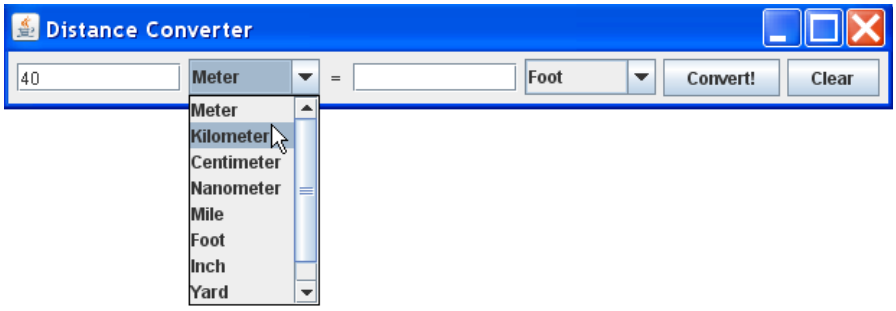
Objectives	1. Create a graphical interface for a distance converter. 2. Practice O-O design by using separate classes for different responsibilities. 3. Practice using an event handler (ActionListener) for user input events.

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

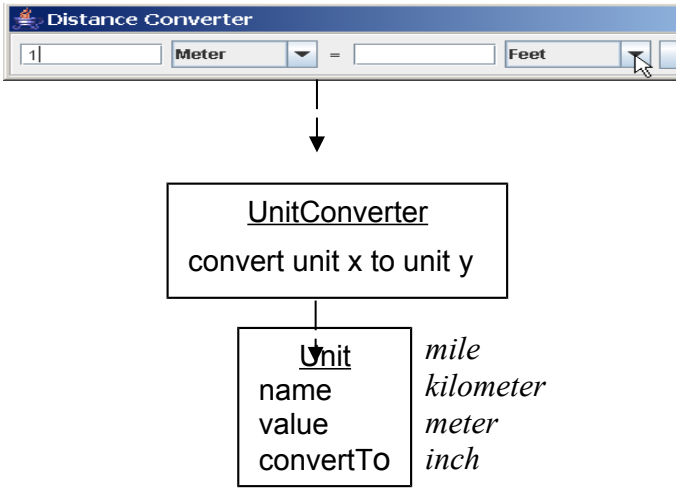


Parts of the Program

Your program needs 3 components, which have different responsibilities.

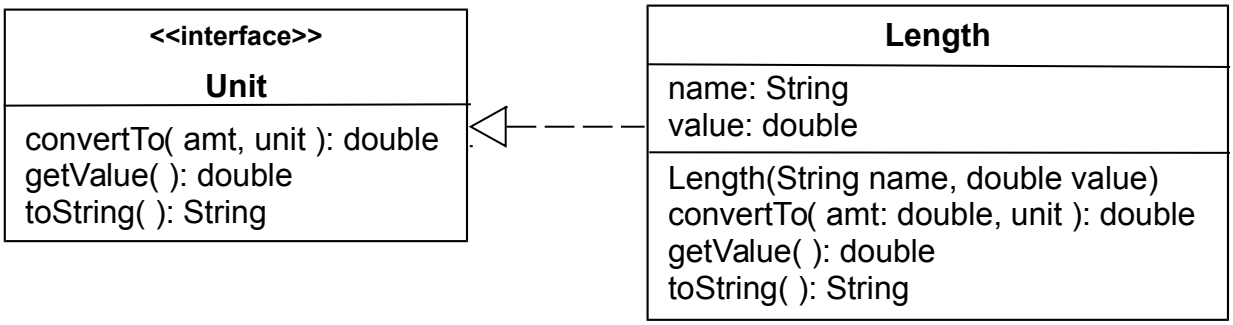
User Interface or View: handles interaction with the user. It handles input from the user and displays results.
It also catches **errors** in input, such as an invalid number, and notifies the user.
Controller: this layer processes the user's request. It knows how the application should behave.

Domain Layer: contains logic and other classes used by application, provides services.



