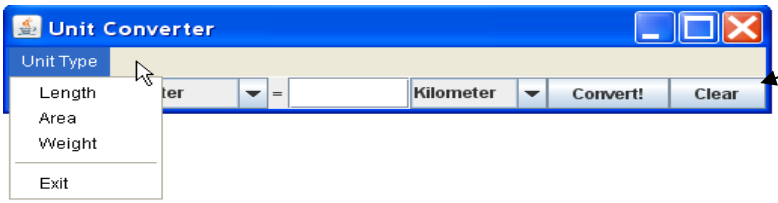
Assignment	Write a general unit converter that can convert several different types of units. You must have at least 4 **kinds of units**, including Length, Weight, Temperature, and at least one other kind of unit.
What to Submit	1. Use the project on Github classroom to submit your work. You will receive an invitation by email The project is unit-converter-*githublogin*. 2. Create a ***runnable JAR file*** of your application and put it in the **root folder** of your project. Name the JAR Converter-*yourname*.jar (use your first name). 3. Use Git commands to commit work to Github. Please **don't use file upload**, students doing that put files in the wrong folder; as a result, their code does not compile. On this assignment, no credit if your code does not compile. 4. Create a **UML class diagram** of your application and submit it on paper.
Evaluation	1. Implements requirements, performs correct conversions, and is usable. 2. Good OO design. Separate UI from application logic; use polymorphism instead of "if". No redundant code. 3. Code quality: follows *Java Coding Convention* for this course, code is well-documented and easy to read.

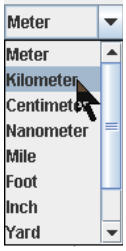
Requirements

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: Length, Area, or Weight. Include an "Exit" option on the menu. See the Java tutorial for how to create a JMenuBar and JMenu.



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 (don't forget to change the color *back* the next time enter is pressed). Use try - catch.

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).

In Eclipse, create a JAR file by right-clicking on the project and choose Export... Then choose "Jar" or "Runnable Jar" as export type.

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

In BlueJ use Project → Create Jar file...

Programming Hints

There are many ways to implement this. To enable polymorphism, you need an interface (or abstract superclass) for different kinds of units -- so that all units look alike. However, you **are not required to use enums** for the unit types. You can devise another solution, such as reading unit data from a file, which may be more flexible (you can add units without changing Java code).

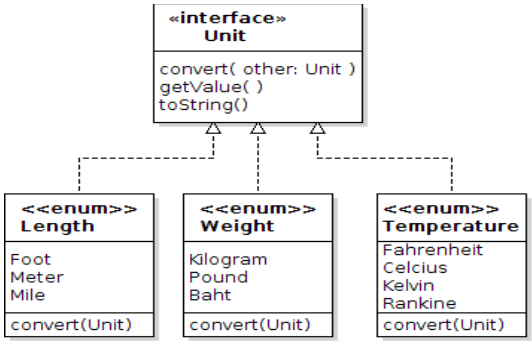
Whatever solution you use, **don't "hardcode" the unit types or unit names into the UI!** The UI and Controller should accept any kind of units. The UI should get names of unit types and units from the UnitConverter.

Some suggestions are:

1. Use separate classes for the user interface (*view*), the controller, and the units.

2. So that the UI can accept any kind of unit, you need to enable *polymorphism*. You need the different kinds of units to "look" alike. Define an *interface* for Unit that specifies the behavior you require of all units.

The actual unit types (Length, Area, Weight) implement this interface. You can use a class or enum for the actual units.



An enum can implement an interface, so Length can be an enum. But a unit type is not *required* to be an enum -- you can use a class instead of an enum. For example, if you want to convert *currencies* (in real time), you might need a class.

3. You need a way to add different kinds of units to the application.

This is similar to the problem of creating different kinds of money for the Coin Purse application. One solution is to define a UnitFactory that knows (1) what *kinds* of units are available, (2) the actual *values* of each unit type.

```
Example code to be added.
```

4. When the user selects a Unit type using a menu item, the UI invokes an *event handler method* in the controller, and this method will ask the UnitFactory for all the units of that type. Then populate the combo boxes with the new units.

```
Example code to be added.
```

5. The numbers may be very large or very small. So avoid displaying unformatted numbers! For example:

$$1 \text{ light-year} = 9.4605284 \times 10^{15} \text{ meters}$$

Try the "%g" format, which automatically chooses between fixed point and scientific notation.
Try this in BlueJ: `String.format("%.5g", x)` with large and small values of x.

Testing

Write and test each unit type first.

Then write and test the UnitFactory. Using a simple command line application may help.