JavaOne

Writing a Large Application in the Real World Based on the Java Foundation Class/Swing (JFC/Swing) API

Troy Henry
Senior Computer Scientist
California Institute of Technology

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.

JavaOne

Qualifications

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

JavaOne

Assumed to Know

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

JavaOne

---

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

JavaOne

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

JavaOne

[illegible] Deep

- Homogenous
- Extensible
- Complete

Homogenous

- Consistent behavior
- Consistent layout

Extensible

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

Complete

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

[illegible]ogeneous [illegible]ping a [illegible] infrastructure

[illegible]

- Swing alone is too low level
- Hard for many programmers to use Swing in a consistent way
- Applications can look very different

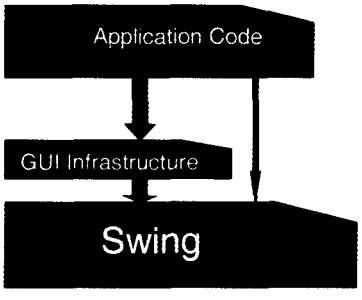
[illegible] Infrastructure Package

- Classes that *complement* Swing objects
- Classes that *combine* Swing objects
- Classes that *build* Swing object in a prescribed way
  - Adjustments to Look & Feel are made in one place

[illegible] Infrastructure Package

- A replacement for Swing
- A layer on top of Swing

[illegible] Infrastructure [illegible] Picture



[illegible] Infrastructure

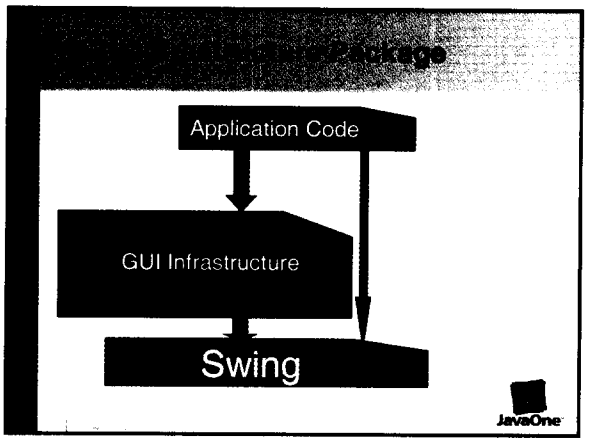
- Over-used ActionListeners
- You lay out every individual swing object
- Frequent GridBagLayout code
- 70% - 80% of your dialogs code interacts with Swing

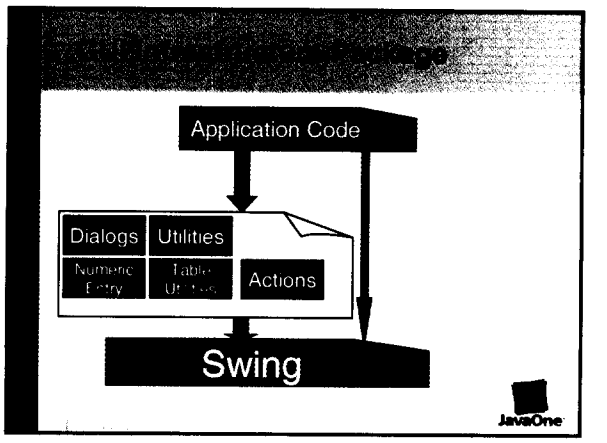
[illegible] You...

