Lecture 1

We look at variations of our temperature conversion GUI. In these variations, the listeners are done in different ways.

More specifically, we make successive revisions to the temperature conversion GUI. The effect is that a program that previously resided in four files (four class definitions) ends up in a single file (one class definition).

These variations represent different design decisions that you would have to choose between.

I: Convenient listener class definitions

The Java library contains abstract class `WindowAdapter`. It looks something like:

```
import java.awt.event.*;

public abstract class WindowAdapter
   implements WindowListener
{
  public void windowOpened(WindowEvent we)
  {
  }

  // Likewise windowActivated, windowClosed,
  // windowDeactivated, windowDeiconified,
  // windowIconified & windowClosing
}
```


I: Convenient listener class definitions, cont'd

Our definition of `GenericWindowLstnr` is now more concise:

```
import java.awt.event.*;

public class GenericWindowLstnr
   extends WindowAdapter
{
  public void windowClosing(WindowEvent we)
  {  System.exit(0);
  }
}
```


II: Anonymous classes

- Can avoid separately defining very simple class definitions by using *anonymous classes*
- Use for 'one-off' classes
  - the class is defined at the same point in the code where an instance is being created
  - the class has no name (an anonymous subclass of some superclass or an anonymous implementer of some interface) & no real constructor of its own
  - much used for listeners

II: Anonymous classes, cont'd

Delete `GenericWindowLstnr`.

In TempConversionGUI, change registration statements from

```
frm.addWindowListener(
      new GenericWindowLstnr());
```

to

```
frm.addWindowListener(new WindowAdapter()
{ public void windowClosing(WindowEvent we)
  {  System.exit(0);
  }
});
```


III: Combining listeners

- One listener object could respond to events from more than one component
- E.g. replace `CompFahrBtnLstnr` and `CompCentBtnLstnr` by `CompTempBtnLstnr` (next slide)
  - Registration changes:

    ```
    compFahrBtn.addActionListener(
          new CompFahrBtnLstnr(this));
    compCentBtn.addActionListener(
          new CompCentBtnLstnr(this));

    CompTempBtnLstnr lstnr =
       new CompTempBtnLstnr(this);
    compFahrBtn.addActionListener(lstnr);
    compCentBtn.addActionListener(lstnr);
    ```

  - Fewer classes, but `actionPerformed` is more complex, less efficient & less cohesive

```
import java.awt.event.*;
public class CompTempBtnLstnr
   implements ActionListener
{
  public CompTempBtnLstnr(
     TempConversionGUI theGUI)
  {  gui = theGUI;
  }

  public void actionPerformed(ActionEvent ae
  {
     if
     {  int cent = gui.getCentField();
        int fahr = (int) Math.round( etc.
        gui.setFahrField(fahr);
     }
     else
     {  int fahr = gui.getFahrField();
        int cent = (int) Math.round( etc.
        gui.setCentField(cent);
     }
  }
  private TempConversionGUI gui;
}
```


IV: The GUI class as a listener

- Could do away with separate listener classes
- The GUI class could act as the listener for the components
- i.e. when an event occurs, the object that responds is the GUI itself
- Quite common but can be a poor design (high coupling)
- Suppose `TempConversionGUI` will act as a listener for its own buttons
  - `TempConversionGUI` will declare `implements ActionListener`
  - therefore it would have to include `actionPerformed` method

IV: The GUI as a listener, cont'd

- The registration statements change from

  ```
  CompTempBtnLstnr lstnr =
     new CompTempBtnLstnr(this);
  compFahrBtn.addActionListener(lstnr);
  compCentBtn.addActionListener(lstnr);
  ```

  to

  ```
  compFahrBtn.addActionListener(this);
  compCentBtn.addActionListener(this);
  ```

- So when the buttons are pressed, it's the `TempConversionGUI` object (i.e. `this`) whose `actionPerformed` method is executed
- A small advantage is that the `actionPerformed` method can be simpler: it's in the same class so methods/variables are now visible


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

public class TempConversionGUI
   extends JPanel
   implements ActionListener
{
   public TempConversionGUI()
   {  // create labels, textfields & buttons
      // and add to window, as before

      // The revised listener registration
      compFahrBtn.addActionListener(this);
      compCentBtn.addActionListener(this);
   }

   // four methods, get/setFahr/CentField,
   // as before
```



```
 public void actionPerformed(ActionEvent ae)
 {  if
    {  int cent =
       int fahr = (int) Math.round( etc.


    }
    else
    {  int fahr =
       int cent = (int) Math.round( etc.


    }
 }

 // variables as before

 // main method as before
}
```