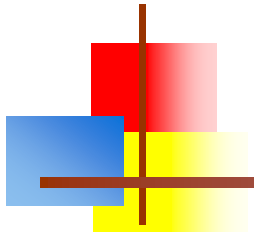
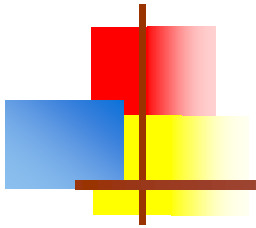
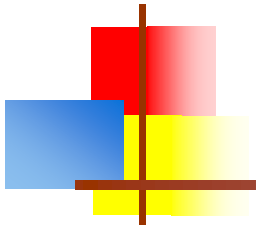


GUI building with Swing



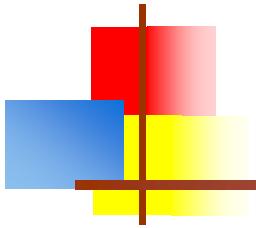
How to build a GUI

- Create a window in which to display things—usually a JFrame (for an application), or a JApplet
- Use the setLayout(LayoutManager ***manager***) method to specify a layout manager
- Create some Components, such as buttons, panels, etc.
- Add your components to your display area, according to your chosen layout manager
- Write some Listeners and attach them to your Components
  - Interacting with a Component causes an Event to occur
  - A Listener gets a message when an interesting event occurs, and executes some code to deal with it
- Display your window



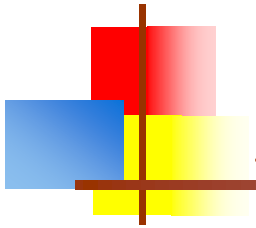
Import the necessary packages

- The Swing components are in javax.swing.*, so you always need to import that for a Swing application
- Swing is built on top of AWT and uses a number of AWT packages, including most of the layout managers, so you need to import java.awt.*
- Most listeners also come from the AWT, so you also need to import java.awt.event.*
- A few listeners, such as DocumentListener and ListSelectionListener, are specific to Swing, so you may need to import javax.swing event.*
- For more complex GUIs, there are additional java.awt.*something* and javax.swing.*something* packages that you may need to import



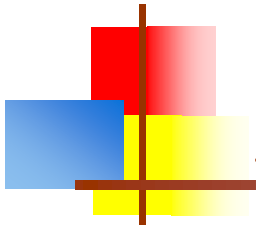
Make a Container

- For an application, your container is typically a JFrame
  - JFrame frame = new JFrame();
  - JFrame frame = new JFrame("Text to put in title bar");
- You can create a JFrame in your "main class"
- It's often more convenient to have your "main class" *extend* JFrame
- For an applet, your "main class" must extend JApplet
- Once your application or applet is up and running, it can create and display various dialogs



Add a layout manager

- The most important layout managers are:
  - BorderLayout
    - Provides five areas into which you can put components
    - This is the default layout manager for both JFrame and JApplet
  - FlowLayout
    - Components are added left to right, top to bottom
  - GridLayout
    - Components are put in a rectangular grid
    - All areas are the same size and shape
  - BoxLayout
    - Creates a horizontal row or a vertical stack
    - This can be a little weird to use
  - GridBagLayout
    - Too complex and a danger to your sanity—***avoid***
    - See http://www.youtube.com/watch?v=UuLaxbFKAcc (Flash, with audio)



