DURHAM COLLEGE
SUCCESS MATTERS

OOP 3200 – Object Oriented Programming II

Week 14 – Java: JavaFX Continued

Week 14 Overview

- ❖ In-class Exercise 13 (Due Friday)
- ❖ Java Quiz 2 (Due Sunday)
- ❖ Java Lab 5 (Due Sunday)
- ❖ **JavaFX (Continued)**

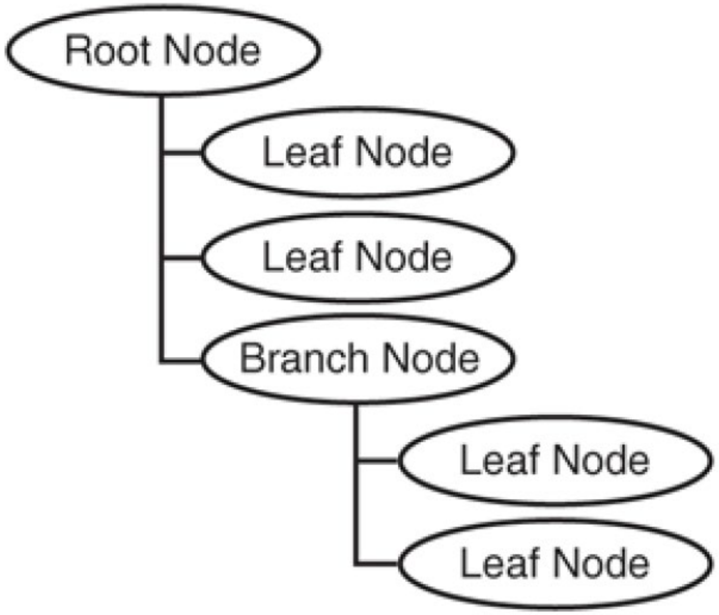
Course Outline (continued)

Week	Date	Topic	Evaluation	Weight
13	Dec 09, 2020	OVERLOADING in JAVA - Describe overloading methods and constructors - Increase knowledge of more complex class creation - Explain static methods and variables - Introduce the concept of inheritance COMPLEX JAVA CLASS CREATION - Discuss inheritance & Polymorphism - Define the terms superclass and subclass - Explain Overriding superclass methods	In-Class Exercises 9: (2%) Java Assignment 4: (6%)	8
14	Dec 16, 2020	JAVA TEMPLATE CLASS CONSTRUCTORS - Abstract classes - Interfaces & Polymorphism - Universal Superclass - Casting Objects	In-Class Exercises 10: (2%) Java Assignment 5: (6%) Java Quiz 1: (5%)	13

JavaFX Continued

JavaFX Review

- Last Week
  - Learned about JavaFX Scene Graph and Node Structure
  - **`Application`** Class
  - **`Stage`** Class (controls the Window)
  - **`Scene`** Class (think of this as the drawing Canvas)
  - Layout Containers (**`HBox`**, **`VBox`**, **`GridPane`**)
  - Basic Controls (**`Label`**, **`Button`**)



JavaFX Continued

- This Week
  - Aligning objects in **HBox** or **VBox** container
  - Spacing objects in an **HBox** or **VBox** container
  - Padding objects in an **HBox** or **VBox** container
  - **GridPane** container (Spacing and padding)
  - Displaying Images (**Image** and **ImageView**)
  - **Button** Controls and Events
  - Reading Input with **TextField** Controls

Aligning Controls in an Hbox Layout Container

- The default alignment of an Hbox layout container is the Left Position.
- You can change the alignment of an **HBox** by calling its **`setAlignment`** method.
- For example, assume **hbox** references an **HBox** object. The following statement causes all controls in the hbox container to be **centered** within the container:

```
hbox.setAlignment(Pos.CENTER);
```

