Almost anyone can write a small GUI application.

We will look at aspects of writing large GUI applications with elegance and finesse for maximum user acceptance.

Java

- Large GUI based applications for 12 years
- Java AWT/Swing for 4 years
- OO Development for 10 years

Some Swing
- Properties
- Action
- javax.swing.Action

1. Developing a GUI Infrastructure
2. Building applications around Actions
3. Polishing the application
4. Installing on multiple platforms

Like a towering sky-scraper, large applications have a deep foundation...
Homogenous
Extensible
Complete

Consistent behavior
Consistent layout

Easy to add more features
Re-usable core components
New components do not have to be written from scratch

Application appears finished
All the implied features are there
- Drag & drop
- Tool tips
- Toolbars
- Help
- Installs easily

Swing alone is too low level
Hard for many programmers to use Swing in a consistent way
Applications can look very different
Classes that complement Swing objects
Classes that combine Swing objects
Classes that build Swing objects in a prescribed way
  - Adjustments to Look & Feel are made in one place

A replacement for Swing
A layer on top of Swing

Over-used ActionListeners
You lay out every individual swing object
Frequent GridBagLayout code
70% - 80% of your dialogs code interacts with Swing

Have a set of predefined, re-usable components
Use very few ActionListeners
Never create an OK button for a dialog
Never write code to do simple number validation

Rarely write code to create basic Swing components
  - JButton
  - JMenu
  - JCheckbox
  - JTextField
  - JRadioButton
Don't use default renderers for JTables
Defines `actionPerformed()`.

Adds property handling:
- `addPropertyChangeListener()`
- `removePropertyChangeListener()`
- `putValue()`
- `getValue()`

Implements everything but `actionPerformed()`.
One Action could have many Buttons
Action is an ActionListener on the Button
Buttons adds a PropertyChangeListener to the Action
The button will reflect Action property changes
- Text
- Tooltip

PropertyChangeListener notifies on when something has changed

Defines actionPerformed()
Adds 6 more methods
Implements everything but actionPerfomed()
Implements actionPerfomed()
To raise a dialog

- Raise a Dialog
- Undo/Redo
- Cut, Copy, Paste
- Save
- Show parts of the User Interface
- Etc.
public class MyDialogAction extends AbstractAction
{
    private MyDialog _dialog;

    public MyDialogAction(JFrame f) {
        putValue(NAME, "Show My Dialog");
        putValue(MENUTEXT_DESCRIPTION, "Show my special dialog");
        putValue(ACCELERATOR_KEY,
                new KeyStroke("0", Event.ALT_MASK));
        _dialog = new MyDialog(f);
    }

    public void actionPerformed(ActionEvent ev) {
        _dialog.setVisible(true);
    }
}
public void actionPerformed(ActionEvent e) {
    _dialog.setVisible(true);
}
1. Load Properties
2. Extract Actions
3. Convert any necessary properties
   - 2. look for cases on property types
   - 1. Add properties to loading
      • We need a success to
      • Abstraction - no way to load properties

- 9. Actions prop
- These lines in a property file

- Every Action has the same type of code
- Nothing primitive is included
- Actions do manage properties, but...
Subclass of AbstractAction
Add Property Loading

public abstract class GeneralAction extends AbstractAction
{
    public GeneralAction(String command) {
        PropDB prop = PropDB.getInstance();
        String s = prop.getProperty(command + "." +
                Action.SHORT_DESCRIPTION);
        if (s == null) putValue(Action.SHORT_DESCRIPTION, s);
        s = prop.getProperty(command + "." + Action.NAME);
        if (s == null) putValue(Action.NAME, s);
        // -- now do mnemonic, accelerator, etc
    }
}

public GeneralAction(String command) {
    PropDB prop = PropDB.getInstance();
    String s;
    s = prop.getProperty(command + "." + SHORT_DESCRIPTION);
    Get the SHORT_DESCRIPTION property
}
If the property exists, then add it to the Action's properties.

Do the same thing with LONG_DESCRIPTION, SMALL_ICON, ACCELERATOR_KEY, MNEMONIC_KEY.
public class MyDialogAction extends GeneralAction {
    // code...
}

public MyDialogAction(JFrame f) {
    super("myDialog");
    _dialog= new MyDialog(f);
    // code...
}

public void actionPerformed(ActionEvent ev) {
    _dialog.setVisible(true);
}

myDialog.Name= "Show My Dialog"
myDialog.ShortDescription= "Show my \ special dialog"
myDialog.Accelerator= "ctrl-d"

This is the command name

Lots of Actions!!!
Application Code
GUI Infrastructure
Swing
Actions that display dialogs take a long time to create
Slow actions affect start-up time
We need lazy initialization

New abstract methods
- build()
- activate()

public void actionPerformed(ActionEvent ev) {
    if (!_built) {
        build();
        _built = true;
    }
    activate();
}

These guys are fine, they don't create a dialog
Toolbars
Splash screens with progress bars
Tool Tips
Help
Effective use of color
Complete & informative error messages
Drag & Drop

Small features greatly affect user acceptance
Simply meeting requirements does not make a great program
Build time in the schedule for adding nice touches
You will not be able to justify them
- Add them anyway
- Everyone will love you, trust me

Will the user be impressed in the first five minutes?
Does it feel easy to use
- Only a novice cares if it is easy to use
- The first few tasks must be easy to do

Always use a splash screen
Generally use a progress bar
Do everything possible to speed it up
- Be meticulous
- Constant vigilance
Complete Part 2
Installing on Multi-platforms

- Easy
- Normal
- JRE

- Very first impression of your application
- Almost Un-noticeable
- Quick
- Network installations are rarely easy!

- Just like any other application
- Feels normal for that platform
- No special request from the installer
- Nothing else to install

- You must guarantee it is there
- You must have control over the version
- Solution – Install your own

- Homogeneous – Build a solid GUI infrastructure
- Extensible – Build your application around Actions
- Complete – Include the small details to make it exceptional
A good program is like a sky-scraper...

If you want to go high, first you better go deep.