Add components to containers

- The usual command is

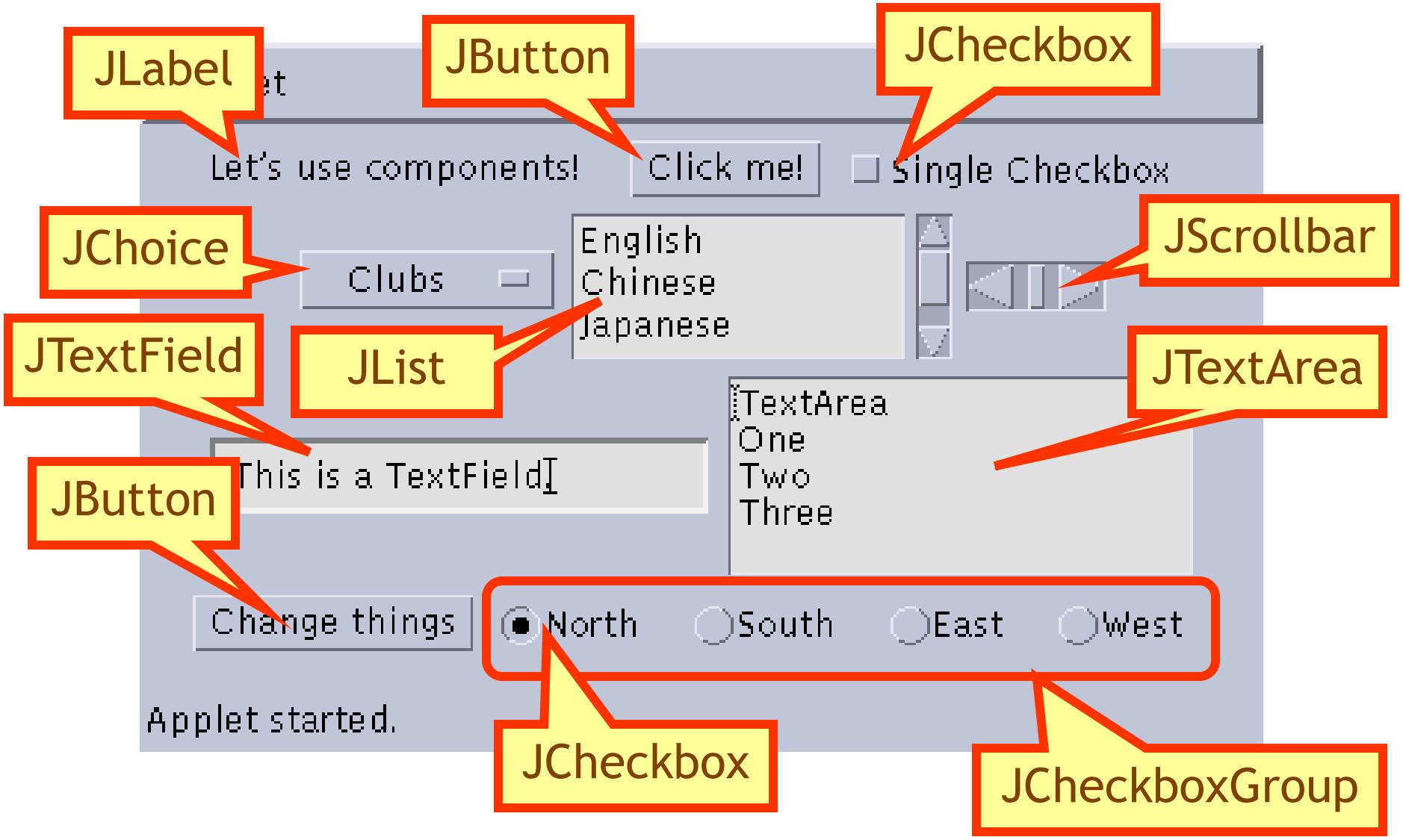
  container.add(*component*);

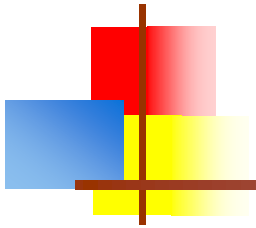
  - For FlowLayout, GridLayout, and BoxLayout, this adds the component to the next available location
  - For BorderLayout, this puts the component in the CENTER by default
- For BorderLayout, it's usually better to use

  container.add(*component*, BorderLayout.*position*);

