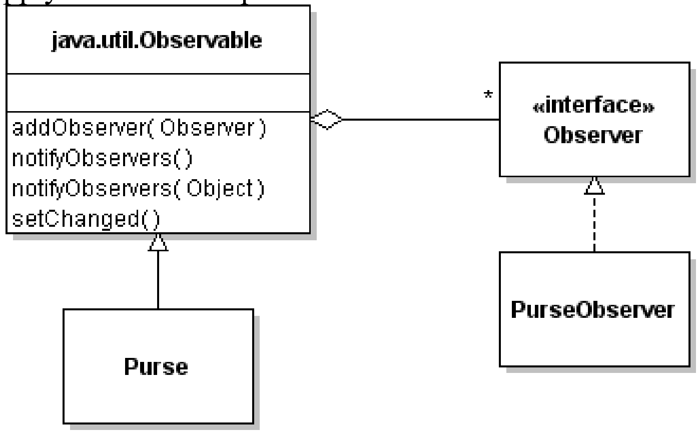
Objectives	Apply the Observer Pattern to the Coin Purse. Create at least 2 graphical observers that display different information about the Purse. Use the package **`coinpurse.gui`** for your graphical observers.
What to Submit	Commit the work to your existing purse project on Github.

Problem 1: Create an Observable Purse

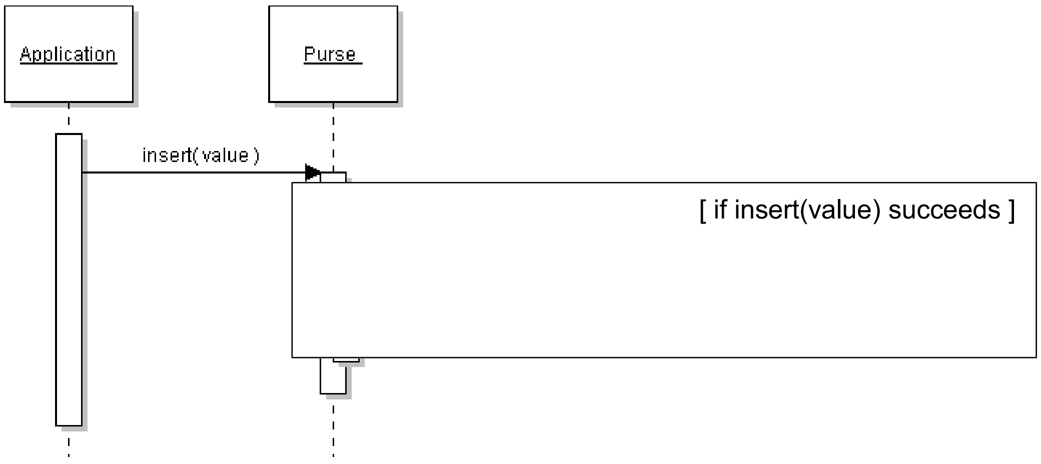
Implement the Observer Pattern for the coin purse, so that other objects can be notified when the state of the Purse changes. Apply the Observer pattern to the Purse like this:



The Purse has to notify observers when the state of the Purse changes.

What methods can change the *state* of the Purse? ________________________________

When these methods are invoked the Purse should notify the observers <u>if something changes</u> (but not if there is no change). The sequence of actions looks like this:



Problem 2: Create Graphical Observers

Create at least 2 graphical observers that shows the status of the purse in a window.

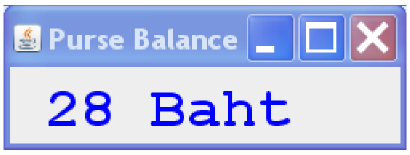
The Observers must *implement* the Observer interface. The update( ) method should update the user interface using information from the Purse when it is invoked.
Here is a console example (your code should use GUI components):

```
public class PurseObserver implements Observer {
    public PurseObserver( ) {
        // nothing to initialize
    }
    /** update receives notification from the purse */
    public void update(Observable subject, Object info) {
        if (subject instanceof Purse) {
            Purse purse = (Purse)subject;
```

```
                int balance = purse.getBalance();
                System.out.println("Balance is: " + balance);
            }
        }
```

Another way to write an Observer is to give it a *reference to Purse* in the constructor. The advantage of that is the observers can initialize their display to correct values, without waiting for update( ) to be called.

2.1 *Purse Balance Observer:* Write an observer to display the balance in the Purse.

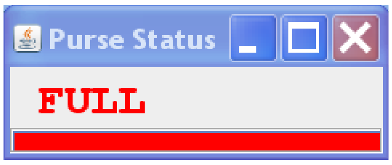


28 Baht

NOTE: Please don't *assume* that the currency is Baht. Can you fix getBalance() to include currency? Consider defining a Money class to represent a quantity of Money, with currency.

