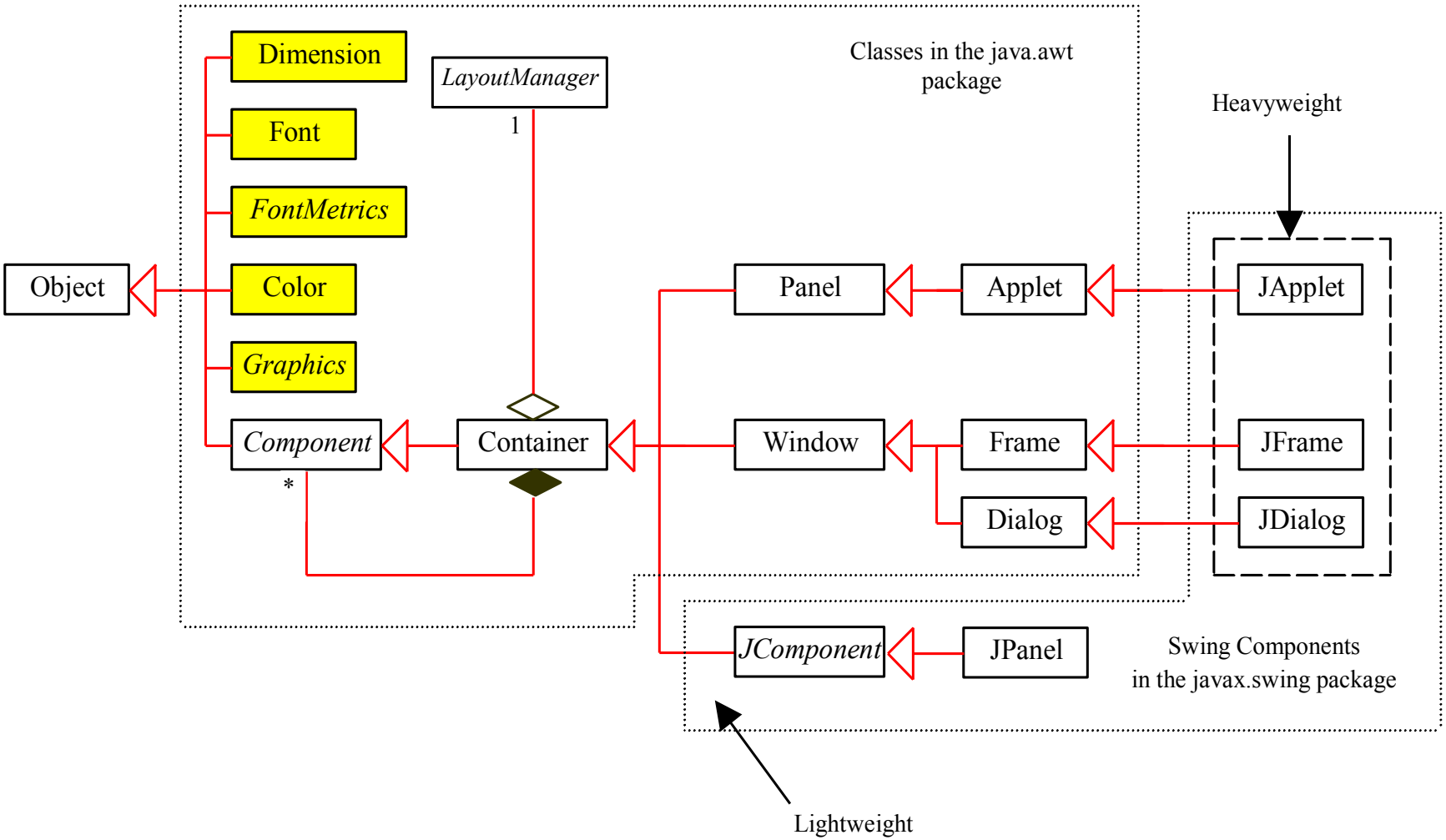
2. Java GUI – Class Hierarchy Swing



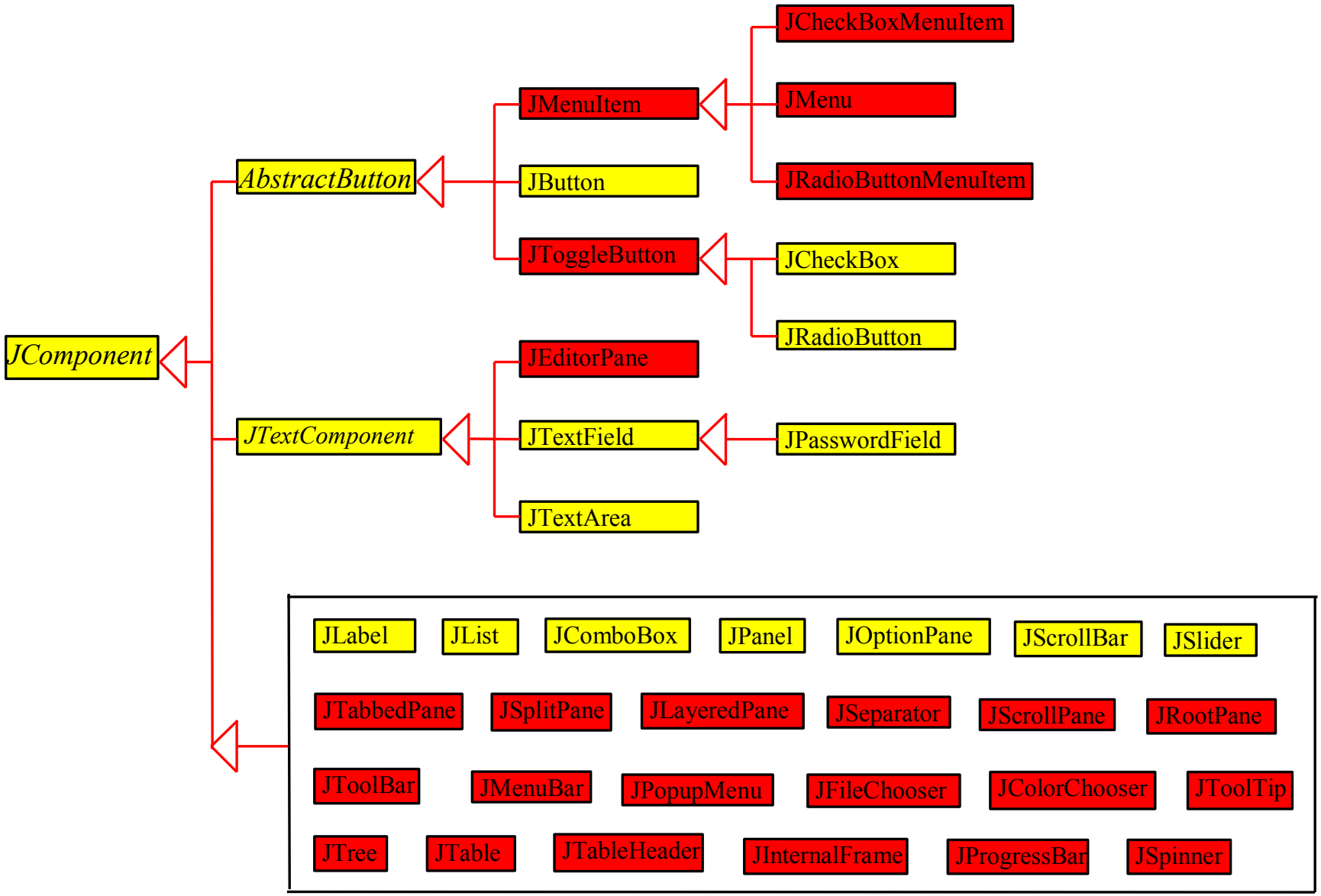
2. Java GUI – Class Hierarchy Swing (Container Classes)



2. Java GUI – Class Hierarchy Swing (Helper Classes)



2. Java GUI – Class Hierarchy Swing (GUI Components)



Frames

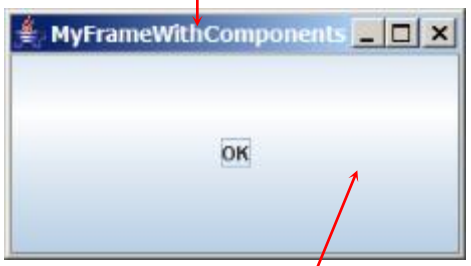
- Frame is a window that is not contained inside another window. Frame is the basis to contain other user interface components in Java GUI applications.
- The JFrame class can be used to create windows.
- For Swing GUI programs, use JFrame class to create widows.

