Event-Driven Programming

Lecture 10

Recall: JavaFX Basics

- So far we've learned about some of the basic GUI classes (e.g. shapes, buttons) and how to arrange them in window(s)
- A big missing piece: interaction
- To have a GUI interact with a user, we have elements respond to user actions, or **event-driven programming**

Big Picture

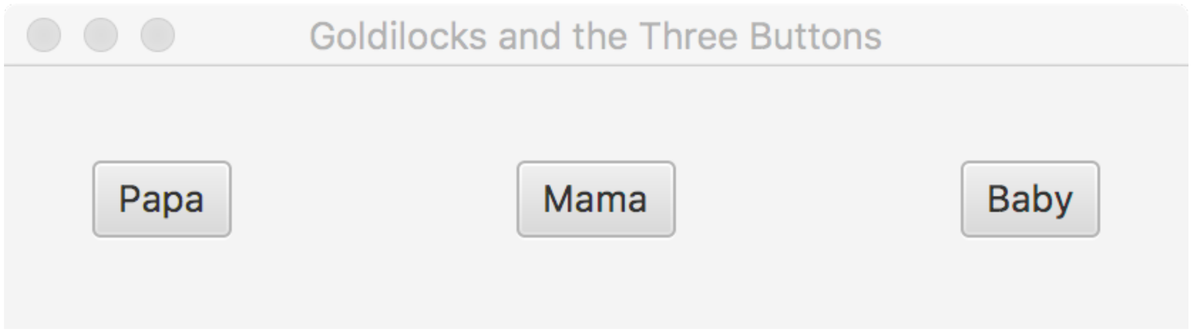
- When GUI elements want to implement event-driven programming, they will offer ways to "handle" an event via a class that implements an interface

- Typical sequence:
  1. Create GUI
  2. "Register" a class to handle event(s), sometimes referred to as a "listener"
  3. Implement handling code in the listener

A Simple Example (1)

1. Make a new JavaFX project
2. Create a GUI with three buttons
   – For now, do NOT use SceneBuilder (we'll get to this later)

```
final HBox pane = new HBox(100);
pane.setAlignment(Pos.CENTER);
final Button btnP = new Button("Papa");
final Button btnM = new Button("Mama");
final Button btnB = new Button("Baby");

pane.getChildren().addAll(btnP, btnM, btnB);

primaryStage.setTitle("Goldilocks and the Three Buttons");
primaryStage.setScene(new Scene(pane));
primaryStage.show();
```



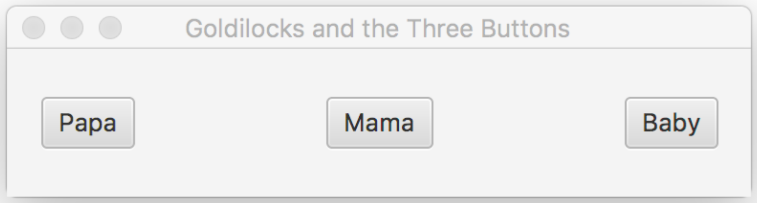
A Simple Example (2)

3. Handle event

   Any class that implements the appropriate interface can be used, including anonymous inner classes and Lambda's

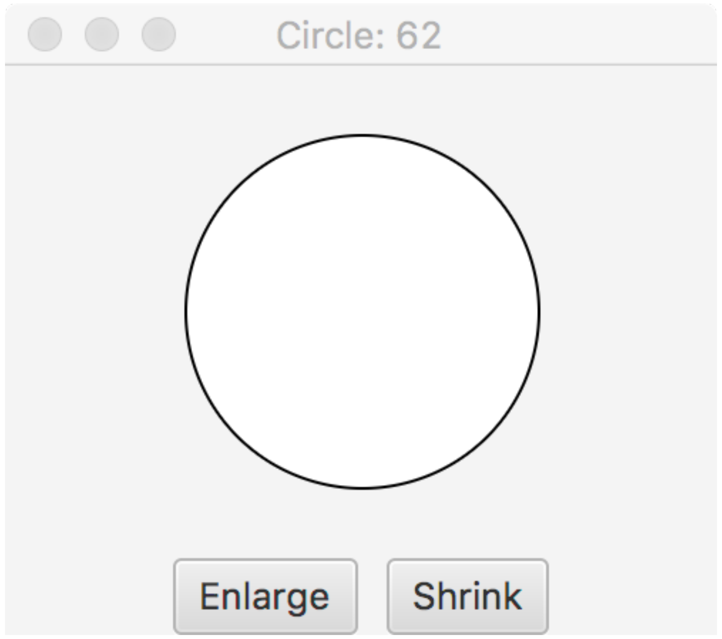
```
private static class JustRight implements
                                    EventHandler<ActionEvent> {
        @Override
        public void handle(ActionEvent event) {
                System.out.printf("Just right :)%n");
        }
}

btnP.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
                System.out.printf("Too Hot!%n");
        }
});

btnM.setOnAction(e->{
        System.out.printf("Too Cold!%n");
});

btnB.setOnAction(new JustRight());
```

```
Too Hot!
Too Cold!
Just right :)
```