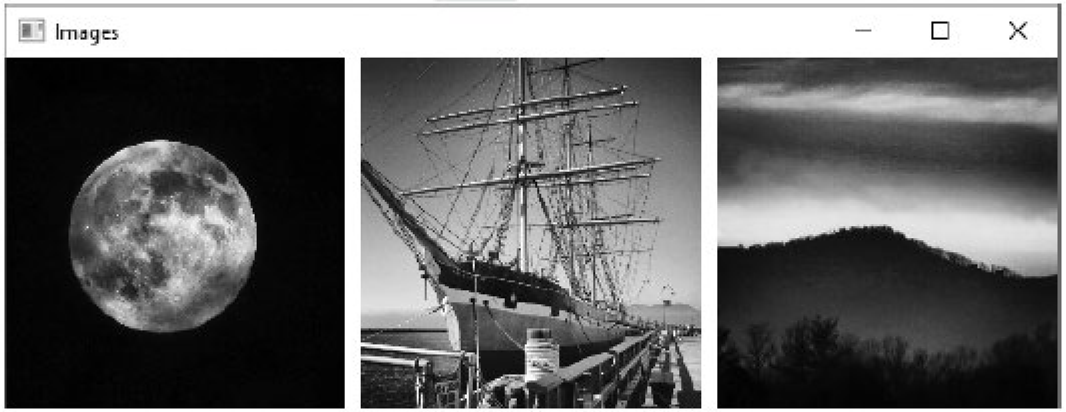
- The argument you pass to the **`setAlignment`** method is an **enum** constant (e.g., TOP_LEFT, TOP_CENTER, TOP_RIGHT, etc.)
- The **Pos** type must be imported from the **`javafx.geometry`** package.

Spacing

- If we want some space to appear between the controls in an **HBox** , we can optionally pass a **spacing value** as the first argument to the **HBox** constructor.

```
HBox hbox = new HBox(10, moonIView, shipIView, sunsetIView);
```

- If the first argument passed to the **HBox** constructor is a **number**, that value is used as the number of pixels to appear between the controls horizontally in the container.



Padding

- **Padding** is space that appears around the inside edge of a container.

- If you want to add padding to an **HBox** , you call the **HBox** object's **`setPadding`** method.

- The **`setPadding`** method takes an **`Insets`** object as its argument.

- When constructing the **`Insets`** object, pass a **number of pixels** as an argument to the constructor.

- For example, the following statement adds **`10 pixels`** of padding to an **Hbox** container named **hbox** :

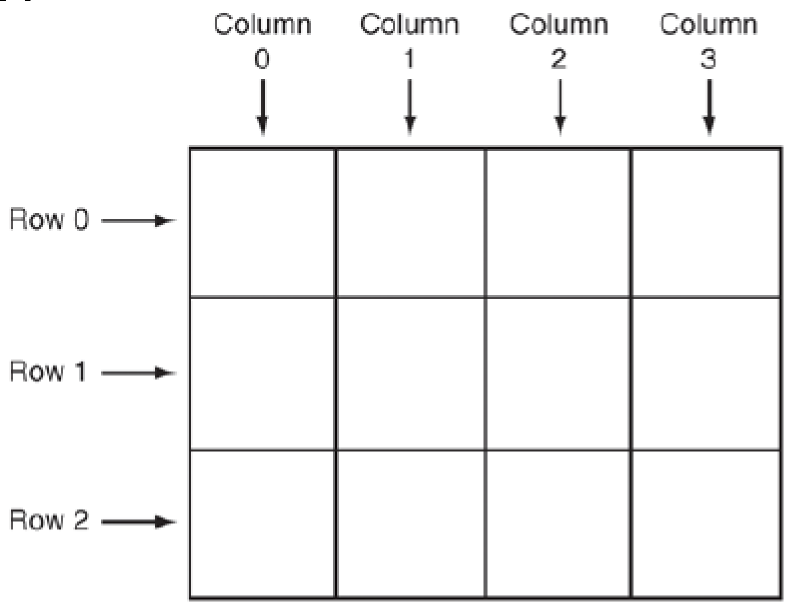
```
hbox.setPadding(new Insets(10));
```

The GridPane Layout Container

- The **GridPane** layout container arranges its contents in a grid with **columns** and **rows**. As a result, a **GridPane** is divided into **cells**, much like a spreadsheet.

- As shown in figure below, the **columns** and **rows** are identified by indexes, beginning at **0**.

- You use these **column** and **row** indexes to refer to specific cells within the **GridPane** .



The **GridPane** Layout Container (continued)

- If you want to create a **GridPane**, you import the **GridPane** class from the **javafx.scene**. layout package.

- Then, you create an instance of the **GridPane** class using its no-arg constructor, as shown here:

```
GridPane gridpane = new GridPane();
```

The **GridPane** Layout Container (continued)

- Once you have created a **GridPane** object, you can call the object’s add method to add controls at specific positions in the grid.

```
gridPaneObject.add(control, column, row);
```

- In the general format, **gridPaneObject** is a reference to a **GridPane**, control is the control you are adding to the **GridPane**, column is the column index, and row is the row index.

