GUI

Lecture Question

Write a class that will react to mouse clicks

- In a package named gui, write a class named LectureMouseHandler that

  - Can be used to react to mouse clicks if added to a GUI as an event listener

  - Takes an Int as a constructor parameter

  - Has a method named currentValue(): Int that returns the value of the constructor parameter

  - Adds the x value of each mouse click to this value

This class is hard!

- If you are feeling overwhelmed

- Relax

- Everyone is feeling overwhelmed

- Hang in there!

Project

- Talk about their project demo 2 at some point

- Teams, but individually graded

The Library

- ScalaFX

  - An interface for JavaFX

  - Allows Scala specific features to be used with JavaFX

- Find the xml for the library and add it to your pom.xml

- Documentation for ScalaFX is lacking

- Documentation for JavaFX is extensive!

  - Look up JavaFX to find new elements

  - Understand the concepts

  - Conver to Scala syntax

GUI

- Lets make a degree converter

  - Input degrees in Fahrenheit

  - Click button to compute and display degrees in Celsius

- We'll build up to this one step at a time
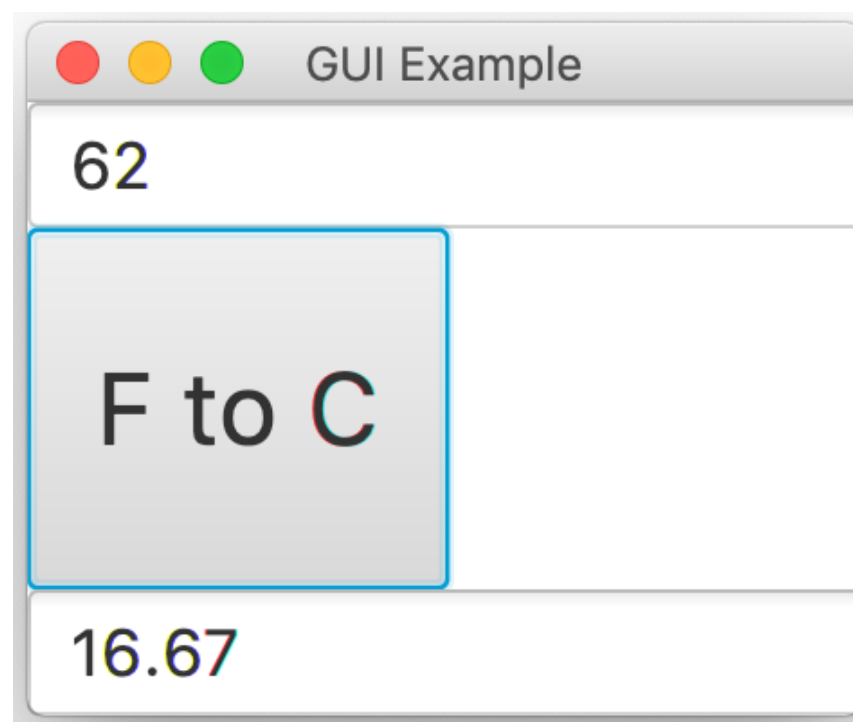
Initializer Code

- This syntax will be used extensively when working with ScalaFX

  - Compacts object creation code

- Add an initializer block of code when creating a new object

  - Valid syntax for any object creation

- Initializer block is in the scope of the object being created

  - Variable **y** is the instance variable **sampleVector.y**

  - Can use **this.y** with the same result

```
val sampleVector: PhysicsVector = new PhysicsVector(1,2,3){
  y=10
}
println(sampleVector)
```

(1.0, 10.0, 3.0)

GUI - JFXApp

- Extend the JFXApp trait

- JFXApp contains a main method and will setup the GUI window

```scala
import scalafx.application.JFXApp
import scalafx.application.JFXApp.PrimaryStage
import scalafx.scene.Scene


object SampleGUI extends JFXApp {

  this.stage = new PrimaryStage {
    title = "Degree Converter"
    scene = new Scene() {
      content = List(
        // GUI Elements will be added here
      )
    }
  }

}
```
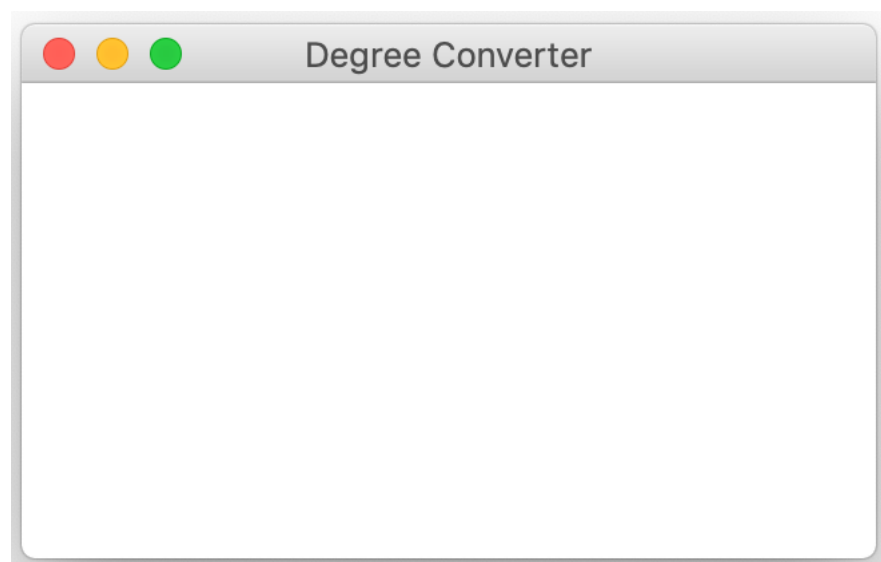
GUI - JFXApp

