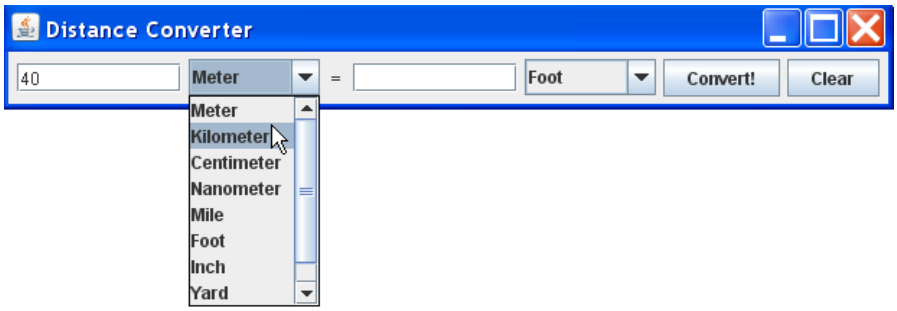
Objectives	1. Create a graphical interface for a distance converter. 2. Practice O-O design by using separate classes for different responsibilities. 3. Practice using an event handler (ActionListener) for user input events.
What to Submit	Submit your project code via git as project name **`converter`**. Write a useful README.md for the

In this lab you will create a graphical unit converter for length units.



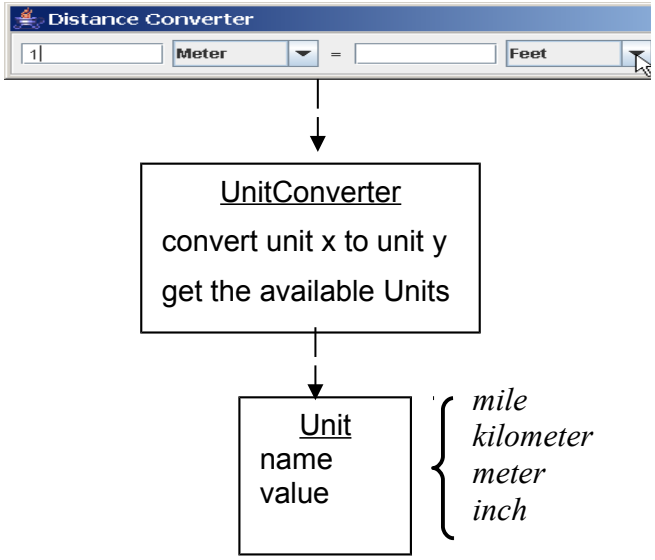
Parts of the Program

Your program needs 3 components, which have different responsibilities.

User Interface or View
: handles interaction with the user. It handles input from the user and displays results.* The UI also catches **errors** in input, such as an invalid number, and notifies the user of errors.

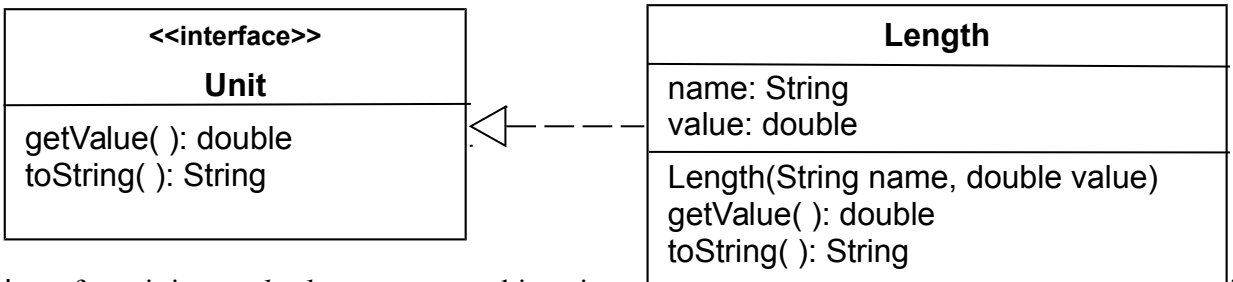
Controller: this component processes the user's request. It knows how the application should behave. It provides an array of Units to the UI.

