Assignment	1. Write a distance converter with graphical interface using JavaFX. 2. Use an event handler for user input events.
What to Submit	Submit your project code via git as project name **converter**. URL of Github Classroom repository will be announced.

In this lab you will create a graphical unit converter for length units.
Iteration 1: Convert Miles to/from Kilometers.

Iteration 2: Convert many length units. Use an enum for the length units.

Setup: Install SceneBuilder

SceneBuilder is a visual layout editor for JavaFX. It saves the layout as a .fxml file. SceneBuilder is usually easier than writing the UI in Java code. It works with most IDE, but you have to download it separately.
1. Download SceneBuilder from **http://gluonhq.com/products/scene-builder/**
2. Tell your IDE where SceneBuilder is installed. For Eclipse use Window -> Preferences -> JavaFX and use the "Browse" button to locate the SceneBuilder executable.

0. Create a JavaFX Project

Create a JavaFX project named **converterfx** in Eclipse or IDE of your choice.
Use the package **converter** for your classes. You need to create 4 components:

Main.java or ConverterApp.java - Main class to launch the program.

ConverterUI.fxml - FXML file for user interface

ConverterController.java - the controller for handling input events from the user

Length - an enumeration (enum) of length units, with values.

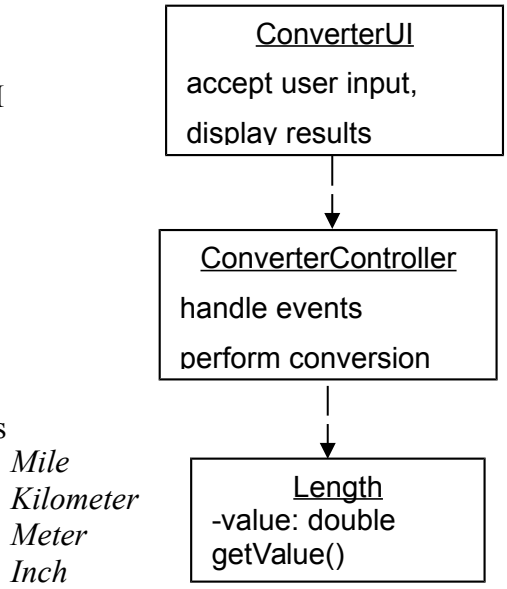
Application Design

Each component has a different set of responsibilities.

User Interface (View): interacts with the user. In some apps the UI also handles events, but in FXML the event handler methods are in a controller.

Controller: initializes data in the UI, and handles events caused by user actions in the UI. The controller knows how the application should behave. It provides the Units to the UI.

Model (Domain Logic): application logic, data, and other classes needed by the application. The converter is quite simple, so there is not much for the domain to do. In other applications the domain contains the majority of program code.

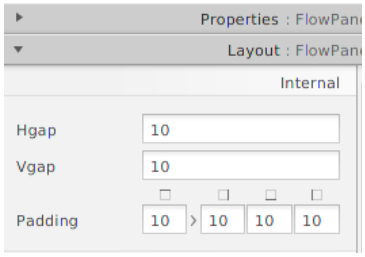


1. Create a User Interface

For the first iteration, use `Labels` for Mile and Kilometer. The UI should look like this:

1. Use a `FlowPane` or `HBox` for the top level layout.

2. Add the components you need. In SceneBuilder, just drag a component from the "Controls" palette into the FlowPane or HBox.



3. Add space between components and around the edge of the window.

   * Select the Container (`FlowPane/HBox`), **not** a component in the container.

   * In the "Layout" palette (right side) set the `Hgap`, `VGap`, and `Padding`.