- Import all relavent code from the scalars library

- Must add to pom and install first

```scala
import scalafx.application.JFXApp
import scalafx.application.JFXApp.PrimaryStage
import scalafx.scene.Scene


object SampleGUI extends JFXApp {

  this.stage = new PrimaryStage {
    title = "Degree Converter"
    scene = new Scene() {
      content = List(
        // GUI Elements will be added here
      )
    }
  }

}
```
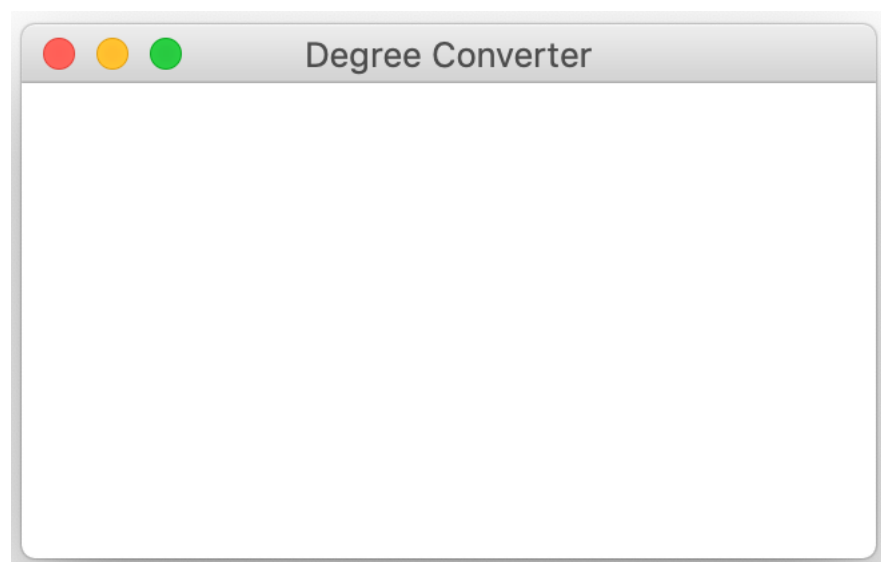

Degree Converter

GUI - JFXApp

- Extend JFXApp from ScalaFX

- JFXApp has a state variable named stage

  - Set the stage to determine what will be displayed

```scala
import scalafx.application.JFXApp
import scalafx.application.JFXApp.PrimaryStage
import scalafx.scene.Scene


object SampleGUI extends JFXApp {

  this.stage = new PrimaryStage {
    title = "Degree Converter"
    scene = new Scene() {
      content = List(
        // GUI Elements will be added here
      )
    }
  }

}
```
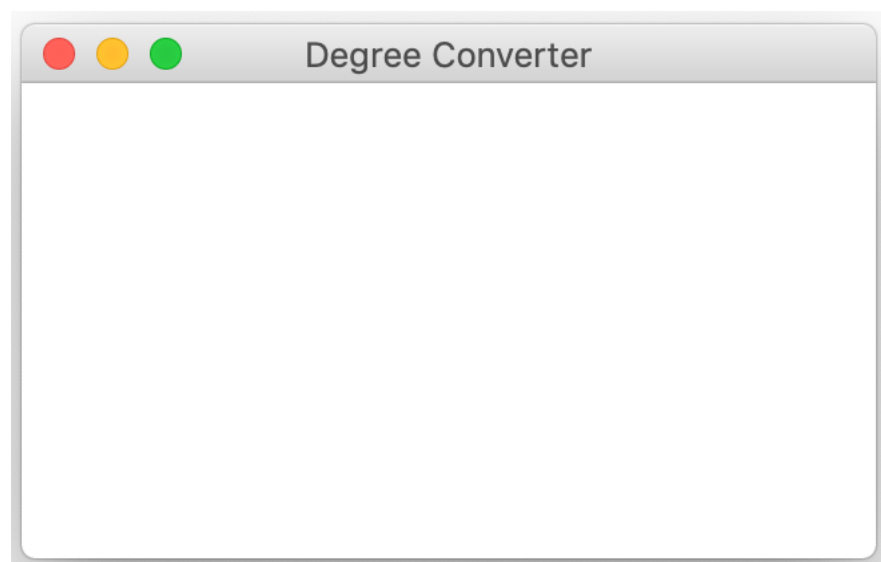
GUI - JFXApp

- Create a new PrimaryStage

- Use an initializer block to set state variables of the stage

- Title is displayed at the top of the window

- Scene will contain all GUI elements to be displayed

```scala
import scalafx.application.JFXApp
import scalafx.application.JFXApp.PrimaryStage
import scalafx.scene.Scene


object SampleGUI extends JFXApp {

  this.stage = new PrimaryStage {
    title = "Degree Converter"
    scene = new Scene() {
      content = List(
        // GUI Elements will be added here
      )
    }
  }

}
```
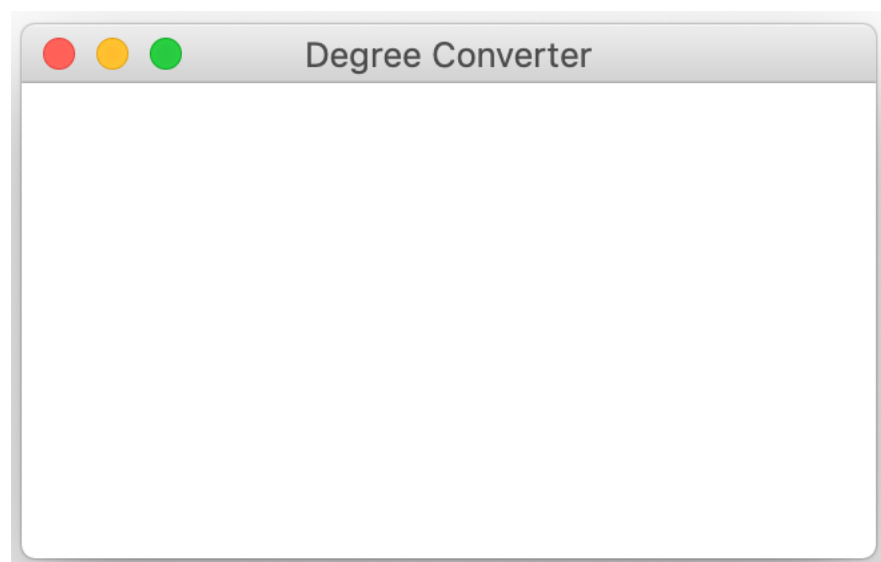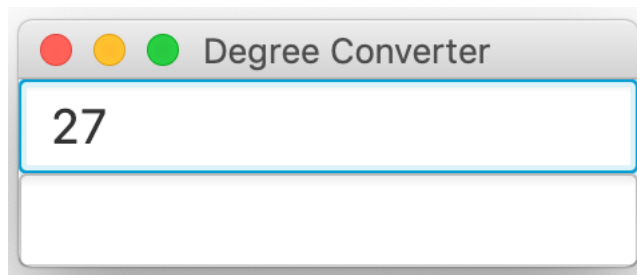
GUI - TextField

