Section 1 - More GUI

This is an optional section C, which goes beyond the required GUI study of this course.

10C.1.1 Tansporter GUI, Next Generation

We will expand on our Transporter to "sanitize" the name that the user types in. We want to make sure it contains two words (first last) and only two, and each one should be at least two characters long. Also, we will not allow anyone to see "Michael Loceff."

Here are the screen shots:

Here is the source:

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

public class Foothill
{
   public static void main(String[] args)
   {
      // establish main frame in which program will run
      TranspoFrame myTranspoFrame = new TranspoFrame("Transporter Room");
      myTranspoFrame.setSize(300, 200);
      myTranspoFrame.setLocationRelativeTo(null);
      myTranspoFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      // show everything to the user
      myTranspoFrame.setVisible(true);
   }
}  // end of class Foothill
```

```
// TranspoFrame class is derived from JFrame class
class TranspoFrame  extends JFrame
{
   private JButton btnMyButton;
   private JTextField txtMyTextArea;
   private JLabel lblMyLabel;

   // TranspoFrame constructor
   public TranspoFrame(String title)
   {
      // pass the title up to the JFrame constructor
      super(title);

      // set up layout which will control placement of buttons, etc.
      FlowLayout layout = new FlowLayout(FlowLayout.CENTER, 5, 20);
      setLayout(layout);

      // 3 controls: a label, a text field and a button
      lblMyLabel = new JLabel("Friend's Name: ");
      txtMyTextArea = new JTextField(10);
      btnMyButton = new JButton("Press Here to See Friend");
      add(lblMyLabel);
      add(txtMyTextArea);
      add(btnMyButton);

      btnMyButton.addActionListener( new SeeFriendListener() );
      txtMyTextArea.addActionListener( new SeeFriendListener() );
   }
```

```
   // inner class for JButton Listener
   class SeeFriendListener implements ActionListener
   {
      // event handler for JButton
      public void actionPerformed(ActionEvent e)
      {
         String strFriendName;
         String[] subNames = {"first", "last"};
         boolean errorFlag;


         strFriendName = txtMyTextArea.getText();
         if (strFriendName == null || strFriendName.length() < 3)
            errorFlag = true;
         else if (strFriendName.indexOf(",") >= 0)
            errorFlag = true;
         else if (strFriendName.equalsIgnoreCase("Michael Loceff"))
         {
            JOptionPane.showMessageDialog(null, "I'm sorry, but no one"
               + " can see Michael Loceff, please choose someone else.");
            return;
         }
         else
         {
            // break into first and last using space
            subNames = strFriendName.split(" ", 0);
            if (subNames.length != 2)
               errorFlag = true;
            else if (subNames[0].length() < 2)
               errorFlag = true;
            else if (subNames[1].length() < 2)
               errorFlag = true;
            else
               errorFlag = false;
         }

         if (errorFlag)
         {
            JOptionPane.showMessageDialog(null,
                  "Name must be '[first][last]', and  [first]\n"
                  + "and [last} must each contain at least 2 letters.");
            return;
         }

         JOptionPane.showMessageDialog(null, "Searching for "
               + subNames[1] + ", " + subNames[0] + " ...");
      }
   } // end inner class SeeFriendListener
} // end class TranspoFrame
```

Do you notice anything else different about this program? What about these two lines:

```
btnMyButton.addActionListener( new SeeFriendListener() );
txtMyTextArea.addActionListener( new SeeFriendListener() );
```

In the original version only the first line appeared. We are allowing the same ***Listener*** class to listen for an event in the **JTextField**. What would an ***action event*** be in a text field? It is considered the event of the user hitting the **enter** key while the focus is in the text field. Normally, users type things in text fields and, when done, hit **enter**. We want this to be an alternate way for the user to say, "I'm done entering my friend's name." This would give the user the option of hitting **enter** rather than pressing a ***button***.

Also, there is no reason to create two ***listener*** objects for these two controls, since each listener is going to do the same thing. To fix this, we create a **SeeFriendListener** object with a name, as opposed to two ***anonymous*** objects as we did above. It would look like this:

```
ActionListener myListener = new SeeFriendListener();
btnMyButton.addActionListener( myListener );
txtMyTextArea.addActionListener( myListener );
```

Look at the ***if/else*** complex that we use to filter out bad user input. This is typical of such input filters, and makes use of the **String** functions like **indexOf(), equals(), length() and split().**

Section 2 - Multiple Windows

10C.2.1 Instantiating a Second JFrame

We have our main **JFrame** described in our class called **TranspoFrame**. So far, if the user entered a legal name (**first *space* last**) we just gave them a **JOptionPane** message dialog as a place holder. But now we want to proceed with the actual program. We would next find out what level of interaction they desire: text chat, listen only, talk and listen, full 3D person delivered into room, etc. In place of that **JOptionPane** dialog, we want to display a second **JFrame** like so:

So, if the name checks out, we will send this second frame to the screen. But how? Clearly, we would do it where we have the OptionPane method call at the end of the **SeeFriendListener** class's **actionPerformed()** method.

```
     if (errorFlag)
     {
        JOptionPane.showMessageDialog(null,
            "Name must be '[first][last]', and  [first]\n"
            + "and [last} must each contain at least 2 letters.");
        return;
      }

      frmSecond = new SecondFrame(strFriendName);
      frmSecond.setSize(500, 250);
      frmSecond.setLocationRelativeTo(null);

      // show second frame to the user
      frmSecond.setVisible(true);
```

Remember, this code is at the tail end of the **actionPerformed()** method and appears after we have done the tests on the name to see if it was legal. If the **errorFlag** is not set to true, then we will proceed to the last few lines that create a new **JFrame** called a **SecondFrame**. But what is a **SecondFrame**? Well, that's the new class we have to design now, and we have to make it display all of the controls you see in the image above.

10C.2.2 Defining the Second JFrame Subclass

Here is a preview of this class, minus the all-important constructor:

```
//SecondFrame class is derived from JFrame class
class SecondFrame  extends JFrame
{
   JLabel lblName, lblTaxes, lblPrice;
   JPanel panelMain, panelLeft, panelCenter, panelRight;
   JComboBox cmboRealityLevels;
   JRadioButton btnLstn, btnTalk, btnFull;
   ButtonGroup btngp;
   JButton btnOK;

   // constructor
   public SecondFrame(String strFriend)
   {
      // to be filled in very shortly ... !
   }
}
```

The important thing to understand is that you can always create a new custom **JFrame** subclass (inherited from the base **JFrame** class), and when you do, you will place the main ***components*** that you expect to display in the new frame into the top of the class as instance members. (They can, and probably should be, declared **private**, but I didn't do that here in order to keep the listing clean looking). Once defined, you can create the new frame wherever you want. Usually

this happens in an event handler because the user clicked some button or hit the enter key, beckoning the new frame to pop up. That's what you see above in the first code fragment: We instantiated it inside the **TranspoFrame's actionPerformed()** method, at the point where we knew we had passed the first tests.

All this is simple, except we haven't said two things:

- What are all the new components, **JPanel, JButtonGroup, ButtonGroup,** etc.?
- How do we get the new **SecondFrame** object to look like the picture above?

You'll have to turn the page to get the answers to these questions.

Section 3 - Panels and Layouts

10C.3.1 BorderLayout

In the past, we used **FlowLayout**, which is the simplest. In **FlowLayout**, whenever you add a new component, it just appears to the right of the previous one in the frame, and when there is no room, it goes on the next line. This is just like ordinary typing with word wrap.

In **BorderLayout**, there are up to five sections of the frame: **NORTH, SOUTH, EAST, WEST** and **CENTER**. You can look in the text for pictures of the placement of each of these, but I think they are generally pretty intuitive. The only thing to note is that you can omit a section and, if you do, some other section will stretch out to take its place.

In our current program, we are going to imbue our **SecondFrame** class with **BorderLayout**, but only use **NORTH, CENTER,** and **SOUTH**. The main part of our **SecondFrame** will occur in the **CENTER** section, while the **NORTH** and **SOUTH** sections simply have text labels in them:

Look at how this is done in the code (we are in the constructor of the **SecondFrame** class):

```
//SecondFrame class is derived from JFrame class
class SecondFrame  extends JFrame
{
   JLabel lblName, lblTaxes;
   JPanel panelMain;
   // other variables temporarily omitted

   // constructor
   public SecondFrame(String strFriend)
   {
      // chain to base class constructor
      super("Transporter Processing Information");

      // set up three main components
      lblName = new JLabel("Before you can see "
            + strFriend + " I need some information.");
      lblTaxes = new JLabel("(Prices are subject to local taxes)");
      panelMain = new JPanel( new GridLayout(1, 3, 10, 10));

      // use border layout and put three above components in frame
      setLayout (new BorderLayout(20, 10));
      add(lblName, BorderLayout.NORTH );
      add(panelMain, BorderLayout.CENTER);
      add(lblTaxes, BorderLayout.SOUTH);

      // more to come in a few moments ...

   }
}
```

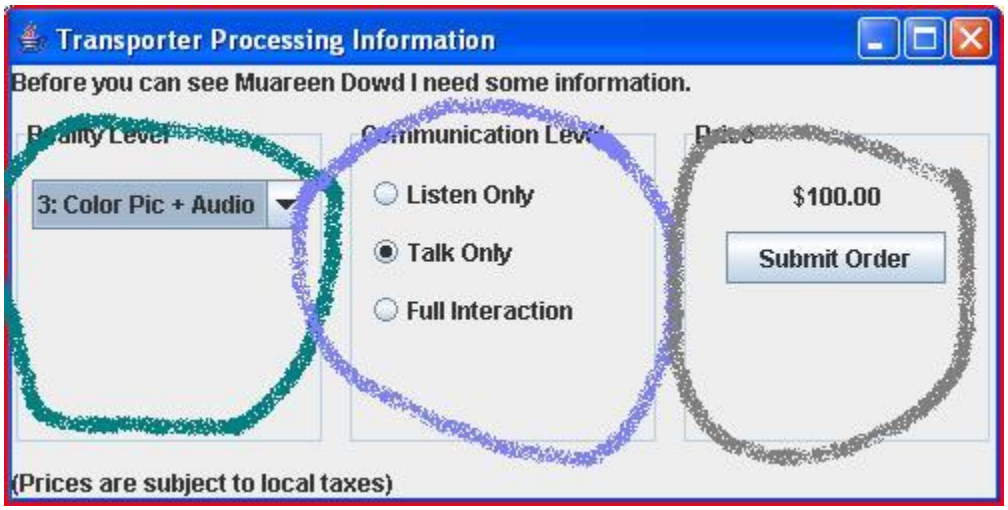
We can almost understand this.

Here are the missing details:

- We're not sure what a **JPanel** is, but whatever it is, we are creating one called **panelMain** and placing it in the large center area of the main frame.
- We created the **JPanel** by passing something called a **GridLayout** object to its constructor. We'll see what that means in the next section.
- When we set the layout, using **setLayout()**, in addition to seeing a **BorderLayout** constructor call, we see two numbers, **(20, 10)** passed to the constructor. These are ***padding values*** (in pixels) describing the gaps between components. We are asking for 20 pixels horizontally and 10 pixels vertically to be placed between the components.
- When we **add()** components to a **BorderLayout** container, we have to specify **NORTH**, **CENTER**, etc. I'll bet you can figure out what that syntax is: **BorderLayout.SOUTH** must be a ***public static member*** of the **BorderLayout** class. Since it is a constant, it's probably also **final**. This is not essential to your understanding this program, but it is important to be able to identify such syntax.

10C.3.2 JPanels and GridLayout

I can't keep the secret any longer. A **JPanel** is an invisible rectangular container in which we place other components. We created one called **panelMain** in our code above and placed it in the center of the above **BorderLayout Jframe** because we wanted to add lots of stuff to it. Now it's time to add that stuff.



Whenever we add things to any container, like a **JFrame** or a **JPanel**, we have to do so using a layout manager. This time we are going to use a **GridLayout**. **GridLayout** allows you to break the frame or panel into a rectangular grid: 2 x 2, 3 x 5, 10 x 10, whatever. In this situation we want three main sub-areas, so we'll create a 1 x 3 grid. This was actually done when we instantiated the **panelMain**, and in that instantiation, we saw not only the 1 x 3 numbers, but also 10, 10. Those 10s were the horizontal and vertical gaps, again, padding the components. So the layout for the **JPanel** was done when we instantiated it.

Now, we are going to place more **JPanels** inside this **Panel**. I'll circle them so you can see the layout:

Once we declare this layout to be **GridLayout** and specify 1x3 (one row, three columns) we can then add exactly three items to the **JPanel**, understood to be placed from left to right. Here is the code:

```
JPanel panelMain, panelLeft, panelCenter, panelRight;

// create three sub-panels to put into panelMain
panelLeft = new JPanel( new FlowLayout(FlowLayout.CENTER, 5, 10));
panelCenter = new JPanel( new FlowLayout(FlowLayout.LEFT, 5, 5));
panelRight = new JPanel( new FlowLayout(FlowLayout.CENTER, 5, 10));
panelMain.add(panelLeft);
panelMain.add(panelCenter);
panelMain.add(panelRight);
```

I wisely called the three **JPanels**, **panelLeft, panelCenter** and **panelRight**. As you can see, when we created them, we decided to go back to **FlowLayout** within those sub-panels. That's

because they are so small and will contain so few components, that we can predict exactly where those components will appear based on the size of the main **SecondFrame** (see the **actionPerformed()** method of the **SeeFriendListener** class for the dimensions) and the size of the combo boxes and buttons we are placing into these panels.

The **FlowLayout** constructors take horizontal and vertical gaps as do all layout manager constructors. We're using 5s and 10s here. I picked those values by trial and error during my program debugging.

10C.3.3 Panel Borders

I said that **JPanels** were invisible. But you can make them visible by doing one of two things (or both):

- Placing components into them (which we shall do)
- Putting borders around them (which we shall also do)

Here's how to put a border around the panels:

```
// place borders around three sub-panels
panelLeft.setBorder(new TitledBorder("Reality Level"));
panelCenter.setBorder(new TitledBorder("Communication Level"));
panelRight.setBorder(new TitledBorder("Price"));
```

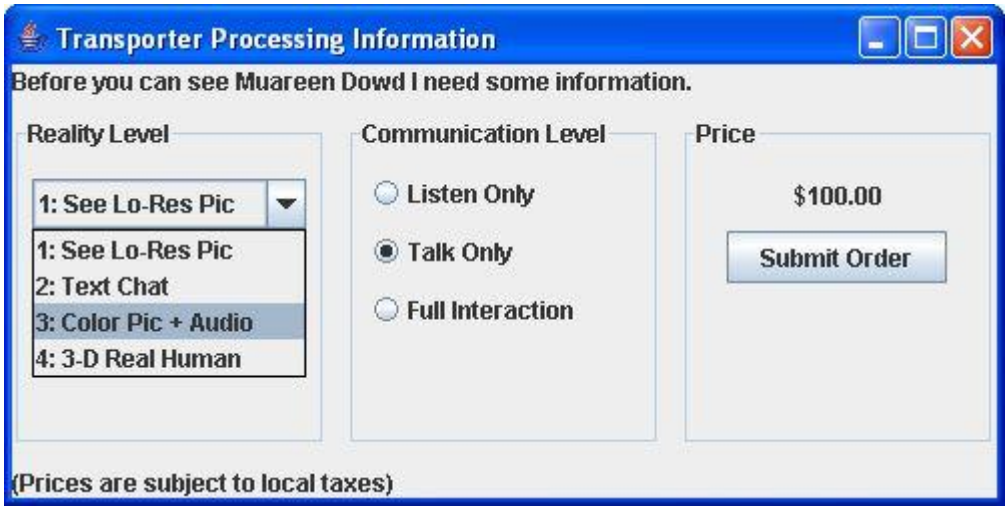
As you can see, we're ***instantiating anonymous*** **TitledBorder** ***objects*** to pass to the **setBorder()** ***instance method*** of the **JPanel** class. (That's actually a good statement for you to analyze.) We need a new import to deal with **JPanel** borders, import **javax.swing.border.*.**

Now we have to add the actual visible components, **JRadioButtons, JComboBoxes** and **JLabels.** That's next.

Section 4 - JComboBoxes and ButtonGroups

10C.4.1 The JComboBox

In the leftmost panel of our center area, called **panel_left**, I placed what you will recognize as a pull-down menu or a **combo box**. The actual class is **JComboBox**. It can contain any number of items. You create an array of Strings, representing the choices in the box, and pass that array to the **JComboBox** constructor. First, here's what we want:



And here's how to create it:

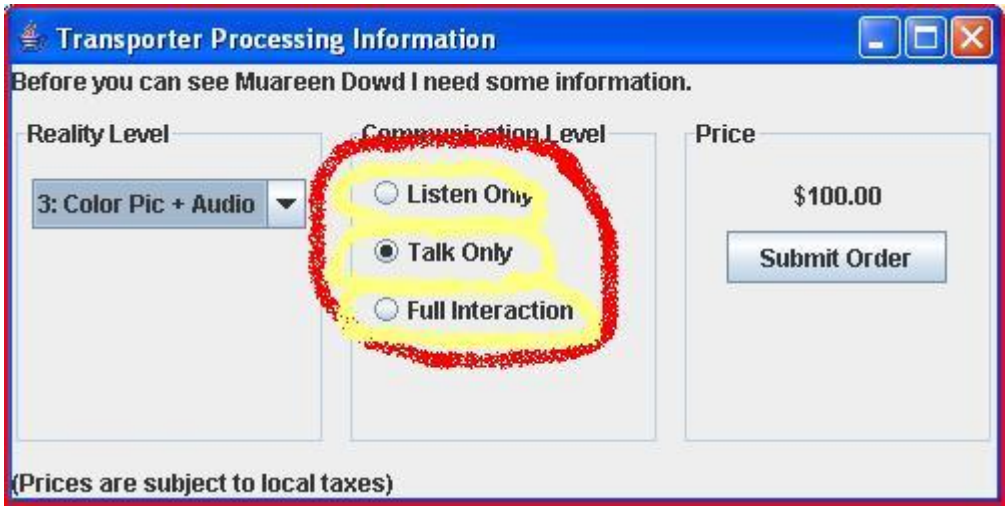
```
   JComboBox cmboRealityLevels;