Model (Domain Logic): application logic and other classes needed by application. This component provides services.
This converter is quite simple, so there is not much for the domain to do. In other applications the domain contains the majority of program code.



1. Write a Unit interface for units

1.1 Define a Unit interface that provides an interface (specification) for behavior of all units.
The toString() method should return a printable name for the unit. getValue() should return the value of 1 unit in a common "standard" unit. For example, if we use meters as the "standard" unit, then `foot.getValue()` would return 0.3048 (1 foot = 0.3048 meter).



The value of a unit is a *multiplier* to convert this unit to a quantity of a "standard" unit. For Length units, let meter be the "standard" unit. The multiplier value to convert other units to meter are:

meter	1.00	*mile*	1609.344
centimeter	0.01	*foot*	0.30480
kilometer	1000.0	*wa*	2.0

toString() should return the name of the unit that will be shown on the UI.

The controller needs a way to get all the units, but you should avoid "hard coding" the length data into the controller. The controller provides a getUnits() method for the UI, but the the controller should call some other object to get the available units (Controllers aren't supposed to do too much or be too smart). Here are 2 solutions (maybe you can think of another). You can implement either one:

1.2 Length as a class that implements Unit, and a UnitFactory to create Units

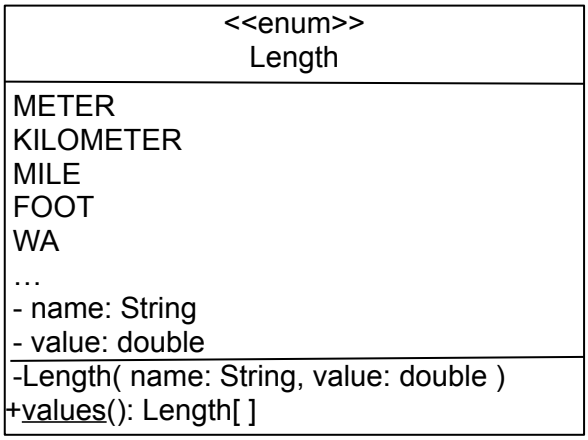
Write a Length class that implements Unit. Then, define a UnitFactory to create units (like the MoneyFactory in purse app). The UnitFactory should read the data for length units from a file. A *Resource Bundle* (as described in the last Coin Purse lab) would be a good choice.
This is a flexible solution because you can add more units without modifying code. When you distribute your application as a Jar file you can include the resource bundle file in the Jar file.

1.3 Length as an Enum

You can define the Lengths as an **enum** instead of a class. Using an **enum**, you define the actual length units inside the enum itself. You also set the properties (name and value) of units in the enum.

An **Enum** defines a datatype with a fixed set of members. It is like a class with a fixed number of instances that are defined as static attributes. You can't create any additional objects from an enum.

See the end of this lab sheet for how to use enum and how to define a Length enum.



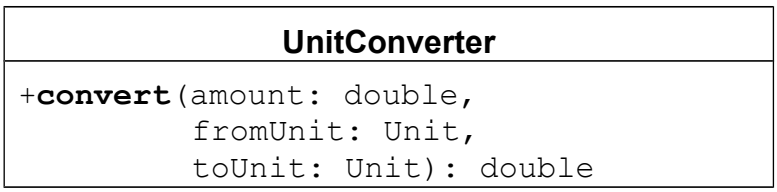
Every enum has an inherited static **`values()`** method that returns all the enum members as an array. So, to get all the Length units, just call **`Length.values()`**.

2. Write the UnitConverter class to Perform Conversions

The UnitConverter class receives requests from the UI to converts a value from one unit to another**.**

It also handles requests from the UI to get all the available units (don't "hardcode" the unit names into the UI).