- We'll add the two text fields to our GUI

- Allow the user to edit one text field to enter a number

- Do not allow the user to edit the other text field to use it for output only


```scala
import scalafx.application.JFXApp
import scalafx.application.JFXApp.PrimaryStage
import scalafx.scene.Scene
import scalafx.scene.control.TextField
import scalafx.scene.layout.VBox


object SampleGUI extends JFXApp {

  val inputDisplay: TextField = new TextField {
    style = "-fx-font: 18 ariel;"
  }

  val outputDisplay: TextField = new TextField {
    editable = false
    style = "-fx-font: 18 ariel;"
  }

  val verticalBox = new VBox(){
    children = List(inputDisplay, outputDisplay)
  }

  this.stage = new PrimaryStage {
    title = "Degree Converter"
    scene = new Scene() {
      content = List(
        verticalBox
      )
    }
  }

}
```
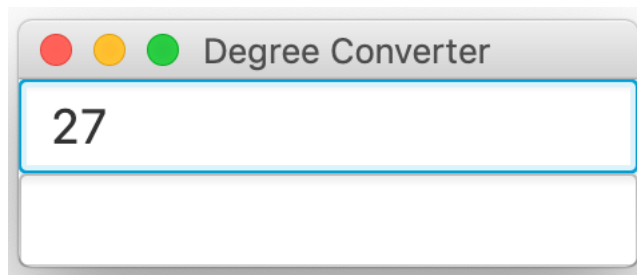
GUI - TextField

- **Important!**

- **Always import from the scalafx package**

  - Only a few exceptions that we'll see later

- The javafx package contains many classes with the same names

  - Auto-import will list these as options. Do not choose them!

```scala
import scalafx.application.JFXApp
import scalafx.application.JFXApp.PrimaryStage
import scalafx.scene.Scene
import scalafx.scene.control.TextField
import scalafx.scene.layout.VBox


object SampleGUI extends JFXApp {

  val inputDisplay: TextField = new TextField {
    style = "-fx-font: 18 ariel;"
  }

  val outputDisplay: TextField = new TextField {
    editable = false
    style = "-fx-font: 18 ariel;"
  }

  val verticalBox = new VBox(){
    children = List(inputDisplay, outputDisplay)
  }

  this.stage = new PrimaryStage {
    title = "Degree Converter"
    scene = new Scene() {
      content = List(
        verticalBox
      )
    }
  }

}
```

Degree Converter

27

GUI - TextField

- Create text fields using initializer blocks to set any state variables we'd like

- Set the style to change the font of the text in the field

- For the output text field

  - Set the editable variable to false so the user cannot edit the text in this field

```scala
import scalafx.application.JFXApp
import scalafx.application.JFXApp.PrimaryStage
import scalafx.scene.Scene
import scalafx.scene.control.TextField
import scalafx.scene.layout.VBox


object SampleGUI extends JFXApp {

  val inputDisplay: TextField = new TextField {
    style = "-fx-font: 18 ariel;"
  }

  val outputDisplay: TextField = new TextField {
    editable = false
    style = "-fx-font: 18 ariel;"
  }

  val verticalBox = new VBox(){
    children = List(inputDisplay, outputDisplay)
  }

  this.stage = new PrimaryStage {
    title = "Degree Converter"
    scene = new Scene() {
      content = List(
        verticalBox
      )
    }
  }

}
```
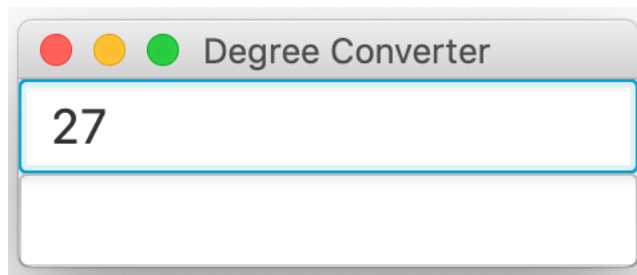
Degree Converter

27

GUI - TextField

- Don't add elements directly to the GUI

- Control the layout of the elements by adding the to a container

  - We'll use VBox to stack the elements vertically

- Add the elements as a List to the children variable of the VBox

```scala
import scalafx.application.JFXApp
import scalafx.application.JFXApp.PrimaryStage
import scalafx.scene.Scene
import scalafx.scene.control.TextField
import scalafx.scene.layout.VBox


object SampleGUI extends JFXApp {

  val inputDisplay: TextField = new TextField {
    style = "-fx-font: 18 ariel;"
  }

  val outputDisplay: TextField = new TextField {
    editable = false
    style = "-fx-font: 18 ariel;"
  }

  val verticalBox = new VBox(){
    children = List(inputDisplay, outputDisplay)
  }

  this.stage = new PrimaryStage {
    title = "Degree Converter"
    scene = new Scene() {
      content = List(
        verticalBox
      )
    }
  }
}
```
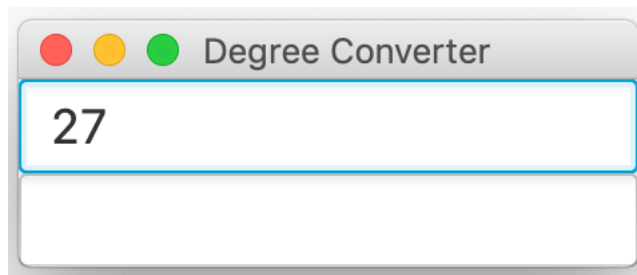
Degree Converter

27

GUI - TextField

- Add the container to our scene

- Scene has a variable named content that takes a list of GUI elements/ containers