Creating Frames

```
import javax.swing.*;
public class MyFrame {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Test Frame");
    frame.setSize(400, 300);
    frame.setVisible(true);
    frame.setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);
  }
}
```

Adding Components in a Frame - Content Pane Delegation in JDK 5

Title bar



Content pane

```
// Add a button into the frame
frame.getContentPane().add(
  new JButton("OK"));
```

```
// Add a button into the frame
frame.add(
  new JButton("OK"));
```

2. Java GUI – Swing JFrame class

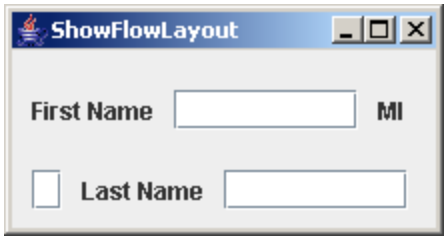
javax.swing.JFrame	
+JFrame()	Creates a default frame with no title.
+JFrame(title: String)	Creates a frame with the specified title.
+setSize(width: int, height: int): void	Specifies the size of the frame.
+setLocation(x: int, y: int): void	Specifies the upper-left corner location of the frame.
+setVisible(visible: boolean): void	Sets true to display the frame.
+setDefaultCloseOperation(mode: int): void	Specifies the operation when the frame is closed.
+setLocationRelativeTo(c: Component): void	Sets the location of the frame relative to the specified component. If the component is null, the frame is centered on the screen.
+pack(): void	Automatically sets the frame size to hold the components in the frame.

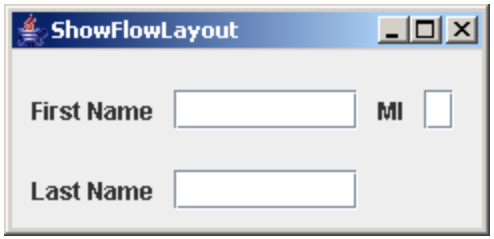
Layout Managers

- Java's layout managers provide a level of abstraction to automatically map your user interface on all window systems.
- The UI components are placed in containers. Each container has a layout manager to arrange the UI components within the container.
- Layout managers are set in containers using the setLayout(LayoutManager) method in a container.

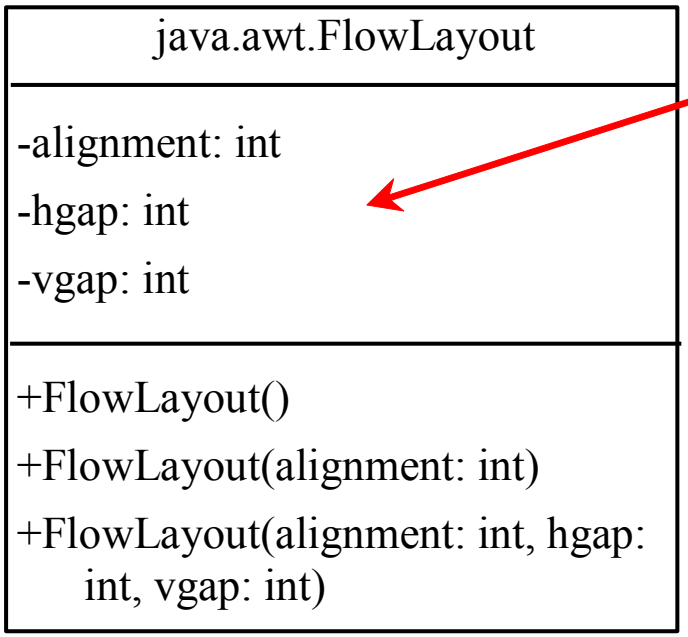
`FlowLayout` Example

Write a program that adds three labels and text fields into the content pane of a frame with a FlowLayout manager.





The FlowLayout Class



The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The alignment of this layout manager (default: CENTER).

The horizontal gap of this layout manager (default: 5 pixels).

The vertical gap of this layout manager (default: 5 pixels).

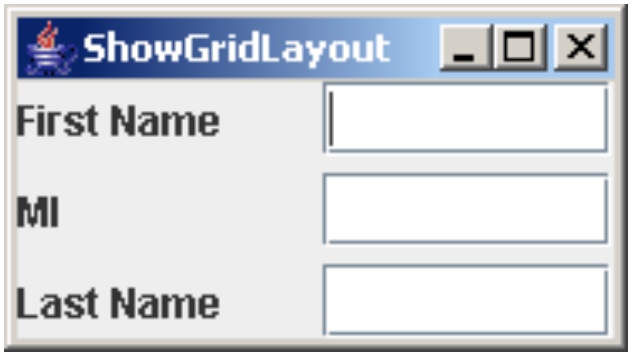
Creates a default FlowLayout manager.

Creates a FlowLayout manager with a specified alignment.

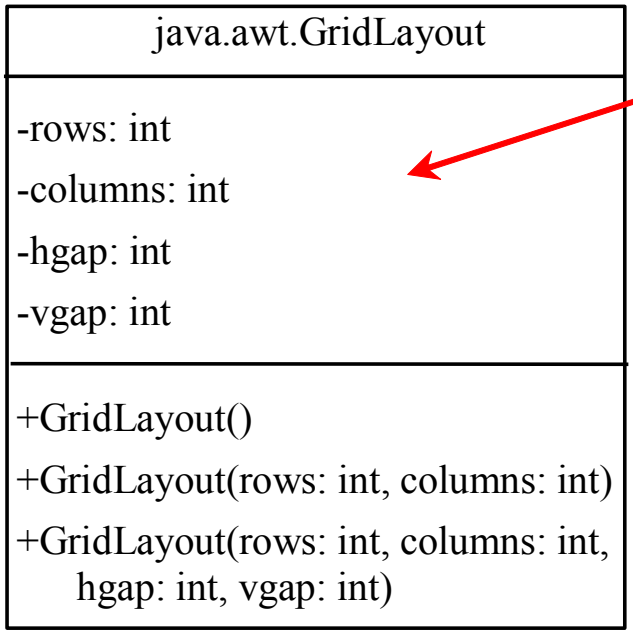
Creates a FlowLayout manager with a specified alignment, horizontal gap, and vertical gap.

`GridLayout` Example

Rewrite the program in the preceding example using a GridLayout manager instead of a FlowLayout manager to display the labels and text fields.



The `GridLayout` Class



The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The number of rows in this layout manager (default: 1).

The number of columns in this layout manager (default: 1).

The horizontal gap of this layout manager (default: 0).

The vertical gap of this layout manager (default: 0).

Creates a default GridLayout manager.

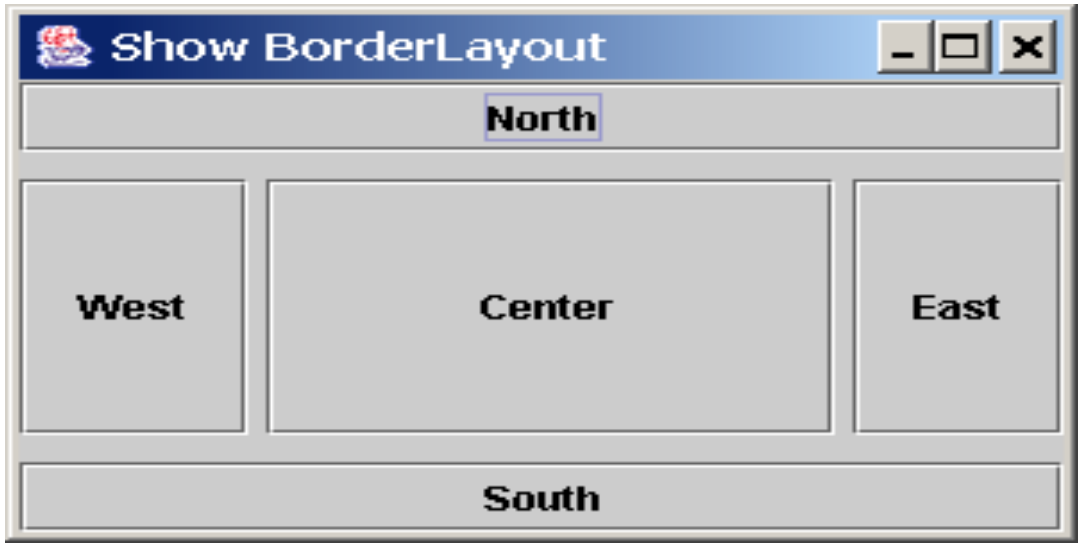
Creates a GridLayout with a specified number of rows and columns.

Creates a GridLayout manager with a specified number of rows and columns, horizontal gap, and vertical gap.

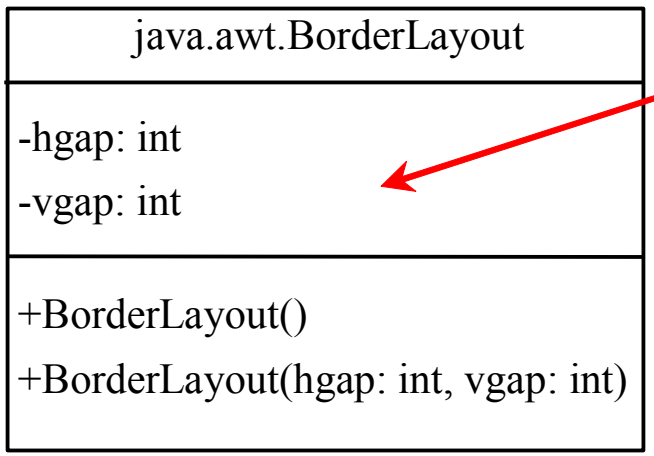
The `BorderLayout` Manager

The `BorderLayout` manager divides the container into five areas: East, South, West, North, and Center. Components are added to a `BorderLayout` by using the add method.

