CMSC 341

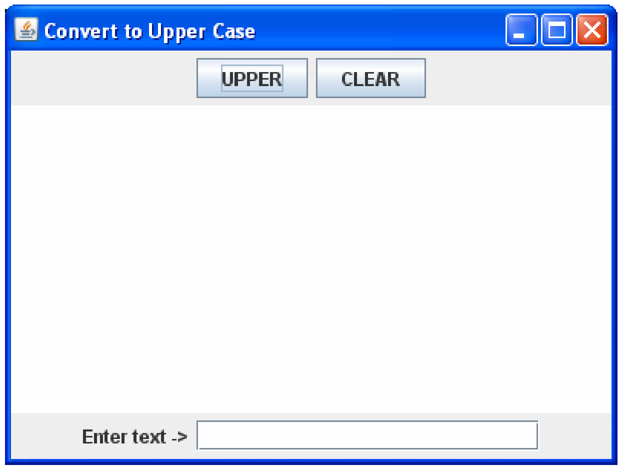
Making Java GUIs Functional

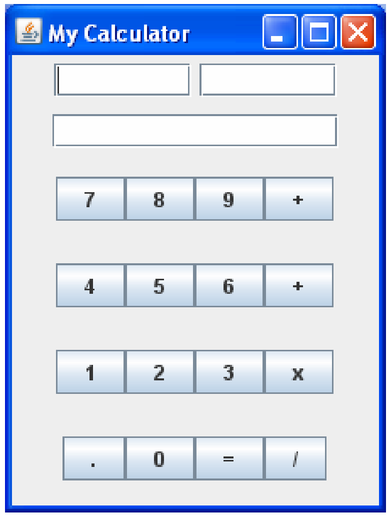
More on Swing

- Great Swing demo at http://java.sun.com/products/plugin/1.3.1_01a/demos/jfc/SwingSet2/SwingSet2Plugin.html
- Just google for “SwingSet Demo Java”
- Now let’s learn how to make GUIs functional

Last Class

- Learned about GUI Programming.
- Created two GUIs
  - UppercaseConverter
  - Calculator
- Now we will make them work.





Events

- Java uses an Event Delegation Model.
- Every time a user interacts with a component on the GUI, events are generated.
- Events are component-specific.
- Events are objects that store information like
  - the type of event that occurred,
  - the source of the event,
  - the time of an event to name a few.

Event Delegation Model

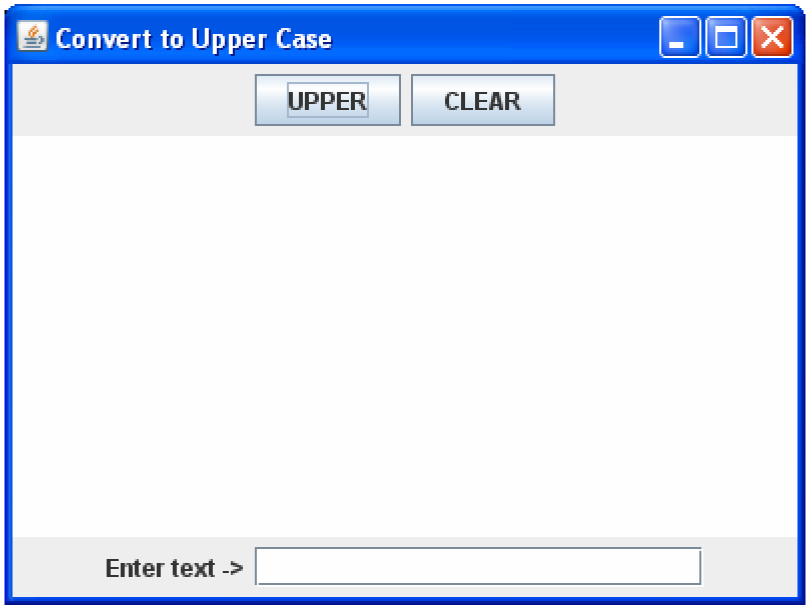
- Once the event is generated, then the event is passed to other objects which handle or react to the event, thus the term event delegation.
- The objects which react to or handle the events are called event listeners.

