Objectives:

1. Exercise use of abstract classes and interfaces.
2. Understand reasons for refactoring a set of classes in order to support various maintenance goals.

Description: The files `cse370/Labs/Lab4/*` comprise a small GUI application that allows users to enter an unsigned integer in some interval `[lowerBound ... upperBound]`. The GUI itself contains a text-entry box, into which the user may type the integer, and a reset button. The text-entry box ensures that any input is valid and within the prescribed range when the user types the enter key. In addition, this text-entry object is being actively monitored (every hundredth of a second) for changes by the main event loop that drives the GUI. When the value in the text-entry box has changed, the event loop queries the new value and prints this value to standard error. The code for the event loop is encapsulated in a function called `processGUIBehavior`. The text-entry box itself, an object of class `IvalText` provides public methods using which the event loop can make these queries.

Unfortunately, this design suffers two problems. First, the monitoring code in the event operates on an object of class `IvalText`. This complicates replacing the text-entry box with another widget, say a graphical slider, that also allows the user to enter an unsigned integer in some range. In fact, we have a class `IvalSlider` that we would like to use in some configurations as a replacement for text-entry objects. It should be easy to swap out an `IvalText` with an `IvalSlider`, but the current design does not allow this. In fact, we should be able to make such an exchange without even recompiling the event-loop code. Second, while there is a reset button in this application, at the moment, it doesn’t do anything. We would like to extend class `Button` so that it is easy to make objects (in this case `IvalSlider` or `IvalText` objects) collaborate with the button.

Tasks:

1. Create an interface class called `Ival`, that abstracts the public interfaces of both `IvalText` and `IvalSlider`. Then go modify the code to use this interface appropriately. After your modifications, it should be possible to edit `app.cc` to use either an `IvalSlider` or an `IvalText` object without having to recompile `processGUIBehavior.cc`. Demonstrate this capability to your TA.
2. Create an interface called ButtonListener that provides an operation buttonPressed, which will be invoked with the label of the button that is pressed. It is intended that any object that implements this interface will be able to listen for buttonPressed events.

3. Now go modify class Button to:
   
   (a) allow objects that implement the ButtonListener interface to register interest in buttonPressed events; and
   
   (b) to notify all registered listeners whenever the button is pressed.

   Note: Whenever the button is pressed, the method Button::pressEvent () will be called.

4. Now go create a class myIvalText that extends class IvalText and implements the ButtonListener interface. When this class services a buttonPressed event, it should reset the value in the text box. Demonstrate this capability to the TA.

5. Repeat the last task for class IvalSlider, creating a new class myIvalSlider. Demonstrate this capability to the TA.