`add(Component, constraint)`, where `constraint` is `BorderLayout.EAST`, `BorderLayout.SOUTH`, `BorderLayout.WEST`, `BorderLayout.NORTH`, or `BorderLayout.CENTER`.



The `BorderLayout` Class



The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The horizontal gap of this layout manager (default: 0).

The vertical gap of this layout manager (default: 0).

Creates a default BorderLayout manager.

Creates a BorderLayout manager with a specified number of horizontal gap, and vertical gap.

Using Panels as Sub-Containers

- Panels act as sub-containers for grouping user interface components.
- It is recommended that you place the user interface components in panels and place the panels in a frame. You can also place panels in a panel.
- To add a component to JFrame, you actually add it to the content pane of JFrame. To add a component to a panel, you add it directly to the panel using the add method.

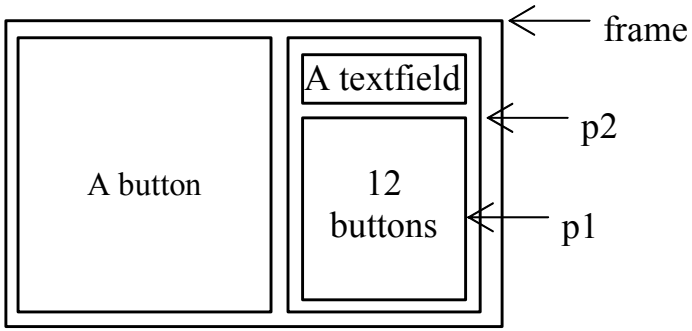
Creating a JPanel

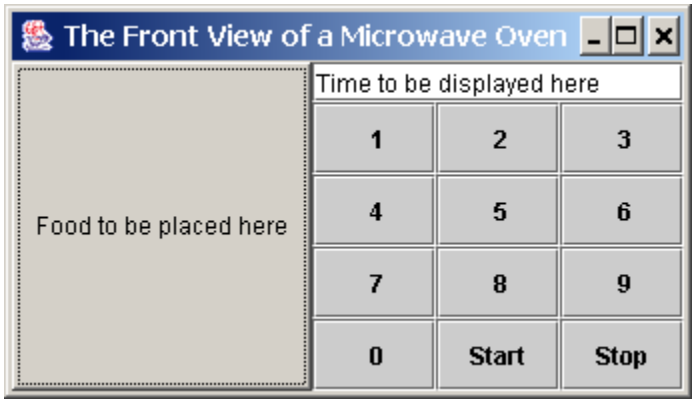
You can use new JPanel() to create a panel with a default FlowLayout manager or new JPanel(LayoutManager) to create a panel with the specified layout manager. Use the add(Component) method to add a component to the panel. For example,

```
JPanel p = new JPanel();
p.add(new JButton("OK"));
```

Testing Panels Example

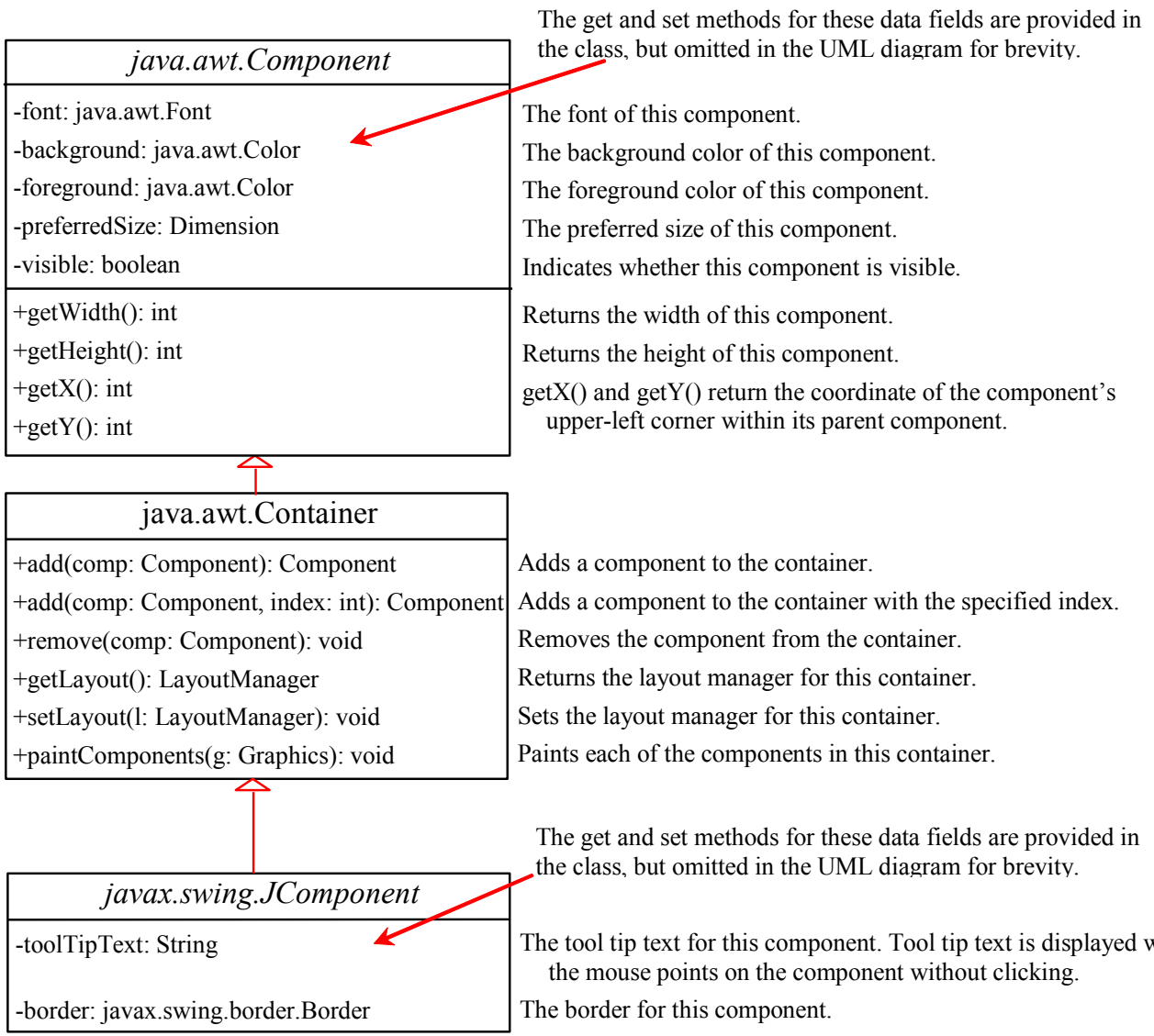
This example uses panels to organize components. The program creates a user interface for a Microwave oven.





Google Windows Builder Pro for Eclipse:
http://download.eclipse.org/windowbuilder/WB/release/R201309271200/3.7/
http://eclipse.org/downloads/download.php?file=/windowbuilder/WB/release/R201309271200/WB_v1.6.1_UpdateSite_for_Eclipse3.7.zip

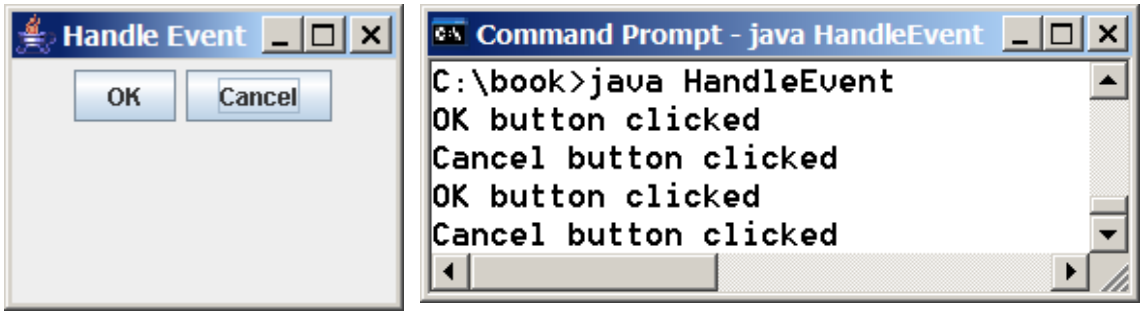
2. Java GUI – Swing Common Features



Handling GUI Events

Source object (e.g., button)

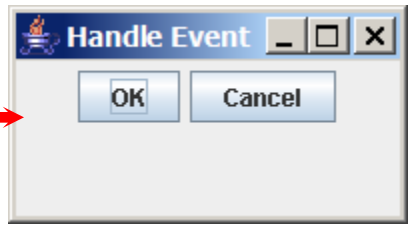
Listener object contains a method for processing the event.



