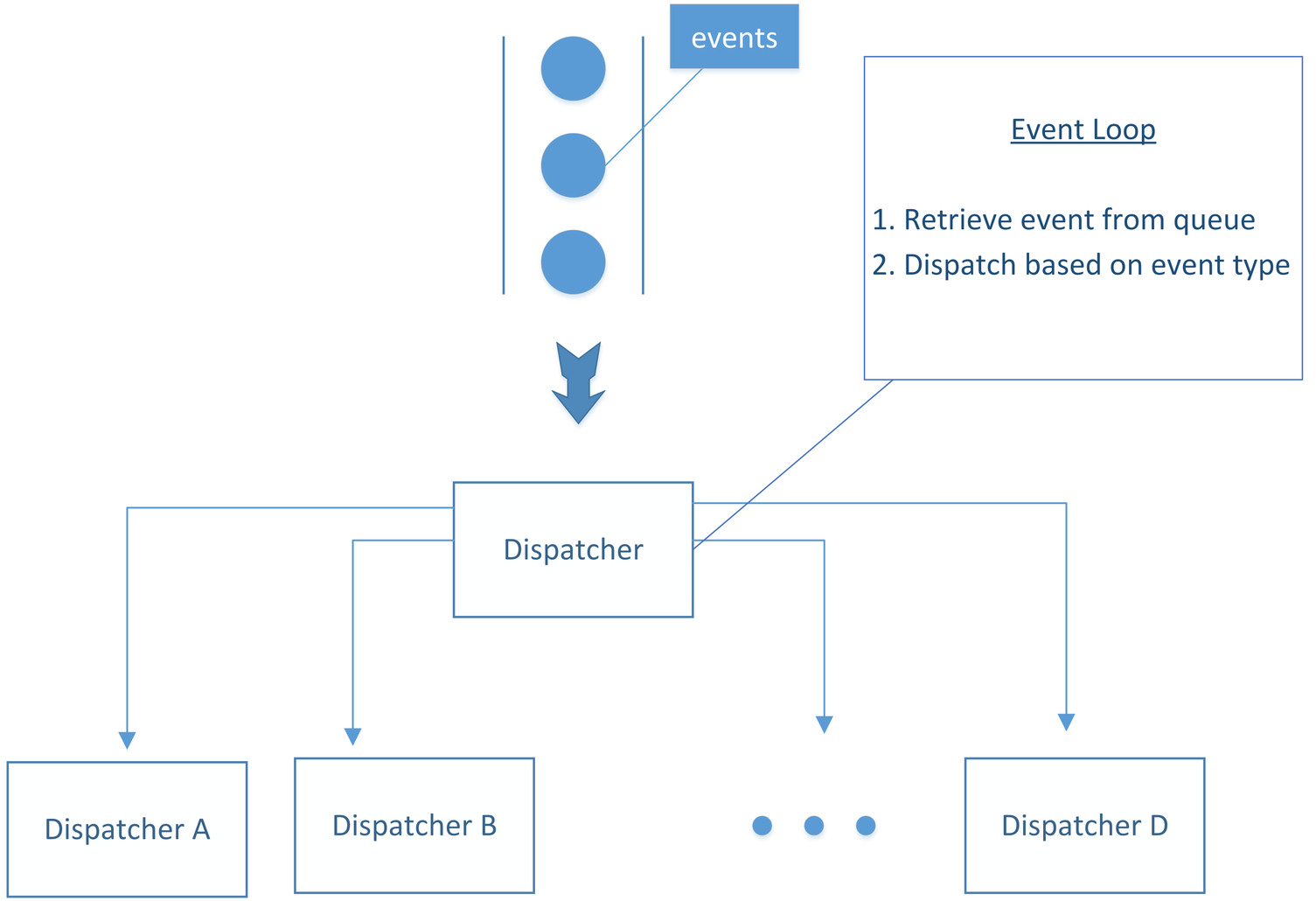
Programming Using Java

Session 10: User Interface Development



Event Handling Pattern

"…process events as they arrive in a queue…"