So, the UnitConverter needs 2 methods to handle requests from the UI:



```
+getUnits( ): Unit[ ]
```

Example: `unitConverter.convert( 3.0, Length.MILES, Length.KILOMETER )`

The conversion has 2 steps:

1) convert 3.0 *from* miles *to* the standard unit (meter): x = 3.0 * 1609.344

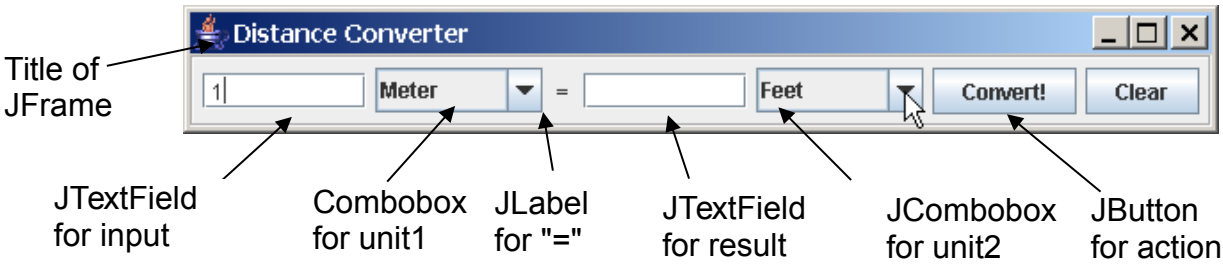
2) convert *from* the stardard unit *to* kilometer: " x / 1000.0

The return value is `3.0*1609.344/1000.0 = 4.828032` (kilometer).

Here are some examples using BlueJ:

```
> UnitConverter uc =  new UnitConverter();
> uc.getUnits( )
[ Length.METER, Length.KILOMETER, Length.CENTIMETER, Length.MILE ...
> uc.convert( 3.0, Length.KILOMETER, Length.METER )
3000.0
```

3. Implement a Graphical User Interface (The Fun Part)



You can use this code as a template. Add more components and complete the code.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;

public class ConverterUI extends
JFrame{
    // attributes for graphical components
    private JButton convertButton;
    private UnitConverter unitconverter;
    //TODO add other attributes

    public ConverterUI( UnitConverter uc
 ) {        this.unitconverter = uc;

      this.setTitle("Length Converter");
      this.setDefaultCloseOperation( ... );
      initComponents( );
    }

    /**
     * initialize components in the window
     */
    private void initComponents() {
//TODO  create components for the UI and position them using a layout manager.

//TODO add components for labels and JComboBoxes

      contents.add( convertButton );
      ActionListener listener = new ConvertButtonListener( );
      convertButton.addActionListener( listener );
      this.pack();          // resize the Frame to match size of components
```

declare attributes for components your application needs to access. Declare JLabel *as local vars in* initComponents() *if you don't need to access them again.*

Constructor does:
(1) receive a reference to the converter class
(2) set some properties of JFrame
(3) call initComponents

```
    }

    /**
     * ConvertButtonListener is an ActionListener that performs an action when
     * the button is pressed.  It is an inner class so it can access private
     * attributes of ConverterUI.
     * It reads the text from a JTextField, convert the value to a number,
     * call the unitconverter to convert, and write result in other text field.
     */
    class ConvertButtonListener implements ActionListener {
      /** The action to perform action when the "convert" button is pressed */
      public void actionPerformed( ActionEvent evt ) {
            String s = inputField1.getText().trim();
            //This line is for testing.  Comment it out after you see how it works.
            System.out.println("actionPerformed: input=" + s);
            if ( s.length() > 0 ) {
                 //TODO  catch errors!  What if the input is not a number?
                  double value = Double.valueOf( s );
                  //TODO get the selected units from the JComboBoxes
                  //TODO invoke the unitconverter to convert value
                  //  then display the result.
                  inputField2.setText( _____________ );
            }
      }
    }  // end of the inner class for ConvertButtonListener
}
```

3.1 Using JComboBox for Units

A JComboBox can hold any kind of values.

Suppose you have an *attribute* named `unit1ComboBox`. You can create a JComboBox and add items to it using:

```
private void initComponents( ) {
      unit1ComboBox = new JComboBox<Unit>( );
      Unit[] lengths = converter.getUnits( );
      for( Unit u : lengths ) unit1ComboBox.addItem( u );
```

We can add *any* objects to a ComboBox. ComboBox will use the object's own toString() to display the object.

Note: JComboBox also has a constructor that accepts an array of Objects as items and adds them all. This way avoids the need to use a loop.