2. Java GUI – Event Handling – Trace Execution

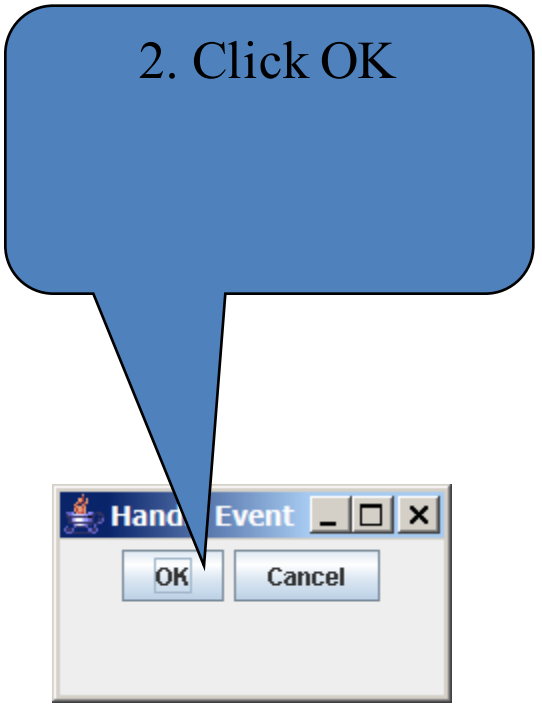
```
public class HandleEvent extends JFrame {
  public HandleEvent() {
    …
   OKListenerClass listener1 = new OKListenerClass();
    jbtOK.addActionListener(listener1);
    …
  }

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

class OKListenerClass implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.out.println("OK button clicked");
  }
}
```

1. Start from the main method to create a window and display it



2. Java GUI – Event Handling – Trace Execution

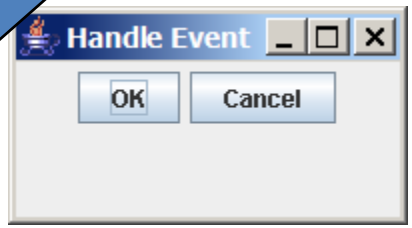
```
public class HandleEvent extends JFrame {
  public HandleEvent() {
    …
   OKListenerClass listener1 = new OKListenerClass();
    jbtOK.addActionListener(listener1);
    …
  }

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

class OKListenerClass implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.out.println("OK button clicked");
  }
}
```

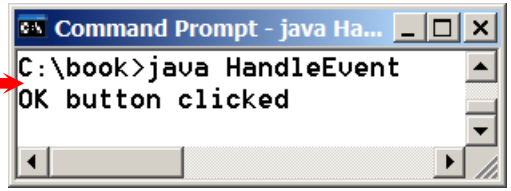


2. Java GUI – Event Handling – Trace Execution

```
public class HandleEvent extends JFrame {
  public HandleEvent() {
    …
   OKListenerClass listener1 = new OKListenerClass();
    jbtOK.addActionListener(listener1);
    …
  }

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

class OKListenerClass implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.out.println("OK button clicked");
  }
}
```

3. Click OK. The JVM invokes the listener's actionPerformed method



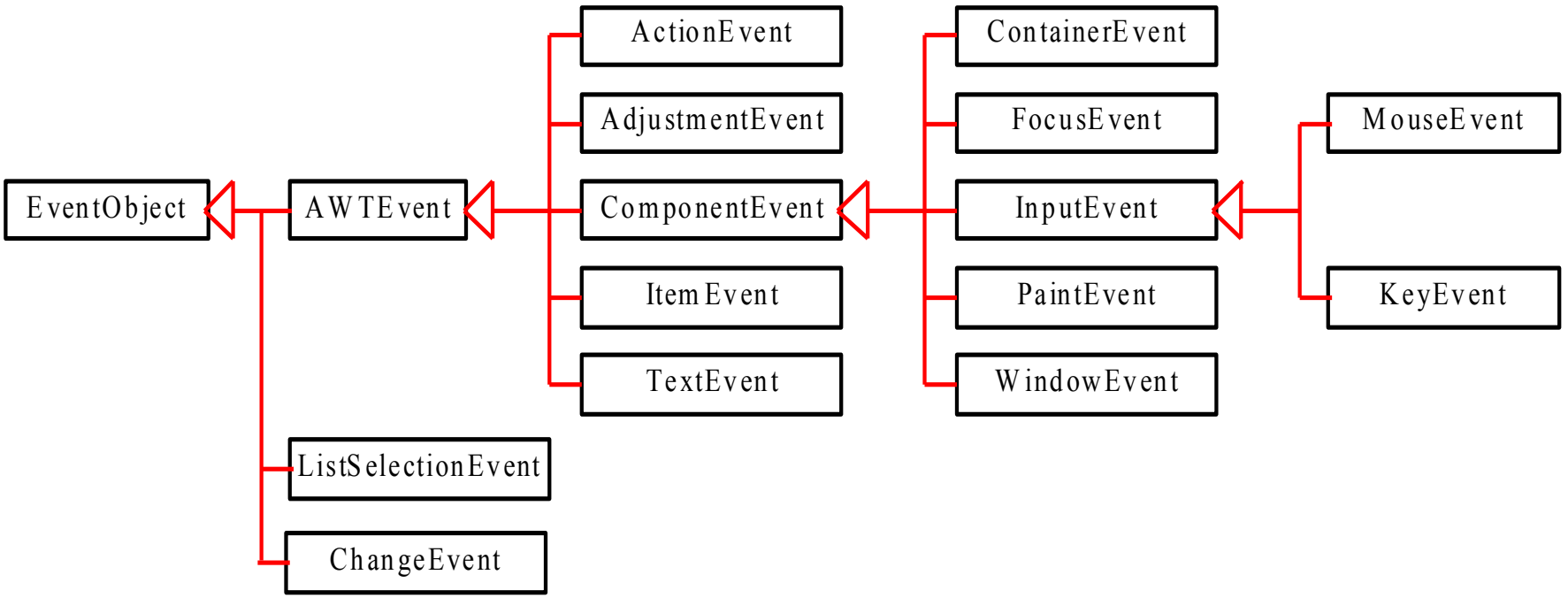


Events

- An *event* can be defined as a type of signal to the program that something has happened.

- The event is generated by external user actions such as mouse movements, mouse clicks, and keystrokes, or by the operating system, such as a timer.

Event Classes



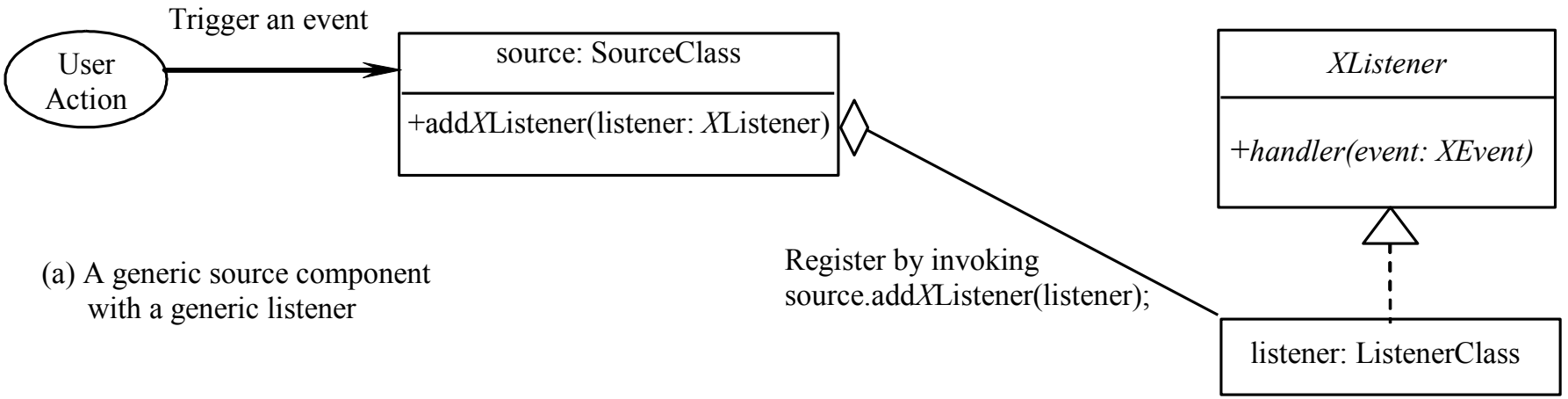
Event Information

An event object contains whatever properties are pertinent to the event. You can identify the source object of the event using the getSource() instance method in the EventObject class. The subclasses of EventObject deal with special types of events, such as button actions, window events, component events, mouse movements, and keystrokes. Table 15.1 lists external user actions, source objects, and event types generated.