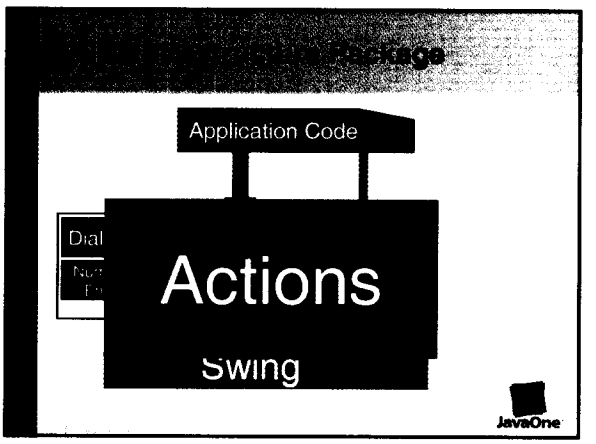
- Have a set of predefined, re-usable components
- Use very few ActionListener
- Never create an OK button for a dialog
- Never write code to do simple number validation

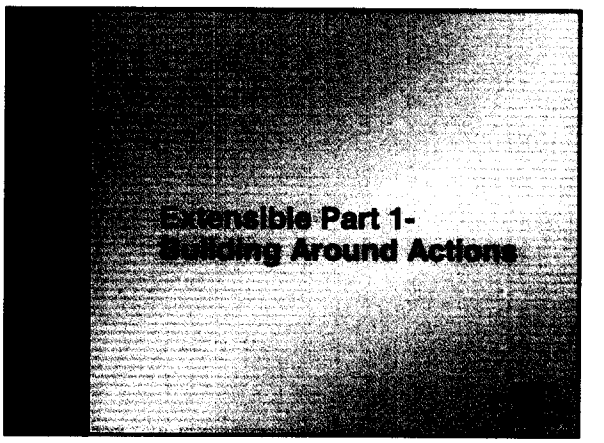
[illegible] You...

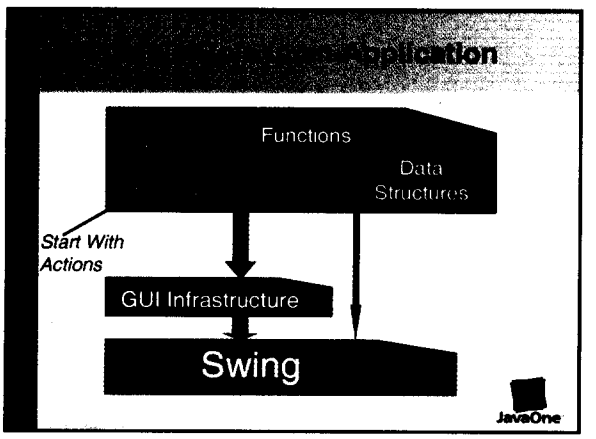
- Rarely write code to create basic Swing components
  - JButton
  - JMenuItem
  - JCheckBox
  - JTextField
  - JRadioButton
- Don't use default renderers for JTables

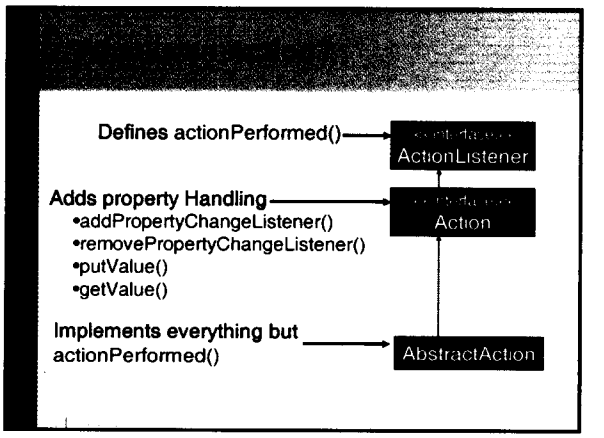

Application Code
GUI Infrastructure
Swing
JavaOne


Application Code
Dialogs
Utilities
Actions
Swing
JavaOne

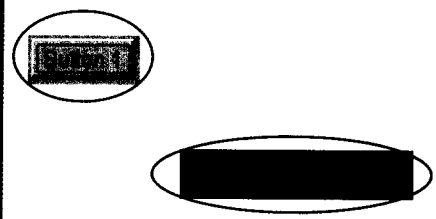

Application Code
Actions
Swing
JavaOne

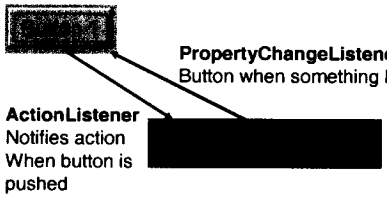

Extensible Part 1-
Building Around Actions


Application
Functions
Data
Structures
Start With
Actions
GUI Infrastructure
Swing
JavaOne