```
      // create ComboBox and add array of items in one step
      unit1ComboBox = new JComboBox( lengths );
```

To get the user-selected value from a JComboBox use the getSelectedItem( ) method.

You probably want to do this in your ActionListener.

```
      Unit unit1 = unit1ComboBox.getSelectedItem( );
```

getSelectedItem( ) returns the type of objects in the JComboBox, so the return value depends on whether you use "new JComboBox()" (contains Objects) or "new JComboBox<Unit>()" (contains Units).

3.2 Write a Main Class to Create Objects and Run the Application

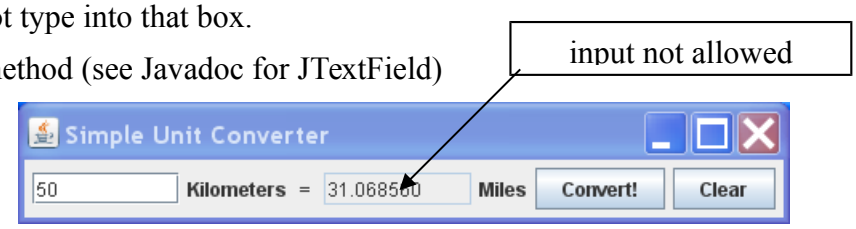
In layered software design, the user interface (upper layer) **should not** create the application or domain level objects. Instead, another class *sets a reference* to those objects into the user interface. This is called "dependency injection". It reduces coupling and encourages software reuse.

Add a "Clear" Button to clear the data from both JTextField. To clear a TextField, set the text to an empty string: `textField.setText("")`.

3.3 Don't Allow Input (Editing) in the Right TextField

The user can type into either text field. We only want him to type into the first text field. Modify the other field so that user cannot type into that box.

Use the setEditable method (see Javadoc for JTextField)



3.4 Pressing ENTER in TextField also Performs the Conversion

Add the *same* ActionListener object for the convertButton to the inputField1, so the user can convert a value by just pressing Enter. (Don't need to click the "Convert" button.)

One ActionListener can be used for more than one component, so you don't need to create a new ActionListener object! Use the *same object* that listens to the "Convert!" button.

3.5 Define your own length units. For example,

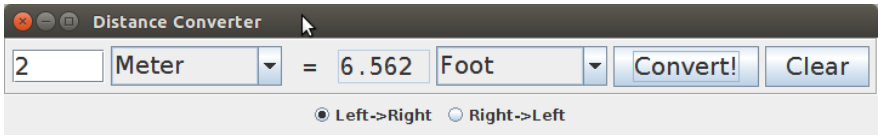
```
1 Light-year = 9460730472580800.0 meter
1 micron = 1.0E-6 meter
```

4. Add a direction to LengthConverter

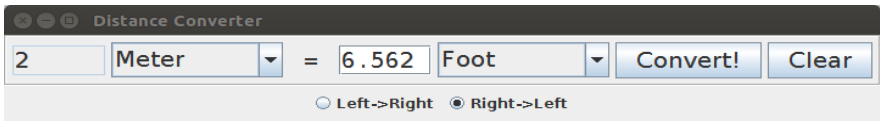
Add a row of RadioButtons to the Distance Converter UI so the user can choose left-to-right conversion, or right-to-left conversion.

When the user clicks a RadioButton you need to change which inputField is editable.

a) Left -> Right conversion mode:



b) Right -> Left conversion mode:



c) Can you design an "auto-detect" conversion mode?

Why a Unit Interface?

You could write the Distance Converter without using "Unit". The UI and converter (controller) could declare all parameters as type Length. So why create a Unit interface? The reason is to make it easy to change and enhance. If the UI and UnitConverter declare all variables as Unit, then the app can convert *any kind of unit*. For example, Area, Volume, or Weight units.

Introduction to Enumeration (Enum)

Java has an enum data type, which is a class having a fixed set of values. A simple enum acts like named constants:

```
public enum Size {
     SMALL,
     MEDIUM,
     LARGE;
}
```

This encourages type safety. If you define a variable of type Size it can only have values from the Size enum:

```
Size mysize = Size.SMALL;        // assign a value from enum
public int setSize(Size s){..}   // declare method than accepts a Size
tshirt.setSize( Size.MEDIUM );   // pass enum value to a method
tshirt.setSize( "MEDIUM" );      // Error. Parameter must be a Size.
```

An enum contains a fixed set of static members (final objects), so its OK to compare values using ==.