  - Anything implementing jfxs.Node

```scala
import scalafx.application.JFXApp
import scalafx.application.JFXApp.PrimaryStage
import scalafx.scene.Scene
import scalafx.scene.control.TextField
import scalafx.scene.layout.VBox


object SampleGUI extends JFXApp {

  val inputDisplay: TextField = new TextField {
    style = "-fx-font: 18 ariel;"
  }

  val outputDisplay: TextField = new TextField {
    editable = false
    style = "-fx-font: 18 ariel;"
  }

  val verticalBox = new VBox(){
    children = List(inputDisplay, outputDisplay)
  }

  this.stage = new PrimaryStage {
    title = "Degree Converter"
    scene = new Scene() {
      content = List(verticalBox)
    }
  }
}
```
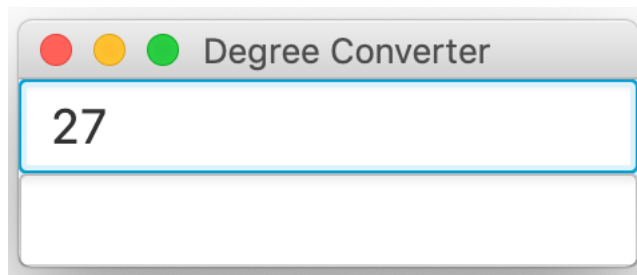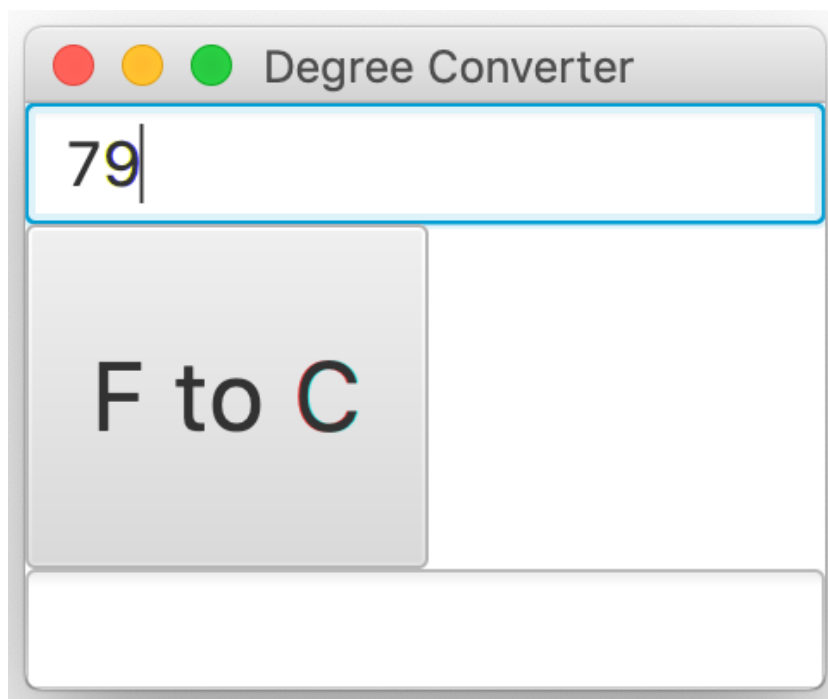
Degree Converter

27

GUI - Button

- Add a button using the same syntax as the text fields

- Use minWidth and minHeight to control the size of the button

```
object SampleGUI extends JFXApp {

  val inputDisplay: TextField = new TextField {
    style = "-fx-font: 18 ariel;"
  }

  val outputDisplay: TextField = new TextField {
    editable = false
    style = "-fx-font: 18 ariel;"
  }

  val button: Button = new Button {
    minWidth = 100
    minHeight = 100
    style = "-fx-font: 28 ariel;"
    text = "F to C"
  }

  val verticalBox: VBox = new VBox(){
    children = List(inputDisplay, button, outputDisplay)
  }

  this.stage = new PrimaryStage {
    title = "Degree Converter"
    scene = new Scene() {
      content = List(verticalBox)
    }
  }

}
```

GUI - Button

- Fair enough

  - We have a full GUI with all the right elements..

- But the button doesn't do anything!

```scala
object SampleGUI extends JFXApp {

  val inputDisplay: TextField = new TextField {
    style = "-fx-font: 18 ariel;"
  }

  val outputDisplay: TextField = new TextField {
    editable = false
    style = "-fx-font: 18 ariel;"
  }

  val button: Button = new Button {
    minWidth = 100
    minHeight = 100
    style = "-fx-font: 28 ariel;"
    text = "F to C"
  }

  val verticalBox: VBox = new VBox(){
    children = List(inputDisplay, button, outputDisplay)
  }

  this.stage = new PrimaryStage {
    title = "Degree Converter"
    scene = new Scene() {
      content = List(verticalBox)
    }
  }

}
```
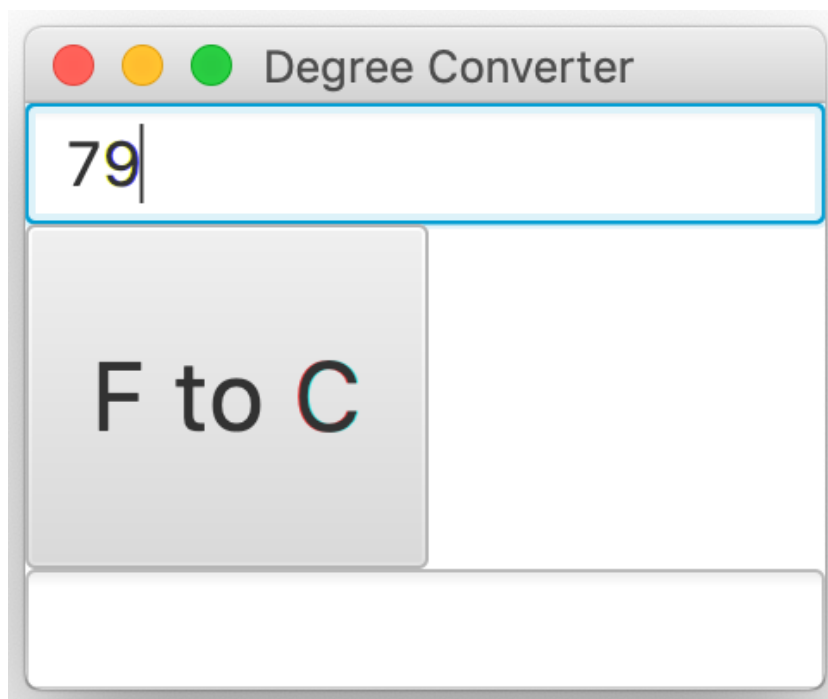
GUI - Button

```
val button: Button = new Button {
  minWidth = 100
  minHeight = 100
  style = "-fx-font: 28 ariel;"
  text = "F to C"
  onAction = new ButtonListener(inputDisplay, outputDisplay)
}
```