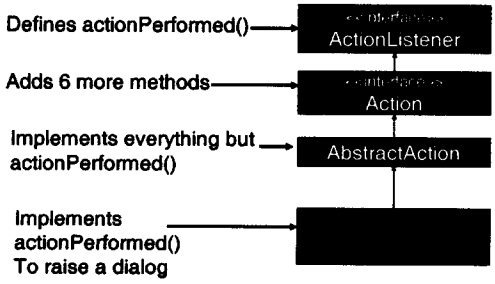

Defines actionPerformed()
ActionListener
Adds property Handling
•addPropertyChangeListener()
•removePropertyChangeListener()
•putValue()
•getValue()
Action
Implements everything but
actionPerformed()
AbstractAction

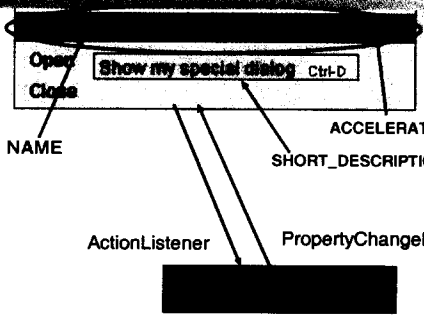
- One Action could have many Buttons
- Action *is an* ActionListener on the Button
- Buttons *adds a* PropertyChangeListener to the Action
- The button will reflect Action property changes
  - Text
  - Tooltip





- Raise a Dialog
- Undo/Redo
- Cut, Copy, Paste
- Save
- Show parts of the User Interface
- Etc.





Action to Raise a Dialog (1)

```
public class MyDialogAction extends AbstractAction {

 private MyDialog _dialog;

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

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

JavaOne

Action to Raise a Dialog (2)

```
public class MyDialogAction extends AbstractAction {

 private MyDialog _dialog;

 public MyDialogAction(JFrame f) {



 }

 public void actionPerformed(ActionEvent ev) {

 }
}
```

JavaOne

Action to Raise a Dialog (3)

```
public class MyDialogAction extends AbstractAction {

 private MyDialog _dialog;

 public MyDialogAction(JFrame f) {



 }

 public void actionPerformed(ActionEvent ev) {

 }
}
```

JavaOne

Action to Raise a Dialog (4)

```
public class MyDialogAction extends AbstractAction {

 private MyDialog _dialog;

 public MyDialogAction(JFrame f) {



 }

 public void actionPerformed(ActionEvent ev) {

 }
}
```

JavaOne

Action to Raise a Dialog (5)

```
public class MyDialogAction extends AbstractAction {

 private MyDialog _dialog;

 public MyDialogAction(JFrame f) {



 }

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

JavaOne

Action to Raise a Dialog (6)

```
public class MyDialogAction extends AbstractAction {

 private MyDialog _dialog;

 public MyDialogAction(JFrame f) {

 }

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

Adding Properties

JavaOne

MyDialogAction Dialog (7)

```
public class MyDialogAction extends AbstractAction {

  private MyDialog _dialog;

  public MyDialogAction(JFrame f) {
    putValue(NAME,                "Show My Dialog");

  }

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

Adding Properties

JavaOne

MyDialogAction Dialog (8)

```
public class MyDialogAction extends AbstractAction {

  private MyDialog _dialog;

  public MyDialogAction(JFrame f) {
    putValue(NAME,                "Show My Dialog");
    putValue(SHORT_DESCRIPTION, "Show my special dialog");

  }

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

Adding Properties

JavaOne

MyDialogAction Dialog (9)