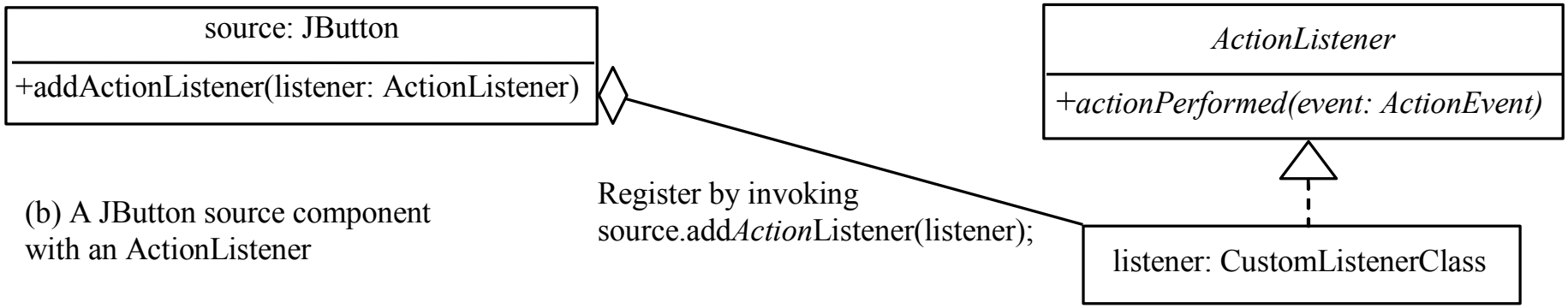
Selected User Actions

User Action	Source Object	Event Type Generated
Click a button	`JButton`	`ActionEvent`
Click a check box	`JCheckBox`	`ItemEvent,ActionEvent`
Click a radio button	`JRadioButton`	`ItemEvent,ActionEvent`
Press return on a text field	`JTextField`	`ActionEvent`
Select a new item	`JComboBox`	`ItemEvent,ActionEvent`
Window opened, closed, etc.	`Window`	`WindowEvent`
Mouse pressed, released, etc.	`Component`	`MouseEvent`
Key released, pressed, etc.	`Component`	`KeyEvent`

The Delegation Model

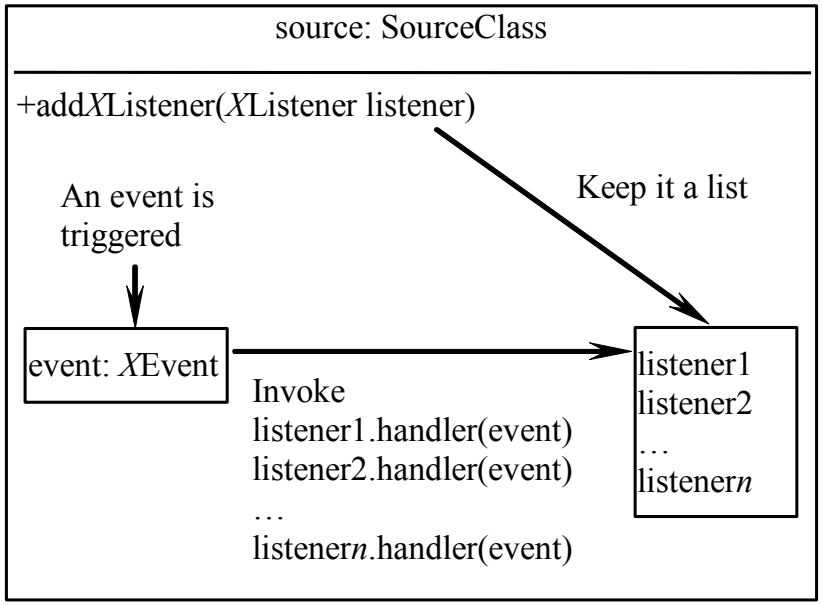


(a) A generic source component with a generic listener

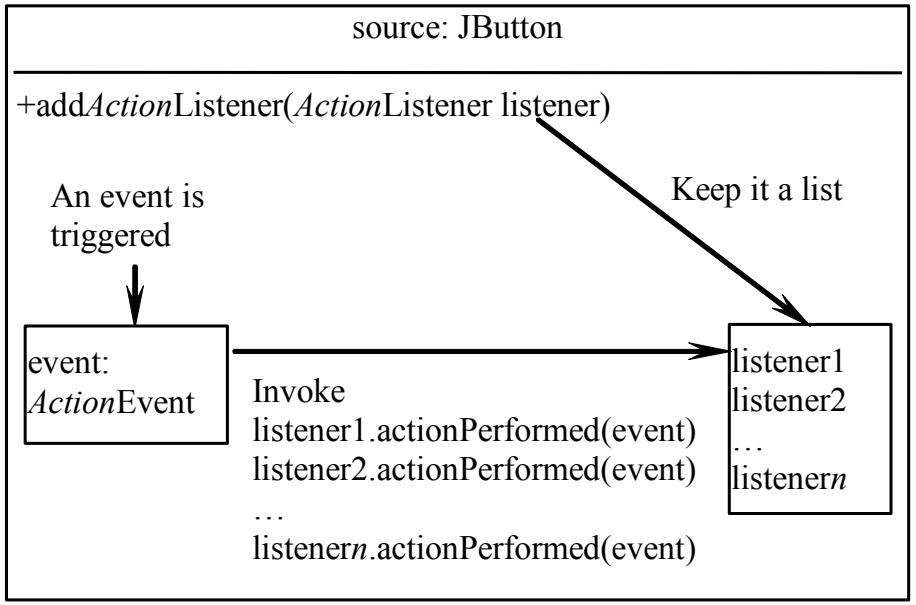


(b) A JButton source component with an ActionListener

Internal Function of a Source Component



(a) Internal function of a generic source object



(b) Internal function of a JButton object

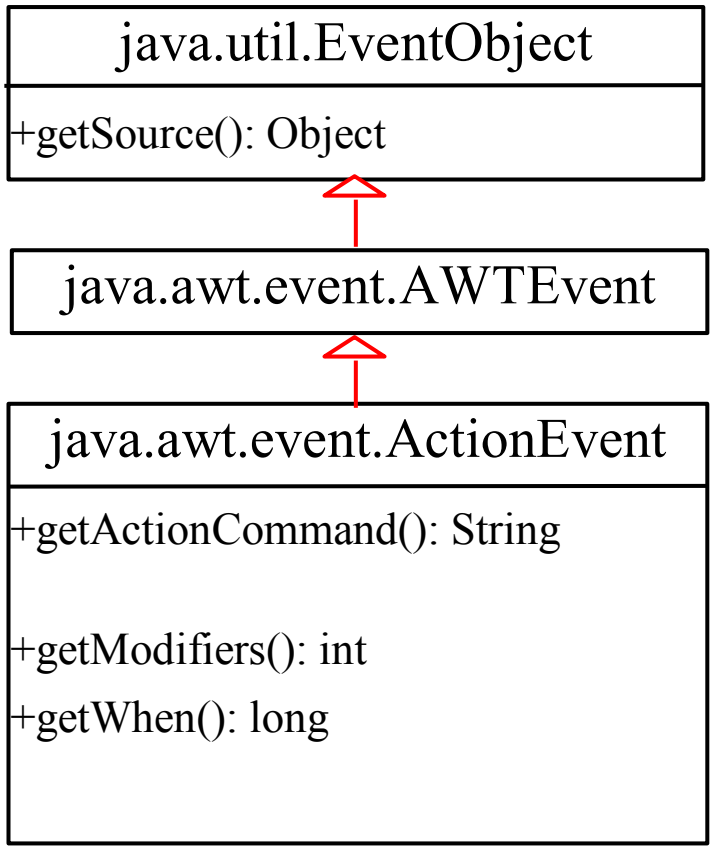
The Delegation Model: Example

```
JButton jbt = new JButton("OK");
ActionListener listener = new OKListener();
jbt.addActionListener(listener);
```

Selected Event Handlers

Event Class	Listener Interface	Listener Methods (Handlers)
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
WindowEvent	WindowListener	windowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
ContainerEvent	ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
MouseEvent	MouseListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseClicked(MouseEvent) mouseExited(MouseEvent) mouseEntered(MouseEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTypeed(KeyEvent)

java.awt.event.ActionEvent



Returns the object on which the event initially occurred.

Returns the command string associated with this action. For a button, its text is the command string.

Returns the modifier keys held down during this action event.

Returns the timestamp when this event occurred. The time is the number of milliseconds since January 1, 1970, 00:00:00 GMT.

Inner Class Listeners

A listener class is designed specifically to create a listener object for a GUI component (e.g., a button). It will not be shared by other applications. So, it is appropriate to define the listener class inside the frame class as an inner class.

Inner Classes

Inner class: A class is a member of another class.

Advantages: In some applications, you can use an inner class to make programs simple.

- An inner class can reference the data and methods defined in the outer class in which it nests, so you do not need to pass the reference of the outer class to the constructor of the inner class.

Inner Classes, cont.

```
public class Test {
  ...
}

public class A {
  ...
}
```

(a)

```
public class Test {
  ...

  // Inner class
  public class A {
    ...
  }
}
```

(b)

```
// OuterClass.java: inner class demo
public class OuterClass {
  private int data;

  /** A method in the outer class */
  public void m() {
    // Do something
  }

  // An inner class
  class InnerClass {
    /** A method in the inner class */
    public void mi() {
      // Directly reference data and method
      // defined in its outer class
      data++;
      m();
    }
  }
}
```

(c)

Inner Classes (cont.)

- Inner classes can make programs simple and concise.
- An inner class supports the work of its containing outer class and is compiled into a class named *OuterClassName*$*InnerClassName*.class. For example, the inner class InnerClass in OuterClass is compiled into *OuterClass*$*InnerClass*.class.

