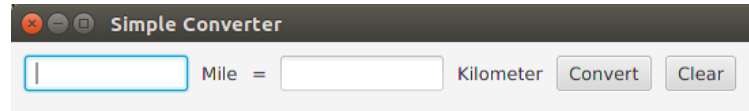


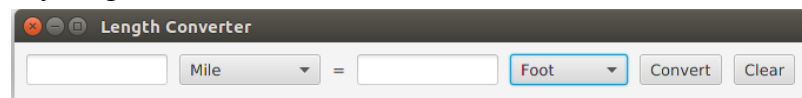
Assignment	1. Write a distance converter with graphical interface using JavaFX. 2. Use an event handler for user input events.
What to Submit	Submit your project code via git as project name converter . URL of Github Classroom repository will be announced.

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 unit values.



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 "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

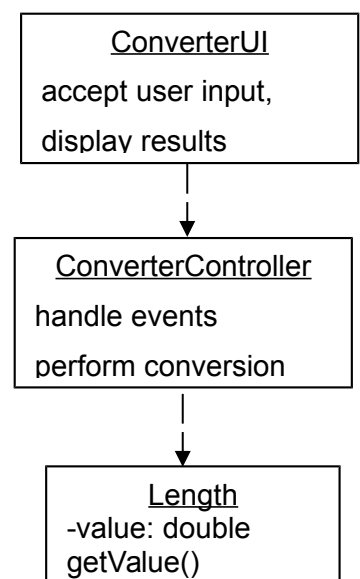
Each component has a different responsibilities.

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

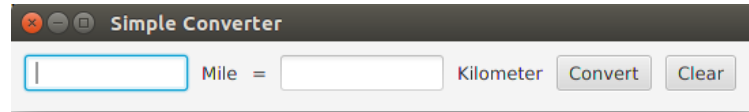
Model (Domain Logic): application logic, data, and other classes needed by application. The converter is quite simple, so there is not much for the domain to do. In other applications the domain contains the majority of program code.

*Mile
Kilometer
Meter
Inch*



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. In the Pane's "Properties" window set the `Hgap`, `VGap`, and `Padding` properties..
3. Preview the UI to check appearance. Adjust Preferred Size of components if necessary.
4. Save the UI (FXML file).

3. Write or Edit the Main class

Eclipse and Netbeans will create a Main class for you. Eclipse also creates a CSS stylesheet (that this project does not use). The code for Main is approximately as shown below.

2.1 Add a title to the window.

2.2 Test that it works. If `getClass().getResource("filename")` cannot find the 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.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 from them and show results. Define fields for these components and annotate them with `@FXML`.

For example:

```
package converter;
```

```

/**
 * UI controller for events and initializing components.
 */
public class ConverterController {
    @FXML
    TextField textfield1;
    @FXML
    TextField textfield2;
}

```

3.2 After saving **ConverterController**, use **SceneBuilder** to (a) specify the name of Controller class (if not already set), and b) set the **fx:id** property on the **TextFields** to the names you use in your controller class.

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 output the result
}

```

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**) to specify that **handleConvert** should be called to handle "Action" events. In **SceneBuilder**, select the "Convert" button and set the "On Action" property to **handleConvert**. "On Action" and other events are in the **Code** palette.

3.6 Run your application and verify that **handleConvert** is called with you click "Convert".

3.7 Improve the **handleConvert** method so that:

a) if user inputs an invalid number, then catch the exception and let the user know (change **TextField** color or border to red, or show a message). Don't print on the console! Use the UI.

If you change the 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 it doesn't have to click the button).

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);

```

<<enum>> Length
Meter Kilometer Mile Foot Wa ...
-value: double {final}
-Length(value: double) +values(): Length[] +getValue(): double

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;

    // constructor must be private
    private Length(double value) { this.value = value; }
    // methods 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 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());

```

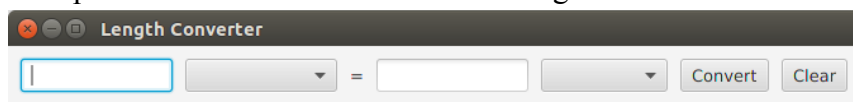
As the same code shows, an enum behaves like a class and the enum members behave like objects.

4.3 Add all these length units to the enum. You can also add others.

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

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 for the 2 ComboBox controls and annotate the attributes with **@FXML**.

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 the Comboxes with Length values, 2) in the **handleConvert** method, *read* the user-selected value from each combobox.

Basic 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 it creates the controller and *after* it assigns values to the **@FXML** attributes. Use this method to put **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.
 */
@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(ActionEvent) handler of textfield1 to our handleConvert method
// this::handleConvert is a "method reference"
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() );
}
```

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**.