Three Players

- Event source which generates the event object
- Event listener which receives the event object and handles it
- Event object that describes the event

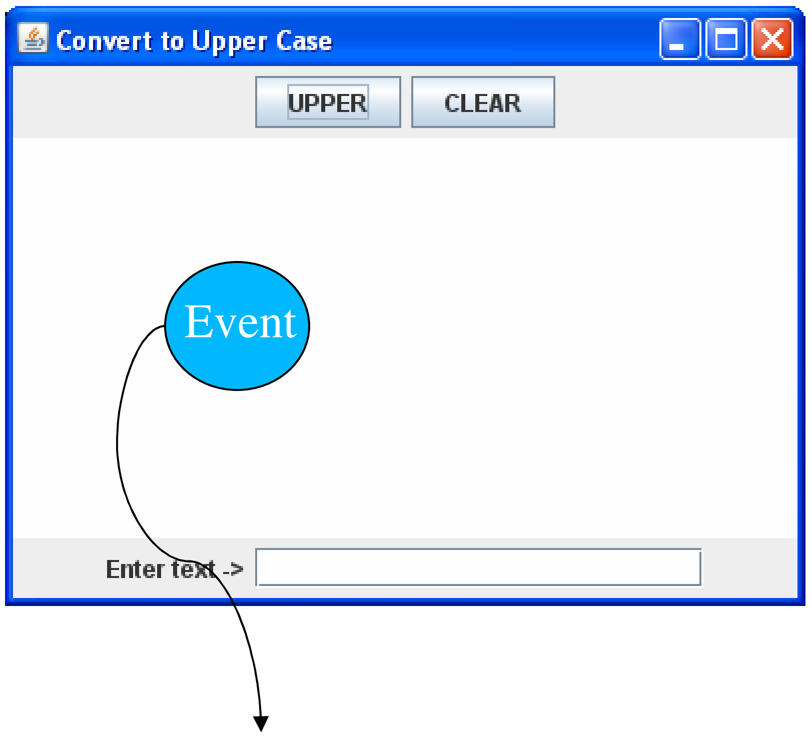
Revisiting our GUI

- We have already created a GUI.
- How many components?
- What are some possible events?



Example

- Click on UPPER *JButton*
- Generates an *ActionEvent*
- Event object is sent to an *ActionListener* that is registered with the UPPER *JButton*
- *ActionListener* handles in *actionPerformed* method.



```
public class Handler implements ActionListener
{
   public void actionPerformed(ActionEvent e){
      System.out.println("Handling " + e);
   }
}
```

Registering Listeners

- By having a class implement a listener interface, it can contain code to handle an event.
- However, unless an instance of the class is registered with the component , the code will never be executed. (Common novice error.)

A Few More Java Events

- FocusEvent – component gains or loses focus
- MouseEvent – mouse is moved, dragged, pressed, released or clicked
- WindowEvent – window is iconified, deiconified, opened or closed
- TextEvent – text is modified
- KeyEvent – key is pressed, depressed or both
- ContainerEvent – components are added or removed from Container

Corresponding Listeners

- FocusEvent – FocusListener
- MouseEvent – MouseListener, MouseMotionListener
- WindowEvent – WindowStateListener, WindowListener, WindowFocusListener
- TextEvent – TextListener
- KeyEvent – KeyListener
- ItemEvent- ItemListener
- ContainerEvent – ContainerListener

Methods for Registering Listeners

- JButton
  - addActionListener(ActionListener a)
  - addChangeListener(ChangeListener c)
  - addItemListener(ItemListener i)
- JList
  - addListSelectionListener(ListSelectionListener l)

UpperCaseConverter Example

- Goal
  - When UPPER button is pressed, the text in the textfield will be converted to upper case and appended into the text area.
  - When CLEAR button is pressed, both the text field and the text area will be cleared.
- Things to consider to accomplish goal
  - What type of events do we need to respond to?
  - What listener interfaces do we need to implement?