- The Button class contains an onAction variable

- This variable determines what the button does when clicked

- Clicking the button creates an **ActionEvent**

  - This event is sent onAction whenever the button is clicked

- onAction must be of type **EventHandler[ActionEvent]**

- Create a ButtonListener (defined on next slide) and pass it references to both text fields

GUI - Button

```scala
import javafx.event.{ActionEvent, EventHandler}
import scalafx.scene.control.TextField

class ButtonListener(inputDisplay: TextField, outputDisplay: TextField) extends EventHandler[ActionEvent] {

  override def handle(event: ActionEvent): Unit = {
    val fahrenheit: Double = inputDisplay.text.value.toDouble
    val celsius = this.fahrenheitToCelsius(fahrenheit)
    outputDisplay.text.value = f"$celsius%1.2f"
  }

  def fahrenheitToCelsius(degreesFahrenheit: Double): Double = {
    val degreesCelsius = (degreesFahrenheit − 32.0) * 5.0 / 9.0
    degreesCelsius
  }

}
```

- Our ButtonListener class will react to button presses

GUI - Button

```scala
import javafx.event.{ActionEvent, EventHandler}
import scalafx.scene.control.TextField

class ButtonListener(inputDisplay: TextField, outputDisplay: TextField) extends EventHandler[ActionEvent] {

  override def handle(event: ActionEvent): Unit = {
    val fahrenheit: Double = inputDisplay.text.value.toDouble
    val celsius = this.fahrenheitToCelsius(fahrenheit)
    outputDisplay.text.value = f"$celsius%1.2f"
  }

  def fahrenheitToCelsius(degreesFahrenheit: Double): Double = {
    val degreesCelsius = (degreesFahrenheit - 32.0) * 5.0 / 9.0
    degreesCelsius
  }

}
```

- ButtonListener extends EventHandler[ActionEvent]

  - This is the type of Button.onAction

- **Important**: EventHandler and ActionEvent must be imported from javafx! (Not scalafx)

  - This applies to all event-based code

GUI - Button

```scala
import javafx.event.{ActionEvent, EventHandler}
import scalafx.scene.control.TextField

class ButtonListener(inputDisplay: TextField, outputDisplay: TextField) extends EventHandler[ActionEvent] {

  override def handle(event: ActionEvent): Unit = {
    val fahrenheit: Double = inputDisplay.text.value.toDouble
    val celsius = this.fahrenheitToCelsius(fahrenheit)
    outputDisplay.text.value = f"$celsius%1.2f"
  }

  def fahrenheitToCelsius(degreesFahrenheit: Double): Double = {
    val degreesCelsius = (degreesFahrenheit - 32.0) * 5.0 / 9.0
    degreesCelsius
  }

}
```

- EventHandler contains a method named handle that we'll override

  - This method is called when our button is pressed

- Input is an instance of the event that was created when the button was pressed

  - The event contains information about the user action

GUI - Button

```scala
import javafx.event.{ActionEvent, EventHandler}
import scalafx.scene.control.TextField

class ButtonListener(inputDisplay: TextField, outputDisplay: TextField) extends EventHandler[ActionEvent] {

  override def handle(event: ActionEvent): Unit = {
    val fahrenheit: Double = inputDisplay.text.value.toDouble
    val celsius = this.fahrenheitToCelsius(fahrenheit)
    outputDisplay.text.value = f"$celsius%1.2f"
  }

  def fahrenheitToCelsius(degreesFahrenheit: Double): Double = {
    val degreesCelsius = (degreesFahrenheit - 32.0) * 5.0 / 9.0
    degreesCelsius
  }

}
```

- Use text.value to get/set the text displayed on the text fields

GUI - Button

```scala
import javafx.event.{ActionEvent, EventHandler}
import scalafx.scene.control.TextField

class ButtonListener(inputDisplay: TextField, outputDisplay: TextField) extends EventHandler[ActionEvent] {

  override def handle(event: ActionEvent): Unit = {
    val fahrenheit: Double = inputDisplay.text.value.toDouble
    val celsius = this.fahrenheitToCelsius(fahrenheit)
    outputDisplay.text.value = f"$celsius%1.2f"
  }

  def fahrenheitToCelsius(degreesFahrenheit: Double): Double = {
    val degreesCelsius = (degreesFahrenheit - 32.0) * 5.0 / 9.0
    degreesCelsius
  }

}
```

- Since this is a full class, we can create additional state/ behavior as desired

- Here we create a helper method for the degree conversion to reduce clutter in the handle method

GUI - Button

```scala
import javafx.event.{ActionEvent, EventHandler}
import scalafx.scene.control.TextField

class ButtonListener(inputDisplay: TextField, outputDisplay: TextField) extends EventHandler[ActionEvent] {

  override def handle(event: ActionEvent): Unit = {
    val fahrenheit: Double = inputDisplay.text.value.toDouble
    val celsius = this.fahrenheitToCelsius(fahrenheit)
    outputDisplay.text.value = f"$celsius%1.2f"
  }

  def fahrenheitToCelsius(degreesFahrenheit: Double): Double = {
    val degreesCelsius = (degreesFahrenheit - 32.0) * 5.0 / 9.0
    degreesCelsius
  }

}
```

- We can now instantiate this class whenever we want a button with this behavior when clicked