  - *position* is one of NORTH, SOUTH, EAST, WEST, or CENTER

Some types of components





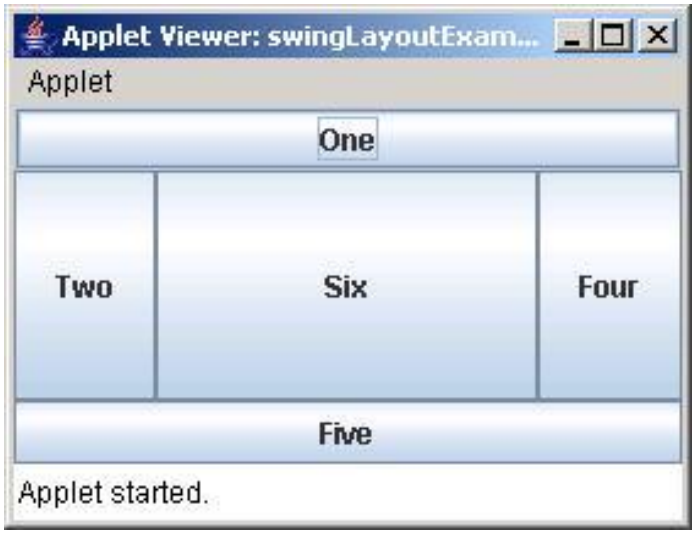
Create components

- JButton button = new JButton("Click me!");
- JLabel label = new JLabel("This is a JLabel");
- JTextField textField1 = new JTextField("This is the initial text");
- JTextField textField2 = new JTextField("Initial text", columns);
- JTextArea textArea1 = new JTextArea("Initial text");
- JTextArea textArea2 = new JTextArea(rows, columns);
- JTextArea textArea3 = new JTextArea("Initial text", rows, columns);
- JCheckBox checkbox = new JCheckBox("Label for checkbox");
- JRadioButton radioButton1 = new JRadioButton("Label for button");
- ButtonGroup group = new ButtonGroup();
  group.add(radioButton1); group.add(radioButton2); *etc.*

- This is just a sampling of the available constructors; see the javax.swing API for all the rest

BorderLayout

```
public class BorderLayoutExample extends JApplet {
    public void init () {
        setLayout(new BorderLayout ());
        add(new JButton("One"),   BorderLayout.NORTH);
        add(new JButton("Two"),   BorderLayout.WEST);
        add(new JButton("Three"), BorderLayout.CENTER);
        add(new JButton("Four"),  BorderLayout.EAST);
        add(new JButton("Five"),  BorderLayout.SOUTH);
        add(new JButton("Six"));
    }
}
```

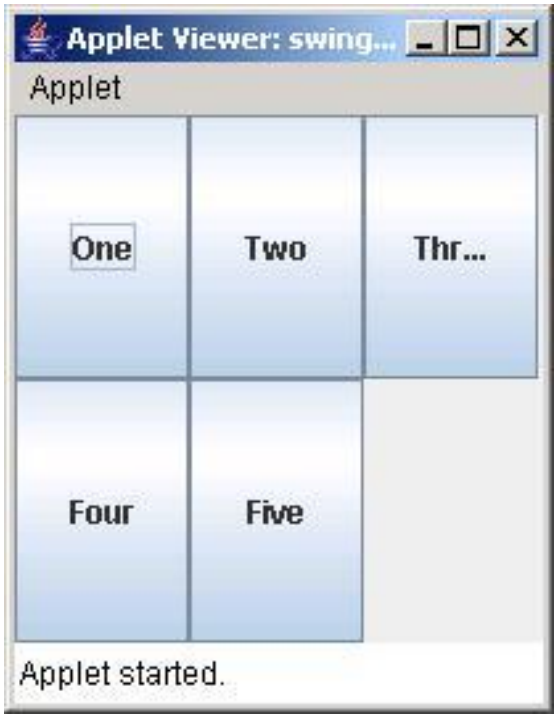


FlowLayout

```
public class FlowLayoutExample extends JApplet {
  public void init () {
    setLayout(new FlowLayout ());
    add(new JButton("One"));
    add(new JButton("Two"));
    add(new JButton("Three"));
    add(new JButton("Four"));
    add(new JButton("Five"));
    add(new JButton("Six"));
  }
}
```

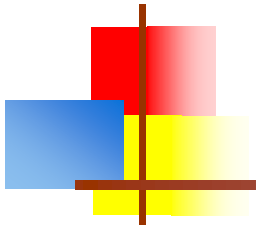
GridLayout

```
public class GridLayoutExample extends JApplet {
    public void init() {
        setLayout(new GridLayout(2, 4));
        add(new JButton("One"));
        add(new JButton("Two"));
        add(new JButton("Three"));
        add(new JButton("Four"));
        add(new JButton("Five"));
    }
}
```



BoxLayout

```
public class BoxLayoutExample extends JApplet {
    public void init () {
        Box box = new Box(BoxLayout.Y_AXIS);
        add(box);
        box.add(new JButton("One"));
        box.add(new JButton("Two"));
        box.add(new JButton("Three"));
        box.add(new JButton("Four"));
        box.add(new JButton("Five"));
        box.add(new JButton("Six"));
    }
}
```



