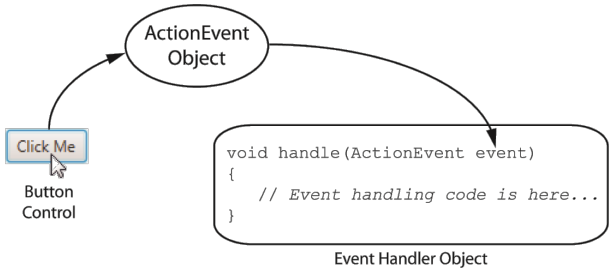
Topic 6: JavaFX

Part b: Event Based Programming

Events and EventHandlers

Handling Events

- An *event* is an action that takes place within a program, such as the clicking of a button.
- When an event takes place, the control that is responsible for the event creates an *event object* in memory.
- The event object contains information about the event.
- The control that generated the event object is know as the *event source*.
- It is possible that the event source is connected to one or more event listeners.



Event Objects

- Event objects are instances of the `Event` class (from the `javafx.event` package), or one of its subclasses.
- For example, a `Button` click generates an `ActionEvent` object. `ActionEvent` is a subclass of the `Event` class.

Event Handlers

- Event handlers are objects.
- You write event handler classes that implement the `EventHandler` interface (from the `javafx.event` package).
- The `EventHandler` interface specifies a `void` method named `handle` that has a parameter of the `Event` class (or one of its subclasses).

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

Registering an Event Handler

- The process of connecting an event handler object to a control is called *registering* the event handler.
- `Button` controls have a method named `setOnAction` that registers an event handler:

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

- When the user clicks the button, the event handler object's `handle` method will be executed.
- EventDemo.java
- Alternatively, every control in javaFX has a method called `addEventHandler` ithat allows us to specify the event that we would like to listen for.
- ImageHandler.java

Anon Classes/Lambda Expressions as Event Handlers

- Recall that a functional interface is an interface that has one, and only one, abstract method.
- The `EventHandler` interface has only one abstract method is a functional interface.
- Any time you are writing Java code to instantiate an anonymous class that implements a functional interface, you should consider using a lambda expression instead.
- A lambda expression is more concise than the code for instantiating an anonymous class.

Reading Inputs

… and more Layouts

Reading Input with `TextField` Controls

- At runtime, the user can type text into a `TextField` control.
- In the program, you can retrieve the text that the user entered.
- The `TextField` class is in the `javafx.scene.control` package.
- To create an empty `TextField`:

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

Reading Input with `TextField` Controls (2)

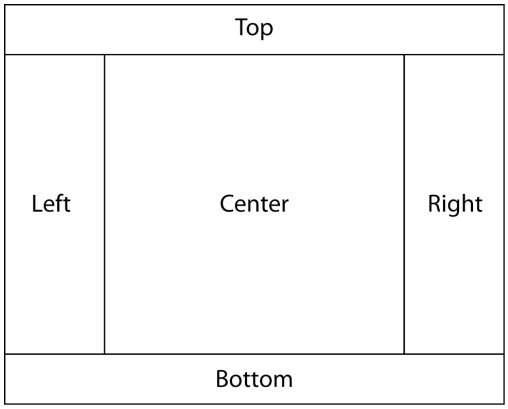
- To retrieve the text that the user has typed into a `TextField` control, call the control's `getText` method.
- The method returns the value that has been entered, as a `String`.
- Example:

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

- KiloConverter.java

The `BorderPane` Layout Container

- The `BorderPane` layout container manages controls in five regions:



- Only one object at a time may be placed into a `BorderPane` region.
- You do not usually put controls directly into a `BorderPane` region.
- Typically, you put controls into another type of layout container, then you put that container into one of the `BorderPane` regions.

The `BorderPane` Layout Container (2)

- The `BorderPane` class is in the `javafx.scene.layout` package.
- Summary of constructors:

Constructor	Description
`BorderPane()`	The no-arg constructor creates an empty BorderPane container.
`BorderPane(`*`center`*`)`	This constructor accepts one argument. The node that is passed as the argument is placed in the BorderPane's center region.
`BorderPane(`*`center`*`, `*`top`*`, `*`right`*`, `*`bottom`*`, `*`left`*`)`	This constructor accepts five nodes as arguments: one to place in each region.

The `BorderPane` Layout Container (3)

- The `BorderPane` class provides the following methods to add controls to specific regions:
  - `setCenter`
  - `setTop`
  - `setBottom`
  - `setLeft`
  - `setRight`
- BorderPaneDemo1.java

Observable List

The `ObservableList` Interface

- Most containers in the javaFX library implement the `ObservableList` interface.
  - It's a special type of list that can trigger an event handler any time an item in the list changes.

- Layout containers keep their children nodes in an `ObservableList`.
- All layout containers have a method named `getChildren()` that returns their `ObservableList` of nodes.

The `ObservableList` Interface (2)

- A few `ObservableList` methods:

Method	Description
`add(`*`item`*`)`	Adds a single item to the list. (This method is inherited from the Collection interface.)
`addAll(`*`item...`*`)`	Adds one or more items to the list, specified by the variable argument list.
`clear()`	Removes all of the items from the list.
`remove(`*`item`*`)`	Removes the object specified by *item* from the list. (This method is inherited from the Collection interface.)
`removeAll(`*`item...`*`)`	Removes one or more items to the list, specified by the variable argument list.
`setAll(`*`item...`*`)`	Clears the current contents of the list and adds all of the items specified by the variable argument list.
`size()`	Returns the number of items in the list. (This method is inherited from the Collection interface.)

The `ObservableList` Interface (3)

- Example: creating an empty `HBox`, then using the `ObservableList`'s `addAll` or `add` method to add nodes to the `HBox`:

```
hbox.getChildren().add(label1);
hbox.getChildren().addAll(label1, label2, label3)
```

- Removing `label1` from the `HBox`:

```
hbox.getChildren().remove(label1);
```