```
if (size == Size.SMALL) System.out.println("You're small");
else if (size == Size.MEDIUM) System.out.println("You are medium");
```

To get all the values of an enum, call the built-in **`values()`** method. For example:

```
for (Size size : Size.values() )
    System.out.println( size );
```

will print SMALL, MEDIUM, LARGE. `Sizes.values()` returns an array `Size[]`.

An enum is really a *class* with a *private constructor*. The enum values are static instances of the class. **An enum can have attributes**. Suppose we want SMALL to cost 0.5, MEDIUM to cost 1.0, and LARGE to cost 2.0. We could add a `cost` attribute to the enum:

```
public enum Size {
     SMALL(0.5),
     MEDIUM(1.0),
     LARGE(2.0);

     public double cost;
     private Size(double c) {
          this.cost = c;
     }
     public double getCost() { return cost; }
}
```

We can get the cost by writing: Size.SMALL.getCost().

Since enum are for *constants*, it is OK to declare the attributes public final so they can be accessed directly but cannot be changed. For example:

```
public enum Size {
     SMALL(0.5),
     MEDIUM(1.0),
     LARGE(2.0);
     public final double cost;
     // enum constructor must be private
     private Size(double c) {
          this.cost = c;
     }
```

```
}
```

Now we can access a value using Size.SMALL.cost.

Length enum

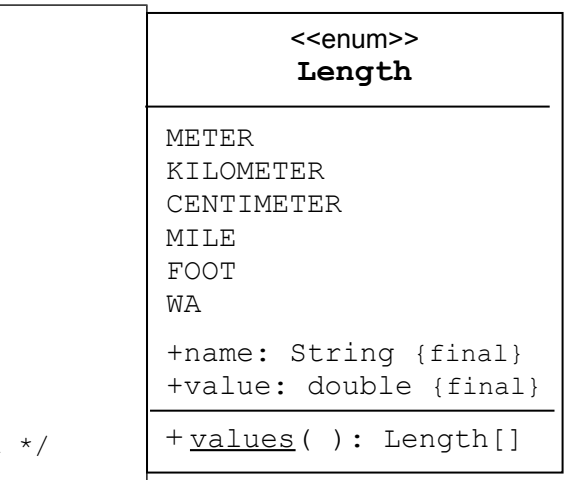
1. Create an enum named Length.java and insert all the length units you want to use:

```
/** A definition of common units of length. */
public enum Length implements Unit {
  /*  Define the members of the enumeration
   *  The attributes are:
   *  name = a string name for this unit,
   *  value = multiplier to convert to meters.
   */
     METER( "Meter", 1.0 ),
     FOOT( "Foot", 0.3048 ),

. . . // more Lengths     ;

     /** name of this unit */
     public final String name;
     /** multiplier to convert this unit to std unit */
     public final double value;

     /** Private constructor for enum members */
     Length(String name, double value){
          //TODO complete this
     }
     /** public properties of the enum members */
     public double getValue() { return value; }
     public String toString() { /* TODO */ }
}
```



The names of *static members* of the enum. Put a comma after each name except the last one.

Each element has a String **name** and a **value** (in meters).

Add more units. For example:

```
1 mile = 1609.344 meter
1 inch = 0.0254 meter
1 foot = 0.3048 meter
1 yard = 3 foot
1 micron = 1.0E-6 meter
1 kilometer = 1000 meter
1 wa = 2 meter (Thai unit)
```

2. *Test the enumeration.* Every enum has a built-in *static* function named **values( )** that returns all the enum members as an array. Using BlueJ Codepad:

```
> Length.MILE.getValue()
1609.344
> Length.WA.toString()
Wa
> Length.WA.getValue()
2.0
```

Java Code to print all the units:

```
Length [] lengths = Length.values( );
for(Length x: lengths) {
    System.out.printf("%s = %f\n", x.toString(), x.getValue());
}
```