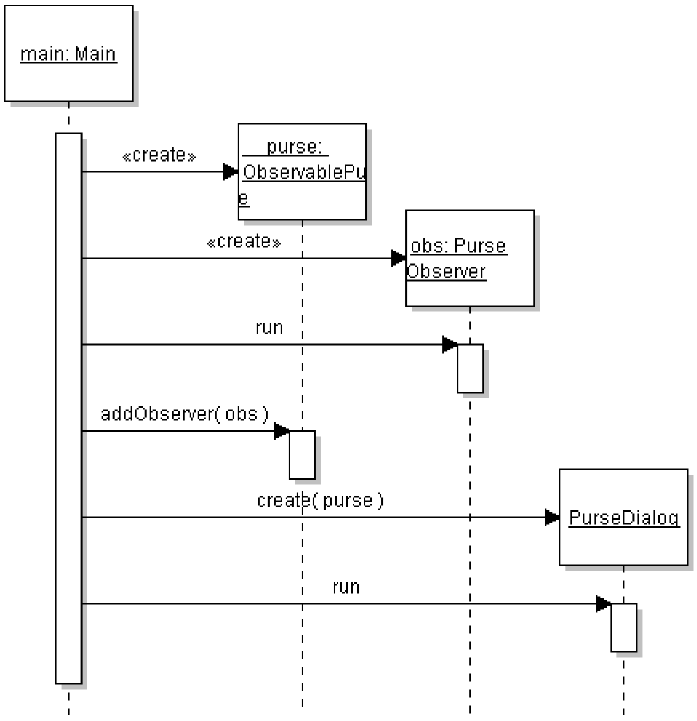
2.2 *Purse Status Observer*: Write an observer that displays "FULL", "EMPTY", or number of objects in the Purse (when not full or empty).

Use a JProgressBar to show the % full, and aTextField to display "FULL", "EMPTY", or number of items.

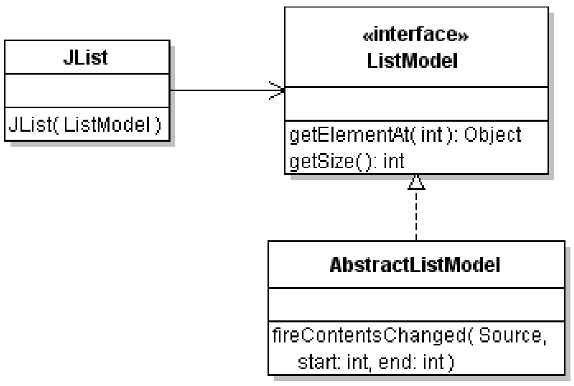


FULL

2.3 Modify the Main class to create the observers and add Observers to the Purse. Main should create a Purse and Observers, and add observers to the Purse.



Problem 3: Use JList to Display Purse Contents


JList
JList(ListModel)
«interface»
ListModel
getElementAt(int): Object
getSize( ): int
AbstractListModel
fireContentsChanged(Source, start: int, end: int)

JList is a Swing component that displays what is in a List. It gets values from a ListModel.

JList calls methods of ListModel to get the data. The data can be any kind of object.

When the list data changes, the ListModel should *notify* JList by calling **fireContentsChanged**.

To use JList, you need to write a ListModel that knows what is in the Purse.

3.1 Modify the Purse and add a method that returns an *immutable list view* of the Purse contents (to protect the Purse). Use **Collections.unmodifiableList( )** to create a *view* of the Purse's list.

Collections.unmodiableList( ) is a *view* of another List, so it changes when the underlying List changes -- you don't need to create a new view each time the purse changes!

3.2 Write a class named PurseListModel that implements ListModel interface. An easy way to write an adapter is to extend the AbstractListModel class. You will only need to write a constructor and two methods: getSize() and getElementAt() methods.

```
/**
 * This class provides a ListModel interface for querying
 * the contents of the purse. It is a kind of adapter.
 */
class PurseListModel extends AbstractListModel<Valuable> . . . {
    // TODO define attributes for what you need to remember
    public PurseListModel( Purse purse ) {
          /* get a reference to purse's list of money */
    }
    // TODO you must implement these methods
    public Valuable getElementAt( int index ); // Get one element
    public int getSize( );         // Returns the size of the list
}
```