GridPane Spacing

- By default, there is **no space** between the columns and rows in a **GridPane** .

- If you want space to appear between the columns in a **GridPane**, you can call the **GridPane** class's **setHgap** method.

- The argument you pass to the **setHgap** method is the number of pixels of space you want between the columns.

- Let's assume gridpane references a **GridPane** object.

```
gridpane.setHgap(10);
```

- This statement will cause **10 pixels** of space to appear between the columns of the **GridPane**.

GridPane Spacing (continued)

- If you want vertical space to appear between the rows in a **GridPane** , you can call the **GridPane** class's **setVgap** method.

- Once again, assume gridpane references a **GridPane** object.

- The following statement calls the **setHgap** method to display **10 pixels** of space between the rows of the **GridPane** object:

```
gridpane.setVgap(10);
```

`GridPane` Padding

- By default, there is **no padding** around the inside edge of a **`GridPane`** . However, you can add padding by calling the **`GridPane`** object's **`setPadding`** method.

- Recall that the **`setPadding`** method takes an Insets object as its argument. When constructing the **`Insets`** object, pass a **number of pixels** as an argument to the constructor.

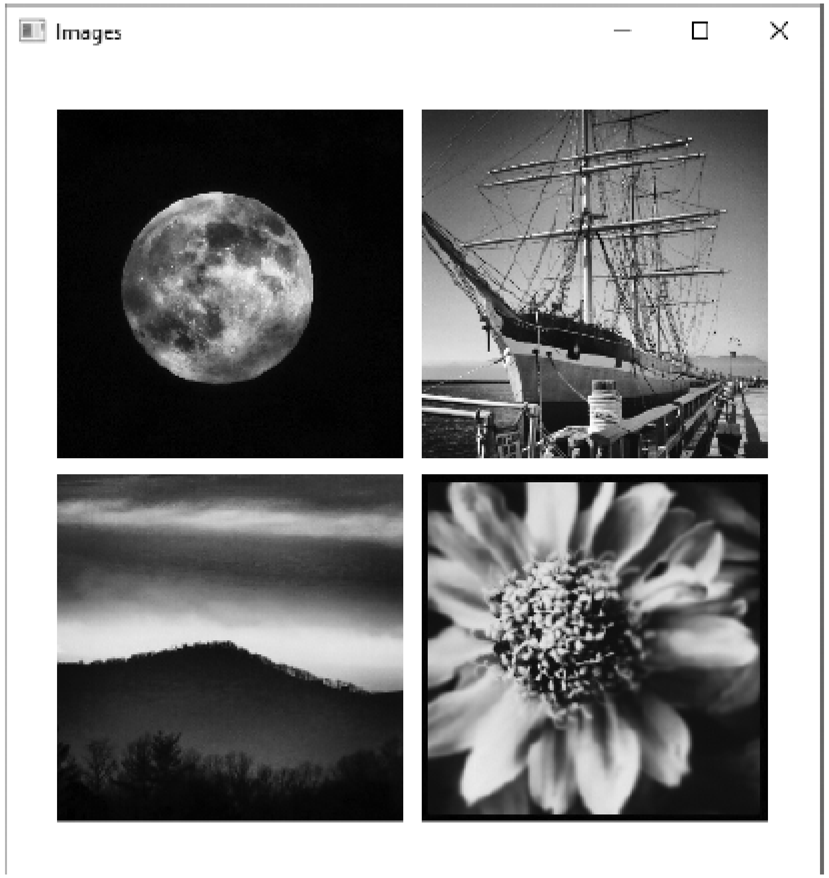
- For example, the following statement adds **`10 pixels`** of padding to a **`GridPane`** container named **`gridpane`** :

- **`gridpane.setPadding(new Insets(10));`**

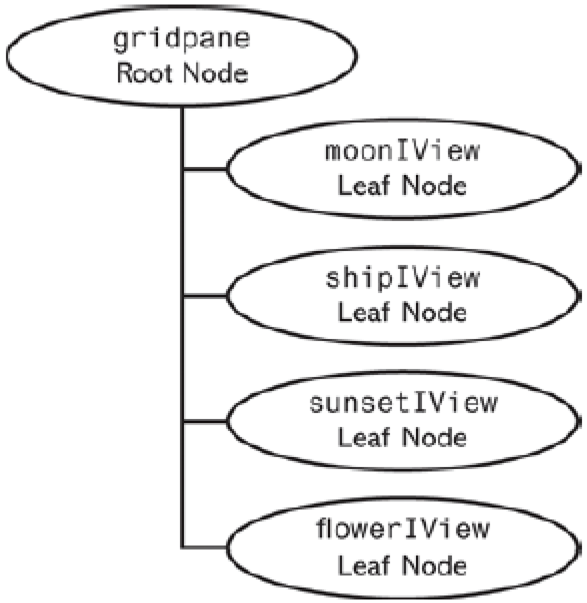
- Recall that the Insets class is in the **`javafx.geometry`** package, so be sure to include the necessary import statement in your program.