GUI - Button

- And our degree converter is complete!
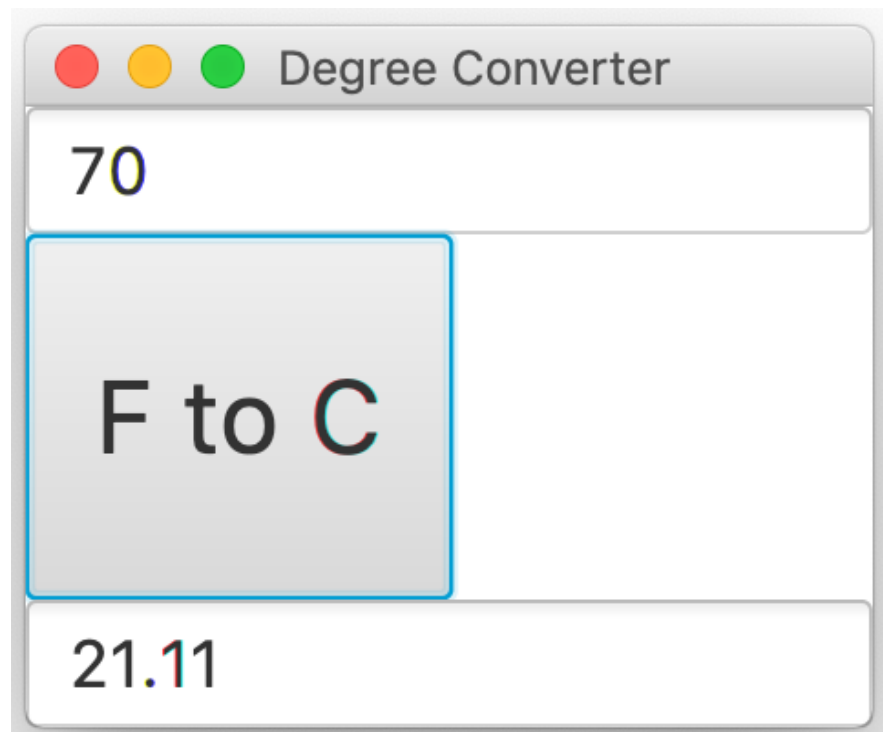

```scala
object SampleGUI extends JFXApp {

  val inputDisplay: TextField = new TextField {
    style = "-fx-font: 18 ariel;"
  }

  val outputDisplay: TextField = new TextField {
    editable = false
    style = "-fx-font: 18 ariel;"
  }

  val button: Button = new Button {
    minWidth = 100
    minHeight = 100
    style = "-fx-font: 28 ariel;"
    text = "F to C"
    onAction = new ButtonListener(inputDisplay, outputDisplay)
  }

  val verticalBox: VBox = new VBox(){
    children = List(inputDisplay, button, outputDisplay)
  }

  this.stage = new PrimaryStage {
    title = "Degree Converter"
    scene = new Scene() {
      content = List(verticalBox)
    }
  }
}
```

Buttons Are Cool

But How Does This Help With Our Project?

Graphics - 2D

- Your project needs some graphics

  - Not just GUI elements like buttons and boxes

- Let's add some simple shapes to a GUI and make them move

- **Note**: This is not an art class. You will never be graded on the aesthetics of your work as long as we can tell what is happening

  - Examples in class will use simple shapes with solid colors. If your entire project looks similar this is fine

  - If you want to add sprites or models, that's fine too but it will not improve your grade. You are graded on functionality, not graphical fidelity

Graphics - 2D

- Coordinate System has inverted y-axis

- Upper left corner is the origin for an element (screen/window)

Graphics - 2D

- Add Shapes to a GUI instead of buttons/text fields

- Circle and Rectangle both extend Shape

```
new Circle {
    centerX = 20.0
    centerY = 50.0
    radius = 20.0
    fill = Color.Green
}
```

```
new Rectangle {
    width = 60.0
    height = 40.0
    translateX = 60.0
    translateY = 10.0
    fill = Color.Blue
}
```

Graphics - 2D

- Circle

  - Defined by center (from upper-left corner of the screen) and radius

```
new Circle {
  centerX = 20.0
  centerY = 50.0
  radius = 20.0
  fill = Color.Green
}
```

- Rectangle

  - Defined by height, width, and translation of upper-left corner (from upper-left corner of the screen)

```
new Rectangle {
  width = 60.0
  height = 40.0
  translateX = 60.0
  translateY = 10.0
  fill = Color.Blue
}
```

Graphics - 2D

- Can add shapes directly to the Scene

- Better organization to add graphics to a new element and add that element to the Scene

  - We'll use a Group for all graphical elements

```scala
var sceneGraphics: Group = new Group {}

val circle: Circle = new Circle {
  centerX = 20.0
  centerY = 50.0
  radius = 20.0
  fill = Color.Green
}
sceneGraphics.children.add(circle)

val rectangle: Rectangle = new Rectangle {
  width = 60.0
  height = 40.0
  translateX = 60.0
  translateY = 10.0
  fill = Color.Blue
}
sceneGraphics.children.add(rectangle)

...

scene = new Scene(windowWidth, windowHeight) {
  content = List(sceneGraphics)
}
```

Graphics - 2D

- A full example GUI using 2d graphics

```scala
object GUI2D extends JFXApp {

  val windowWidth: Double = 800
  val windowHeight: Double = 600

  val playerCircleRadius:Double = 20

  var allRectangles: List[Shape] = List()
  var sceneGraphics: Group = new Group {}

  val player: Circle = new Circle {
    centerX = Math.random() * windowWidth
    centerY = Math.random() * windowHeight
    radius = playerCircleRadius
    fill = Color.Green
  }

  sceneGraphics.children.add(player)

  this.stage = new PrimaryStage {
    this.title = "2D Graphics"
    scene = new Scene(windowWidth, windowHeight) {
      content = List(sceneGraphics)
    }

    val update: Long => Unit = (time: Long) => {
      for (shape <- allRectangles) {
        shape.rotate.value += 0.5
      }
    }

    AnimationTimer(update).start()
  }

}
```

Graphics - 2D

- Set the height and width of the scene is pixels

- Create a player as a blue circle

- Add the graphics to the scene as a group

```scala
object GUI2D extends JFXApp {

  val windowWidth: Double = 800
  val windowHeight: Double = 600

  val playerCircleRadius:Double = 20

  var allRectangles: List[Shape] = List()
  var sceneGraphics: Group = new Group {}

  val player: Circle = new Circle {
    centerX = Math.random() * windowWidth
    centerY = Math.random() * windowHeight
    radius = playerCircleRadius
    fill = Color.Green
  }

  sceneGraphics.children.add(player)

  this.stage = new PrimaryStage {
    this.title = "2D Graphics"
    scene = new Scene(windowWidth, windowHeight) {
      content = List(sceneGraphics)
    }

    val update: Long => Unit = (time: Long) => {
      for (shape <- allRectangles) {
        shape.rotate.value += 0.5
      }
    }

    AnimationTimer(update).start()
  }

}
```

Graphics - 2D

- We'll get to the rectangles later

- First

  - What is this AnimationTimer?