Nested layouts

- A JPanel is both a JContainer and a Component
  - Because it's a container, you can put other components into it
  - Because it's a component, you can put it into other containers
- All but the very simplest GUIs are built by creating several JPanels, arranging them, and putting components (possibly other JPanels) into them
- A good approach is to draw (on paper) the arrangement you want, then finding an arrangement of JPanels and their layout managers that accomplishes this

An example nested layout

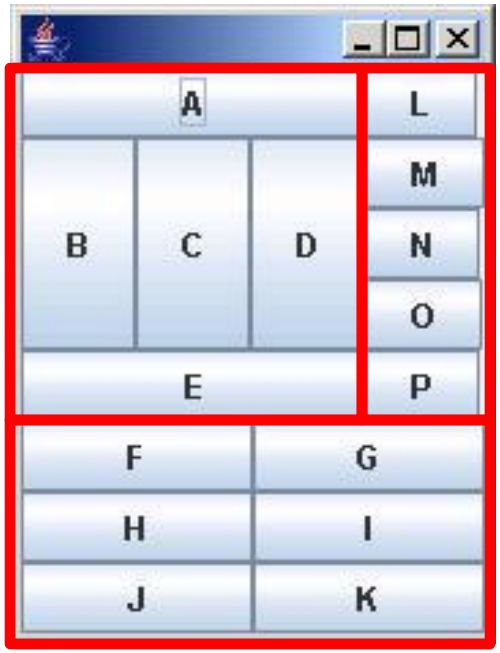
- Container container = new *JFrame() or JApplet()*;

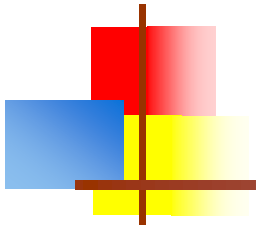
```
JPanel p1 = new JPanel();
p1.setLayout(new BorderLayout());
p1.add(new JButton("A"), BorderLayout.NORTH);
      // also add buttons B, C, D, E

JPanel p2 = new JPanel();
p2.setLayout(new GridLayout(3, 2));
p2.add(new JButton("F"));
     // also add buttons G, H, I, J, K

JPanel p3 = new JPanel();
p3.setLayout(new BoxLayout(p3, BoxLayout.Y_AXIS));
p3.add(new JButton("L"));
     // also add buttons M, N, O, P

 container.setLayout(new BorderLayout());
 container.add(p1, BorderLayout.CENTER);
 container.add(p2, BorderLayout.SOUTH);
 container.add(p3, BorderLayout.EAST);
```





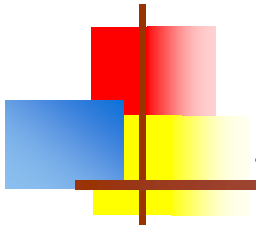
Create and attach listeners

- JButton okButton = new JButton("OK");

- okButton.addActionListener(new MyOkListener());

- ```
  class MyOkListener implements ActionListener {
      public void actionPerformed(ActionEvent event) {
          // code to handle okButton click
      }
  }
  ```

- A small class like this is often best implemented as an anonymous inner class
  ```



Anonymous inner classes

- Anonymous inner classes are convenient for short code (typically a single method)

  `b.addActionListener(`***anonymous inner class***`);`

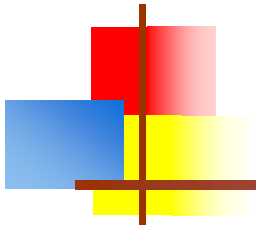
- The *anonymous inner class* can be either:

  `new` ***Superclass*****`(`*****args*****`) { `*****body*****` }`**

  or

  `new` ***Interface*****`() { `*****body*****` }`**

- Notice that no class name is given--only the name of the superclass or interface
  - If it had a name, it wouldn't be anonymous, now would it?
- The *args* are arguments to the superclass's constructor (interfaces don't have constructors)



Using an anonymous inner class

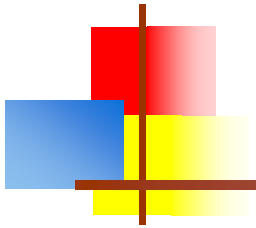
- Instead of:
  - okButton.addActionListener(**new** MyOkListener());

```
class MyOkListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        // code to handle OK button click
    }
}
```

- You can do this:
  - okButton.addActionListener(**new ActionListener**() {