  // put a combo box in panelLeft
   String[] strRealityLevels
         = {"1: See Lo-Res Pic", "2: Text Chat",
         "3: Color Pic + Audio", "4: 3-D Real Human"};
   cmboRealityLevels = new JComboBox(strRealityLevels);
   panelLeft.add(cmboRealityLevels);
```

10C.4.2 Button Groups

As you can see from the picture, the center panel, **panelCenter**, has three radio buttons. ***Radio buttons***, or **JRadioButtons**, have "radio-button-ness" by virtue of two properties that you take for granted:

1. They are small round buttons, not large square ones.
2. When you push one, any others will get un-pushed.

We create effect #1 by simply declaring them as **JRadioButton** objects. But effect #2 doesn't happen automatically. To do that we have to place all the **JRadioButton** objects into a group called a **ButtonGroup**. (That's right, no J). When you do that, the buttons know about one another and will turn off and on in synchronization with each other.



Here is the code:

```
JRadioButton btnLstn, btnTalk, btnFull;
ButtonGroup btngp;

// put a button group in panelCenter
btnLstn = new JRadioButton("Listen Only");
btnTalk = new JRadioButton("Talk Only");
btnFull = new JRadioButton("Full Interaction");
btngp = new ButtonGroup();
btngp.add(btnLstn);
btngp.add(btnTalk);
btngp.add(btnFull);

// place buttons in panel, top to bottom
panelCenter.add(btnLstn);
panelCenter.add(btnTalk);
panelCenter.add(btnFull);
```

10C.4.3 The Final Controls

The rightmost **JPanel, panelRight,** is going to get two controls we have already seen, **JLabel** and **JButton**. Here is the code:

```
// put the price and ok button in panelRight
lblPrice = new JLabel("$100.00");
btnOK = new JButton("Submit Order");
panelRight.add(lblPrice);
panelRight.add(btnOK);
```

And with that, we have completed the design of the **SecondFrame**. I'll put the code for the entire program on the next page.

Section 5 - TranspoFrame - The New Listing

10C.5.1 Listing

We've already discussed every line of this program somewhere, so here it is. Copy and paste it into your IDE and have fun with it. We have to learn next how to handle the new events that we are throwing at it. For now, the second frame is sort of brain dead. We'll bring it to life shortly.

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

public class Foothill
{
   public static void main(String[] args)
   {
      // establish main frame in which program will run
      TranspoFrame myTranspoFrame = new TranspoFrame("Transporter Room");
      myTranspoFrame.setSize(300, 200);
      myTranspoFrame.setLocationRelativeTo(null);
      myTranspoFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      // show everything to the user
      myTranspoFrame.setVisible(true);
   }
}

// TranspoFrame class is derived from JFrame class
class TranspoFrame  extends JFrame
{
   private JButton btnMyButton;
   private JTextField txtMyTextArea;
   private JLabel lblMyLabel;

   // TranspoFrame constructor
   public TranspoFrame(String title)
   {
      // pass the title up to the JFrame constructor
      super(title);

      // set up layout which will control placement of buttons, etc.
      FlowLayout layout = new FlowLayout(FlowLayout.CENTER, 5, 20);
      setLayout(layout);

      // Frame will have 3 controls: a label, a text field and a button
      lblMyLabel = new JLabel("Friend's Name: ");
      txtMyTextArea = new JTextField(10);
      btnMyButton = new JButton("Press Here to See Friend");
      add(lblMyLabel);
      add(txtMyTextArea);
      add(btnMyButton);
```

```
   ActionListener myListener = new SeeFriendListener();
   btnMyButton.addActionListener( myListener );
   txtMyTextArea.addActionListener( myListener );
}

// inner class for JButton and JTextField Listener
class SeeFriendListener implements ActionListener
{
   SecondFrame frmSecond;

   // event handler for JButton
   public void actionPerformed(ActionEvent e)
   {
      String strFriendName;
      String[] subNames;
      boolean errorFlag;

      strFriendName = txtMyTextArea.getText();
      if (strFriendName == null || strFriendName.length() < 3)
         errorFlag = true;
      else if (strFriendName.indexOf(",") >= 0)
         errorFlag = true;
      else if (strFriendName.equalsIgnoreCase("Michael Loceff"))
      {
         JOptionPane.showMessageDialog(null,
               "I'm sorry, but no one can "
               + "see Michael Loceff, please choose someone else.");
         return;
      }
      else
      {
         // break into first and last using space
         subNames = strFriendName.split(" ", 0);
         if (subNames.length != 2)
            errorFlag = true;
         else if (subNames[0].length() < 2)
            errorFlag = true;
         else if (subNames[1].length() < 2)
            errorFlag = true;
         else
            errorFlag = false;
      }

      if (errorFlag)
      {
         JOptionPane.showMessageDialog(null,
               "Name must be '[first][last]', and  [first]\n"
               + "and [last} must each contain at least 2 letters.");
         return;
      }

      frmSecond = new SecondFrame(strFriendName);
      frmSecond.setSize(500, 250);
      frmSecond.setLocationRelativeTo(null);
```

```
         // show second frame to the user
         frmSecond.setVisible(true);

      }
   } // end inner class SeeFriendListener
} // end class TranspoFrame

//SecondFrame class is derived from JFrame class
class SecondFrame  extends JFrame
{
   JLabel lblName, lblTaxes, lblPrice;
   JPanel panelMain, panelLeft, panelCenter, panelRight;
   JComboBox cmboRealityLevels;
   JRadioButton btnLstn, btnTalk, btnFull;
   ButtonGroup btngp;
   JButton btnOK;

   // constructor
   public SecondFrame(String strFriend)
   {
      // chain to base class constructor
      super("Transporter Processing Information");

      // set up three main components
      lblName = new JLabel("Before you can see "
            + strFriend + " I need some information.");
      lblTaxes = new JLabel("(Prices are subject to local taxes)");
      panelMain = new JPanel( new GridLayout(1, 3, 10, 10));

      // use border layout and put three above components in frame
      setLayout (new BorderLayout(20, 10));
      add(lblName, BorderLayout.NORTH );
      add(panelMain, BorderLayout.CENTER);
      add(lblTaxes, BorderLayout.SOUTH);

      // create three sub-panels to put into panelMain
      panelLeft = new JPanel( new FlowLayout(FlowLayout.CENTER, 5, 10));
      panelCenter = new JPanel( new FlowLayout(FlowLayout.LEFT, 5, 5));
      panelRight = new JPanel( new FlowLayout(FlowLayout.CENTER, 5, 10));
      panelMain.add(panelLeft);
      panelMain.add(panelCenter);
      panelMain.add(panelRight);

      // place borders around three sub-panels
      panelLeft.setBorder(new TitledBorder("Reality Level"));
      panelCenter.setBorder(new TitledBorder("Communication Level"));
      panelRight.setBorder(new TitledBorder("Price"));

      // put a combo box in panelLeft
      String[] strRealityLevels = {"1: See Lo-Res Pic", "2: Text Chat",
            "3: Color Pic + Audio", "4: 3-D Real Human"};
      cmboRealityLevels = new JComboBox(strRealityLevels);
      panelLeft.add(cmboRealityLevels);
```

```
        // put a button group in panelCenter
        btnLstn = new JRadioButton("Listen Only");
        btnTalk = new JRadioButton("Talk Only");
        btnFull = new JRadioButton("Full Interaction");
        btngp = new ButtonGroup();
        btngp.add(btnLstn);
        btngp.add(btnTalk);
        btngp.add(btnFull);

        // place buttons in panel, top to bottom
        panelCenter.add(btnLstn);
        panelCenter.add(btnTalk);
        panelCenter.add(btnFull);

        // put the price and ok button in panelRight
        lblPrice = new JLabel("$100.00");
        btnOK = new JButton("Submit Order");
        panelRight.add(lblPrice);
        panelRight.add(btnOK);

    }
}
```

