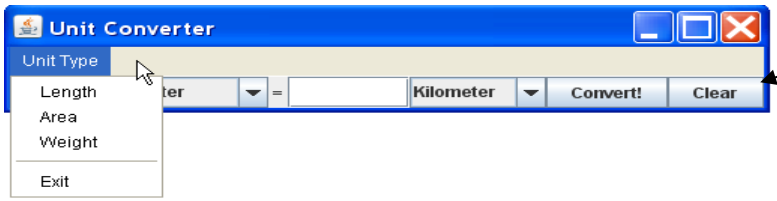
Assignment	Write a unit converter that can convert several different types of units. You must have at least 4 **kinds of units**, including Length, Weight, Temperature, and one other kind of unit (your choice).
What to Submit	1. Use the project on Github Classroom to submit your work. 2. Use an *interface* for Units, so that your converter can work with many different kinds of unit using polymorphism. 3. Create a descriptive project README.md on Github that include a **UML class diagram** of your application. The UML does should omit trivial methods like toString(), equals(), or getter methods, so its easier to understand. 4. Create a ***JAR file*** of your application and put it in the **root folder** of your project. Name the JAR UnitConverter.jar. In README.md include instructions for how to run your JAR file.
Evaluation	1. Implements requirements, performs correct conversions, and is usable. 2. Good OO design. Modular code that uses polymorphism to mimimize dependency on specific units (only dependency is in UnitFactory). 3. Code quality: uses *Java Coding Convention* for this course, code is well-documented and easy to read. 4. Quality of README. Correct English and clear, concise instructions for how to run the JAR file.

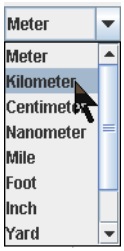
Requirements

1. Write a general unit converter that can convert values of different types of units, including Length, Weight, Temperature, and at least one other kind of unit.

2. Provide a **menu** to select the unit type to show in the UI. Include an "Exit" option on the menu.



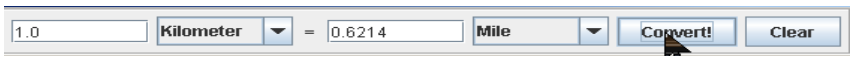
Button to clear the input fields.



3. When the user selects a type of unit, update the combo boxes to show **only that type of unit** in the combo-box. Don't mix unit types (e.g. meter and gram).

4. For each unit type include at least: 3 metric units (such as meter, cm, micron), 2 English units (such as foot, mile, acre, pound), and at least 1 Thai unit (wa, rai, thang). Exception: for Temperature there may not be enough standard unit types.

5. User should be able to convert in either direction: left-to-right or right-to-left. The converter should be smart enough to determine whether it should convert left-to-right or right-to-left, but give preference to left-to-right.



6. Program should **never crash** and **never print on the console**! Catch exceptions and handle them.

7. If user enters an *invalid value* you should catch it and change text color to RED or change TextField border to red. Don't forget to change the color *back* the next time any input is received!

8. Don't "hardcode" the unit information into the UI. The Unit Type menu must **not** contain the words "Length", "Weight", "Tempature", etc. Get the unit types from the UnitFactory!

9. Use polymorphism. The Controller should never refer to "Length", "Weight", "Temperature" -- everything is a unit. You should not need to *cast* unit values.

10. Use encapsulation: each unit type should have its own convert( ) method to convert values. This enables the app to handle non-linear conversion such as Temperature.

11. Create a UML class diagram for your application design.

12 Create a runnable JAR file. A runnable JAR file contains all your project classes in one JAR file and has a designated "main" class. You run a JAR by typing: `java -jar myjarfile.jar`

You can also run it by double clicking the JAR file's icon (on some operating systems).

Be careful that the JAR file includes your *.fxml file. Otherwise, the JAR won't work.

Programming Hints

To enable polymorphism, you need an interface for different kinds of units -- so that the Java code for the UI and event handlers can treat all the units the same. There is no special code for Length, Area, or Weight in the UI or event handlers.