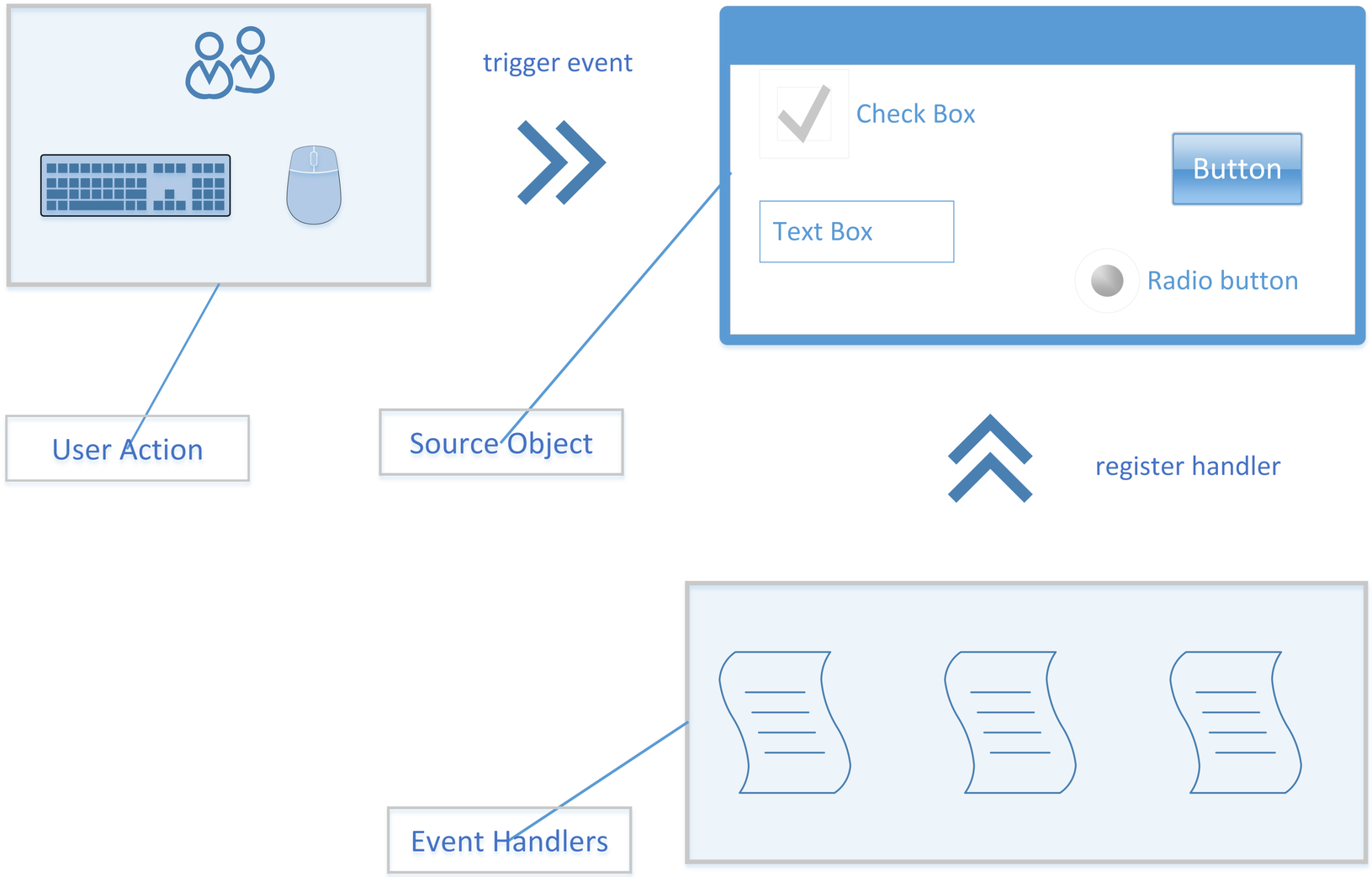
Event Handling Psuedocode

1. event objects are obtained from queue
2. events are sent to type-specific handlers

- developer codes how program reacts to system events
- sequence of events is unknown

1.

```
public class Dispatcher {
  public void main(String[] args) { ...
    Event e;

    while (true) {
      e = Queue.getNextEvent();
      if (e.type == QUIT) break;
      switch (e.type) {
       case EVENT1: eventHandler1(e);
       case EVENT2: eventHandler2(e);
       …
       default:
        // ignore event or raise exception
      }
    }
  }
}
```