Section 6 - Item Listeners

10C.6.1 Being a Good Listener

You've already seen an example of event handlers methods defined in listener classes. In our transporter, when the user either pressed the "See Friend" button or hit the enter key while typing in the friends name, this caused an **ActionEvent** to be triggered, which created an implicit call to the controls' event handler. If you recall from our first lecture on events, this was a three step process:

1. **Place the control** somewhere in our container frame
2. **Define the listener** class that will handle the event for the control.
3. **Register our listener** with the ***control*** that it will listen to.

Here you see these three steps as we have taken them in our previous program (although step 3 appears before step 2 because of the way the classes must be created):

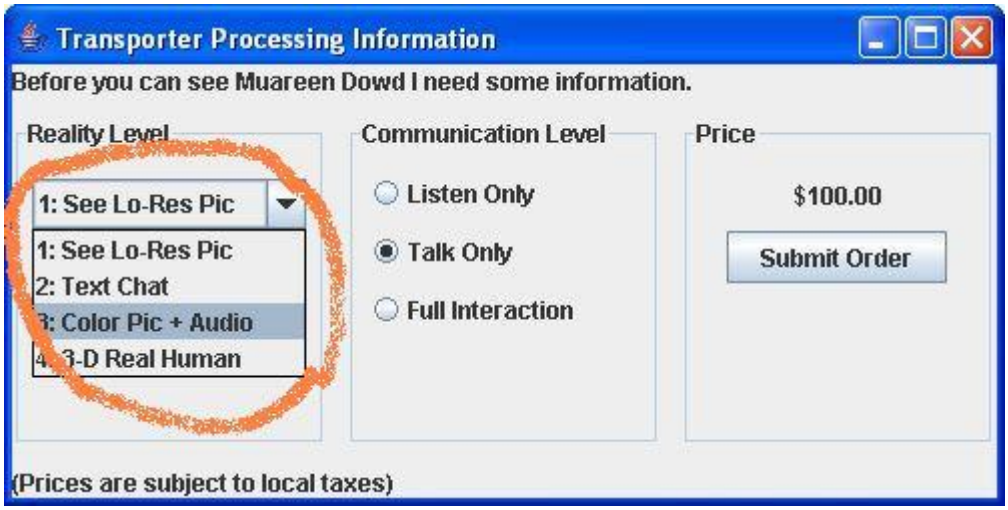
```
// TranspoFrame class is derived from JFrame class
class TranspoFrame  extends JFrame
{
   private JButton btnMyButton;

   // TranspoFrame constructor
   public TranspoFrame(String title)
   {
      // 1. place the button
      btnMyButton = new JButton("Press Here to See Friend");
      add(btnMyButton);

      // 3. register an action listener
      SeeFriendListener myListener = new SeeFriendListener();
      btnMyButton.addActionListener( myListener );
   }

   // 2. define the listener
   class SeeFriendListener implements ActionListener
   {

      // event handler for JButton
      public void actionPerformed(ActionEvent e)
      {
         // take action based on the event
      }
   } // end inner class SeeFriendListener

} // end class TranspoFrame
```

As it happened, both the event of pressing the button and the event of hitting a return in the text field both generated the same kind of event, an **ActionEvent**. Not only that, but we actually took the exact same action in either case. This is what allowed us to have only one listener class and to allow one object of that class handle either event.