Exercise

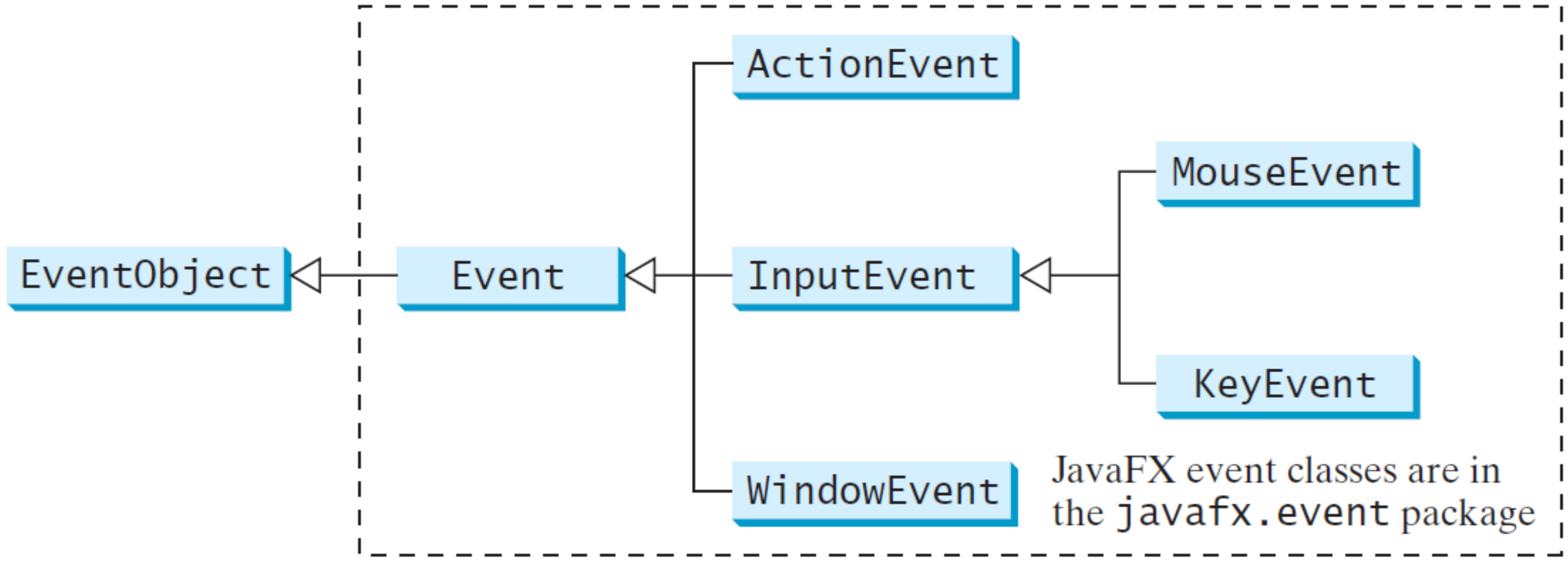
- Create a JavaFX application that allows you to grow/shrink a circle via buttons



Solution

```
StackPane sp = new StackPane();
Circle c = new Circle(50);
c.setStroke(Color.BLACK);
c.setFill(Color.WHITE);
sp.getChildren().add(c);

HBox hBox = new HBox();
hBox.setSpacing(10);
hBox.setAlignment(Pos.CENTER);
Button btnEnlarge = new Button("Enlarge");
btnEnlarge.setOnAction(e->{c.setRadius(c.getRadius()+2);});
Button btnShrink = new Button("Shrink");
btnShrink.setOnAction(e->{c.setRadius(c.getRadius()-2);});
hBox.getChildren().add(btnEnlarge);
hBox.getChildren().add(btnShrink);

BorderPane borderPane = new BorderPane();
borderPane.setCenter(sp);
borderPane.setBottom(hBox);
BorderPane.setAlignment(hBox, Pos.CENTER);
Scene scene = new Scene(borderPane, 250, 200);
primaryStage.titleProperty().bind(c.radiusProperty().asString("Circle: %.0f"));
primaryStage.setScene(scene);
primaryStage.show();
```

JavaFX Events



Event Information

- An event object contains whatever properties are related 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