Implementing an ActionListener

- Create as a separate class
  - No access to data in *JFrame*
- Create as an inner class
  - Access to *JFrame* data
  - Must instantiate an object of this class to pass to *addActionListener* method
- Make the *JFrame* implement the interface
  - Access to *JFrame* data
  - No need to instanciate an object of this class – have the *this* reference

Implementing ActionListener

```
import java.awt.event.*;
public class UpperCaseConverter extends JFrame implements
   ActionListener
{  //omitted code
    upper = new JButton("UPPER");
    clear = new JButton("CLEAR");
    upper.addActionListener(this);
     clear.addActionListener(this);
   //omitted code
   public void actionPerformed(ActionEvent e){
    Object obj = e.getSource();
    if(obj == clear) System.out.println("Clear");
    else if(obj == upper) System.out.println("Upper");
   }
}
```

Good to test for expected interaction as you go

Implement Desired Behavior

```
public void actionPerformed(ActionEvent e)
{
    Object obj = e.getSource();
    if(obj == clear){
        input.setText("");
        output.setText("");
    }
    else if(obj == upper){
        String result = input.getText();
        StringBuffer buffer = new
            StringBuffer(output.getText());
        buffer.append(result.toUpperCase()+ "\n");
        output.setText(buffer.toString());
    }
}
```

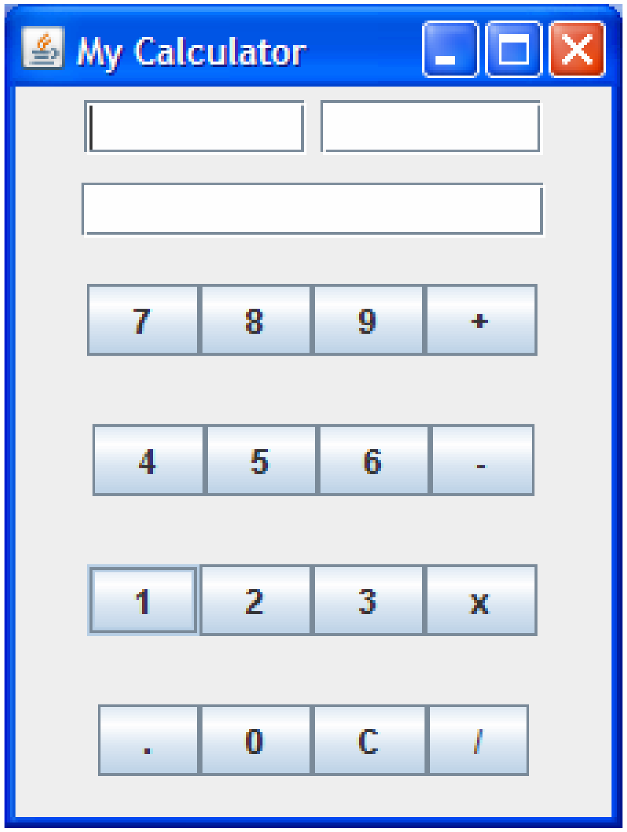
*JButton*s, *JLabel*s, *JTextField*s and *JTextArea*s all have *setText* method to change their content

Adding Functionality to the Calculator

- Need capability for telling the number to go to the left or right TextField.
  - If click and holding the ctrl button then number goes to the left, else the right.
- Need to be able to perform operations.
  - Use the operators themselves for the operations.
- Need to be able to clear fields.
  - Convert the equal sign to a C for clear.