```
    public void actionPerformed(ActionEvent event) {
        // code to handle OK button click
    }
);
```

- Keep anonymous inner classes very short (typically just a call to one of your methods), as they can really clutter up the code

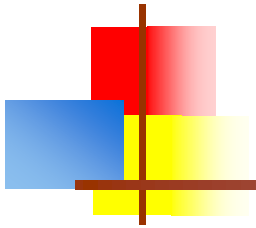


Suggested program arrangement 1

```
class SomeClass {
    // Declare components as instance variables
    JFrame frame; // Can also define them here if you prefer
    JButton button;
    public static void main(String[] args) {
        new SomeClass().createGui();
    }
    // Define components and attach listeners in a method
    void createGui() {
        frame = new JFrame();
        button = new JButton("OK");
        frame.add(button); // (uses default BorderLayout)
        button.addActionListener(new MyOkListener());
    }
    // Use an inner class as your listener
    class MyOkButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            // Code to handle button click goes here
        }
    }
}
```

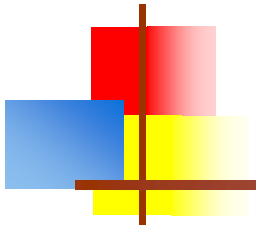
Suggested program arrangement 2

```
class SomeClass extends JFrame {
    // Declare components as instance variables
    // JFrame frame; // Don't need this
    JButton button;
    public static void main(String[] args) {
        new SomeClass().createGui();
    }
    // Define components and attach listeners in a method
    void createGui() {
        // frame = new JFrame();  // Don't need this
        button = new JButton("OK");
        add(button); // Was: frame.add(button);
        button.addActionListener(new MyOkListener());
    }
    // Use an inner class as your listener
    class MyOkButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            // Code to handle button click goes here
        }
    }
}
```



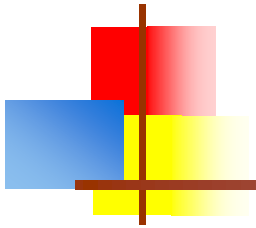
Components use various listeners

- JButton, JMenuItem, JComboBox, JTextField:
  - addActionListener(ActionListener)
    - public void actionPerformed(ActionEvent event)
- JCheckBox, JRadioButton:
  - addItemListener(ItemListener)
    - public void itemStateChanged(ItemEvent event)
- JSlider
  - addChangeListener(ChangeListener)
    - public void stateChanged(ChangeEvent event)
- JTextArea
  - **getDocument().**addDocumentListener(DocumentListener)
    - public void insertUpdate(DocumentEvent event)
    - public void removeUpdate(DocumentEvent event)
    - public void changedUpdate(DocumentEvent event)



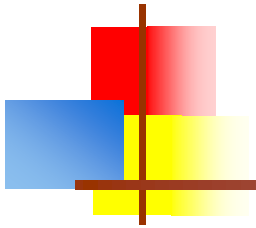
Getting values

- Some user actions normally cause the program to *do* something: clicking a button, or selecting from a menu
- Some user actions set values to be used *later:* entering text, setting a checkbox or a radio button