Example User Actions & Handlers

User Action	*Source Object*	*Event Type Fired*	*Event Registration Method*
Click a button	`Button`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Press Enter in a text field	`TextField`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Check or uncheck	`RadioButton`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Check or uncheck	`CheckBox`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Select a new item	`ComboBox`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Mouse pressed	`Node, Scene`	`MouseEvent`	`setOnMousePressed(EventHandler<MouseEvent>)`
Mouse released			`setOnMouseReleased(EventHandler<MouseEvent>)`
Mouse clicked			`setOnMouseClicked(EventHandler<MouseEvent>)`
Mouse entered			`setOnMouseEntered(EventHandler<MouseEvent>)`
Mouse exited			`setOnMouseExited(EventHandler<MouseEvent>)`
Mouse moved			`setOnMouseMoved(EventHandler<MouseEvent>)`
Mouse dragged			`setOnMouseDragged(EventHandler<MouseEvent>)`
Key pressed	`Node, Scene`	`KeyEvent`	`setOnKeyPressed(EventHandler<KeyEvent>)`
Key released			`setOnKeyReleased(EventHandler<KeyEvent>)`
Key typed			`setOnKeyTyped(EventHandler<KeyEvent>)`

MouseEvent

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.

KeyEvent

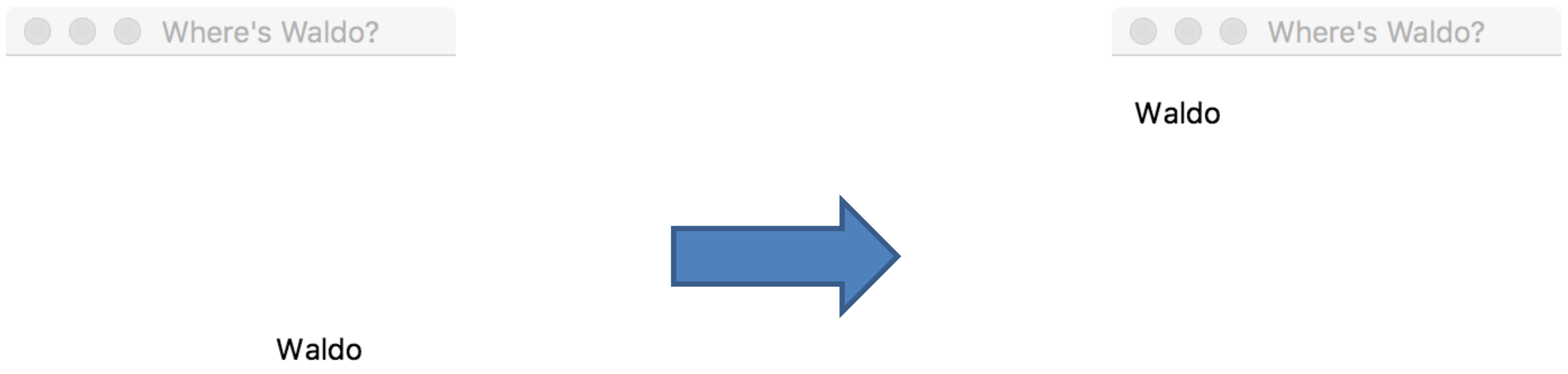
javafx.scene.input.KeyEvent	
`+getCharacter(): String`	Returns the character associated with the key in this event.
`+getCode(): KeyCode`	Returns the key code associated with the key in this event.
`+getText(): String`	Returns a string describing the key code.
`+isAltDown(): boolean`	Returns true if 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.

KeyCode Constants

Constant	*Description*	*Constant*	*Description*
HOME	The Home key	CONTROL	The Control key
END	The End key	SHIFT	The Shift key
PAGE_UP	The Page Up key	BACK_SPACE	The Backspace key
PAGE_DOWN	The Page Down key	CAPS	The Caps Lock key
UP	The up-arrow key	NUM_LOCK	The Num Lock key
DOWN	The down-arrow key	ENTER	The Enter key
LEFT	The left-arrow key	UNDEFINED	The keyCode unknown
RIGHT	The right-arrow key	F1 to F12	The function keys from F1 to F12
ESCAPE	The Esc key	0 to 9	The number keys from 0 to 9
TAB	The Tab key	A to Z	The letter keys from A to Z

Exercise

Write a JavaFX program that allows the user to control the position of the text “Waldo” via Left/Down/Up/Right arrow keys



Solution

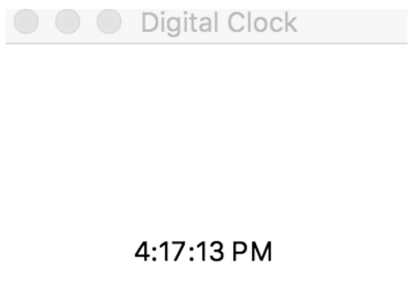
```
Pane pane = new Pane();

Text text = new Text(50, 50, "Waldo");
pane.getChildren().add(text);
text.setOnKeyPressed(e -> {
	switch (e.getCode()) {
		case DOWN:
			text.setY(text.getY() + 5);
			break;
		case UP:
			text.setY(text.getY() - 5);
			break;
		case LEFT:
			text.setX(text.getX() - 5);
			break;
		case RIGHT:
			text.setX(text.getX() + 5);
			break;
		default:
			break;
	}
});

Scene scene = new Scene(pane, 200, 200);
primaryStage.setTitle("Where's Waldo?");
primaryStage.setScene(scene);
primaryStage.show();

text.requestFocus();
```

