CHAPTER 8

■ ■ ■

Introduction to JavaFX

QUESTIONS AND EXERCISES

1. What is JavaFX?

Answer:

JavaFX is an open source Java-based GUI framework for developing rich client applications. It is also the successor of Swing in the arena of GUI development technology in the Java platform. The JavaFX library is available as a public Java API. JavaFX contains several features that make it a preferred choice for developing rich client applications:

- JavaFX is written in Java, enabling you to take advantage of all Java features such as multithreading, generics, lambda expressions, etc. You can use any Java editor of your choice, such as NetBeans, to author, compile, run, debug, and package your JavaFX application.
- JavaFX supports data binding through its libraries.
- JavaFX code can be written using any JVM-supported scripting languages such as Visage, Groovy, Scala, Nashorn, etc.
- JavaFX offers two ways to build a UI: using Java code and using FXML. FXML is an XML-based scriptable markup language to define a UI declaratively. You can download the Scene Builder tool for Java 9 from `http://gluonhq.com/products/scene-builder`.
- JavaFX provides a rich set of multimedia support such as playing back audios and videos. It takes advantage of available codecs on the platform.
- JavaFX lets you embed web content in applications.
- JavaFX provides out-of-the-box support for applying effects and animations, which are important for developing gaming applications. In JavaFX, you can achieve sophisticated animations by writing just a few lines of code.

2. What is the fully qualified name of the class from which your JavaFX application class must inherit?

Answer:

`javafx.application.Application`

3. To run a JavaFX application with UI components, your module must export the package containing your JavaFX application class to at least one of the JavaFX modules. What is the name of that JavaFX module?

Answer:

`javafx.graphics`

Commented [KS1]: Removed Question #4, which was a duplicate of question #2.

4. What is the signature of the `start()` method that your JavaFX application class needs to override?

Answer:

```
void start(Stage stage)
```

5. Do you need to include a "public static void main(String[] args)" method with your application class to run your JavaFX application?

Answer:

No. the Java launcher (the `java` command) does not require a `main()` method to launch a JavaFX application. If the class that you want to run inherits from the `Application` class, the `java` command launches the JavaFX application by automatically calling the `Application.launch()` method for you.

6. What is a stage? How does the primary stage get created? How do you show a stage?

Answer:

Similar to a stage in the real world, a JavaFX stage displays a scene. A scene has visuals—such as text, shapes, images, controls, animations, effects, etc.—with which the user may interact, as is the case with all GUI-based applications.

The JavaFX application launcher create a primary stage without a scene. The stage look-and-feel is different depending on the environment your application is run in.

You must show the stage to see the visuals contained in its scene. The `show()` method of the stage to show the stage.

7. What is a scene and how do you add a scene to a stage?

Answer:

An instance of the `Scene` class, which is in the `javafx.scene` package, represents a scene. A stage contains one scene. A scene contains visuals—such as text, shapes, images, controls, animations, effects, etc.

You create a `Scene` object, add controls to it, and call the `setScene(Scene value)` method of the `Stage` class to set the scene for the stage.

8. Explain the life cycle of a JavaFX application with the threads involved in executing code in each stage.

Answer:

JavaFX runtime creates several threads that are used to perform different tasks at different stages in the application. The JavaFX runtime creates, among other threads, two threads named

- JavaFX-Launcher
- JavaFX Application Thread

The `launch()` static method of the `Application` class create these threads. During the lifetime of a JavaFX application, the JavaFX runtime calls the following methods of the JavaFX application class in order:

- The no-args constructor
- The `init()` method
- The `start()` method

- The `stop()` method

The JavaFX runtime creates the instance of the specified application class on the JavaFX Application Thread.