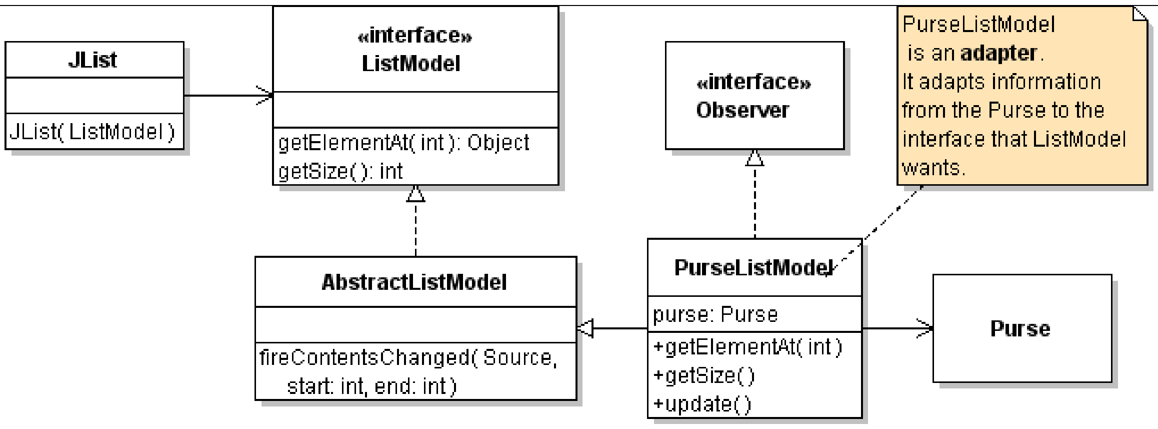
3.3 PurseListModel should be an *Observer*. It gets notified when the Purse changes.

3.4 PurseListModel needs to "update" the JList display when the Purse changes. When the Purse notifies Observers, the PurseListModel notifies JList by calling fireContentsChanged( ):

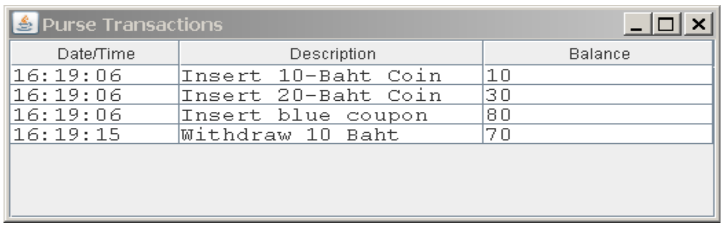
```
    public void update(Observable subject, Object info) {
         //TODO set size = number of items in the purse
         //Notify the JList that ListModel has changed.
         fireContentsChanged( this, 0, size-1 );
    }
```

3.5 Once you have a PurseListModel, you can create a JList and show it in a frame.

```
ListModel<Valuable> listModel = new PurseListModel( purse );
JList<Valuable> listview = new JList<>( listModel );
//TODO ensure that listModel is an observer of Purse
//TODO add JList to a JFrame or JDialog window
```


JList
JList(ListModel)
«interface» ListModel
getElementAt(int): Object
getSize(): int
«interface» Observer
PurseListModel is an **adapter**. It adapts information from the Purse to the interface that ListModel wants.
AbstractListModel
fireContentsChanged(Source, start: int, end: int)
PurseListModel
purse: Purse
+getElementAt(int)
+getSize()
+update()
Purse

Optional: Use a JTable to Display Transactions



Date/Time	Description	Balance
16:19:06	Insert 10-Baht Coin	10
16:19:06	Insert 20-Baht Coin	30
16:19:06	Insert blue coupon	80
16:19:15	Withdraw 10 Baht	70

JTable is a Swing component that display data in a table. It gets values from a TableModel.

There are several ways to create a TableModel. For simple tables, you can use a DefaultTableModel. For more flexible tables, extend AbstractTableModel (similar to AbstractListModel but you need to write more methods).

The Purse Transactions will be an Observer of the Purse. Design the code for this yourself.

To create a JTable with a DefaultTableModel use:

```
TableModel model = new DefaultTableModel();
// our table data will have 3 columns
model.addColumn("Date/Time");
model.addColumn("Description");
model.addColumn("Balance");
JTable table = new JTable( model );
// the table might be long, so add scroll bars using a JScrollPane
JScrollPane pane = new JScrollPane( table );
pane.setVerticalScrollBarPolicy(
                JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
table.setFont( font ); // optional: set Font in table
pane.setMinimumSize(new Dimension(80,30));
this.add( pane ); // add the scrollpane, NOT the table
```

In Purse, when you notify Observers you can send them an Object as parameter, for example:

```
notifyObservers("Deposited 1,000 Baht");
```

The object in notifyObservers is the second parameter of the update( ) method in Observer.

In your Observer, you can add a new row to the table model using:

```
     model.addRow( new Object[] { time, info, balance} );
```

when you change a DefaultTableModel it automatically invokes a "fire" method to notify the view (JTable).
The type of data in each column can be different. For example, a DateTime object, a String, and a Double. JTable will choose a format to match the datatype in each column. You can also override the getColumnClass method and specify the class for each column, which may improve formatting.