Inner Classes (cont.)

- An inner class can be declared public, protected, or private subject to the same visibility rules applied to a member of the class.
- An inner class can be declared static. A static inner class can be accessed using the outer class name. A static inner class cannot access nonstatic members of the outer class

Anonymous Inner Classes

- An anonymous inner class must always extend a superclass or implement an interface, but it cannot have an explicit extends or implements clause.
- An anonymous inner class must implement all the abstract methods in the superclass or in the interface.
- An anonymous inner class always uses the no-arg constructor from its superclass to create an instance. If an anonymous inner class implements an interface, the constructor is Object().
- An anonymous inner class is compiled into a class named OuterClassName$*n*.class. For example, if the outer class Test has two anonymous inner classes, these two classes are compiled into Test$1.class and Test$2.class.

Anonymous Inner Classes (cont.)

Inner class listeners can be shortened using anonymous inner classes. An *anonymous inner class* is an inner class without a name. It combines declaring an inner class and creating an instance of the class in one step. An anonymous inner class is declared as follows (AnonymousListnerDemo.java):

```
new SuperClassName/InterfaceName() {
  // Implement or override methods in superclass or interface
  // Other methods if necessary
}
```

Java FX Prerequisites

- Latest Java JDK 8 (includes JavaFX 8): http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- Scene Builder 2.0 or greater: http://www.oracle.com/technetwork/java/javase/downloads/sb2download-2177776.html
- Eclipse 4.3 or greater with e(fx)clipse plugin. The easiest way is to download the preconfigured distro from the e(fx)clipse website. As an alternative you can use an update site for your Eclipse installation: http://www.eclipse.org/efxclipse/install.html

Java FX Eclipse Configurations

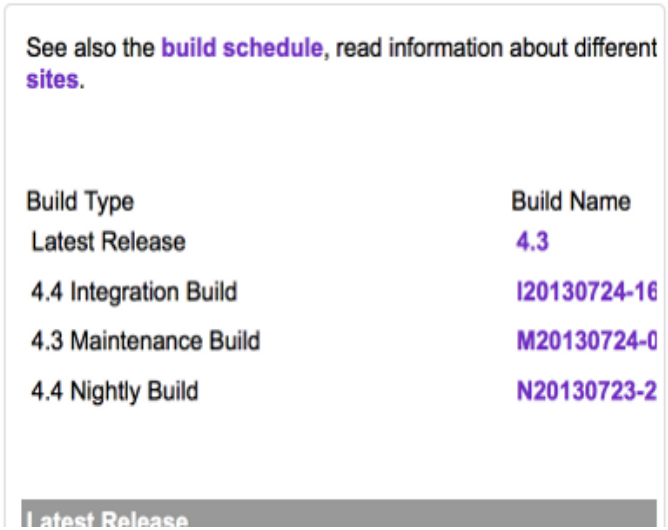
Update sites

To install the e(fx)clipse tooling into your already existing Eclipse you have to add an Update site. There's no released yet available so you need to add the release update-site http://download.eclipse.org/efxclipse/updates-released/1.0.0/site and http://download.eclipse.org/modeling/tmf/xtext/updates/composite/releases/ for the needed xtext dependencies.

If you are not familiar with update-sites you can follow the short guide below or use a pre-packaged version

Eclipse 4.4

1.

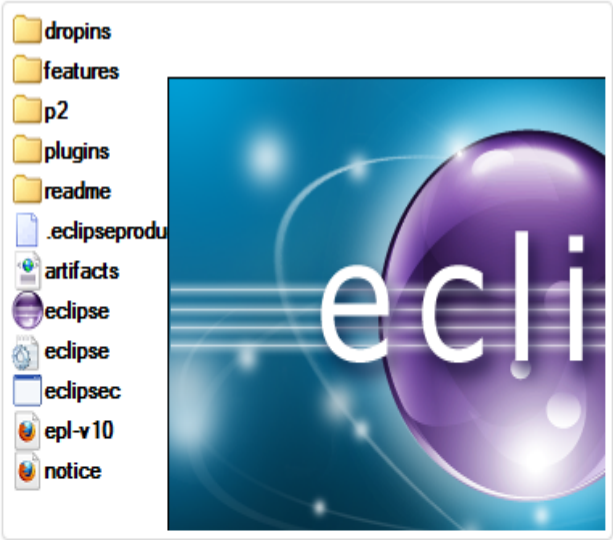


Download the latest release of the Eclipse 4.4 SDK.

http://www.eclipse.org/efxclipse/install.html

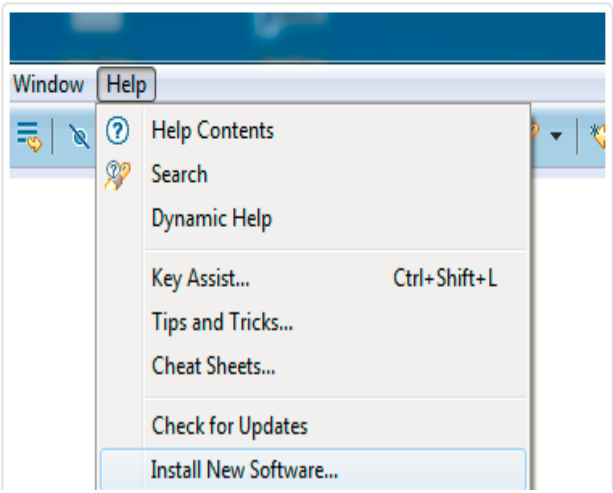
Java FX Eclipse Configurations

2.



Fire up your Eclipse IDE if you have not done so already.

3.

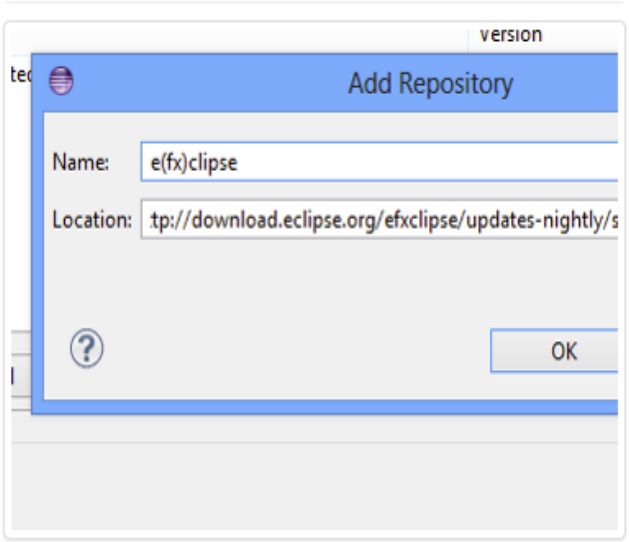


e(fx)clipse is an extension for your Eclipse IDE which is delivered as a so called p2 repository. Extensions like this can be installed using the "Install New Software" wizard.

http://www.eclipse.org/efxclipse/install.html

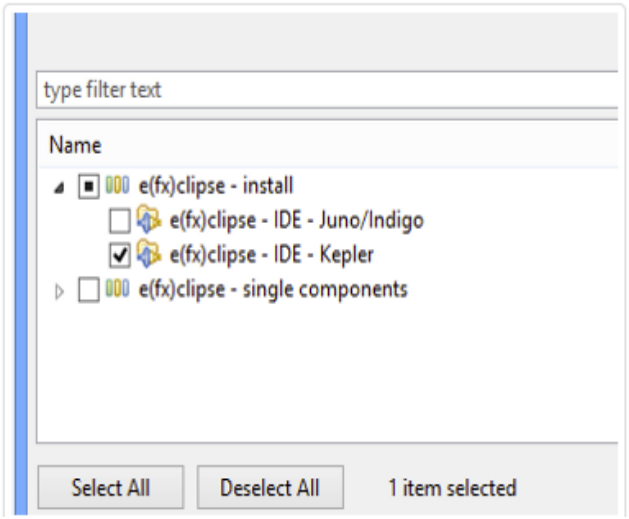
Java FX Eclipse Configurations

4.



Eclipse does not know about the location of the e(fx)clipse p2 repository so you need to add the repository as a software site. The address is: http://download.eclipse.org/efxclipse/updates-released/1.0.0/site

5.



In the tree of installable features, check only the entry corresponding to your release:

- e(fx)clipse - IDE

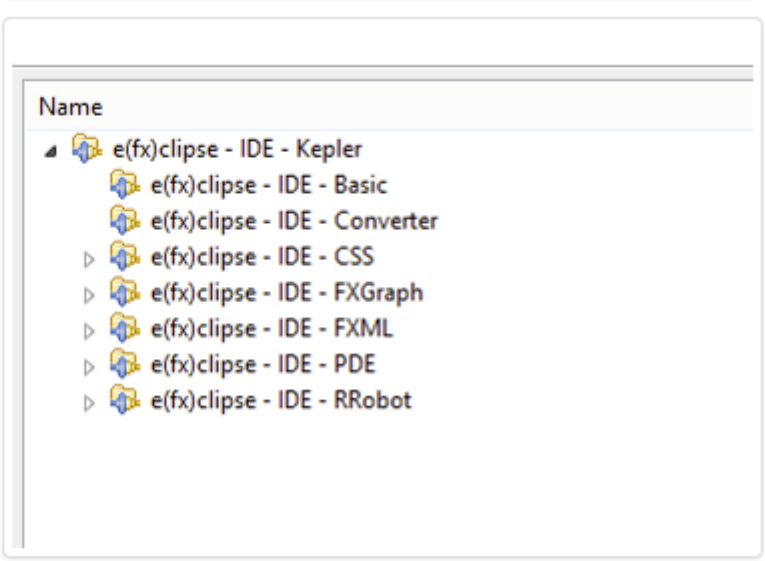
Also, make sure the "Contact all update sites during install to find required software" is checked.