The JavaFX-Launcher thread calls the `init()` method of the application class. The `init()` method implementation in the `Application` class is empty. You can override this method in your application class. It is not allowed to create a `Stage` or a `Scene` on the JavaFX-Launcher thread. They must be created on the JavaFX Application Thread. Therefore, you cannot create a `Stage` or a `Scene` inside the `init()` method. Attempting to do so throws a runtime exception. It is fine to create UI controls, for example, buttons, shapes, etc. in the `init()` method.

The JavaFX Application Thread calls the `start(Stage stage)` method of the application class. Note that the `start()` method in the `Application` class is declared `abstract`, and you must override this method in your application class.

At this point, the `launch()` method waits for the JavaFX application to finish.

When the application finishes, the JavaFX Application Thread calls the `stop()` method of the application class. The default implementation of the `stop()` method is empty in the `Application` class. You will have to override this method in your application class to perform your logic when your application stops.

9. Explain two ways that a JavaFX application may get terminated.

Answer:

A JavaFX application may be terminated explicitly or implicitly. You can terminate a JavaFX application explicitly by calling the `exit()` static method of the `Platform` class. When this method is called, after or from within the `start()` method, the `stop()` method of the `Application` class is called, and then the JavaFX Application Thread is terminated. At this point, if there are only daemon threads running, the JVM will exit. If the `Platform.exit()` method is called from the constructor or the `init()` method of the `Application` class, the `stop()` method of the `Appplication` class may not be called.

A JavaFX application may be terminated implicitly when the last window is closed. This behavior can be turned on or turned off using the static `setImplicitExit(boolean implicitExit)` method of the `Platform` class. Passing `true` to this method turns this behavior on. Passing `false` to this method turns this behavior off. By default, this behavior is turned on. When this behavior is turned on, the `stop()` method of the `Application` class is called

before terminating the JavaFX Application Thread. Terminating the JavaFX Application Thread does not always terminate the JVM. The JVM terminates if all running non-daemon threads terminate. If the implicit terminating behavior of the JavaFX application is turned off, you must call the `exit()` method of the `Platform` class to terminate the application.

10. Write a simple JavaFX program with the following nodes: A `Label`, a `TextField`, a `Button`, and a `Circle`. The `TextField` contains an initial value of 100, which is the radius of the `Circle` in pixel. When the user changes the value in the `TextField`, the `Circle` should be redrawn to reflect the new radius. All nodes should be visible all the time. That is, you need to resize the screen to fit the `Circle`.

Solution:

```
// DrawCircle.java
package com.jdojo.javafx.exercises;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.control.Label;
import javafx.scene.control.Spinner;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;


public class DrawCircle extends Application {
    Circle circle = new Circle();
    Spinner<Double> radiusFld = new Spinner<>(50, 1000, 100, 20);;

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

    @Override
    public void start(Stage stage) {
        // Arrange the label and the spinner at the top
        HBox hbox = new HBox(5);
        hbox.getChildren().addAll(new Label("Radius"), radiusFld);

        // Have a circle in a stack pane
        StackPane stack = new StackPane();
        stack.getChildren().add(new Group(circle));
```

```
        // Add all controls to a border pane
        BorderPane pane = new BorderPane();
        pane.setTop(hbox);
        pane.setCenter(stack);

        // Bind the circle radius to the spinner value
        circle.radiusProperty().bind(radiusFld.valueProperty());

        // Make sure the window is resized when the circle's radius changes
        radiusFld.valueProperty().addListener(radius -> {
           stage.sizeToScene();
        });


        // Set the stage properties and make it visible
        Scene scene = new Scene(pane);
        stage.setTitle("Resizable Circle");
        stage.setScene(scene);
        stage.sizeToScene();
        stage.show();
    }
}
```

11. Write a program using a JavaFX observable list that prints the sum of all integers in a list to the standard output. When the elements of the list changes, the new sum is printed.