4. Preview the UI to check appearance. If necessary, adjust Preferred Size of components.

5. Save the UI (fxml file).

2. Write or Edit the Main class

Eclipse, IntelliJ, and Netbeans will create a Main class for you. Eclipse also creates a CSS stylesheet (this project does not use the stylesheet). The code for Main is approximately as shown below.
2.1 Add a **title** to the window.
2.2 Let the window auto-size itself (Eclipse sets the size to 400 x 400 -- delete that).
2.3 Test that it works. If `getClass().getResource("`*filename.fxml*`")` cannot find the FXML file then try adding the directory to the filename, e.g. `"converter/ConverterUI.fxml"`.

```
package converter;

public class Main extends Application {
    @Override
    public void start(Stage stage) {
        try {
            Parent root = (Parent)FXMLLoader.load(
                          getClass().getResource("ConverterUI.fxml") );
            Scene scene = new Scene(root);
            stage.setScene(scene);
            stage.sizeToScene( );
            stage.setTitle("App Title Goes Here");
            stage.show();
        } catch(Exception e) {
            System.out.println("Exception creating scene: "+e.getMessage());
        }
    }

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

3. Write a ConverterController to Handle Events

The IDE (Eclipse) may have created this class for you. If not, create it yourself.

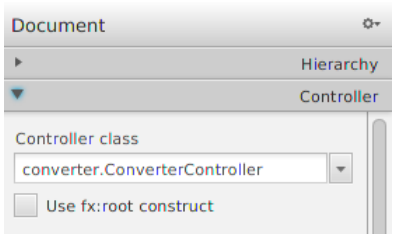
3.1 The controller needs access to the 2 TextField in the ConverterUI, so it can get text or set the result. Define fields for these components and annotate them with `@FXML`.

For example:

```
package converter;
import javafx.scene.control.TextField;
/**
 * UI controller for events and initializing components.
 */
public class ConverterController {
    @FXML
    private TextField textfield1;
    @FXML
    private TextField textfield2;
```

3.2 Save ConverterController, then use SceneBuilder to
(a) specify the name of Controller class (if not already set),
(b) set the **fx:id** property on the TextFields to the names of the attributes in the Controller. The fx:id in the the "Code" palette on right side. You must select the component you want before setting its fx:id.

Specify the Controller:



Set fx:id of a TextField:

3.3 Run the Main class to verify the application works.

3.4 Write a handler method to perform conversion. 1 mile = 1.609344 kilometer. Here is example code:

```
    /**
     * Convert a distance from one unit to another.
     */
    public void handleConvert(ActionEvent event) {
        // read values from textfield(s)
        String text = textfield1.getText().trim();
        // This is for testing
        System.out.println("handleConvert converting "+text);
        // perform the conversion and write the result in textfield2
    }
```

Display results to 4-digit accuracy. If a value is very large or small, use the **`%g`** number format **`"%.4g"`**.

3.5 Edit the UI (in SceneBuilder). Specify that when the user clicks the "Convert" button, invoke **`handleConvert`** to "handle" the event. Button click is an "On Action" event, so set the "On Action" property of the Convert button. "On Action" and other events are in the Code palette (right-side, bottom).

3.6 Run the application; verify that handleConvert is called with you click "Convert".

3.7 Improve logic of the **`handleConvert`** method so that:

(a) if user inputs an invalid number, then catch the error and let the user know: change TextField color to red, or show a message. Don't print on the console! Use the UI.
If you change the text color or border, change it back after a correct conversion or "Clear" is pressed!

b) Convert from mile to kilometer (left-to-right) or kilometer to mile (right-to-left) depending on where the user types.

3.8 Write a **handleClear** method to clear the Textfields.

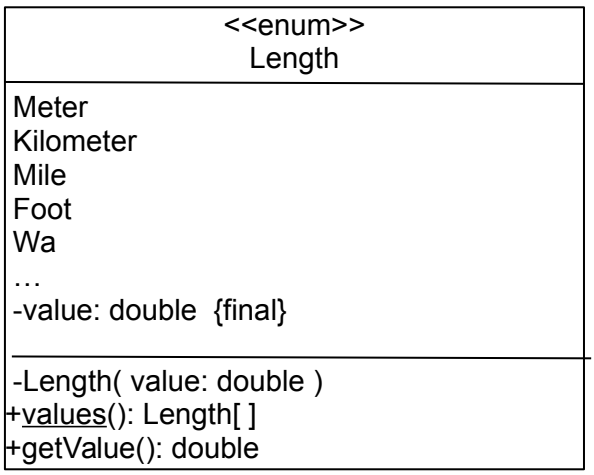
3.9 Enable the user to press ENTER in a TextField, so it causes a conversion (so he doesn't have to click the button). Hint: TextField also has an "On Action" property.

4. Write a Length enum for Length units

See the end of this document for introduction to enum type.

We want the converter to handle many different units. Since the units are fixed a simple way to handle this is using an **enum**. enum is really a class with a fixed set of static instances (defined inside the enum).
We will use an enum for Length units, so we can write code like this:

```
double x = Length.Mile.getValue(); // meters per mile
System.out.printf("1 mile = %.3f meters", x);
```



4.1 Create a Length "enum" (similar to creating a class).
Each enum member will have one attribute which is the value of the length unit *in meters*.

```
public enum Length {
    // you must write the enum members first, separated by comma
    Mile(1609.344),
    Kilometer(1000.0),
    Meter(1.0);

    // attributes of the enum members
    private final double value;

    // enum constructor must be private
    private Length(double value) { this.value = value; }
    // methods are just like in a class
    public double getValue() { return this.value; }
}
```

The **value** is a *multiplier* to convert this unit to a quantity of a "standard" unit. For Length, let meter be the "standard" unit. Hence, Meter has a value of 1.0. Kilometer has a value of 1000.0.

4.2 Add all these length units to the enum. You can also add others. How about *inch* or *light-year*?

meter	1.0000
centimeter	0.0100
kilometer	1,000.0
mile	1609.344
foot	0.30480
wa	2.00000
AU	149,597,870,700 (Astronomical Unit is avg. distance from Sun to the Earth)

4.3 Write a test class to print all the Length units. Print their names and values. Use the built-in static **values()** method to get all the Length members. The `values()` method is an *automatic* method in *every enum*. It has *no relation* to the getValue() method we defined for our Length enum.

```
    // example how to get the values
    Length[] lengths = Length.values();
    // use the values
    for(Length x : lengths)
        System.out.println(x.toString()+" = "+x.getValue());
```

As the above code shows, an enum behaves like a class and the enum members behave like objects.

5. Modify the ConverterUI to use a ComboBox for Length Units

5.1 In SceneBuilder, replace the Label for "Kilometer" and "Mile" with ComboBox controls, so the user can select units from a drop-down list. It should look something like this:

5.2 The controller class will *set* the values of the ComboBox using the Length enum, and *read* the user's selection. So, in ConverterController define attributes (fields) for the 2 ComboBox controls and annotate the attributes with `@FXML`.

5.3 In SceneBuilder, set the fx:id of the ComboBox controls to the variable names you defined to ConverterController. This is how you connect controls (in the UI) to variables in the controller.

6. Modify ConverterController to Use ComboBox

You need to make two changes to the controller: 1) initialize both ComboBox with Length units, 2) in the **`handleConvert`** method, *read* the user-selected value from each ComboBox.

ComboBox Syntax: ComboBox has a type parameter, which is the type of item in the ComboBox. We will put Length objects in the combo boxes, so the ComboBox should have type <Length>:

```
@FXML
private ComboBox<Length> combobox1; // any variable name you like
```

Some ComboBox methods we will use are:

Put values in a ComboBox: `combobox.getItems().addAll( values )`

Get the selected value: `Length x = combobox1.getValue( )` // may be null if nothing selected

6.1 Initialize values of the ComboBoxes for Length units. If a JavaFX controller has a public method named **`initialize()`**, then JavaFX will automatically call this method when in creates the controller and *after* it assigns values to the @FXML attributes. Use this method to set the Length units in the ComboBoxes;