JavaFX Animations

- JavaFX provides the **`Animation`** class with the core functionality for all animations
- Look to **`PathTransition`** for movement along a path
- Look to **`FadeTransition`** for opacity change over a given time
- The **`Timeline`** class supports general animation across specified time intervals

Example

```
BorderPane pane = new BorderPane();
Text text = new Text(50, 50, "");
pane.setCenter(text);

Scene scene = new Scene(pane, 200, 200);
primaryStage.setTitle("Digital Clock");
primaryStage.setScene(scene);
primaryStage.show();

EventHandler<ActionEvent> eH = e->{
    final LocalDateTime dt = LocalDateTime.now();
    text.setText(String.format("%d:%02d:%02d %sM",
        dt.getHour()%12, dt.getMinute(),
        dt.getSecond(), dt.getHour()>=12?"P":"A"));
};

Timeline a = new Timeline(new KeyFrame(Duration.millis(1000), eH));
a.setCycleCount(Timeline.INDEFINITE);
a.play();
```

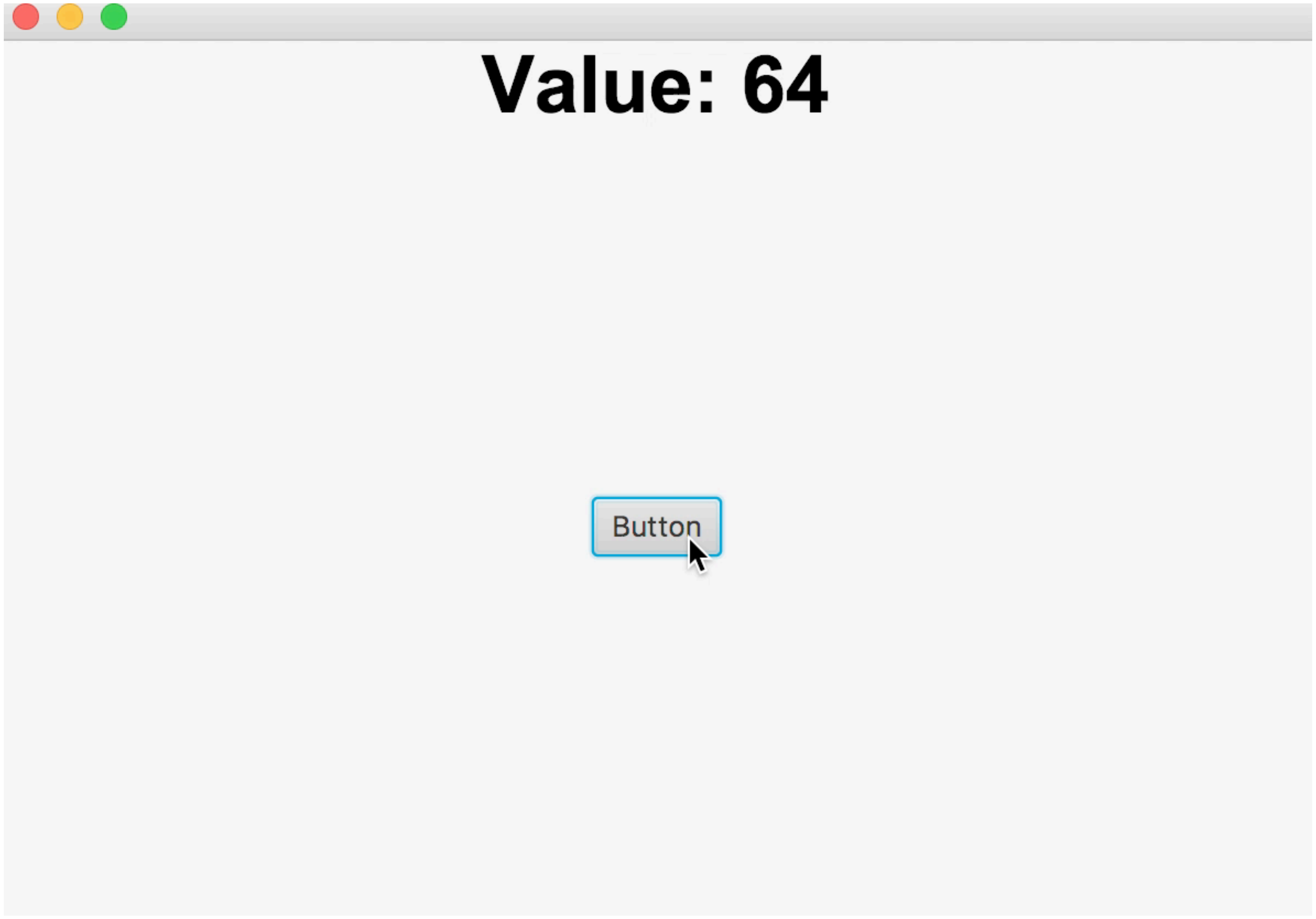


```
4:17:13 PM
```

Using SceneBuilder

1. Make a class that extends **`Application`** and implements **`Initializable`**
2. Create an FXML file, open in SceneBuilder
3. Lower left, Controller: set "Controller class" to your class from drop-down list
4. For any element you wish to access in code…
   a) Create an instance variable of the appropriate type, annotate with @FXML
   b) Click element; right, code: set "fx:id" to variable name from drop-down list
5. For any events to handle, either…
   a) Choose instance method via "On Action"; OR
   b) Register event handler in **`initialize`** method
6. Fill in **`start`** method for FXML load