Solution:

```
// SummingList.java
package com.jdojo.javafx.exercises;

import javafx.collections.FXCollections;
import javafx.collections.ListChangeListener;
import javafx.collections.ObservableList;

public class SummingList {
    public static void main(String[] args) {
        // Create a list with some elements
        ObservableList<Integer> list = FXCollections.observableArrayList();

        // Add a ChangeListener to the list
        list.addListener((ListChangeListener.Change<? extends Integer> change) -> {
            // Comute the sum
            int sum = list.stream().mapToInt(Integer::intValue).sum();

            // Print the new sum of inetgers in the list
            System.out.print("List: " + list);
            System.out.println(", Sum: " + sum);
        });
```

```
        // Add items to the list
        list.add(10);
        list.add(20);
        list.addAll(30, 40);

        // Remove an item from the list
        list.removeAll(30);
    }
}
```

12. What are event capturing and event bubbling phases? If you want to disable a specific type of event on a node and all its child nodes, what kind of handler will you be writing: an event filter or event handler?

Commented [KS2]: Moved will before you

Answer:

During the capture phase, an event travels from the head to tail of its event dispatch chain. During the bubbling phase, an event travels from the tail to head of its event dispatch chain.

If you want to disable a specific type of event on a node and all its child nodes, you need to add an event event filter to the node?

13. Modify the `EventHandling` class in the source code for this chapter. Add another `CheckBox` with a label "Disable Mouse Clicks". When this `CheckBox` is selected, the mouse-clicked event for the entire stage is disabled - you cannot use mouse to click the two `CheckBox` nodes or any other nodes. When you unselect the `CheckBox`, the mouse-clicked event should work as it works in the example program. (Hint: Try disabling the mouse-pressed event as well to disable selection of CheckBox nodes.)

Commented [KS3]: changed ". With" to with"

Solution:

Use the keyboard (press the spacebar) to unselect the checkboxes after you select the "Disable Mouse Clicks" checkbox.

```
// EventHandling.java
package com.jdojo.javafx.exercises;

import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
```

```
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.input.MouseEvent;
import static javafx.scene.input.MouseEvent.MOUSE_CLICKED;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

public class EventHandling extends Application {
    private CheckBox consumeEventCbx = new CheckBox("Consume Mouse Click at Circle");
    private CheckBox disableEventCbx = new CheckBox("Disable Mouse Clicks");

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

    @Override
    public void start(Stage stage) {
        Circle circle = new Circle(50, 50, 50);
        circle.setFill(Color.CORAL);

        Rectangle rect = new Rectangle(100, 100);
        rect.setFill(Color.TAN);

        VBox checkBoxContainer = new VBox(5);
        checkBoxContainer.getChildren().addAll(consumeEventCbx, disableEventCbx);

        HBox root = new HBox();
        root.setPadding(new Insets(20));
        root.setSpacing(20);
        root.getChildren().addAll(circle, rect, checkBoxContainer);

        Scene scene = new Scene(root);

        // Register mouse-clicked event handlers to all nodes,
        // except to the rectangle and the checkbox
        EventHandler<MouseEvent> handler = e -> handleEvent(e);
        EventHandler<MouseEvent> circleMeHandler = e -> handleEventforCircle(e);
        EventHandler<MouseEvent> disableFilter = e -> handleEventFilter(e);

        stage.addEventHandler(MOUSE_CLICKED, handler);
        stage.addEventFilter(MOUSE_CLICKED, disableFilter);
        scene.addEventHandler(MOUSE_CLICKED, handler);
        root.addEventHandler(MOUSE_CLICKED, handler);
        circle.addEventHandler(MOUSE_CLICKED, circleMeHandler);

        stage.setScene(scene);
        stage.setTitle("Event Handling");
        stage.show();
    }
```

```
    public void handleEventFilter(MouseEvent e) {
        if (disableEventCbx.isSelected()) {
            e.consume();
        }
    }

    public void handleEvent(MouseEvent e) {
        print(e);
    }

    public void handleEventforCircle(MouseEvent e) {
        print(e);
        if (consumeEventCbx.isSelected()) {
            e.consume();
        }
    }

    public void print(MouseEvent e) {
        String type = e.getEventType().getName();
        String source = e.getSource().getClass().getSimpleName();
        String target = e.getTarget().getClass().getSimpleName();

        // Get coordinates of the mouse relative to the event source
        double x = e.getX();
        double y = e.getY();

        System.out.println("Type=" + type + ", Target=" + target
                + ", Source=" + source
                + ", location(" + x + ", " + y + ")");
    }
}
```