  - You *can* listen for events from these, but it's not usually a good idea
  - Instead, *read* their values when you need them
    - String myText = myJTextField.getText();
    - String myText = myJTextArea.getText();
    - boolean checked = myJCheckBox.isSelected();
    - boolean selected1 = myJRadioButton1.isSelected();



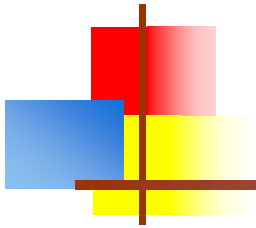
Enabling and disabling components

- It is poor style to remove components you don't want the user to be able to use
  - "Where did it go? It was here a minute ago!"
- It's better to *enable* and *disable* controls
  - Disabled controls appear "grayed out"
  - The user may wonder *why?*, but it's still less confusing
- *anyComponent*.setEnabled(*enabled*);
  - Parameter should be true to enable, false to disable



Dialogs

- A dialog (small accessory window) can be modal or nonmodal
  - When your code opens a modal dialog, it waits for a result from the dialog before continuing
  - When your code opens a nonmodal dialog, it does so in a separate thread, and your code just keeps going
- Sun supplies a few simple (but useful) *modal* dialogs for your use
- You can create your own dialogs (with JDialog), but they are *nonmodal* by default



Message dialogs

- JOptionPane.showMessageDialog(parentJFrame,
  "This is a JOptionPane \"message\" dialog.");

- Notice that showMessageDialog is a static method of JOptionPane
- The "parentJFrame" is typically your main GUI window (but it's OK to use null if you don't have a main GUI window)

Confirm dialogs

- int yesNo =
  JOptionPane.showConfirmDialog(parentJFrame,
  "Is this what you wanted to see?");
- if (yesNo == JOptionPane.YES_OPTION) { ... }

Input dialogs

- String userName =
  JOptionPane.showInputDialog(parentJFrame,
  "What is your name?")

Option dialogs

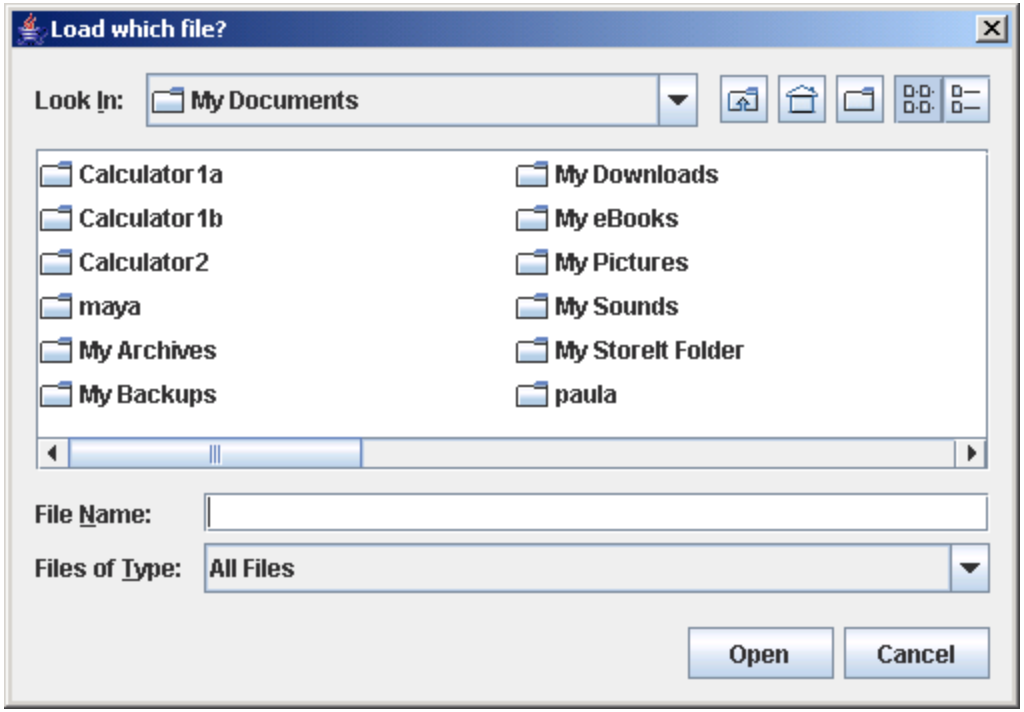
```
Object[] options =
    new String[] {"English", "Chinese", "French", "German" };
int option =
    JOptionPane.showOptionDialog(parentJFrame,
                                 "Choose an option:",
                                 "Option Dialog",
                                 JOptionPane.YES_NO_OPTION,
                                 JOptionPane.QUESTION_MESSAGE,
                                 null,
                                 options,
                                 options[0]); // use as default
```

- Fourth argument could be JOptionPane.YES_NO_CANCEL_OPTION
- Fifth argument specifies which icon to use in the dialog; it could be one of ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, or PLAIN_MESSAGE
- Sixth argument (null above) can specify a custom icon

Load file dialogs