Slightly Modified GUI

- Notice the change
  - Changed '=' to 'C'
  - Changed all references from "equals" to "clears" in the code



Add Listeners

```
plus.addActionListener(this);
minus.addActionListener(this);
mult.addActionListener(this);
div.addActionListener(this);
clears.addActionListener(this);
dot.addActionListener(this);
for(int i = 0; i < 10 ; i++)
  numbers[i].addActionListener(this);
```

Implementing the *actionPerformed* Method

- First step is to implement the skeleton code that will recognize the different locations that are clicked.
- Second step is to code for clicks with ctrl key pressed and not pressed.
- Third step is to add desired behavior.
  - Helper methods would be helpful for the converting of text to floats and for the various arithmetic operations.

More ActionEvent Methods

```
public void
actionPerformed(ActionEvent e)
{
 String command = e.getActionCommand();
 System.out.println(command);
 int modifiers = e.getModifiers();
 if(modifiers == ActionEvent.CTRL_MASK)
    System.out.println("CTRL PRESSED");
}
```

Problem

- Unfortunately, the code on the previous code can not differentiate between a button click with the control key down and a button click alone.
- Next… try *MouseListener* interface.
  - *mousePressed*
  - *mouseReleased*
  - *mouseExited*
  - *mouseClicked*
  - *mouseEntered*

Changing to a MouseListener

- Change all ***ActionListener*** references to ***MouseListener*** references
- Remove ***actionPerformed*** method and add:

```
public void mouseClicked(MouseEvent e){
 int button = e.getButton();
 System.out.println(button);
 String modifiers =
    e.getMouseModifiersText(e.getModifiers());
 System.out.println(modifiers);
}
public void mouseReleased(MouseEvent e){}
public void mousePressed(MouseEvent e){}
public void mouseEntered(MouseEvent e){}
public void mouseExited(MouseEvent e){}
```

Determines which button was pressed, right or left

States whether the Ctrl, Alt or Shift buttons were pressed

Output

- After a left click then right click on a number output is:

```
1
Button1
3
Meta+Button3
```

- After left click then right click on a number with ctrl down output is:

```
1
Ctrl+Button1
3
Meta+Ctrl+Button3
```

mouseClicked Method

- Need to use *getSource* method to determine which button was pressed.
- Easiest way to differentiate is left click and right click
- Left click ->left operand
- Right click -> right operand
- For operators doesn’t matter

Functional *mouseClicked* Method

```
public void mouseClicked(MouseEvent e){
   int button = e.getButton();  JTextField dest = null;
   if(button == 1) dest = operand1;  //left click == left operand
   if(button == 3) dest = operand2;  //right click == right operand
   Object src = e.getSource();
   if(src == clears) clear();  //helper method
   else if(src == mult||src == div||src == plus||src == minus)
     performOperation(src); //helper method
   else{
     int i = 0;
     for(; i < numbers.length; i++)
         if(src == numbers[i]) break;
     StringBuffer text = new StringBuffer(dest.getText());
     if (src == dot) text.append(dot.getText());
     else text.append(numbers[i].getText());
     dest.setText(text.toString());
   }
}
```

Helper Method

```
private void performOperation(Object src){
 float f1 = 0;float f2 = 0;
 try {
     f1 = Float.parseFloat(operand1.getText());
     f2 = Float.parseFloat(operand2.getText());
 }catch (NumberFormatException e){
     output.setText("Invalid Number Format");
 }
 try{
     float ans = 0;
     if(src == mult) ans = f1 * f2;
     else if(src == plus) ans = f1 + f2;
     else if(src == minus) ans = f1 - f2;
     else if(src == div) ans = f1 / f2;
     output.setText(Float.toString(ans));
 } catch (Exception e) {
     output.setText("Invalid Operation");
 }
}
```

Adapter Classes

- In the previous implementation, we implemented four empty methods.
- We can create a listener class that extends its corresponding adapter class.
- Adapter classes provide the empty implementation of all the methods in a listener interface
- We only need to override the method(s) whose behavior we want to influence.

Anonymous Inner Classes

- Adapter classes are often implemented as anonymous inner classes.

```
mult.addListener(new MouseAdapter(){
    public void mouseReleased(){
    // specialized code just for mult
    // that will only be executed when mouse is
    // released on the ‘x’ JButton
    }
});
```