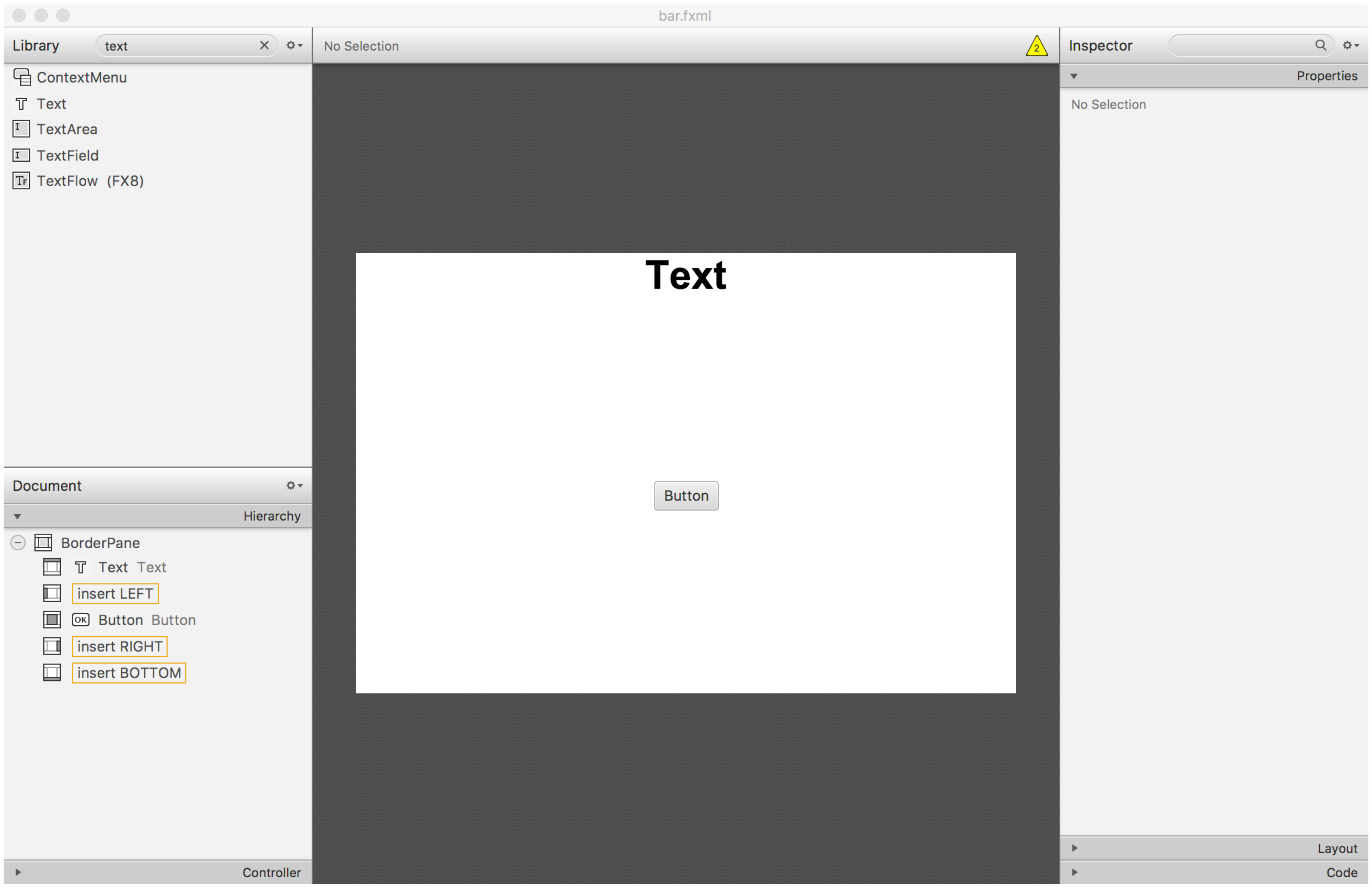
Button -> Random Text


Value: 64

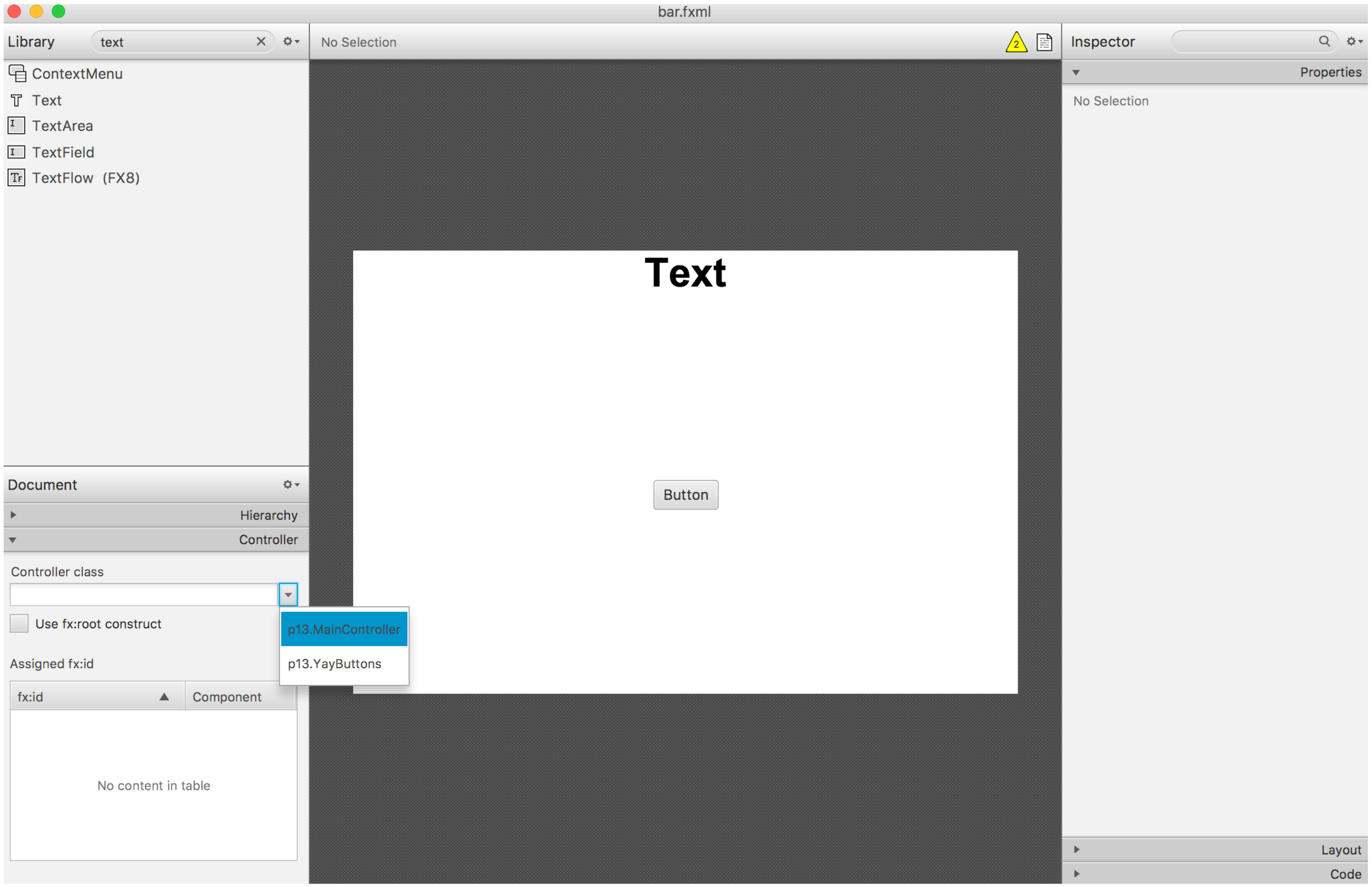
Example (1)

```
public class MainController extends Application implements Initializable {
    @Override public void start(Stage primaryStage) throws Exception {
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
    }