GridPane Padding (continued)

Images in a **GridPane**



GridPane Scene Graph



Displaying Images

- Displaying an image in a JavaFX application is a two-step process:
  - you load the image into memory
  - you display the image

- This requires two classes from the JavaFX library:
  - **`Image`**
  - **`ImageView`** .

- Both of these classes are in the **`javafx.scene.image`** package.

Displaying Images (continued)

- The **Image** class can load an image from the computer's local file system, or from an Internet location.
- The class supports the **BMP**, **JPEG**, **GIF**, and **PNG** file types.
- To load an image of any of these types, create an instance of the **Image** class, passing the constructor a **string argument** that specifies the file's name and location.

```
Image image = new Image("file:HotAirBalloon.jpg");
```

- In this example, the **string** that we are passing to the constructor specifies a file named **HotAirBalloon.jpg**.
- The part of the string that reads **file:** is a **protocol identifier** indicating that the file is on the local computer.

Displaying Images (continued)

- Because no path was given, it is assumed the file is in the same directory or folder as the program's executable **`.class`** file.

- If you want to load an image from a different location, you can specify a path.

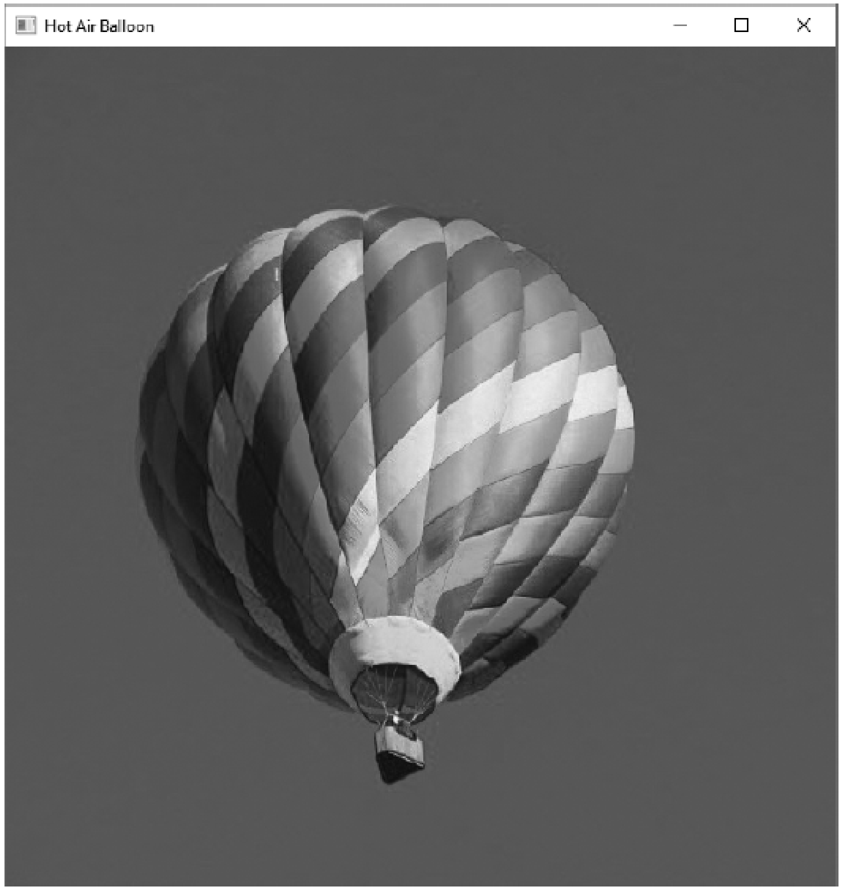
```
Image image = new Image("file:C:\\Images\\HotAirBalloon.jpg");
```

- Next, you create an **instance** of the **`ImageView`** class, passing a reference to the **`Image`** object as an argument to the constructor.