  - What is that strange type of the update variable?

```scala
object GUI2D extends JFXApp {

  val windowWidth: Double = 800
  val windowHeight: Double = 600

  val playerCircleRadius:Double = 20

  var allRectangles: List[Shape] = List()
  var sceneGraphics: Group = new Group {}

  val player: Circle = new Circle {
    centerX = Math.random() * windowWidth
    centerY = Math.random() * windowHeight
    radius = playerCircleRadius
    fill = Color.Green
  }

  sceneGraphics.children.add(player)

  this.stage = new PrimaryStage {
    this.title = "2D Graphics"
    scene = new Scene(windowWidth, windowHeight) {
      content = List(sceneGraphics)
    }

    val update: Long => Unit = (time: Long) => {
      for (shape <- allRectangles) {
        shape.rotate.value += 0.5
      }
    }

    AnimationTimer(update).start()
  }

}
```

Graphics - 2D

- AnimationTimer

- We'll get to the rectangles later

- First

  - What is this AnimationTimer?

  - What is that strange type of the update variable?

```scala
object GUI2D extends JFXApp {

  val windowWidth: Double = 800
  val windowHeight: Double = 600

  val playerCircleRadius:Double = 20

  var allRectangles: List[Shape] = List()
  var sceneGraphics: Group = new Group {}

  val player: Circle = new Circle {
    centerX = Math.random() * windowWidth
    centerY = Math.random() * windowHeight
    radius = playerCircleRadius
    fill = Color.Green
  }

  sceneGraphics.children.add(player)

  this.stage = new PrimaryStage {
    this.title = "2D Graphics"
    scene = new Scene(windowWidth, windowHeight) {
      content = List(sceneGraphics)
    }

    val update: Long => Unit = (time: Long) => {
      for (shape <- allRectangles) {
        shape.rotate.value += 0.5
      }
    }

    AnimationTimer(update).start()
  }

}
```

Graphics - Animation

- The **update** variable stores a function

  - Yes, you can do that!

- The type of a function is:

  - All the input types

  - An arrow =>

  - The output type

- The type of the update variable is a function that takes a Long as a parameter and outputs Unit

```scala
// define a function for the action timer (Could also use a method)
// Rotate all rectangles (relies on frame rate. lag will slow rotation)
val update: Long => Unit = (time: Long) => {
  for (shape <- allRectangles) {
    shape.rotate.value += 0.5
  }
}

// Start Animations. Calls update 60 times per second (takes update as an argument)
AnimationTimer(update).start()
```

Graphics - Animation

- We can define a function using syntax similar to creating a method

  - The input parameters in parentheses

  - An arrow => (as opposed to just = for methods)

  - The function body in braces {}

    - Can omit the braces for 1 line functions

```scala
// define a function for the action timer (Could also use a method)
// Rotate all rectangles (relies on frame rate. lag will slow rotation)
val update: Long => Unit = (time: Long) => {
  for (shape <- allRectangles) {
    shape.rotate.value += 0.5
  }
}

// Start Animations. Calls update 60 times per second (takes update as an argument)
AnimationTimer(update).start()
```

Graphics - Animation

- **ActionTimer** is used for animations on a GUI

- Create and start the timer to start animations

```
// define a function for the action timer (Could also use a method)
// Rotate all rectangles (relies on frame rate. lag will slow rotation)
val update: Long => Unit = (time: Long) => {
  for (shape <- allRectangles) {
    shape.rotate.value += 0.5
  }
}

// Start Animations. Calls update 60 times per second (takes update as an argument)
AnimationTimer(update).start()
```

Graphics - Animation

- **ActionTimer** constructor takes a function (or method) as an argument of type (Long) => Unit

  - This function is called 60 times per second (If possible)

  - The long is the current epoch time in nanoseconds

- The update function will be called 60 times/second

```scala
// define a function for the action timer (Could also use a method)
// Rotate all rectangles (relies on frame rate. lag will slow rotation)
val update: Long => Unit = (time: Long) => {
  for (shape <- allRectangles) {
    shape.rotate.value += 0.5
  }
}

// Start Animations. Calls update 60 times per second (takes update as an argument)
AnimationTimer(update).start()
```

Graphics - User Inputs

- Cool.. but it looks like update was rotating rectangles. We don't have any rectangles!

- Let's allow the user to add rectangles by clicking the GUI

```scala
class MouseEventHandler() extends EventHandler[MouseEvent] {

  val rectangleWidth: Double = 60
  val rectangleHeight: Double = 40

  override def handle(event: MouseEvent): Unit = {
    drawRectangle(event.getX, event.getY)
  }

  def drawRectangle(centerX: Double, centerY: Double): Unit = {
    val newRectangle = new Rectangle() {
      width = rectangleWidth
      height = rectangleHeight
      translateX = centerX - rectangleWidth / 2.0
      translateY = centerY - rectangleHeight / 2.0
      fill = Color.Blue
    }
    GUI2D.sceneGraphics.children.add(newRectangle)
    GUI2D.allRectangles = newRectangle :: GUI2D.allRectangles
  }

}
```

```scala
scene = new Scene(windowWidth, windowHeight) {
  // add an EventHandler[MouseEvent] to draw a rectangle when the player clicks the screen
  addEventHandler(MouseEvent.MOUSE_CLICKED, new MouseEventHandler())
}
```

Graphics - User Inputs

- We'll add an event handler directly to the Scene

- Specify the type of event to be handled
    - Mouse clicks in this example

- Provide an EventHandler that can handle that type of event

- Remember: Use javafx types for events and event handlers

```scala
class MouseEventHandler() extends EventHandler[MouseEvent] {

  val rectangleWidth: Double = 60
  val rectangleHeight: Double = 40

  override def handle(event: MouseEvent): Unit = {
    drawRectangle(event.getX, event.getY)
  }

  def drawRectangle(centerX: Double, centerY: Double): Unit = {
    val newRectangle = new Rectangle() {
      width = rectangleWidth
      height = rectangleHeight
      translateX = centerX - rectangleWidth / 2.0
      translateY = centerY - rectangleHeight / 2.0
      fill = Color.Blue
    }
    GUI2D.sceneGraphics.children.add(newRectangle)
    GUI2D.allRectangles = newRectangle :: GUI2D.allRectangles
  }

}
```

```scala
scene = new Scene(windowWidth, windowHeight) {
  // add an EventHandler[MouseEvent] to draw a rectangle when the player clicks the screen
  addEventHandler(MouseEvent.MOUSE_CLICKED, new MouseEventHandler())
}
```

