Assignment	1. Write a distance converter with graphical interface using JavaFX. 2. Use an enum for the names and values of Length units. 3. Create a ***runnable* JAR file** named LengthConverter.jar.
What to Submit	Submit your project code to Github Classroom, including LengthConverter.jar. The assignment URL will be posted on Google Classroom.

In this lab you will create a graphical unit converter for length units. These instructions show how to create the UI entirely in Java code. There is another (older) lab sheet showing how to create the layout using SceneBuilder and FXML, but the other parts are written in Java, same as this lab.

Iteration 1: Convert Miles to/from Kilometers.

Iteration 2: Convert many length units. Use an enum for the length units and ComboBox in the UI.

Create User Interface in Code

A previous version of this lab shows how to use SceneBuilder to create the UI. These instructions show how to create the UI layout in code, which can be done in any IDE and is useful for learning about components. If you want to use SceneBuilder to layout the UI and create an FXML file, you can do that.

0. Requirements

If you are using JDK 8-10 then JavaFX is included. You don't need to add any libraries.

If you are using Java JDK 11, then install JavaFX 11 from https://gluonhq.com/products/javafx.

Also for JDK 11, to add JavaFX to a project in your IDE see: https://openjfx.io/openjfx-docs/

There is more information about JavaFX on the course web at https://skeoop.github.io/

1. Create a JavaFX Project

Create a JavaFX project in an IDE of your choice.
Use the package **`converter`** for your classes. The main class should be named `ConverterApp`.
If the IDE uses a different name, just use Refactor -> Rename to change it (even VS Code can do this).

2. Iteration 1 Simple Converter

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

2.1 In the initComponents() method, create Label, TextField, and Button objects to match the appearance above. The component classes are: Label, TextField, and Button in package javafx.scene.control.

2.2 For the root element you can use a FlowPane, HBox, or any layout you like except AnchorPane (absolute layout, which is no good).

To learn about layouts, see: https://www.tutorialspoint.com/javafx/javafx_layout_panes.htm

2.3 Set a **title** to the window.

2.4 Add **padding** around the edges of the window and **space** between components. Specify CENTER alignment. (This is covered in the example video posted on Google Classroom.)

2.5 Test that it works and displays correctly.

2.6 Are the TextFields too small? You can specify the width using textField.setPrefWidth(double). This means to set the *preferred width.* Note that textField.setWidth() won't work! (See Javadoc for why.)

```
package converter;

public class ConverterApp extends javafx.application.Application {
    @Override
    public void start(Stage stage) {
        Pane root = initComponents();
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.sizeToScene( ); // optional, doesn't always work
        stage.setTitle("App Title Goes Here");
        stage.show();
    }

    /** initialize components for the scene graph to display. */
    private Pane initComponents() {
        FlowPane pane = new FlowPane();
        pane.setPadding( ... );
        // create and add components
        TextField numberField1 = new TextField();
        Label label1 = new Label("Mile");
        ...
        Button convertButton = new Button("Convert!");
        pane.getChildren.addAll(numberField1, label1, ...);
    }

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

Note: In older versions of JavaFX it was necessary to put the code for start() inside a try - catch block as shown below. The try - catch is always needed if you use FXMLLoader to laod the FXML from a file.

```
    public void start(Stage stage) {
        try {
            Pane root = initComponents();
            Scene scene = new Scene(root);
            stage.setScene(scene);
            ...
            stage.show();
        } catch(Exception e) {
            System.out.println("Exception creating scene: "+e.getMessage());
        }
    }

    private Pane initComponents() throws Exception { ... }
}
```

3. Add Event Handler

3.1 Add an EventHandler to the "Convert!" button that converts Mile to Kilometer or Kilometer to Mile, and displays the result in a TextField.
The logic should be: if both text fields are empty then do nothing. If the left textfield contains a value then convert left-to-right (miles to km). Otherwise if the right textfield contains a value then convert right-to-left (km to miles).

For the event handler, add an *inner class* that implements EventHandler, as shown below:

```
    class ConvertHandler implements EventHandler<ActionEvent> {
        private static final double MILES_PER_KM = 0.62137119;

        @Override
        private void handle(ActionEvent event) {
             String text = numberField1.getText().trim();
             // etc.
             // convert Mile to Kilometer or Kilometer to Mile
             // put the result in the other TextField & format it correctly
        }

    }

    // in initComponents...

    convertButton.setOnAction( new ConvertHandler() );
```

3.2 ConvertHandler needs to read the values from the numberField1 and numberField2 components, but those variables cannot be accessed because they are local variables in initComponents().

Change them to be attributes (fields) of the ConverterApp object. You should still initialize them (create TextField objects) in initComponents().