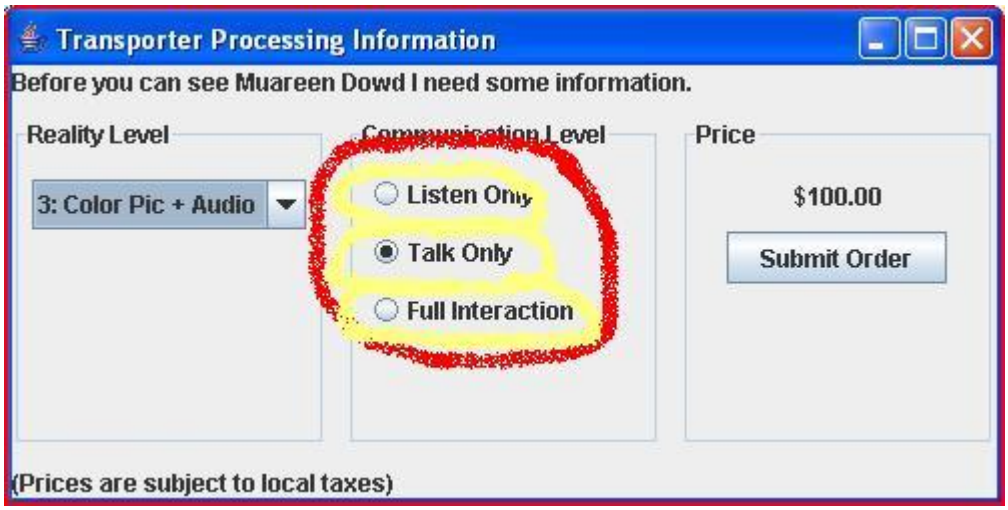
10C.6.2 ItemEvent Listeners

In most programs we have more than one type of event, and even those controls that generate the same kinds of events may require different instances of the listener class to handle them, because the actions are disparate.

Our **JComboBox** and **JRadioButtons** are an example of a ***controls*** that will require a new kind of event for us, the **ItemEvent**. As you can imagine, this event is triggered if the user selects an item from the combo box:



or presses one of the Communication Level buttons:



The difference between this new **ItemEvent** and the **ActionEvent** that we have already used, is

- The inner listener class that we define needs to implement an **ItemListener** interface.
- The event handler method will be called **itemStateChanged().**
- Inside the **itemStateChanged()** method, we will normally read the combo box state using the **getSelectedIndex()** method, which returns 0 through n-1, where 0 is the first list item and n-1 is the last.

Here is the code that creates and registers the **ItemListeners** (placed at the end of the **SecondFrame** constructor, where these controls are all established):

```
// register the listeners
cmboRealityLevels.addItemListener(new RealityLevelListener());
btnLstn.addItemListener(new CommLevelListener());
btnTalk.addItemListener(new CommLevelListener());
btnFull.addItemListener(new CommLevelListener());
```

And here are the two **ItemListener** classes defined (inner classes to **SecondFrame** class):

```
// inner class for JButton Listener
class RealityLevelListener implements ItemListener
{
   public void itemStateChanged(ItemEvent e)
   {
      realityState = cmboRealityLevels.getSelectedIndex();
      updatePrice();
   }
} // end RealityLevelListener

// inner class for JButton Listener
class CommLevelListener implements ItemListener
{
   public void itemStateChanged(ItemEvent e)
   {
      if (btnLstn.isSelected())
            commState = 0;
      else if (btnTalk.isSelected())
            commState = 1;
      if (btnFull.isSelected())
            commState = 2;
      updatePrice();
   }
} // end CommLevelListener
```

As you see, with the combo box, we can get the selected index in a single statement using **getSelectedIndex()**. With radio buttons, we have to test each radio button until we find one that is checked using **isSelected().**

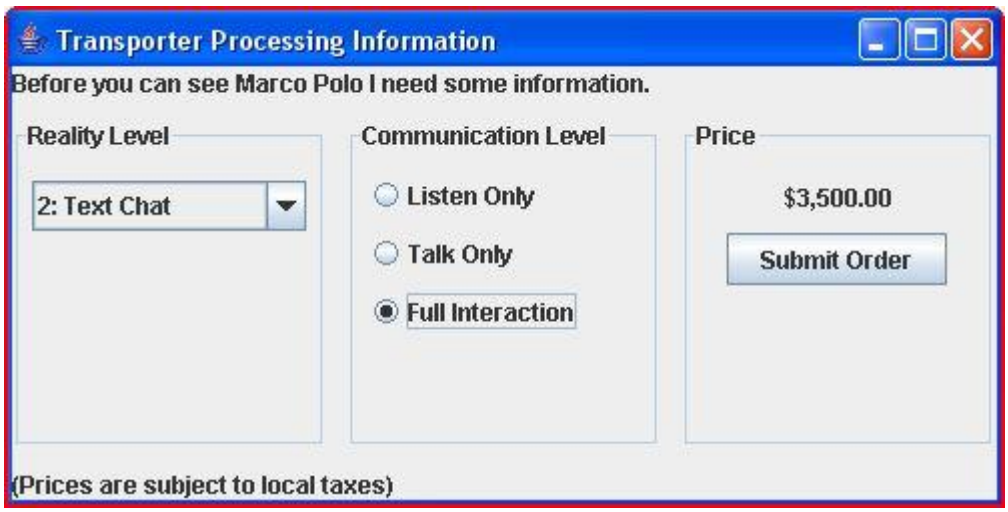
We have left, unexplained, the meaning of the new instance variables **commState** and **realityState** as well as the meaning of the **updatePrice()** method. These will be discussed in the next section.