You may, however, install the FX Runtime feature into your IDE but we strongly recommend composing your own target platform.

http://www.eclipse.org/efxclipse/install.html

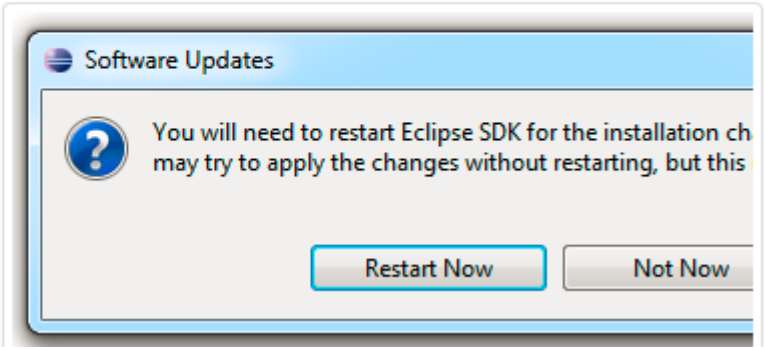
Java FX Eclipse Configurations

6.



Go through the wizard and let p2 do its job.

7.



When requested, restart Eclipse.

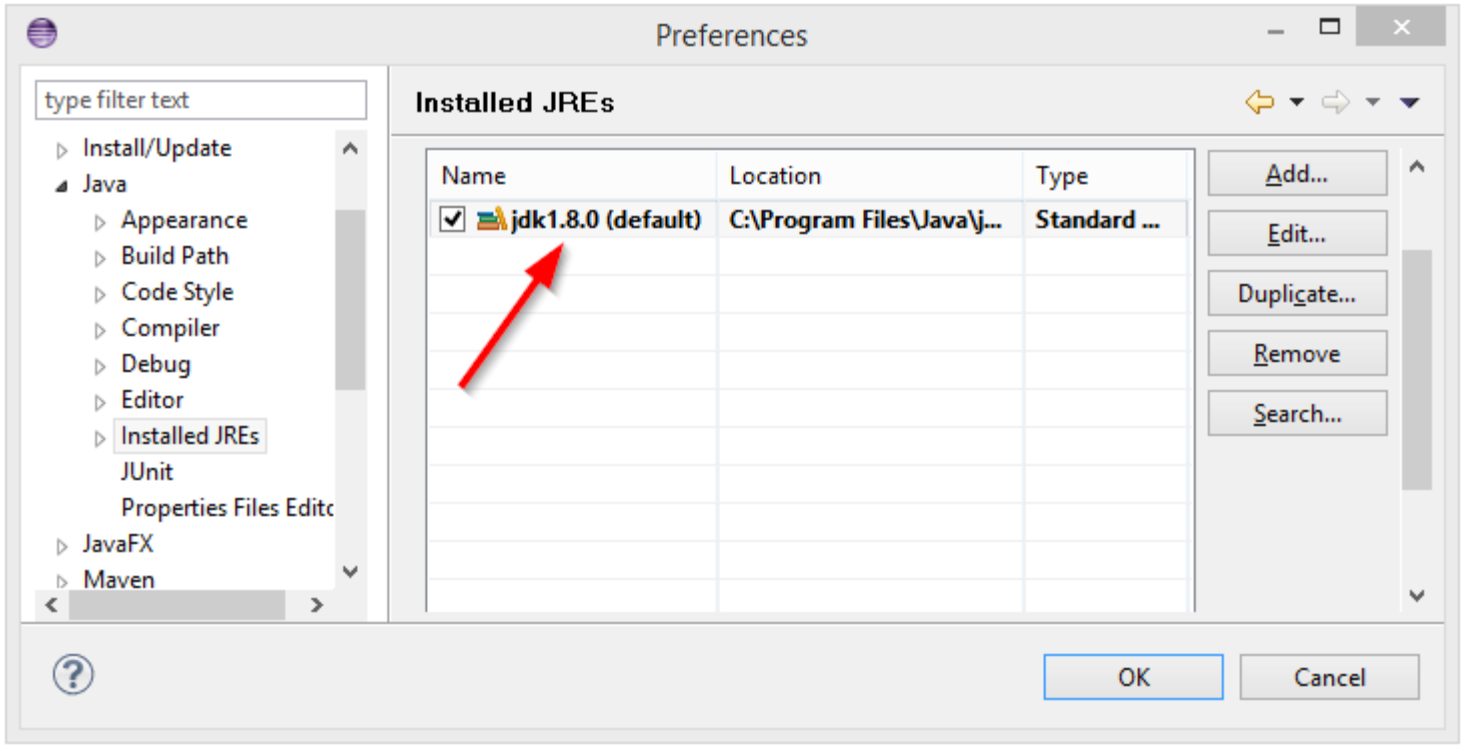
http://www.eclipse.org/efxclipse/install.html

Java FX Eclipse Configurations

It is need to tell Eclipse to use JDK 8 and also where it will find the Scene Builder:

1. Open the Eclipse Preferences and navigate to *Java | Installed JREs*.
2. Click *Add...*, select *Standard VM* and choose the installation *Directory* of your JDK 8.

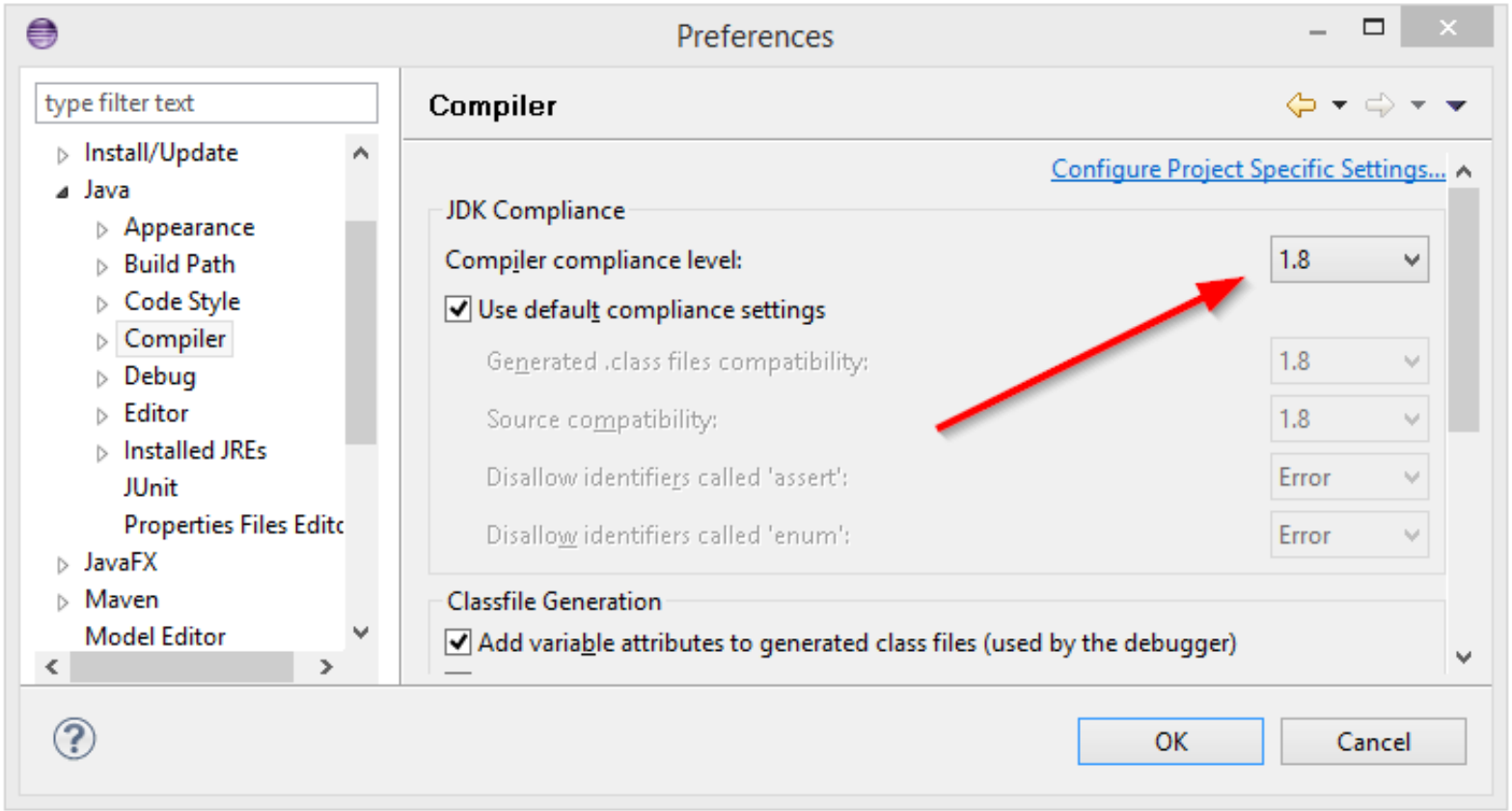
3. Remove the other JREs or JDKs so that the **JDK 8 becomes the default.**



