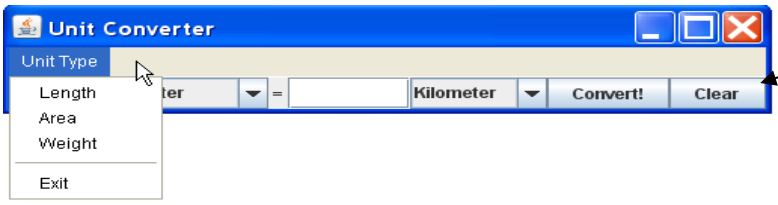
Assignment	Write a general unit converter that can convert several different types of units. You must have at least **4 kinds of units**, including Length, Area, Weight, and at least one other kind of unit.
What to Submit	1. Use the project on Github classroom to submit your work. You will receive an 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-5910xxxxxx.jar (use your student id). 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.

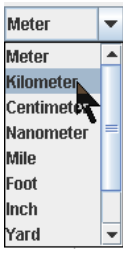
Requirements

1. Write a general unit converter that can convert values of different types of units, including Length, Area, and Weight.

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).

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. Avoid "`if`" and "`switch`" statements as much as possible! Encapsulate information about units and unit types. See below for suggestions.

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

10. 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.

In BlueJ use Project → Create Jar file...

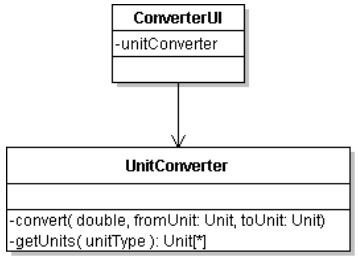
Programming Hints

There are many ways to implement this. To enable polymorphism, you need an interface for different kinds of units. However, you **are not required to use enums** for the units. Reading unit data from a file is another solution, that is more flexible (you can add units without changing Java code).

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

Some suggestions are:

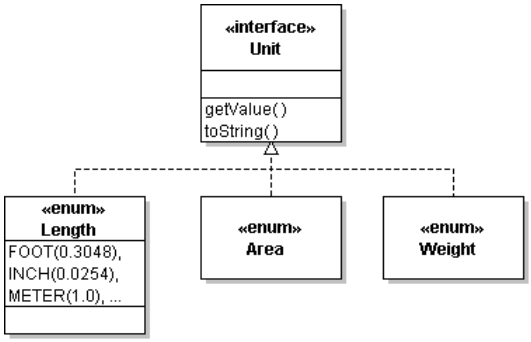
1. Use separate classes for the user interface (*view*) and the UnitConverter class that performs the conversion. The UI handles events (like button press) and asks the UnitConverter to do the work.



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 UnitConverterUI.

The UnitConverterUI should *ask* the controller class (UnitConverter) for the kinds of units it knows about.

For example, the UnitConverter can have a method named getUnitTypes() that returns an array of UnitTypes. The UnitTypes can be an enum, too (but doesn't have to be). Such as:

```
public enum UnitType {
      Length,
      Area,
      Weight,       // this can include mass units, too
      AnotherUnit;  // your choice
}
```

4. When the user selects a Unit type using a menu item, the UI *asks* the UnitConverter for all the units of that type. The UI uses the result to add units to the comboboxes.

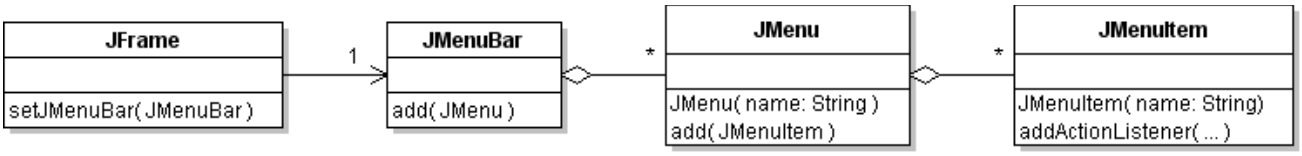
```
UnitType utype = UnitType.Length;
// get all the Length units from the UnitConverter
Unit [] units = converter.getUnits( utype );
// populate the user interface with these units
for( Unit u : units ) comboBox1.add( u );
```

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.

6. In Swing, a JFrame contains a JMenuBar. The JMenuBar contains one or more JMenu. Each JMenu contains one or more JMenuItem. You can also add *Action* objects directly to a JMenu and it will create a menu item for each Action. If you add *Action* directly to JMenu then you don't need to write an *ActionListener*, because each *Action* is its own *ActionListener*.




```
class ExitAction extends AbstractAction {
      public ExitAction( ) {
            super.setName("Exit");
      }
      public void actionPerformed(ActionEvent evt) {
            //TODO exit the program
      }
}
JMenu menu = new JMenu("File");
menu.add( new JMenuItem("...") );
menu.add( new ExitAction() );
JMenuBar menubar = new JMenuBar();
menuBar.add( menu );
frame.setJMenuBar( menubar );
```

The *Java Tutorial* has examples of creating a JMenuBar and menus.

Testing

Write and test the UnitConverter class first. Then write the UI.

An effective programming style is to code a little, test, code a little more, test again, etc.

What Class Does the Actual Conversion?

In the Distance Converter lab, the UnitConverter class has a convert( ) method. This works for distance units, but what about temperature? Temperative conversion is an affine function. Or currency (conversion rate changes)? A more general solution would be for the units to convert themselves! The UnitConverter.convert( ) method can just call another convert method of the actual unit classes. Assigning the convert() function to the units themselves is example of the *Information Expert* principle (units are the information expert for how to convert to other units).

Example Unit Conversion

Suppose the User asks to convert 2 kilometers to feet (plural of "foot").

The UI object calls `convert( 2.0, Length.KILOMETER, Length.FOOT )`

The `convert` method computes amount * fromUnit / toUnit:

```
result = 2.0 * 1000.0 / 0.3048
```