```
JFileChooser chooser = new JFileChooser();
chooser.setDialogTitle("Load which file?");
int result = chooser.showOpenDialog(enclosingJFrame);
if (result == JFileChooser.APPROVE_OPTION) {
    File file = chooser.getSelectedFile();
    // use file
}
```

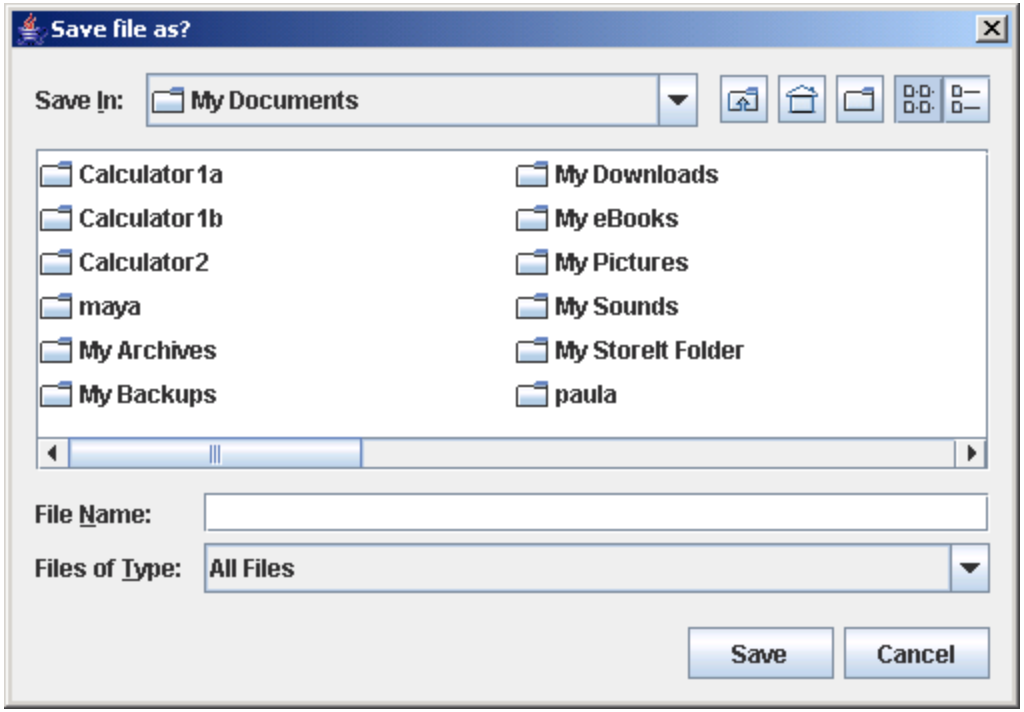
- You could also test for CANCEL_OPTION or ERROR_OPTION
- You will get back a File object; to use it, you must know how to do file I/O

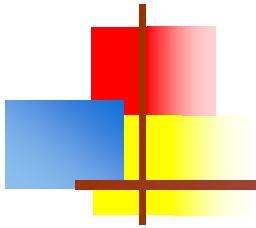


Save file dialogs

```
JFileChooser chooser = new JFileChooser();
chooser.setDialogTitle("Save file as?");
int result = chooser.showSaveDialog(enclosingJFrame);
if (result == JFileChooser.APPROVE_OPTION) {
    File file = chooser.getSelectedFile();
    // use file
}
```

- You could also test for CANCEL_OPTION or ERROR_OPTION
- You will get back a File object; to use it, you must know how to do file I/O

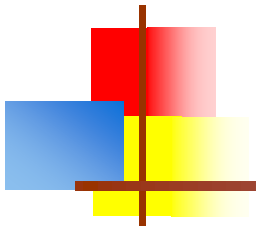




Quitting the program

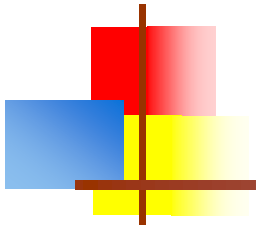
- *gui*.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

- Other options are DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE, and DISPOSE_ON_CLOSE



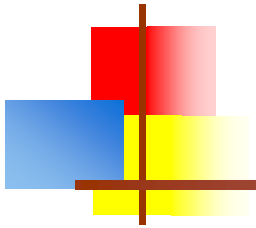
Summary I: Building a GUI

- Create a container, such as JFrame or JApplet
- Choose a layout manager
- Create more complex layouts by adding JPanels; each JPanel can have its own layout manager
- Create other components and add them to whichever JPanels you like



Summary II: Building a GUI

- For each active component, look up what kind of Listeners it can have
- Create (implement) the Listeners
  - often there is one Listener for each active component
  - Active components can share the same Listener
- For each Listener you implement, supply the methods that it requires
- For Applets, write the necessary HTML



The End

It should be noted that no ethically-trained software engineer would ever consent to write a DestroyBaghdad procedure. Basic professional ethics would instead require him to write a DestroyCity procedure, to which Baghdad could be given as a parameter.

--Nathaniel S Borenstein