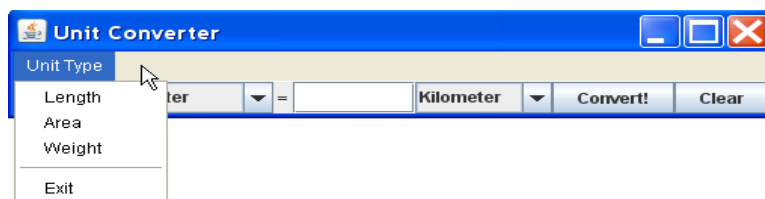


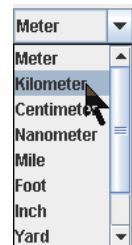
Assignment	Write a general unit converter that can convert several different types of units. You must have at least 4 kinds of units , including Length, Area, Weight, and at least one other kind of unit.
What to Submit	<ol style="list-style-type: none"> 1. Create a project named pa3 and commit it to Bitbucket. 2. Create a <i>runnable JAR file</i> of your application with the name <code>dist/Converter-bxxxxx.jar</code> (your student id) inside your pa3 project. Please don't create an extra layer of folders when committing to Bitbucket. That is, avoid this: <code>b5710xxxxxxx/pa3/PA3/src</code>, etc. 3. Create a UML class diagram of your application and submit it on paper.
Evaluation	<ol style="list-style-type: none"> 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 <i>Java Coding Convention</i> 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.



3. When the user selects a type of unit, you must 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.

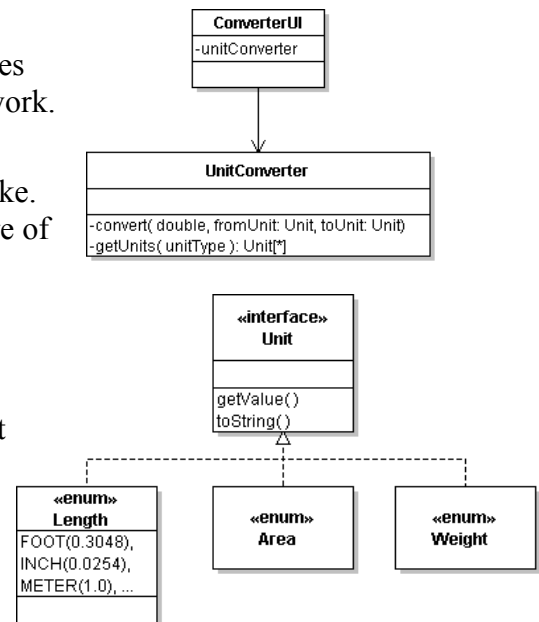
In BlueJ use Project → Create Jar file...

Programming Hints

There are many ways to implement this. Here is one approach, but you can invent your own solution.

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. An enum can implement an interface, so `Length` can be an enum. (But a unit type doesn't *have* 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* another class for the units, preferably the `UnitConverter` class.

This was done in Lab 7 where the `UnitConverter` had a factory method to get length units.

Don't "hardcode" the units into the UI class. The UI should accept any kind of units.

4. You also need a way for the `UnitConverterUI` to discover *what kinds of units* are available. Try not to hard-code this into the UI. Two ways to do this are:

- (a) a method in the `UnitConverter` class, such as `getUnitTypes()`.
- (b) an enum of `UnitTypes`, such as:

```

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

5. 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 );
  
```

1 meter = 0.000000000000000105702 light-year

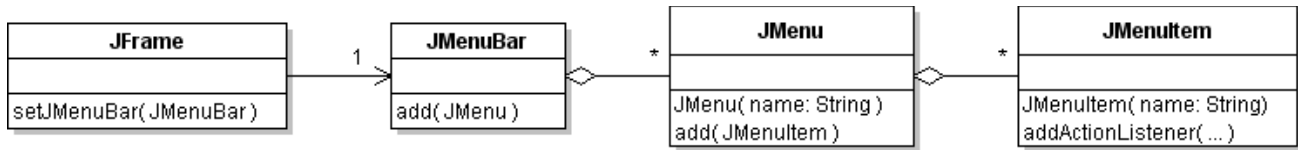
6. The numbers may be very large or very small. So avoid displaying unformatted numbers! For example:

1 light-year = 9.4605284 x 10¹⁵ 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`.

7. 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() );
  
```

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

Testing

Write and test the `UnitConverter` class first. Then write the UI. Write code to test the `UnitConverter`.

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

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`:

$$\text{result} = 2.0 * 1000.0 / 0.3048$$

What about light-years? How many *wa* equals one *light-year*?