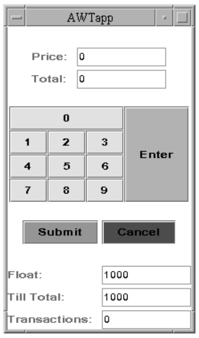
Lecture 1

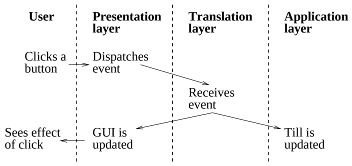
We write a shop till program. The architecture we use keeps different concerns quite separate and is essential for complex programs with GUIs.



A good architecture for programs with GUIs

- *Application layer* or *Model*
  (e.g. `Till` object)
  These are object(s) created from classes that define & manage application data; they have nothing to do with GUIs
- *Presentation layer* or *Views*
  (e.g. `TillGUI` object)
  These are object(s) that create & arrange visual components; they must accurately reflect the model visually to users
- *Translation layer* or *Controllers*
  (listeners, e.g. `TillController`)
  These are objects whose handler methods are invoked in response to events; they send messages to the model and the GUI so that the model and GUI update themselves in response to events

The architecture, cont'd



Clean separation of concerns:

- High cohesion: each object's methods/variables are directed at a single purpose
- Low coupling: minimal dependencies between objects
- Therefore, more readable, re-usable, maintainable. E.g. makes it easier to replace the 'view' (GUI) with a different view

The `Till` class definition: guess the rest!

This is a 'normal' class definition (constructors, getters, setters, etc.). It has no mention of GUI components or events.

```
public Till(int theTillFloat)
public int getTillFloat()
public int getTakings()
public int getTillTotal() // float+takings
public int getTransactionCount()
public int getTotalOwed()

public void ringUpItem(int theItemPrice)
{  amountOwedByThisCust += theItemPrice;
}
```



```
public void submitTransaction()
{  transactionCount++;
   takings += amountOwedByThisCust;
   amountOwedByThisCust = 0;
}

public void cancelTransaction()
{  amountOwedByThisCust = 0;
}

private int tillFloat;
private int takings;
private int transactionCount;
private int amountOwedByThisCust;
```


Highlights of the `TillGUI` class definition

This class definition creates & lays out visual components, and registers listeners. It doesn't send any messages to other parts of the program.

```
public class TillGUI extends JPanel
{  public TillGUI(
         TillController theController)
   { super();

     controller = theController;

     // Create the four main GUI panels.
     JPanel custP = createCustPanel();
     JPanel numPadP = createNumPadPanel();
     JPanel actionP = createActionPanel();
     JPanel tillTotP = createTillTotPanel();

     // Layout the 4 panels one beneath the
     // other using GridBagLayout
   }
```

```
public int getPriceEntered()
{  return
      Integer.parseInt(priceTF.getText());
}

public void setPriceEntered(
   int thePriceEntered)
{  priceTF.setText("" + thePriceEntered);
}

// Similar definitions for setTotalOwed,
// setTillFloat, setTillTotal &
// setTransactionCount

// The 4 helper methods, create & layout
// the labels, textfields & buttons.
// They also register listeners...
```



```
// Registering numeric buttons' listener
// Uses an anonymous class

ActionListener lstnr = new ActionListener()
{ public void actionPerformed(ActionEvent ae
  { String btnText = ae.getActionCommand();
    int digit = Integer.parseInt(btnText);
    int priceSoFar = getPriceEntered();
    int newPrice = priceSoFar * 10 + digit;
    setPriceEntered(newPrice);
  }
};
for (int i = 0; i < 10; i++)
{ digitB[i].addActionListener(lstnr);
}

// Registering Enter/Submit/Cancel listener
// Uses the 'controller' created earlier

enterB.addActionListener(controller);
submitB.addActionListener(controller);
cancelB.addActionListener(controller);
```


The `TillController` class: listener for 3 buttons

This class definition 'mediates' between the GUI class and the `Till` class. It can send messages to both. In response to events, it therefore updates the `Till` and updates the GUI.

```
public class TillController
   implements ActionListener
{
   public TillController()
   {
   }

   public void setTill(Till theTill)
   {  till = theTill;
   }

   public void setGUI(TillGUI theGUI)
   {  gui = theGUI;
      updateGUI();
   }
```



```
private void updateGUI()
{  gui.setPriceEntered(0);
   gui.setTotalOwed(till.getTotalOwed());
   gui.setTillFloat(till.getTillFloat());
   gui.setTillTotal(till.getTillTotal());
   gui.setTransactionCount(
            till.getTransactionCount());
}
```



```
public void actionPerformed(ActionEvent ae)
{  String str = ae.getActionCommand();
   if (str.equals(TillGUI.ENTER))
   {  till.ringUpItem(gui.getPriceEntered())
      updateGUI();
   }
   else if (str.equals(TillGUI.SUBMIT))
   {  till.ringUpItem(gui.getPriceEntered())
      till.submitTransaction();
      updateGUI();
   }
   else // i.e. cancel
   {  till.cancelTransaction();
      updateGUI();
   }
}

private Till till;
private TillGUI gui;
```


The `main` method

```
public static void main(String[] args)
{  Till till = new Till(1000);
   TillController controller =
            new TillController();
   TillGUI gui =
           new TillGUI(controller);
   controller.setTill(till);
   controller.setGUI(gui);

   JFrame frm = new JFrame();
   Container contentPane =
            frm.getContentPane();
   contentPane.add(gui);
   frm.pack();
   frm.setLocation(100, 100);
   frm.setVisible(true);
}
```