```
public class MyDialogAction extends AbstractAction {

  private MyDialog _dialog;

  public MyDialogAction(JFrame f) {
    putValue(NAME,                "Show My Dialog");
    putValue(SHORT_DESCRIPTION, "Show my special dialog");
    putValue(ACCELERATOR_KEY,
          KeyStroke.getKeyStroke( "d", Event.ALT_MASK ));

  }

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

Adding Properties

JavaOne

MyDialogAction Dialog (10)

```
public class MyDialogAction extends AbstractAction {

  private MyDialog _dialog;

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

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

Create the Dialog

JavaOne

MyDialogAction Dialog (11)

```
public class MyDialogAction extends AbstractAction {

  private MyDialog _dialog;

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

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

Every button push

JavaOne

Extensible Part 2
Extending Action & Using Property Files

[illegible] Actions Even Cooler

- Actions do manage properties, *but...*
  - Using putValue() is tedious
  - Every Action has the same type of code
- Add property files!

JavaOne

Expanding Property Use

- Properties are named pieces of information your application uses
- `java.util.Properties` manages properties
  - Maintains a database of properties
  - Reads properties from a file

JavaOne

Property File Example

```
myDialog.Name= Show My Dialog

myDialog.ShortDescription= Show my \
                           special dialog
myDialog.Accelerator=ctrl-d
```

These lines are in a property file
e.g. – actions.prop

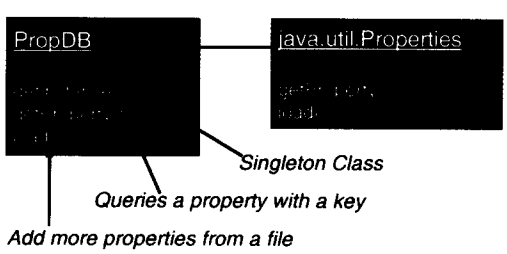
JavaOne

Adding Loaded Properties to AbstractAction

- AbstractAction – no way to load properties
- *We need a Subclass to*
  1. Add properties loading
  2. Look for certain property types
  3. Convert any necessary properties

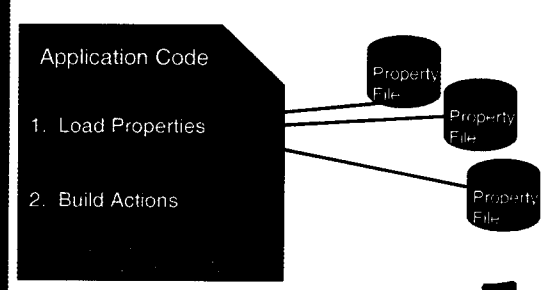
JavaOne

[illegible] Class



JavaOne

Action's Role in Application



JavaOne

GeneralAction

- Subclass of AbstractAction
- Add Property Loading

GeneralAction

Defines actionPerformed() → ActionListener

Adds 6 more methods → Action

Implements everything but actionPerformed() → AbstractAction

Adds *property loading* but does not implement actionPerformed() → GeneralAction

GeneralAction Code (1)

```
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
      //

 }
}
```

GeneralAction Code (2)

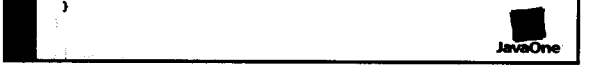
```
public abstract class GeneralAction extends AbstractAction
{

 public GeneralAction(String command)  {

 }
}
```

Command parameter is the base name for properties

GeneralAction Code (3)

```
public abstract class GeneralAction extends AbstractAction
{

 public GeneralAction(String command)  {
   PropDB prop= PropDB.getInstance();

 }
}
```

Get the Singleton property class



GeneralAction Code (4)

```
public abstract class GeneralAction extends AbstractAction

 public GeneralAction(String command)  {
   PropDB prop= PropDB.getInstance();

   String s;
   s= prop.getProperty(command + "." + SHORT_DESCRIPTION);

 }
```