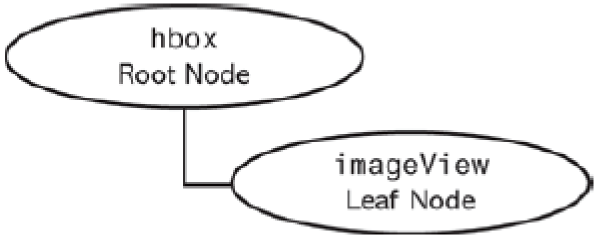
```
ImageView imageView = new ImageView(image);
```

Displaying Images (continued)

`Image` and **`ImageView`** Example Output



Scene Graph for Displaying an **`Image`**



Loading Images from an Internet Location

- The **`string`** that you pass as an argument to the **`Image`** class constructor specifies a **protocol** used to load the **image**.

- The example you saw in previous code snippet uses the **`file:`** protocol to load the image from the local file system.

- If you want to load an image from an Internet location, you can use the **`http:`** protocol.

```
Image image = new Image("http://www.somelocation.com/images/HotAirBalloon.jpg");
```

Setting the Size of an Image

- By default, the ImageView class displays an image at its full size.

- You can resize the image with the **`ImageView`** class's **`setFitWidth`** and **`setFitHeight`** methods.

- Let's assume **`myImageView`** references an **`ImageView`** object.

```
myImageView.setFitWidth(100);
myImageView.setFitHeight(100);
```

- After this code executes, the image size of the **`ImageView`** object will be set to **`100 pixels`** wide by **`100 pixels`** high.

Preserving the Image's Aspect Ratio

- The ImageView class has a method named **`setPreserveRatio`** that you can call to make sure an image's aspect ratio is preserved.

- Let's assume that **`myImageView`** references an **`ImageView`** object.

```
myImageView.setPreserveRatio(true);
```

Changing an `ImageView`'s Image

- You can change the image that is displayed by an **`ImageView`** object by calling the object's **`setImage`** method.
- The **`setImage`** method takes, as an argument, a reference to an Image object you want to display.
- For example, let's assume **`myImageView`** references an **`ImageView`** object, and **`myImage`** references an Image object.
- The following code shows an example of calling the **`setImage`** method:

```
myImageView.setImage(myImage);
```

Button Controls and Events

- A **Button** is a rectangular control that appears as a button with a caption written across its face.
- Typically, when the user clicks a **Button** control (either with a mouse, or by touching it on a touch screen), an action takes place.
- To create a **Button** control, you will use the Button class, which is in the **javafx.scene.control** package.
- Once you have written the necessary import statement, you will create an instance of the **Button** class.

```
Button myButton = new Button("Click Me");
```

Handling Events

- An **event** is an **action** that takes place while a program is running.
- For example, each time the user clicks a **`Button`** control, an event takes place.
- When an **event** takes place, the control responsible for the event creates an **event object** that contains information about the event.
- The GUI control that created the event object is known as the **event source**.
- Event objects are instances of the **`Event`** class (from the **`javafx.event`** package), or one of its subclasses.
- When a **`Button`** control is clicked, an **event object** that is an instance of the **`ActionEvent`** class is created.
- **`ActionEvent`** is a subclass of the **`Event`** class, and it is also in the **`javafx.event`** package.

Handling Events (continued)

- But what happens to the event object once it is generated by an event source?
- It’s possible the event source is connected to one or more **event handlers**.
- An **event handler** is an object that responds to events.
- If an **event source** is connected to an **event handler**, a specific method in the event handler is called and the event object is passed as an argument to the method.
- This process is sometimes referred to as **event firing**.

Writing Event Handlers

- When you are writing a GUI application, it is your responsibility to write the **event handler** classes that your application needs.

- When you write an event handler class, it must implement the **EventHandler** interface, which is in the **`javafx.event`** package.

- The **EventHandler** interface specifies a **void method** named **`handle`**.

- The **`handle`** method must have a parameter of the **`Event`** class, or one of its subclasses.

Writing Event Handlers (continued)

- For example, suppose we want to write an **event handler** class that can respond to **`Button`** clicks.

- Recall that when a **`Button`** control is clicked, the event object that is generated is of the **`ActionEvent`** type.