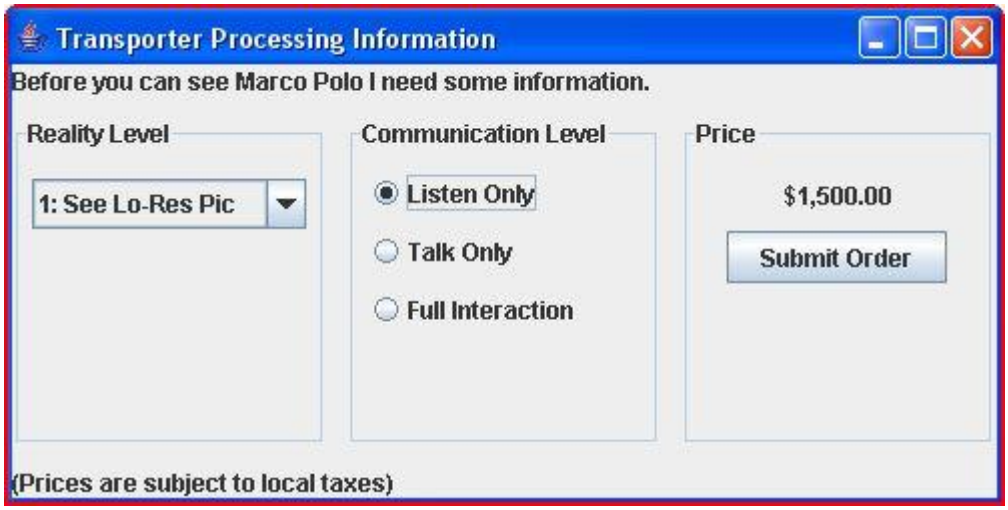
Section 7 - State Variables

10C.7.1 Keeping Track of Several User Selections

In our transporter project, we want the price of the service to be a function of the Reality Level that the user selects. Seeing a low resolution still-frame picture of your friend with no interaction is not as exciting -- or costly -- as seeing a color picture and hearing his or her voice. And if you want your friend beamed-in to your room, in full three dimensional form, that is going to cost you big time.

Here are the screens shots showing the effect of different user choices in the left and center panels:





Our strategy will be to respond to the combo box change by reading the new Reality Level and, based on what it was, modify the price label in the right panel. Since this price is going to not only reflect the Reality Level, but also the Communication Level (center panel) we will have to read both of these panels' values. The smartest way to do this is to create two new ***instance members,*** **realityState** (0-3) and **commState**, (0-2) that will store the most recent selections from the user. We then add a private method called **updatePrice()** that will first, calculate the new price based on the latest **realityState** and **commState** values, and second, update **lblPrice** with the new amount (by calling its instance method **lblPrice.setText()**).

Variables that remember the state of a user interface for us are, reasonably, called ***state variables***. Since we are designing the **SecondFrame** ***class***, these ***state variables*** naturally fall inside that class as instance variables. Similarly, the **updatePrice()** method will be an instance method for this class, since it must access the instance state variables.

```
//SecondFrame class is derived from JFrame class
class SecondFrame  extends JFrame
{
   // other variables omitted

   // state variables
   private int realityState;  // 0-3
   private int commState;     // 0-2

   // constructor
   public SecondFrame(String strFriend)
   {
      // body omitted except for last statement:

      updatePrice();
   }

   private void updatePrice()
   {
      int intPrice;
      NumberFormat bucks = NumberFormat.getCurrencyInstance(Locale.US);

      intPrice = 1000*(realityState+1) + 500*(commState+1);

      // beaming people across spacetime is expensive
      if (realityState == cmboRealityLevels.getItemCount() - 1)
         intPrice = 500000;

      // update the pricetag
      lblPrice.setText(bucks.format(intPrice));
   }

   // inner class for JButton Listener
   class RealityLevelListener implements ItemListener
   {
   } // end RealityLevelListener

   // inner class for JButton Listener
   class CommLevelListener implements ItemListener
   {
   } // end CommLevelListener
}
```

A nice improvement on this version would be to split the **updatePrice()** into two methods, **recalculatePrice()** and **updatePriceLabel()**. Based on their names, what do you think these two methods would do? Why is this a good idea?

We have reached another plateau in our transporter application. It is getting long, but that's unavoidable. The concepts, however, are simple and compact. The length comes from the fact that there are several controls and panels in the GUI. Many of the steps that we take in this program, are repeated two or more times, creating the larger size of the program, without really adding any complexity.

The entire listing appears in the next section with no further commentary. It is complete and ready for copying and pasting into your Java IDE.

Section 8 - Transporter Price Version

10C.8.1 Transporter Listing Only

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.text.*;
import java.util.*;

public class Foothill
{
   public static void main(String[] args)
   {
      // establish main frame in which program will run
      TranspoFrame myTranspoFrame = new TranspoFrame("Transporter Room");
      myTranspoFrame.setSize(300, 200);
      myTranspoFrame.setLocationRelativeTo(null);
      myTranspoFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      // show everything to the user
      myTranspoFrame.setVisible(true);
   }
}

// TranspoFrame class is derived from JFrame class
class TranspoFrame  extends JFrame
{
   private JButton btnMyButton;
   private JTextField txtMyTextArea;
   private JLabel lblMyLabel;

   // TranspoFrame constructor
   public TranspoFrame(String title)
   {
      // pass the title up to the JFrame constructor
      super(title);

      // set up layout which will control placement of buttons, etc.
      FlowLayout layout = new FlowLayout(FlowLayout.CENTER, 5, 20);
      setLayout(layout);
```

```
   // Frame will have 3 controls: a label, a text field and a button
   lblMyLabel = new JLabel("Friend's Name: ");
   txtMyTextArea = new JTextField(10);
   btnMyButton = new JButton("Press Here to See Friend");
   add(lblMyLabel);
   add(txtMyTextArea);
   add(btnMyButton);