GUI Event Handling

Java UI History

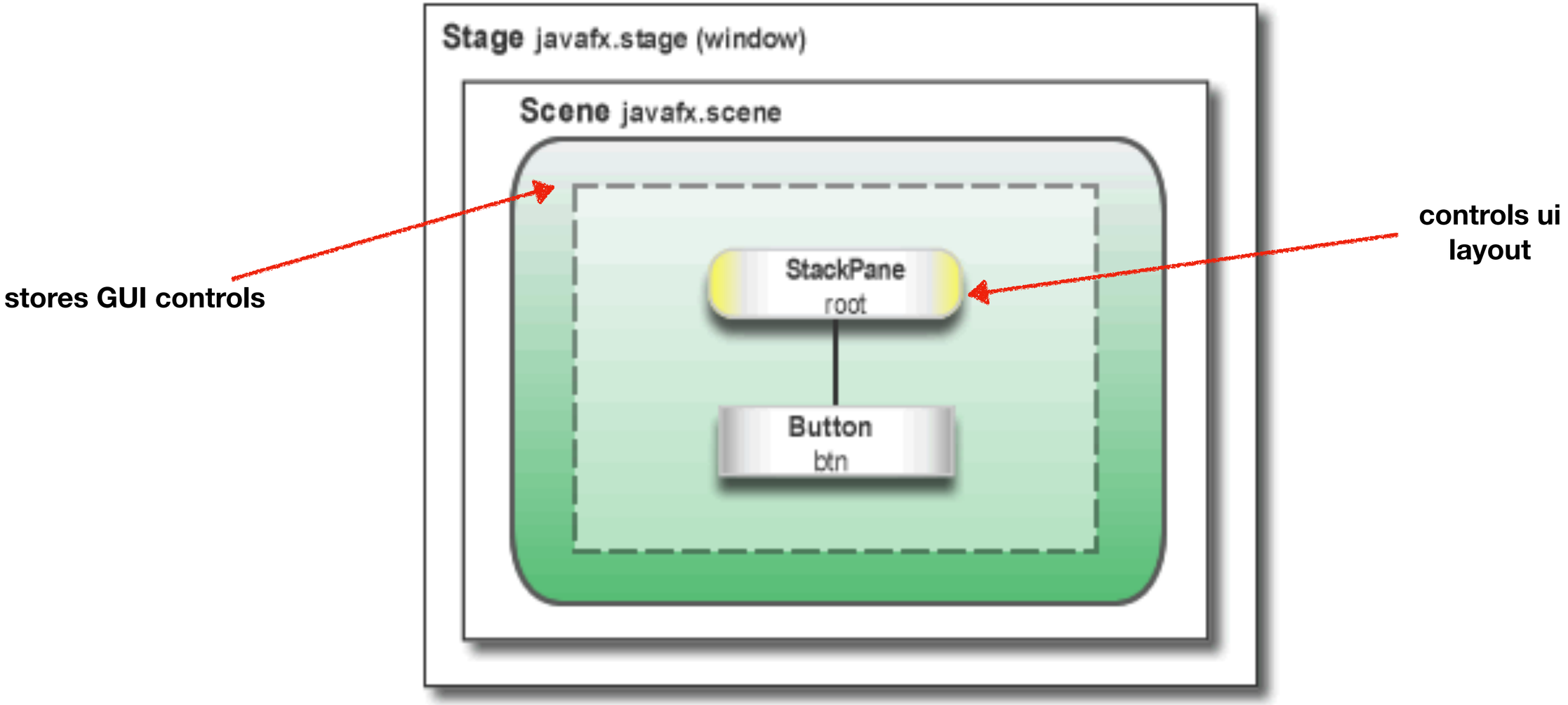
1. AWT
    - components translate into platform equivalents
        - look / feel vary
2. Swing
    - pluggable look and feel
    - modified MVC (view and controller are together
3. JavaFX
    - MVC
    - multithreading, multiple threads can execute while sharing system resources

JavaFX Benefits

1. GUIs are created a lot faster than in Swing and AWT
2. Easy integration of sounds, images and videos and of web content
3. not compiled into bytecode
4. localized at runtime
5. used for any JVM language
6. Code is simplified in JavaFX by strict MVC Architecture
7. JavaFX can be integrated in Swing applications, allowing for a smoother transition

GUI Event Handling

- event source object, fires an action event
- event object,
- event handler object, capable of processing event fired



JavaFX UI Hierarchy

"…stage contains a nested set of objects…"

JavaFX Event Handlers

- extends javafx.application.Application
- create UI controls/bind handlers in Application.start()
- must be instance of EventHandler<T extends Event> interface
- must register with the event source object source.setOnAction(handler)

```
public class HelloWorld extends Application {

  @Override
  public void start(Stage primaryStage) {
    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!");
        }
      }

    ); ...
}
```