```
package converter;

public class ConverterApp extends javafx.application.Application {
    private TextField numberField1;
    private TextField numberField2;
```

3.3 Format the conversion result. The result (double) may be huge or tiny. Don't display numbers as 0.0000000025 or 333333333.333. You should use a **format** so that the value is readable and doesn't show many insignificant digits. The `%g` format does this -- see the Javadoc for Formatter class for explanation.

The `%g` format will select either fixed point or scientific format (1.0e+3) depending on the value. An example is `"%10.5g"`. You should try `%g` with different values for width and decimals using Jshell: String.format("%10.5g", x) for different x. You can even use `"%.5g"` or `"%g"` with no field size.

3.4 **Test it.** Are conversions correct?

3.5 **Add the ConvertHandler to the TextFields, too,** so the user can press ENTER in a TextField and conversion is done immediately. No need to click "Convert".

Reuse the same ConvertHandler object -- don't create a new one for each component (a bad idea).

4. Add Event Handler for the Clear Button

Define a separate event handler for the Clear button. Clear the text fields.

5. Push Your Code to Github

The repositoryfor this lab does not contain any code. In your project directory (not the "src" subdirectory) do:

1. **`git init`**
2. add a `.gitignore` file. You can use the one from other labs as example.
3. add a `README.md` containing Markdown that briefly explains what this code does.
4. commit to you local repo: `git commit -m "initial code checkin"`
5. Add Github as remote. The instructions are shown on Github when you visit an empty repository.
   `git remote add origin git@github.com:OOP2020/lab12-`*`yourgithubid`*`.git`
   `git push -u origin master`
6. Next time you can use simply "git push", without the "-u origin master".

Iteration 2: Length Units and ComboBox

In the second part of this lab you will define an *enumeration* for different length units, add an external class to handle unit conversion and getting units, and modify the UI to use a Combobox to show units:

6. Write a Length enum for Length units

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

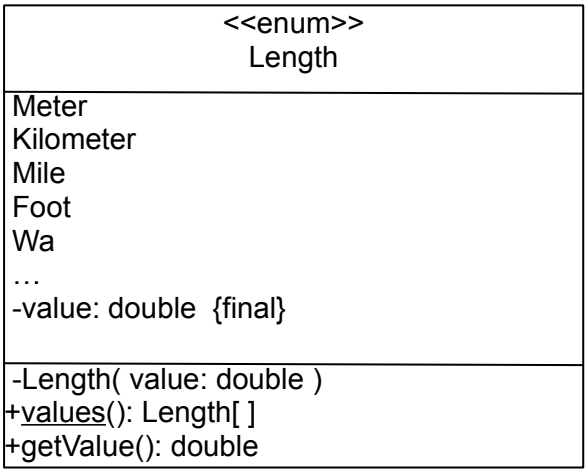
We want the converter to handle many different units. Since the units are a fixed set of values, a good way to handle them is an enum. enum is a Java type similar to a class with a fixed set of members that are defined inside the enum. When you compile an enum the result *is* a class (Length.class).

Enum can have attributes and methods just like a class; an enum constructor (which is optional) must be private.

Here's an example of using an enum for Length units:

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

```
1 mile = 1609.344 meters
```



6.1 Create a Length "enum". 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
    Kilometer(1000.0),
    Mile(1609.344),
    Meter(1.0);

    // attribute of the enum members is number of meters per 1 unit
    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** attribute is a *multiplier* to convert this unit to meters (the "base" unit for length). Hence, Meter has a value of 1.0. Kilometer has a value of 1000.0.

6.2 Add all these length units to the enum. You can add other length units you like.

meter	1.0
centimeter	0.01
kilometer	1,000.0
mile	1,609.344
foot	0.30480
wa	2.0
light-year	9,460,730,472,580,800

6.3 Write test code to print all the Length units. Print the Length 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*. `values()` has *no relation* to the getValue() method we defined for our own Length enum.

```
    // get all the members of the Length enum
    Length[] lengths = Length.values();
    // print the members and their value attribute
    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.

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

JavaFX has a Combobox class with a type parameter: ComboBox<T>. It displays a drop-down list of items. The JavaFX tutorial has examples, but they are *too complex*:

https://docs.oracle.com/javafx/2/ui_controls/combo-box.htm

You don't need to use FXCollections or ObservableList to initialize a ComboBox!

To add values to a combobox, use getItems().add( ) or getItems().addAll( ).