Get the SHORT_DESCRIPTION property

GeneralAction Code (5)

```
public abstract class GeneralAction extends AbstractAction
{

  public GeneralAction(String command)  {
    PropDB prop= PropDB.getInstance();

    String s;
    s= prop.getProperty(command + "." + SHORT_DESCRIPTION);
    if (s != null) putValue(SHORT_DESCRIPTION, s);

}
```

If the property exists, then add it to the Action's properties

GeneralAction Code (6)

```
public abstract class GeneralAction extends AbstractAction
{

  public GeneralAction(String command)  {
    PropDB prop= PropDB.getInstance();

    String s;
    s= prop.getProperty(command + "." + SHORT_DESCRIPTION);
    if (s != null) putValue(SHORT_DESCRIPTION, s);

    s= prop.getProperty(command + "." + NAME);
    if (s != null) putValue( NAME, s);

}
```

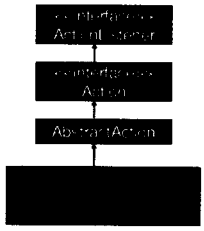
Do the same thing with NAME

GeneralAction Code (7)

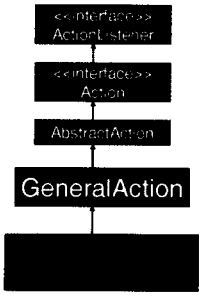
```
public abstract class GeneralAction extends AbstractAction
{

  public GeneralAction(String command)  {
    PropDB prop= PropDB.getInstance();

    String s;
    s= prop.getProperty(command + "." + SHORT_DESCRIPTION);
    if (s != null) putValue(SHORT_DESCRIPTION, s);

    s= prop.getProperty(command + "." + NAME);
    if (s != null) putValue( NAME, s);
      //
      // -- now do mnemonic, accelerator, etc
      //

}
```

Do the same thing with LONG_DESCRIPTION, SMALL_ICON, ACCELERATOR_KEY, MNEMONIC_KEY

GeneralAction Extends



GeneralAction Extends



MyDialogAction Now Extends GeneralAction

```
public class MyDialogAction extends GeneralAction {

  private MyDialog _dialog;

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


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

MyDialogAction Now Extends GeneralAction (2)

```
public class MyDialogAction extends GeneralAction {

  private MyDialog _dialog;

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

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

"*myDialog*" to GeneralAction

JavaOne

MyDialogAction Now Extends GeneralAction (3)

```
public class MyDialogAction extends GeneralAction {

  private MyDialog _dialog;

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

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

Short!!!

Same!!!

JavaOne

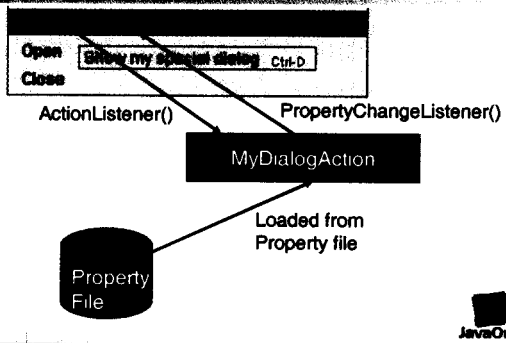
Property File Example - Again