There are several ways to implement this. Here is one way that involves only 2 extra pieces of code:

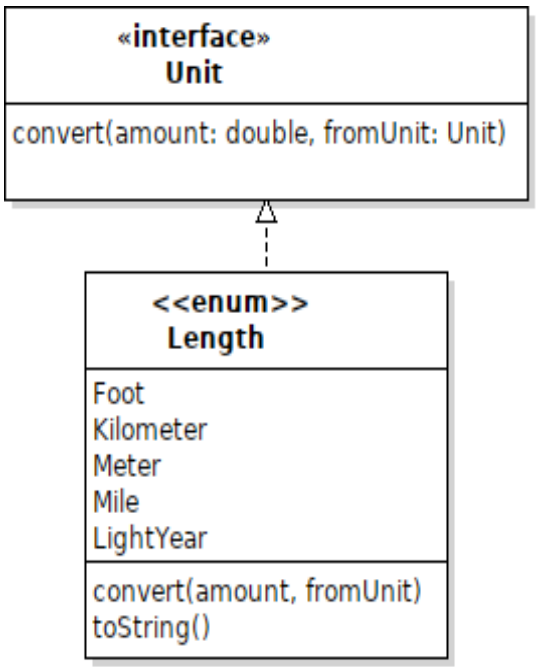
1. a Unit interface that every unit (Length, Area, etc) implements. (Enum can implement an interface.)

2. a Factory class to get the units and get the names of the types of units (for display in a menu).

Unit InterfaceDefine an interface for the required behavior of all units.

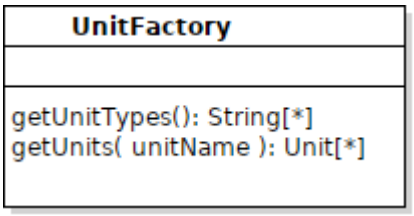
Study your code for Length Converter to find what methods you need. (That is, don't just blindly copy the UML shown here.)

Its a good idea to let the units convert themselves by including a convert method in each unit class. That's more flexible than having an event handler in the UI perform the conversion. For example, converting Celsius to Fahrenheit.



Unit Factory Class

For the UI to display different kinds of units, without hard-coding the units into the UI, define a UnitFactory class that is responsible for getting the units and telling the UI what kinds of units are available.



For the *names* of the types of units you can use an Enum, or just use Strings. A UnitType enum may be cleaner code but the use of Strings is not too clumsy, since the unit names are only needed in the Factory class.

The Factory methods can be static or instance. If you use instance methods, you should make the Factory be a Singleton (there are slides that show how to implement a Singleton class -- its easy).

For static methods, using the factory would look like this:

```
String[] unittypes = UnitFactory.getUnitTypes();
String unitname = unittypes[0]; // the first kind of unit, e.g. "Length"
Unit[] units = UnitFactory.getUnits( unitname );
```

Whatever solution you use, **don't "hardcode" the names of unit types into the UI!** The UI and controller should accept any kind of units. The strings "Length", "Area", "Weight", etc. should not appear anywhere in the UI or controller code.

Test Your Code in the Lab12 Length Converter

After defining a Unit interface and making Length implement Unit, see if you can replace "Length" with "Unit" everywhere in your Lab 12 Length Converter code. Declare ComboBoxes as ComboBox<Unit> instead of ComboBox<Length>, etc.

The only exception is 1 line of code to get the Length values:

```
private ComboBox<Unit> unitbox1;

private Pane initComponents() {
    Unit[] units = Length.values();   // the only dependency on Length

    unitbox1 = new ComboBox<Unit>();
    unitbox1.getItems().addAll( units );
    unitbox1.setValue( units[0] );    // initial value to show in combobox
```

If you can get this to work wihout any *cast* then your code is on the right track.

Next write another enum such as **`Weight`**. Test if you can replace Length with Weight in your Lab12 code and have the application still work without any other changes:

```
private Pane initComponents() {
    Unit[] units = Weight.values();

    rest of the code is not changed
```

Does it show weight units now? Does it convert weights? If so, you are making good progress.

Next, write a simple UnitFactory class with getUnits() method to get values for a type of unit. In your Lab12 code, see if you can use the UnitFactory instead of depending on the Weight enum:

```
private Pane initComponents() {
    Unit[] units = UnitFactory.getUnits("Weight");

    rest of the code is not changed
```

Does the code still work?

When you add a menu for selecting unit type, you can eliminate the String constant for "Weight" and use whatever unit the user selects from the menu.

You should also add a separate method to load unit values into the combo-boxes, instead of doing it in initComponents. Then your menu handler can call that method. This avoids duplicate code.

This design is OK, but using Strings for unit types isn't type safe. This isn't a big deal, since the use of Strings is localized to the UnitFactory. You could define a UnitType enum that contains names of the units:

```
public enum UnitType {
    Length,
    Volume,
    Temperature,
    Weight;
}
```

and modify the methods of UnitFactory to accept and return UnitType instead of String.