```
/**
 * JavaFX calls the initialize() method of your controller when
 * it creates the UI form, after the components have been created
 * and @FXML annotated attributes have been set.
 *
 * This is a hook to initialize anything your controller or UI needs.
 */
@FXML
public void initialize() {
    // This is for testing
    System.out.println("Running initialize");
    if (unitbox1 != null) {
      unitbox1.getItems().addAll( Length.values() );
      unitbox1.getSelectionModel().select(0); // select an item to show
    }
    if (unitbox2 != null) {
      unitbox2.getItems().addAll( Length.values() );
      unitbox2.getSelectionModel().select(1); // select an item to show
    }
}
```

Note: its not needed here, but you can use `initialize()` to assign event handlers to UI controls, too.

You could write:

```
// Set the Action event handler of textfield1 to our handleConvert method.
// this::handleConvert is a "method reference" (Java 8 syntax)
textfield1.setOnAction( this::handleConvert );
```

6.2 Modify the handleConvert() method to get the selected Length unit from each ComboBox.

```
public void handleConvert(ActionEvent event) {
    // read values from textfield(s)
    String text = textfield1.getText().trim();
    // read the Length from combobox
    Length unit1 = unitbox1.getValue();  // this may be null
    // This is for testing
    System.out.printf("got values %s %s\n", text, unit1.toString() );
```

6.3 Convert the length units using same rules as in Problem 3.

Avoid writing duplicate code and **never throw exception** even if some input is invalid.

Reference

How to use JavaFX ComboBox: https://docs.oracle.com/javafx/2/ui_controls/combo-box.htm

JavaFX Tutorial: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm

Introduction to Enumeration (Enum)

Java has an enum data type, which is a class having a fixed set of values. A simple enum is a collection of named constants:

```
public enum Size {
      SMALL,
      MEDIUM,
      LARGE;
}
```

This encourages *type safety*. If you define a variable of type Size it can only have values from the Size enum:

```
Size mysize = Size.SMALL;        // assign a value from enum
public int setSize(Size s){..}   // declare method than accepts a Size
tshirt.setSize( Size.MEDIUM );   // pass enum value to a method
tshirt.setSize( "MEDIUM" );      // Error. Parameter must be a Size.
```

Comparing Values: An enum is a fixed set of static final members, so its OK to compare values using ==.

```
if (size == Size.SMALL) System.out.println("You're small");
else if (size == Size.MEDIUM) System.out.println("You are medium");
```

Built-in Methods:

`Enum.values()` To get all the values of an enum, call the built-in **`values()`** method. `Sizes.values()` returns an array `Size[]`. For example:

```
// Get the Sizes as an array:
Size[] sizes = Size.values();
// Print all the sizes
for ( Size size : Size.values() ) System.out.println( size );
SMALL
MEDIUM
LARGE
```

`name()` and **`toString()`**: These methods return the name of the enum member, exactly as specified in the enum. You can override toString() to display a different name (often useful!). name() is final, so you cannot override it.

Enum can have attributes and methods

An enum is really a ***class*** with a ***private constructor***. The enum values are static instances of the class. **An enum can have attributes and methods**.

Suppose we want SMALL to cost 0.5, MEDIUM to cost 1.0, and LARGE to cost 2.0. We could add a `cost` attribute and getCost() method to the enum:

```
public enum Size {
      SMALL(0.5),
      MEDIUM(1.0),
      LARGE(2.0);

      public double cost;
      private Size(double c) {
            this.cost = c;
      }
      public double getCost() { return cost; }
}
```

We can get the cost for some item by writing:

```
Size size = Size.SMALL:
double total = quantity * size.getCost();
```

Since enum are for *constants*, it is OK to declare the attributes public final so they can be accessed directly but cannot be changed. For example:

```
public enum Size {
     SMALL(0.5),
     MEDIUM(1.0),
     LARGE(2.0);
     public final double cost;
     // enum constructor must be private
     private Size(double c) {
          this.cost = c;
     }
}
```

Now we can get a cost by writing Size.SMALL.cost.