   SeeFriendListener myListener = new SeeFriendListener();
   btnMyButton.addActionListener( myListener );
   txtMyTextArea.addActionListener( myListener );
}

// inner class for JButton Listener
class SeeFriendListener implements ActionListener
{
   SecondFrame frmSecond;

   // event handler for JButton
   public void actionPerformed(ActionEvent e)
   {
      String strFriendName;
      String[] subNames;
      boolean errorFlag;

      strFriendName = txtMyTextArea.getText();
      if (strFriendName == null || strFriendName.length() < 3)
         errorFlag = true;
      else if (strFriendName.indexOf(",") >= 0)
         errorFlag = true;
      else if (strFriendName.equalsIgnoreCase("Michael Loceff"))
      {
         JOptionPane.showMessageDialog(null,
               "I'm sorry, but no one can "
               + "see Michael Loceff, please choose someone else.");
         return;
      }
      else
      {
         // break into first and last using space
         subNames = strFriendName.split(" ", 0);
         if (subNames.length != 2)
            errorFlag = true;
         else if (subNames[0].length() < 2)
            errorFlag = true;
         else if (subNames[1].length() < 2)
            errorFlag = true;
         else
            errorFlag = false;
      }
```

```
         if (errorFlag)
         {
            JOptionPane.showMessageDialog(null,
                  "Name must be '[first][last]', and  [first]\n"
                  + "and [last} must each contain at least 2 letters.");
            return;
         }

         frmSecond = new SecondFrame(strFriendName);
         frmSecond.setSize(500, 250);
         frmSecond.setLocationRelativeTo(null);

         // show second frame to the user
         frmSecond.setVisible(true);

      }
   } // end inner class SeeFriendListener
} // end class TranspoFrame

//SecondFrame class is derived from JFrame class
class SecondFrame  extends JFrame
{
   JLabel lblName, lblTaxes, lblPrice;
   JPanel panelMain, panelLeft, panelCenter, panelRight;
   JComboBox cmboRealityLevels;
   JRadioButton btnLstn, btnTalk, btnFull;
   ButtonGroup btngp;
   JButton btnOK;

   // state variables
   private int realityState;  // 1-4
   private int commState;     // 1-3

   // constructor
   public SecondFrame(String strFriend)
   {
      // chain to base class constructor
      super("Transporter Processing Information");
      realityState = commState = 0;

      // set up three main components
      lblName = new JLabel("Before you can see " + strFriend
            + " I need some information.");
      lblTaxes = new JLabel("(Prices are subject to local taxes)");
      panelMain = new JPanel( new GridLayout(1, 3, 10, 10));

      // use border layout and put three above components in frame
      setLayout (new BorderLayout(20, 10));
      add(lblName, BorderLayout.NORTH );
      add(panelMain, BorderLayout.CENTER);
      add(lblTaxes, BorderLayout.SOUTH);
```

```
    // create three sub-panels to to into panelMain
    panelLeft = new JPanel( new FlowLayout(FlowLayout.CENTER, 5, 10));
    panelCenter = new JPanel( new FlowLayout(FlowLayout.LEFT, 5, 5));
    panelRight = new JPanel( new FlowLayout(FlowLayout.CENTER, 5, 10));
    panelMain.add(panelLeft);
    panelMain.add(panelCenter);
    panelMain.add(panelRight);

    // place borders around three sub-panels
    panelLeft.setBorder(new TitledBorder("Reality Level"));
    panelCenter.setBorder(new TitledBorder("Communication Level"));
    panelRight.setBorder(new TitledBorder("Price"));

    // put a combo box in panelLeft
    String[] strRealityLevels = {"1: See Lo-Res Pic", "2: Text Chat",
          "3: Color Pic + Audio", "4: 3-D Real Human"};
    cmboRealityLevels = new JComboBox(strRealityLevels);
    panelLeft.add(cmboRealityLevels);

    // put a button group in panelCenter
    btnLstn = new JRadioButton("Listen Only");
    btnLstn.setSelected(true);    // default
    btnTalk = new JRadioButton("Talk Only");
    btnFull = new JRadioButton("Full Interaction");
    btngp = new ButtonGroup();
    btngp.add(btnLstn);
    btngp.add(btnTalk);
    btngp.add(btnFull);

    // place buttons in panel, top to bottom
    panelCenter.add(btnLstn);
    panelCenter.add(btnTalk);
    panelCenter.add(btnFull);

    // put the price and ok button in panelRight
    lblPrice = new JLabel("-- temporary --");
    btnOK = new JButton("Submit Order");
    panelRight.add(lblPrice);
    panelRight.add(btnOK);

    // register the listeners
    cmboRealityLevels.addItemListener(new RealityLevelListener());
    btnLstn.addItemListener(new CommLevelListener());
    btnTalk.addItemListener(new CommLevelListener());
    btnFull.addItemListener(new CommLevelListener());

    // adjust price to default state
    updatePrice();
}
```

```
   private void updatePrice()
   {
      int intPrice;
      NumberFormat bucks = NumberFormat.getCurrencyInstance(Locale.US);

      intPrice = 1000*(realityState+1) + 500*(commState+1);

      // beaming people across spacetime is expensive
      if (realityState == cmboRealityLevels.getItemCount() - 1)
         intPrice = 500000;

      // update the pricetag
      lblPrice.setText(bucks.format(intPrice));
   }

   // inner class for JButton Listener
   class RealityLevelListener implements ItemListener
   {
      public void itemStateChanged(ItemEvent e)
      {
         realityState = cmboRealityLevels.getSelectedIndex();
         updatePrice();
      }
   } // end RealityLevelListener

   // inner class for JButton Listener
   class CommLevelListener implements ItemListener
   {
      public void itemStateChanged(ItemEvent e)
      {
         if (btnLstn.isSelected())
               commState = 0;
         else if (btnTalk.isSelected())
               commState = 1;
         if (btnFull.isSelected())
               commState = 2;
         updatePrice();
      }
   } // end CommLevelListener
}
```