Toward a GUI-programming model

- We want to adhere to language-independent ideals
  - Concepts move from GUIs in Java to …
  - javax.swing and java.awt offer thousands of choices
    - Too many to have to understand/find comfort in, but …
- We want to write reasonable, robust , GUI applications
  - Actually write code, not simply adhere to lofty ideals
  - Show me the code!
- Simple, extensible, re-usable conceptual framework
  - How to develop GUIs, how to extend
  - Ask Questions

One GUI Conceptual Framework

- Create a JPanel for the GUI contentPane
  - Provide a BorderLayout, organize hierarchically
  - Ok to use GridLayout, FlowLayout, … nested
- Create Buttons, Menu-items, and other widgets
  - Bind each event-generator to a listener
  - Do not dispatch within a listener on event source
    - No "if event-generator is button A do this"
- Use anonymous inner classes, or named inner classes
  - Process events, created and attached close-to-source
  - Make a button, make a button-listener

Click on a button, display the click

```
ActionListener textDisplayer = new ActionListener(){
    public void actionPerformed(ActionEvent e)
    {
        showText(e.getActionCommand());
    }
};
```

- What does an ActionListener do?
  - Listens for an event, e.g., from Button, Menu, …
  - Processes the command/event
- How do anonymous classes work?
  - Note: ActionListener is an *interface*, but object created!
  - See what Eclipse refactoring will do with this

Making a Move: View and Controller

```
ActionListener moveMaker = new ActionListener(){
  public void actionPerformed(ActionEvent e)
  {
    int val = Integer.parseInt(e.getActionCommand());
    myControl.makeMove(new PuzzleMove(val));
  }
};
```

- We know this will be bound to a specific type of button
  - Not generic, completely application specific
  - Turns swing/GUI event into application event: Move
- Controllers should be programmed abstractly
  - Don't base code on a GUI toolkit, separate concerns