1. Write the Unit interface and Length enum or class

Start by writing a Unit interface and Length class that can convert a value from one length unit to another. You don't really need a Unit interface for this lab, but it will help when you generalize the unit converter in PA3.



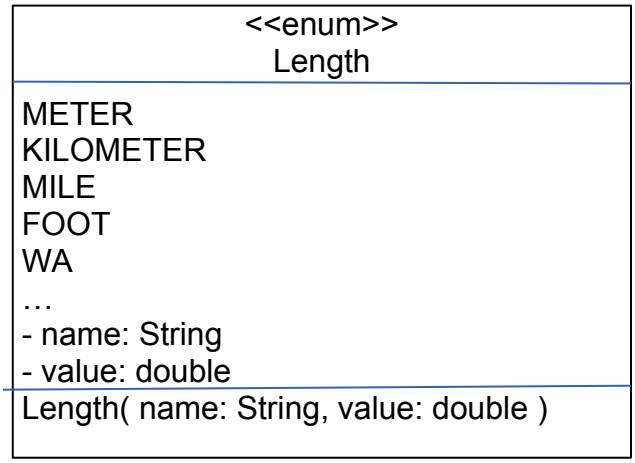
The value of a unit is a *multiplier* to convert this unit to a quantity of a base unit. For Length units, let meter be the base unit. To convert other units to meter, the multiplier values are:

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.

1.1 Using an Enum

Since we have a fixed collection of Length units, we can use an enum instead of a class. The only disadvantage of an enum is that the only way to add more Length units (like LIGHT_YEAR) is by editing the Length enum and recompiling it.



convertTo(double amount, Unit unit) converts an amount from one length unit to another length unit. I For example:

```
Length mile = LENGTH.MILE;
Length km = LENGTH.KILOMETER;
```

to convert 3 miles to kilometers you invoke

```
mile.convertTo(3.0, km ).
```

The conversion has 2 steps:

1) convert 3.0 *from* miles *to* the base unit: 3.0 * 1609.344

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

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

1.1 (Alternate Design) Length class

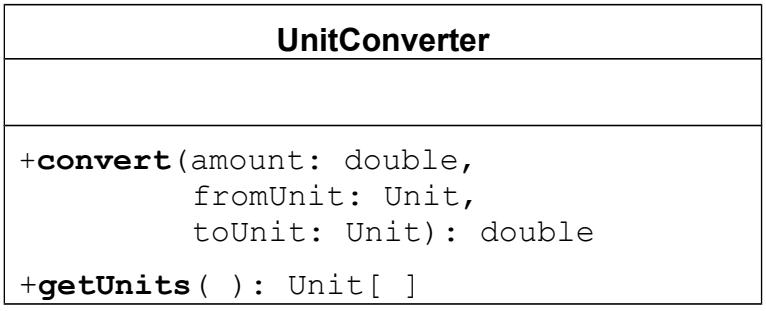
For the unit converter, you can use a class instead of enum.

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 receives requests from the UI to get all the available units (so the UI knows what is should display).

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



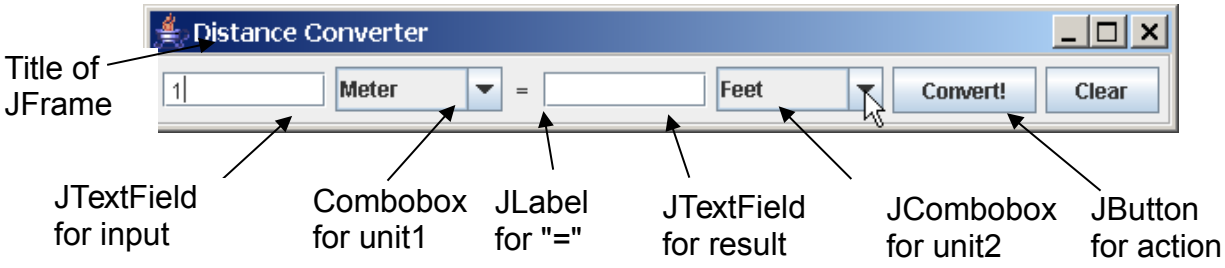
(If you didn't write a Unit interface, then use Length instead of Unit in these methods.)

