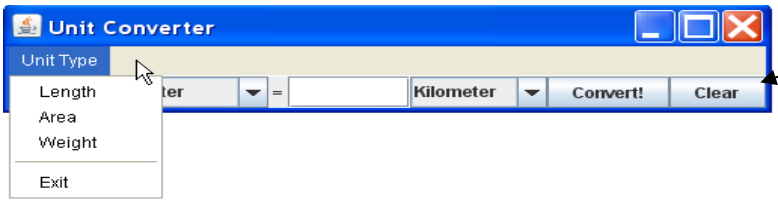
Assignment	Write a general unit converter that can convert several different types of units. You must have at least **4 kinds of units**, including Length, Area, Weight, and at least one other kind of unit.
What to Submit	1. Create a project named **`pa1`** and commit it to Github. 2. Create a *runnable JAR file* of your application and save in a "`dist`" subdirectory of your project directory. Use the name: `dist/`Converter-58xxxxxxx.jar (using your student id). Please don't create an extra layer of folders when committing to Git. That is, **don't** do this: `b5710xxxxxxx/`**`pa1/PA1`**`/src`, etc. 3. 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: code is easy to read and well-documented. Follows the *Java Coding Convention*. 4. Program catches input errors and doesn't crash.

Requirements

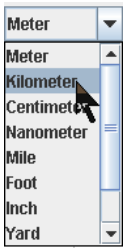
1. Write a general unit converter that can convert values of different types of units, including Length, Area, and Weight.

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. So, if he selects "Area" then only units of area are shown.



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 something is input). 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.

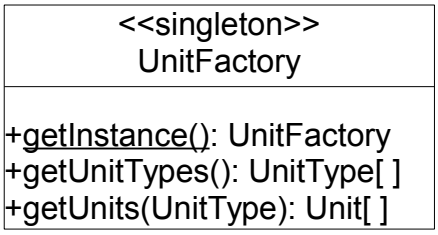
10. Create a runnable JAR file. A runnable JAR file contains all your project classes in one JAR file and has a designated "main" class. You run a JAR by typing: `java -jar myjarfile.jar`

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...

Use a Factory and Singleton Pattern



1. Create a UnitFactory class with the methods shown here.

UnitType[] getUnitTypes( ) returns array of known unit types

Unit[] getUnits(UnitType utype) returns all the Units of type utype.

2. Make UnitFactory be a Singleton so we only have **one** UnitFactory.
Define a static getInstance() method to get the singleton instance. See the lecture slides for how to write a Singleton.

3. Define UnitType to represent a type of unit (like Length, Area). You can use an enum or class. UnitType should have a toString() method so you can use it to configure the Menu of unit types.

Programming Suggestions

1. Use the Controller (UnitConverter) as a Facade or Intermediary

We don't want to add unnecessary *coupling* to the ConverterUI class. Since the UI already knows about the UnitConverter class, let the UnitConverter class act as briddge or intermediary.

When the UI wants the UnitTypes it calls unitConverter.getUnitTypes(), and the UnitConverter calls the UnitFactory.

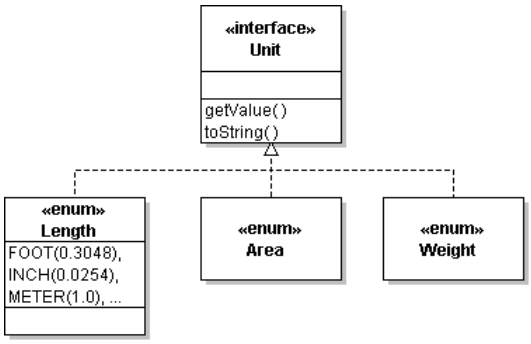
When the UI wants all the Units (of one type) it calss unitConverter.getUnits(UnitType) and the unitConverter calls the Factory.

This extra layer can be a "overdesign", but it helps to reduce coupling and make your application easier to maintain.

```
UnitType utype = UnitType.Length;
// get all the Length units from the UnitConverter
Unit [] units = converter.getUnits( utype );  // converter calls UnitFactory
// add units to comboboxes in UI.
//TODO: You must remove the old units first.
for( Unit u : units ) comboBox1.add( u );
```

'

2. All the actual unit types (Length, Area, Weight) should implement the Unit interface. But a unit 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. Don't "hardcode" the units into the ConverterUI class.

The UI should accept any kind of units.

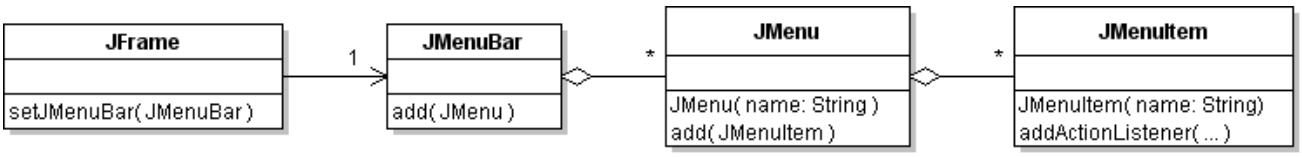
```
public enum UnitType {
      Length,
      Area,
      Weight,       // this can include mass units, too
      AnotherUnit;  // your choice
}
```

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

1 light-year = 9.4605284×10^{15} 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.

5. In Swing, a JFrame contains a JMenuBar. The JMenuBar contains one or more JMenu objects. Each JMenu contains JMenuItem objects. You can use *ActionListener* to know when a JMenuItem is selected. 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 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 code a little, test the code, code a little more, test again, etc.

Coding Requirements

1. Write complete Javadoc in all classes. Include @author, @param, @return tags.

2. Avoid duplicate code.

3. Code should be well formatted and easy to read.