source object

handler registers with source

JavaFX Event Handlers

- create scenes/panes
- bind controls to panes

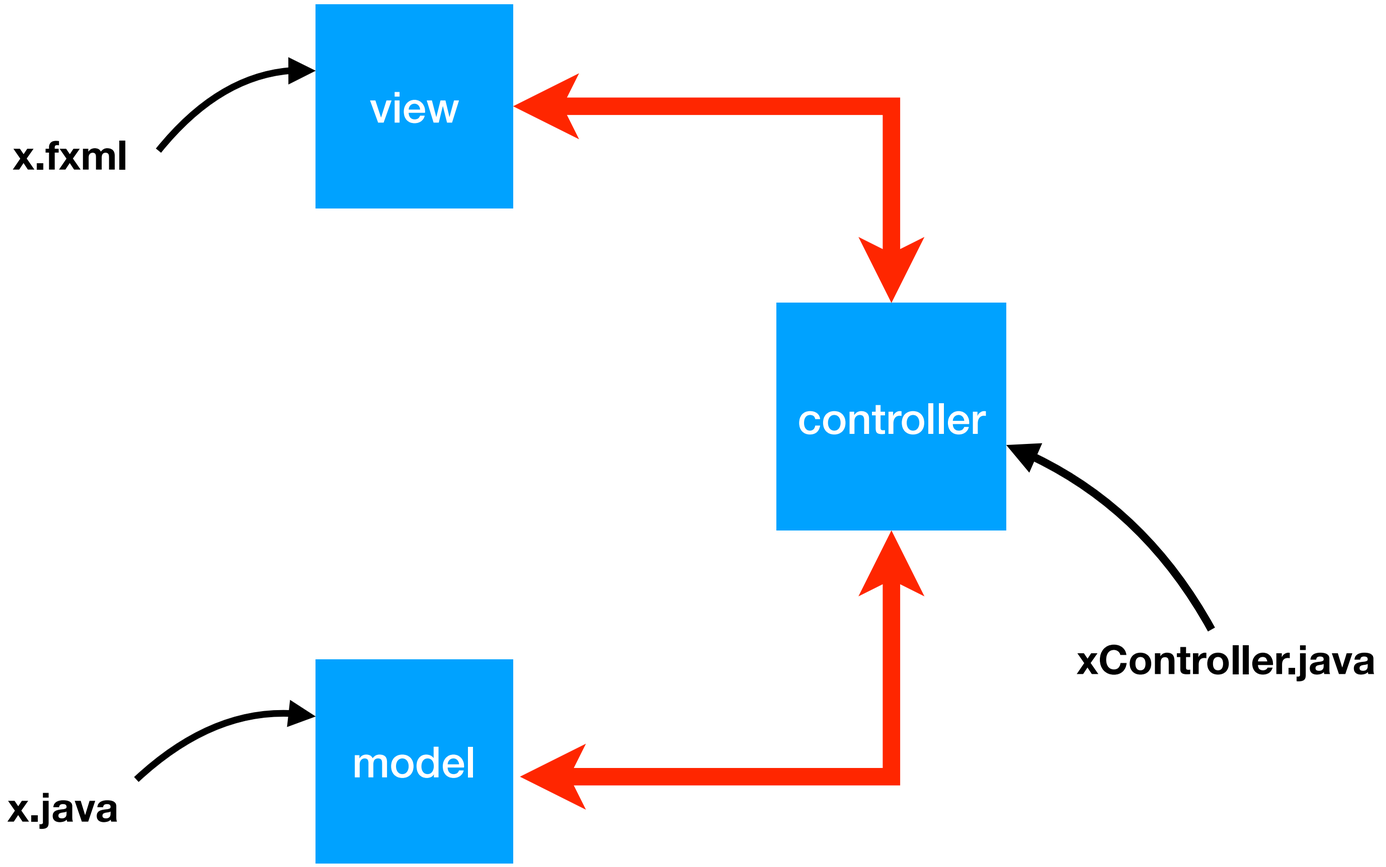
```
public class HelloWorld extends Application {

  @Override
  public void start(Stage primaryStage) {
    ….

    StackPane root = new StackPane();
    root.getChildren().add(btn);

    Scene scene = new Scene(root, 300, 250);
    primaryStage.setTitle("Hello World!");
    primaryStage.setScene(scene);
    primaryStage.show();

  }

}
```

add to pane

configure scene

Using FXML

- markup language to create UI
- allows use of MVC



```
@Override
public void start(Stage stage) throws Exception {

  Parent root = FXMLLoader.load
  (getClass().getResource("fxml_example.fxml"));

  Scene scene = new Scene(root, 300, 275);
  stage.setTitle("FXML Welcome");
  stage.setScene(scene);
  stage.show();

}
```

FXML Application

"…retrieves UI resource definitions…"

```
<Label text="Password:"
GridPane.columnIndex="0" GridPane.rowIndex="2"/>

<PasswordField fx:id="passwordField" GridPane.columnIndex="1"
GridPane.rowIndex="2"/>
```

```
<HBox spacing="10" alignment="bottom_right" GridPane.columnIndex="1"
GridPane.rowIndex="4"> <Button text="Sign In"
onAction="#handleSubmitButtonAction"/>
</HBox>

<Text fx:id="actiontarget"
GridPane.columnIndex="0" GridPane.columnSpan="2" GridPane.halignment="RIGHT"
GridPane.rowIndex="6"/>
```

FXML File

"…defines UI resources…"

```
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class FXMLExampleController {

   @FXML private Text actiontarget;
   @FXML protected void handleSubmitButtonAction (ActionEvent event)  {
      actiontarget.setText("button pressed");
   }
}
```

FXML Controller File

"…logic to update controls…"