Graphics - User Inputs

- Override handle(event_type) just like we did for action events on button clicks

- Since this is a mouse event we can access the (x, y) location of the click

- Add a rectangle at that location

  - We access the GUI2D object in this example to add the rectangles to the GUI

```scala
class MouseEventHandler() extends EventHandler[MouseEvent] {

  val rectangleWidth: Double = 60
  val rectangleHeight: Double = 40

  override def handle(event: MouseEvent): Unit = {
    drawRectangle(event.getX, event.getY)
  }

  def drawRectangle(centerX: Double, centerY: Double): Unit = {
    val newRectangle = new Rectangle() {
      width = rectangleWidth
      height = rectangleHeight
      translateX = centerX - rectangleWidth / 2.0
      translateY = centerY - rectangleHeight / 2.0
      fill = Color.Blue
    }
    GUI2D.sceneGraphics.children.add(newRectangle)
    GUI2D.allRectangles = newRectangle :: GUI2D.allRectangles
  }

}
```

```scala
scene = new Scene(windowWidth, windowHeight) {
  // add an EventHandler[MouseEvent] to draw a rectangle when the player clicks the screen
  addEventHandler(MouseEvent.MOUSE_CLICKED, new MouseEventHandler())
}
```

Graphics - Animation

- Now, when a user clicks the GUI it adds a rectangle

- Since we rotate all rectangles in update, these rectangles will rotate

- This example shows how to use the concepts

  - In a full app, you would apply physics or other behavior in update

```
// define a function for the action timer (Could also use a method)
// Rotate all rectangles (relies on frame rate. lag will slow rotation)
val update: Long => Unit = (time: Long) => {
  for (shape <- allRectangles) {
    shape.rotate.value += 0.5
  }
}

// Start Animations. Calls update 60 times per second (takes update as an argument)
AnimationTimer(update).start()
```

Graphics - User Inputs

- Similar concept to handle keyboard inputs to move the player

```scala
class KeyEventHandler(player: Circle) extends EventHandler[KeyEvent]{

  val playerSpeed: Int = 10

  override def handle(event: KeyEvent): Unit = {
    keyPressed(event.getCode)
  }

  def keyPressed(keyCode: KeyCode): Unit = {
    keyCode.getName match {
      case "W" => player.translateY.value -= playerSpeed
      case "A" => player.translateX.value -= playerSpeed
      case "S" => player.translateY.value += playerSpeed
      case "D" => player.translateX.value += playerSpeed
      case _ => println(keyCode.getName + " pressed with no action")
    }
  }

}
```

```scala
scene = new Scene(windowWidth, windowHeight) {
  content = List(sceneGraphics)
  // add an EventHandler[KeyEvent] to control player movement
  addEventHandler(KeyEvent.KEY_PRESSED, new KeyEventHandler(player))
}
```

Graphics - User Inputs

- Inherit the EventHandler[KeyEvent] class for keyboard inputs

  - Listen for key events {KEY_PRESSED, KEY_RELEASED, KEY_TYPED}

- Each event has a key code identifying which key was used

```scala
class KeyEventHandler(player: Circle) extends EventHandler[KeyEvent]{

  val playerSpeed: Int = 10

  override def handle(event: KeyEvent): Unit = {
    keyPressed(event.getCode)
  }

  def keyPressed(keyCode: KeyCode): Unit = {
    keyCode.getName match {
      case "W" => player.translateY.value -= playerSpeed
      case "A" => player.translateX.value -= playerSpeed
      case "S" => player.translateY.value += playerSpeed
      case "D" => player.translateX.value += playerSpeed
      case _ => println(keyCode.getName + " pressed with no action")
    }
  }

}
```

```scala
scene = new Scene(windowWidth, windowHeight) {
  content = List(sceneGraphics)
  // add an EventHandler[KeyEvent] to control player movement
  addEventHandler(KeyEvent.KEY_PRESSED, new KeyEventHandler(player))
}
```

Graphics - User Inputs

- Use match/case to react to different keys

  - Similar to switch/case in other languages

- Use underscore for a default case

```scala
class KeyEventHandler(player: Circle) extends EventHandler[KeyEvent]{

  val playerSpeed: Int = 10

  override def handle(event: KeyEvent): Unit = {
    keyPressed(event.getCode)
  }

  def keyPressed(keyCode: KeyCode): Unit = {
    keyCode.getName match {
      case "W" => player.translateY.value -= playerSpeed
      case "A" => player.translateX.value -= playerSpeed
      case "S" => player.translateY.value += playerSpeed
      case "D" => player.translateX.value += playerSpeed
      case _ => println(keyCode.getName + " pressed with no action")
    }
  }

}
```

```scala
scene = new Scene(windowWidth, windowHeight) {
  content = List(sceneGraphics)
  // add an EventHandler[KeyEvent] to control player movement
  addEventHandler(KeyEvent.KEY_PRESSED, new KeyEventHandler(player))
}
```

Lecture Question

Write a class that will react to mouse clicks

- In a package named gui, write a class named LectureMouseHandler that

  - Can be used to react to mouse clicks if added to a GUI as an event listener

  - Takes an Int as a constructor parameter

  - Has a method named currentValue(): Int that returns the value of the constructor parameter

  - Adds the x value of each mouse click to this value

Graphics - 3D

How to draw an owl

1.

2.

1. Draw some circles   2. Draw the rest of the focking owl

Graphics - 3D

- ScalaFX has 3D shapes

    - Add to group same as 2D shapes

    - Add a material to set the color

- Create a 5x5x5 grey cube at (0, -2.5, -10)

- Coordinates

    - Up is negative y (Gravity is positive)

```scala
new Box(5, 5, 5) {
  material = new PhongMaterial(Color.Grey)
  drawMode = DrawMode.Fill
  transforms.add(new Translate(0, -2.5, -10))
}
```

Graphics - 3D

- Add a camera to the scene to control the player's view

```
var camera_ = new PerspectiveCamera(true) {
  transforms.addAll(new Rotate(0, Rotate.YAxis), new Rotate(0, Rotate.XAxis), new Translate(-5, -3, -50))
}

...

this.scene = new Scene(root, windowWidth, windowHeight) {
  fill = Color.AliceBlue
  camera = camera_
}
```