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

Example (2)


bar.fxml
Library
text
ContextMenu
Text
TextArea
TextField
TextFlow (FX8)
No Selection
Text
Button
Inspector
Properties
No Selection
Layout
Code
Document
Hierarchy
BorderPane
Text Text
insert LEFT
Button Button
insert RIGHT
insert BOTTOM
Controller

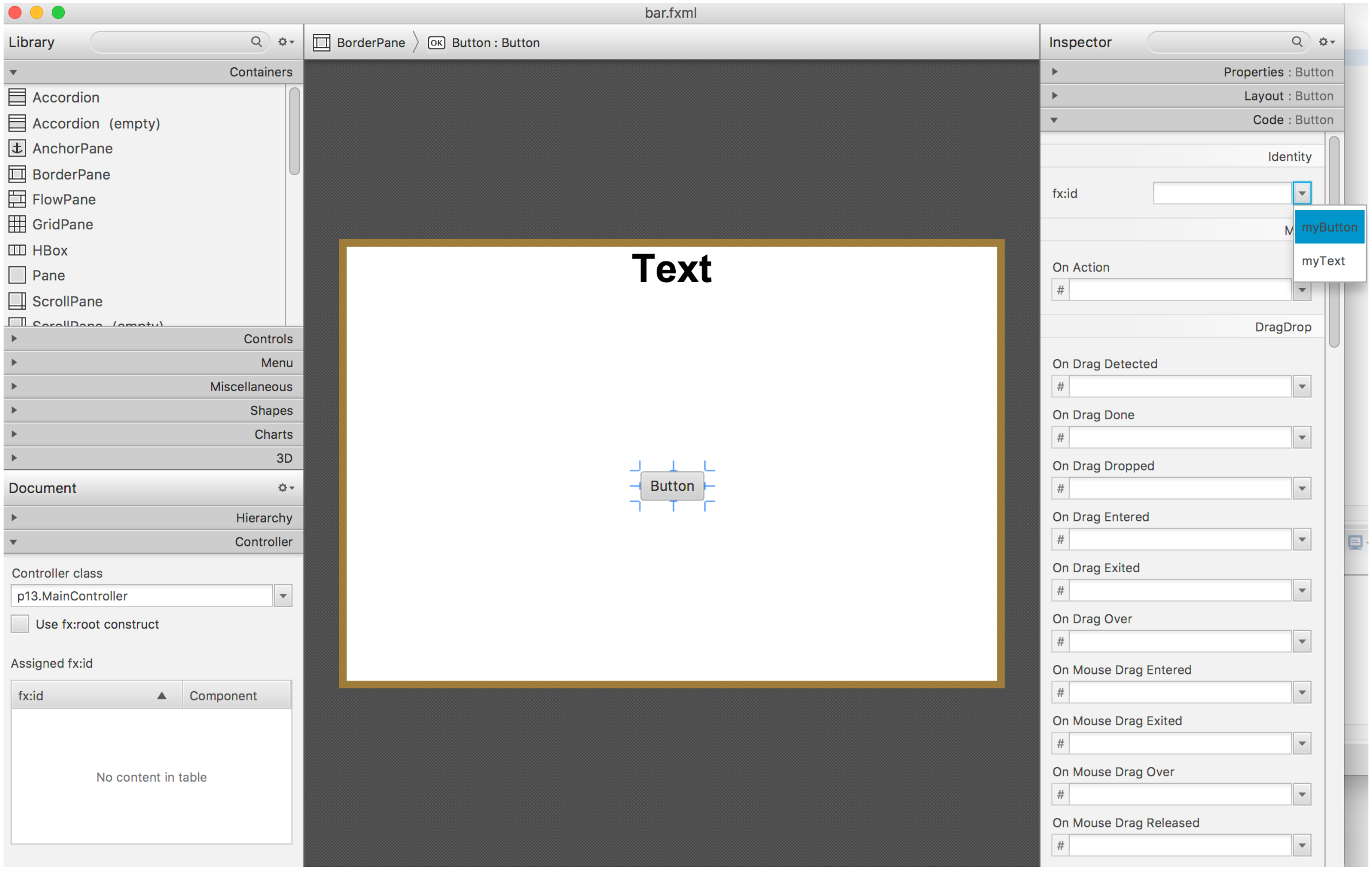
Example (3)


bar.fxml
Library
text
ContextMenu
Text
TextArea
TextField
TextFlow (FX8)
No Selection
Inspector
Properties
No Selection
Text
Button
Document
Hierarchy
Controller
Controller class
p13.MainController
p13.YayButtons
Use fx:root construct
Assigned fx:id
fx:id
Component
No content in table
Layout
Code

Example (4a)