For example:

```
ComboBox<String> food = new ComboBox<String>();
food.getItems().addAll("Pizza", "Salad", "Fried Rice");
// select current item to display on ComboBox
food.setValue("Pizza");
```

7.1 Create ComboBox for the Length units on the left side and right side of "=". The ConvertHandler will need to get the user's selected unit (just like it gets the value from the TextField) so these variables should be attributes of ConverterApp, but initialized inside initComponents().

```
public class ConverterApp extends javafx.application.Application {
    private ComboBox<Length> unitbox1;
    private ComboBox<Length> unitbox2;
    private TextField numberField1;

    // inside initComponents:
        unitbox1 = new ComboBox<Length>();
        unitbox1.getValues().addAll( Length.values() );
        // add unitbox1 and unitbox2 to the UI root element
        // Replace the Label for "Mile" and "Kilometer" with ComboBox.
        pane.getChildren.addAll(numberField1, unitbox1, equalsLabel, ...);
```

7.2 In **`ConvertHandler`**, get the current unit from each ComboBox and use the units in the conversion (instead of Miles and Kilometers as in Iteration 1).

Since unitbox1 was defined as ComboBox<Length> it will return a Length unit. This makes the conversion *easy*. No need to use hard-coded values for lengths.

```
Length unit1 = unitbox1.getValue();  // currently selected Length unit
```

For example:

```
class ConvertHandler implements Handler<ActionEvent> {
    public void handle(ActionEvent event) {
        // read values for left-side of "=" sign
        String text1 = textfield1.getText().trim();
```

```
        Length unit1 = unitbox1.getValue();
        // This is for testing
        System.out.printf("Left side: %s %s\n", text1, unit1.toString() );
```

7.3 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.

8. (Recommended) Let the Length Units Convert Themselves

Since the Length enum know everything necessary to convert from one unit to another, you could add a method to Length to help perform the conversion. For example:

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

    /**
     * convert a quanity one Length unit to another Length unit
     */
    public double convert(double amount, Length fromUnit) {
        // convert from this length unit to another one
    }
```

In ConvertHandler.handle( ) you invoke this method to perform the conversion. For example

```
Length fromUnit = Length.Mile;
Length toUnit = Length.Meter;
double result = toUnit.convert( 1.0, fromUnit ); // convert 1 mile to meters
```

This is not required, but it will help in Programming Assignment 3.

9. Create a Runnable JAR file named LengthConverter.jar

IDEs can create a JAR file containing code and other files. For it to be "runnable" you must specify a "main" class when you create the JAR. You can also create a JAR file using the `jar` command. Change directory to the output directory for your project (bin/ or out/production/lab12) and enter:

```
jar -c -f LengthConverter.jar -e converter.ConverterApp converter/*
```

Put the **`LengthConverter.jar`** file in your project's root directory and commit to Git.

Verify that your Jar file is runnable by running it yourself. The command is:

```
> java -jar LengthConverter.jar
```

For Java 11 you need to specify the *module path* for JavaFX. Enter:

```
> java --module-path /path/to/javafx11/lib/ --add-modules javafx.controls -jar LengthConverter.jar
```

To save typing, put the "--module-path /path/to/javafx11/lib/ --add-modules javafx.controls" in a file named "`vmargs`" (any file name is OK) and type:

```
> java @vmargs -jar LengthConverter.jar
```

Test & Review Your Code. Then Push to Github

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 promotes *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 void setSize(Size s){..} // declare method than accepts a Size
tshirt.setSize( Size.MEDIUM );  // pass enum value to a method
tshirt.setSize( mysize );        // pass value using a variable
tshirt.setSize( "MEDIUM" );      // Error. Parameter must be a Size.
```

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

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

Built-in Methods:

Enum.values() Returns all the values of an enum as an array. `Sizes.values()` returns an array `Size[]`. For example:

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

name(): returns the *name* of the enum member as a String, exactly as specified in the enum.

ordinal(): returns an *int* that is the ordinal value of this member, Size.SMALL.ordinal() is 0.

toString(): the default toString returns the name of the enum member (which can be ugly). You can override toString() to display a different name (often useful!).

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 Sizes to have a price: SMALL is 5.0, MEDIUM is 10.0, LARGE is 20.0. Add a `price` attribute and specify the value of price when you declare the enum members:

```
public enum Size {
     SMALL(5.0),
     MEDIUM(10.0),
     LARGE(20.0);

     private double price;
     private Size(double p) {     // constructor accepts price param.
          this.price = p;
```

```
     }
     public double getPrice() { return price; }
}
```

We can get the cost for some item by writing:

```
Size size = Size.SMALL:

double total = quantity * size.getPrice();
```

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(5.0),
     MEDIUM(10.0),
     LARGE(20.0);

     public final double price;
     // enum constructor must be private
     private Size(double p) {
          this.price = p;
     }
}
```

Now we can get a cost by writing: `double total = quantity * Size.SMALL.price;`

Its more concise code than calling getPrice().