14. Standard buttons such as `Yes`, `No`, and `Cancel` are displayed in different orders on different operating systems. You have to pack two buttons, Yes and No, in a horizontal bar where the buttons ordering depends on the operating system. Name the control that you will use to achieve this.

Answer:

`javafx.scene.control.ButtonBar`

15. What is the use of a `Group` node? Name two panes that acts as a container.

Commented [KS4]: Added "a"

Answer:

A `Group` node contains an `ObservableList` of children that are rendered in order whenever this node is rendered. A `Group` node takes on the collective bounds of its children and is not directly resizable. Any transform, effect, or state applied to a `Group` node will be applied to all children of that group.

`BorderPane` and `StackPane` are examples of two panes that act as containers.

16. Name and define five UI controls.

Answer:

Control	Description
`Label`	A non-editable text control that is typically used to display the label for another control.
`Button`	Represents a command button control. It can display text and an icon. It generates an `ActionEvent` when it is activated.
`RadioButton`	Represents a radio button. It is used to provide a mutually exclusive choice from the list of choices.
`CheckBox`	Represents a three-state selection control. The three states are *checked*, *unchecked*, and *undefined*.
`TextField`	Represents a single-line text input control.

17. What types of shapes does JavaFX support - 2D only, 3D only, or both?

Answer:

JavaFX supports both 2D shapes and 3D shapes.

18. What is the use of a `Canvas` in JavaFX?

Answer:

A `Canvas` offers a drawing surface to draw shapes, images, and text using drawing commands.

19. What are effects, transformations, and animation in JavaFX?

Answer:

An effect is a filter that accepts one or more graphical inputs, applies an algorithm on the inputs, and produces an output. Typically, effects are applied to nodes to create visually appealing user interfaces. Examples of effects are shadow, blur, warp, glow, reflection, blending, different types of lighting, etc. The JavaFX library provides several effect-related classes. An effect is a conditional feature. Effects applied to nodes will be ignored if they are not available on the platform.

A transformation is a mapping of points in a coordinate space to themselves, preserving distances and directions between them. Several types of transformations can be applied to points in a coordinate space. JavaFX supports the following types of transformation: Translation, Rotation, Shear, Scale, and Affine. An instance of the abstract `Transform` class represents a transformation in JavaFX. The `Transform` class contains common methods and properties used by all types of transformations on nodes. It contains factory methods to create specific types of transformations.

In JavaFX, animation is defined as changing the property of a node over time.

20. What is FXML?

Answer:

FXML is an XML-based language for building a user interface for JavaFX applications. You can use FXML to build an entire scene or part of a scene. FXML allows application developers to separate the logic for building the UI from the business logic. If the UI part of the application changes, you do not need to recompile the JavaFX code; you can change the FXML using a text editor and rerun the application. You still use JavaFX to write business logic in the Java programming language. An FXML document is an XML document.

21. Write a program that will print the default printer name. If there is no default printer installed, it should print a message "No printer found".

Solution:

```
// DefaultPrinter.java
package com.jdojo.javafx.exercises;

import javafx.print.Printer;

public class DefaultPrinter {
```

```
    public static void main(String[] args) {
        Printer defaultPrinter = Printer.getDefaultPrinter();
        if (defaultPrinter != null) {
            String name = defaultPrinter.getName();
            System.out.println("Default printer name: " + name);
        } else {
            System.out.println("No printer found.");
        }
    }
}
```