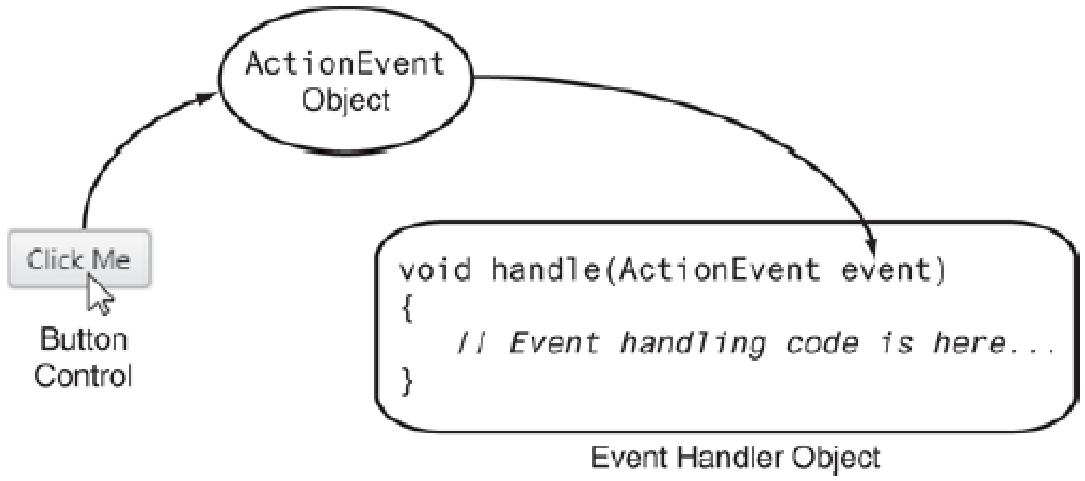
- This is the general format of an event handler class that handles events of the **`ActionEvent`** type:

```
class ButtonClickHandler implements EventHandler<ActionEvent>
{
    @Override
    public void handle(ActionEvent event)
    {
        // Write event handling code here.
    }
}
```

Writing Event Handlers (continued)

- Once you have written an event **handler** class, you create an object of that class, and **connect** the event handler object with a **control**.

- When a **`Button`** control is clicked, it generates an **`ActionEvent`** object, and it automatically executes the handle method of the event handler object to which it is connected, passing the **`ActionEvent`** object as an argument.



```
void handle(ActionEvent event)
{
   // Event handling code is here...
}
```

Registering an Event Handler

- Once you have written an event handler class, you create an object of the class, and connect the object to a control.
- The process of **connecting an event handler** object to a control is called **registering the event handler**.
- **`Button`** controls have a method named **`setOnAction`** , which is used for registering event handlers.
- You simply call the **`Button`** ’s **`setOnAction`** method, passing a reference to the event handler object as an argument.
- This will connect the **`Button`** control to the event handler object

Registering an Event Handler (continued)

❖ For example, suppose we create a **`Button`** control named **`myButton`**, and we have written an event handler class named **`ButtonClickHandler`**.

❖ The following code registers an object of the **`ButtonClickHandler`** class with the **`Button`** control:

```
myButton.setOnAction(new ButtonClickHandler());
```

❖ Now, when the user clicks on the **`Button`** control, the **`ButtonClickHandler`** object’s handle method will be automatically executed.

Reading Input with `TextField` Controls

- One of the primary controls that you will use to get data from the user is the **`TextField`** control.

- A **`TextField`** control appears as a rectangular area. When the application is running, the user can type text into a **`TextField`** control.

- In the program, you can retrieve the text that the user entered and use that text in any necessary operations.

Reading Input with **`TextField`** Controls (continued)

- To create a **`TextField`** control, you will use the **`TextField`** class, which is in the **`javafx.scene.control`** package.

- Once you have written the necessary import statement, you will create an instance of the **`TextField`** class.

- The following statement shows an example:

```
TextField myTextField = new TextField();
```

Reading Input with **`TextField`** Controls (continued)

- This statement above creates an empty **`TextField`**.

- Optionally, you can pass a string to the constructor to display initial text in the **`TextField`**

```
TextField myTextField = new TextField("Example data");
```

Reading Input with **`TextField`** Controls (continued)

- To retrieve the text that the user has typed into a **TextField** control, you call the control's **`getText`** method. The method returns the value that has been entered into the **`TextField`** as a **`String`**.

- For example, assume **`myTextField`** is a **`TextField`** control.

- The following code reads the value that has been entered into **`myTextField`** , and assigns it to a **`String`** variable named `input` :

```
String input;
input = myTextField.getText();
```