http://code.makery.ch/java/javafx-8-tutorial-part1/

Java FX Eclipse Configurations

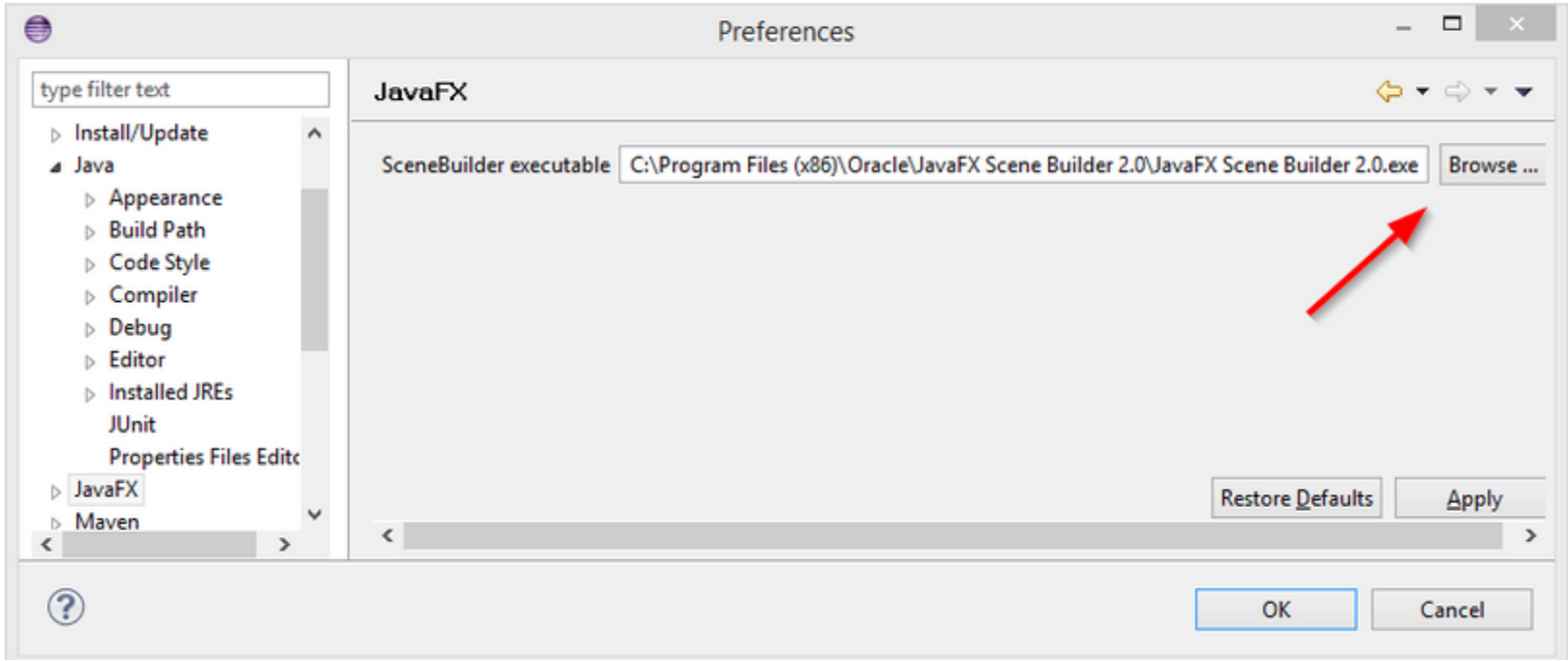
4. Navigate to *Java* | *Compiler*. Set the **Compiler compliance level to 1.8.**



http://code.makery.ch/java/javafx-8-tutorial-part1/

Java FX Eclipse Configurations

5. Navigate to the *JavaFX* preferences. Specify the path to your Scene Builder executable.



http://code.makery.ch/java/javafx-8-tutorial-part1/

2. Java GUI – Java FX

Java FX Sample

```
package eu.ase.gui;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello World!");
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);
        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}
```

https://docs.oracle.com/javafx/2/get_started/hello_world.htm | http://code.makery.ch/java/javafx-8-tutorial-part1/

Java FX Sample

Important things about the basic structure of a JavaFX application:

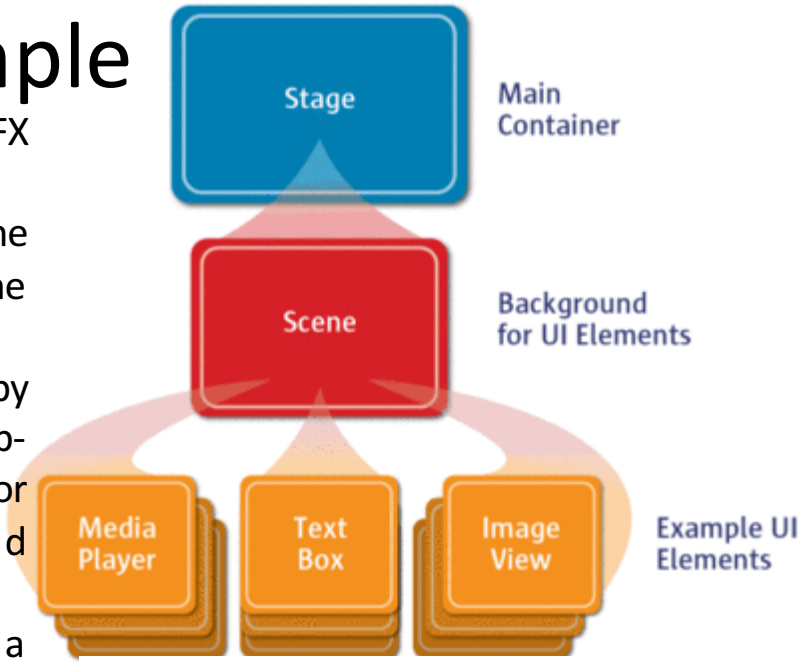
• The main class for a JavaFX application extends the javafx.application.Application class. The start() method is the main entry point for all JavaFX applications.

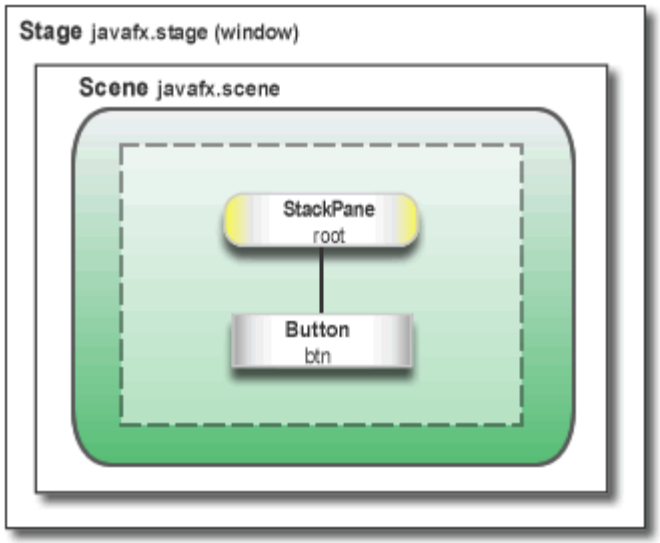
• A JavaFX application defines the user interface container by means of a stage and a scene. The JavaFX Stage class is the top-level JavaFX container. The JavaFX Scene class is the container for all content. Example HelloWorld creates the stage and scene and makes the scene visible in a given pixel size.

• In JavaFX, the content of the scene is represented as a hierarchical scene graph of nodes. In this example, the root node is a StackPane object, which is a resizable layout node. This means that the root node's size tracks the scene's size and changes when the stage is resized by a user.

• The root node contains one child node, a button control with text, plus an event handler to print a message when the button is pressed.

• The main() method is not required for JavaFX applications when the JAR file for the application is created with the JavaFX Packager tool, which embeds the JavaFX Launcher in the JAR file.





Java Swing
for easy sharing

Section Conclusions

Java Swing as GUI development approach is going to be replaced by Java FX

Java Swing is good for understanding call-back and event – delegate mechanisms, as well as, anonymous inner classes

Event-driven programming might be implemented with Java Swing

For evaluation Java FX is mandatory / Java Swing is kept for back-ward compatibility reasons

Share knowledge, Empowering Minds

Communicate & Exchange Ideas

Questions & Answers!

But wait…

There's More!

Thanks!

Java Programming – Software App Development
End of Lecture 14 – Java SE Multithreading 2 + GUI