Here are some examples:

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

Write and test both methods.

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



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;



    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 layout manager.

      Container contents = this.getContentPane();
      LayoutManager layout = new FlowLayout( );
      contents.setLayout( layout );
      convertButton = new JButton("Convert");
//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
    }

    class ConvertButtonListener implements ActionListener {

      /** method to perform action when the 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  handle 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 converter to convert value and display
                  // the results.
                  inputField2.setText( _____ );
            }
      }
    }  // end of the inner class for ConvertButtonListener
}
```

declare attributes for components your application needs to access. You may declare JLabel as local vars in initComponents() if you don't need to access their contents again.

Constructor does:
(1) receive a reference to the application
(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, and write result in other field.

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.

JComboBox also has a constructor that accepts an array of Objects as items and adds them all. This way avoids the need for 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 he getSelectedItem( ) method.

You probably want to do this in your ActionListener.

```
public void actionPerformed( ActionEvent evt ) {
      // get the selected item from first ComboBox
      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

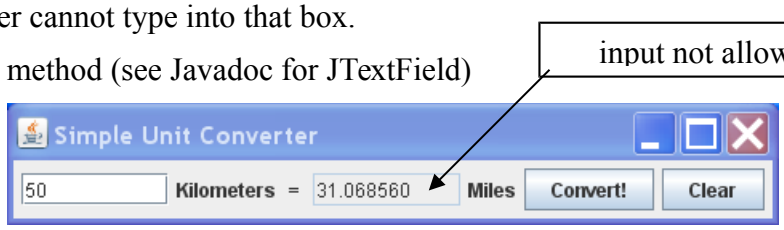
In layered software design, the user interface (upper layer) **should not** create the application or domain level objects. Instead, another class *sets a reference* to those objects into the user interface. This is called "dependency injection". It reduces coupling and encourages software reuse.

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)



1.9 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 on 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.

2.7 Define your own length units. For example,

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

Enumeration

Java has an enum data type, which is a class having a fixed set of values.

A simple enum contains just named constants:

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

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

Size mysize = Size.SMALL; // assign a value from enum

mysize = 1; // ERROR

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

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

An enum is really a *class* with a *private constructor*. The enum values are static instances of the class. You can create an enum with attributes, too. Suppose we want SMALL to have a value 0.5, MEDIUM to be 1.0, and LARGE to be 2.0.

```
public enum Size {
     SMALL(0.5),
     MEDIUM(1.0),
     LARGE(2.0);
     private double value;
     private Size(double v) {
          this.value = v;
     }
}
```

We'd access the values just like any other object: Size.SMALL.getValue( ).

Since enum are for constants, it is common to declars the attributes public final so they can be accessed directly but not changed. For example:

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

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

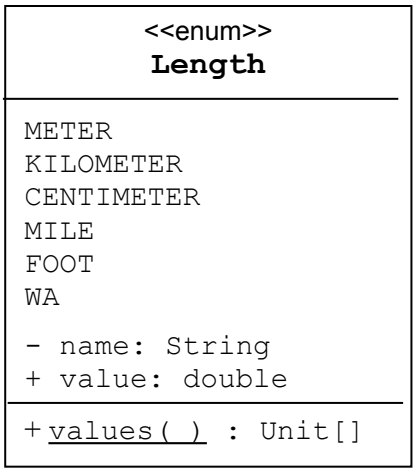
1. Create an enum named Length.java and insert all the length units you want to use:

```
/** A definition of common units of length. */
public enum Length {
  /*  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 );

//TODO add more length units

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

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



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 wa = 2 meter (Thai unit)
```

2. For convenience, the **value** attributes of each enum member is **public final** so that we can easily access the value. For example: `Length.MILE.value` is the same as `Length.MILE.getValue()`. Since the value is *final*, this doesn't break the encapsulation.

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

```
> Length.MILE.getValue()
1609.344
> Length.WA.toString()
Wa
> Length.WA.value
2.0
// print all the units
> Length [ ] array = Length.values();
> for( Length u : array ) System.out.println( u + " = " + u.getValue() );
```