```
public class MainController extends Application implements Initializable {
    @FXML
    Button myButton;

    @FXML
    Text myText;

    @Override
    public void start(Stage primaryStage) throws Exception {
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
    }

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

Example (4b)



Controller class: p13.MainController

Use fx:root construct

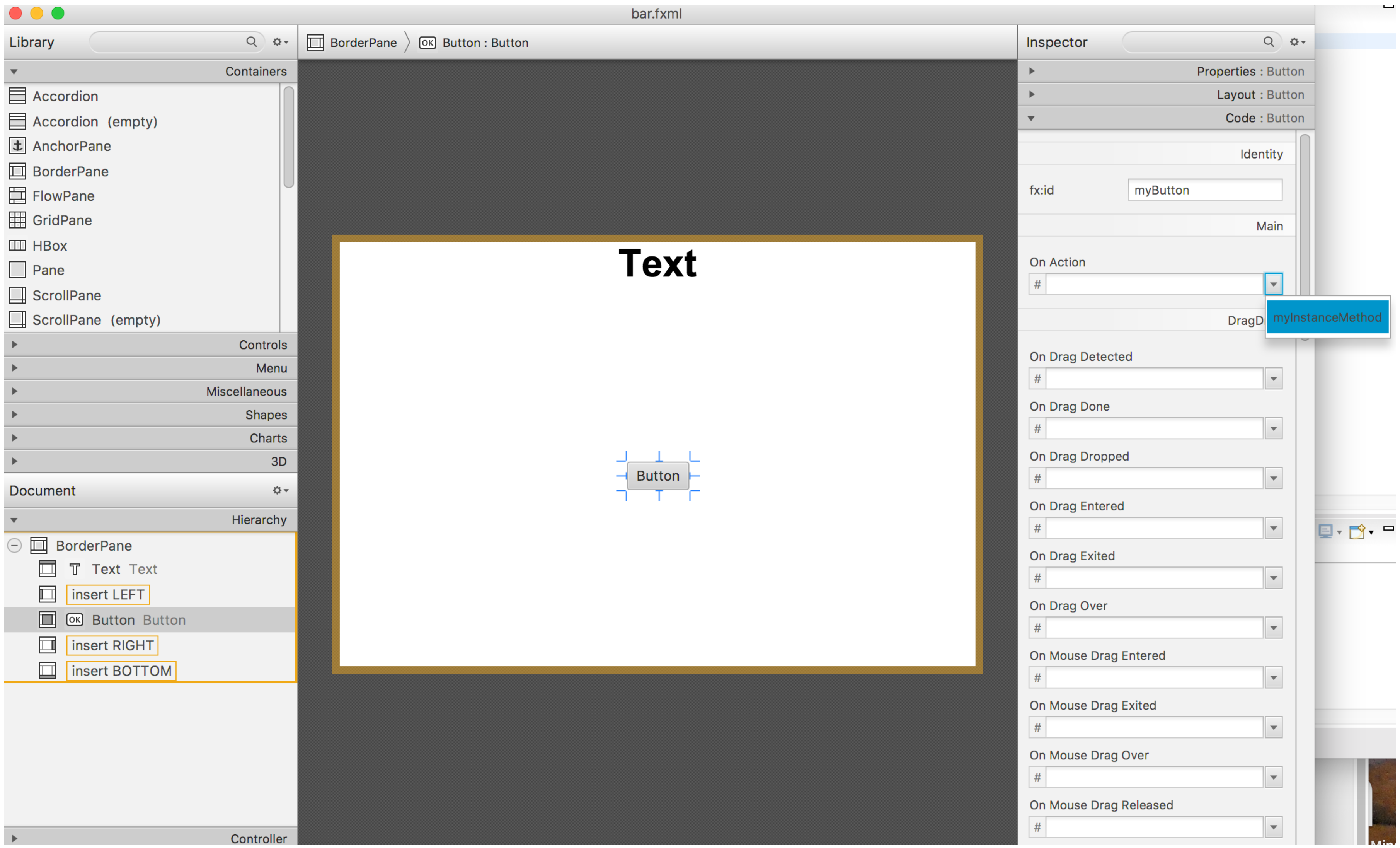
Assigned fx:id

fx:id	Component
No content in table	

fx:id: myButton / myText

On Action: #

Example (5a)



fx:id myButton

On Action #

myInstanceMethod

Example (5b)

```
public class MainController extends Application implements Initializable {
	@FXML
	Button myButton;

	@FXML
	Text myText;

	@Override
	public void start(Stage primaryStage) throws Exception {
	}

	@Override
	public void initialize(URL location, ResourceBundle resources) {
		myButton.setOnAction(e->{
			myText.setText(String.format("Value: %d", (new Random()).nextInt(100)));
		});
	}

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

Example (6)

```
public class MainController extends Application implements Initializable {
	@FXML
	Button myButton;

	@FXML
	Text myText;

	@Override
	public void start(Stage primaryStage) throws Exception {
		final FXMLLoader loader = new FXMLLoader(getClass().getResource("bar.fxml"));
		final Pane p = loader.load();

		primaryStage.setScene(new Scene(p));
		primaryStage.show();
	}

	@Override
	public void initialize(URL location, ResourceBundle resources) {
		myButton.setOnAction(e->{
			myText.setText(String.format("Value: %d", (new Random()).nextInt(100)));
		});
	}

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

Take Home Points

- Event-driven programming is a way of delegating responsibility for handling an event to a class
- JavaFX makes heavy use of this model via the EventHandler interface
- You now have the basics to create interactive GUIs via code and/or SceneBuilder