```
myDialog.Name= Show My Dialog

myDialog.ShortDescription= Show my \
                           special dialog

myDialog.Accelerator=ctrl-d
```

This is the command name

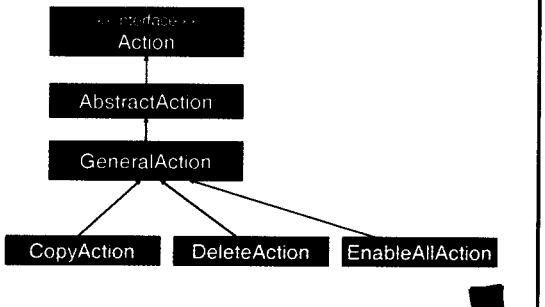
JavaOne

Example - Using Property Files



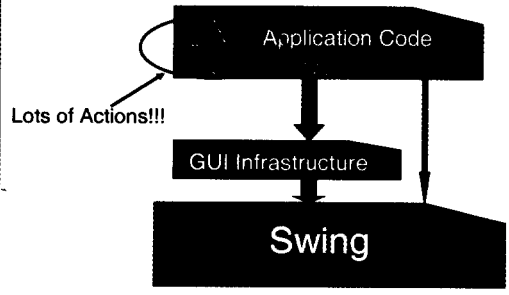
JavaOne

Actions Extend GeneralAction



JavaOne

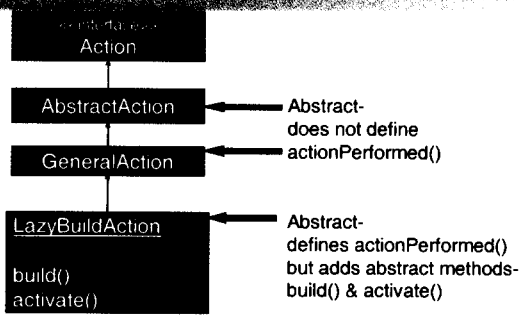
Many Actions



How Can We Improve Startup Time?

- Actions that display dialogs take a long time to create
- Slow actions affect start-up time
- We need lazy initialization

LazyBuildAction



LazyBuildAction

- New abstract methods
  - build()
  - activate()

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

Build the first time

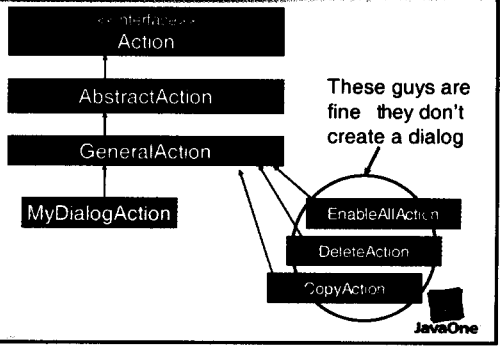
LazyBuildAction

- New abstract methods
  - build()
  - activate()

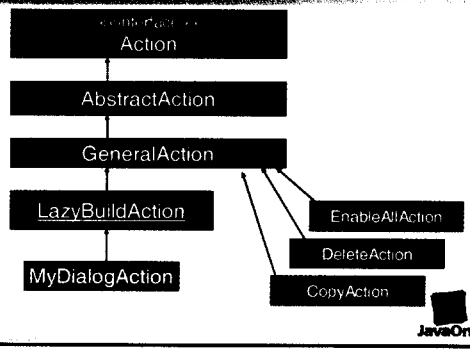
```
public void actionPerformed(ActionEvent ev) {
    if (!_built) {
        build();
        _built= true;
    }
    activate();
}
```

Activate every time

Where Should LazyBuildAction?



Where Should LazyBuildAction?



```
public class MyDialogAction extends LazyBuildAction {

 private MyDialog _dialog;
 private JFrame   _f;

 public MyDialogAction(JFrame f){
     super("myDialog");
     _f= f;
 }

 protected void build(){
     _dialog= new MyDialog(_f,data);
 }

 protected void activate(){
     _dialog.setVisible(true);
 }
}
```

JavaOne

```
public class MyDialogAction extends LazyBuildAction {

 private MyDialog _dialog;
 private JFrame   _f;

 public MyDialogAction(JFrame f){
     super("myDialog");
     _f= f;
 }

 protected void build(){
     _dialog= new MyDialog(_f,data);
 }

 protected void activate(){
     _dialog.setVisible(true);
 }
}
```

JavaOne

```
public class MyDialogAction extends LazyBuildAction {

 private MyDialog _dialog;
 private JFrame   _f;

 public MyDialogAction(JFrame f){
     super("myDialog");
     _f= f;
 }

 protected void build(){
     _dialog= new MyDialog(_f,data);
 }

 protected void activate(){
     _dialog.setVisible(true);
 }
}
```

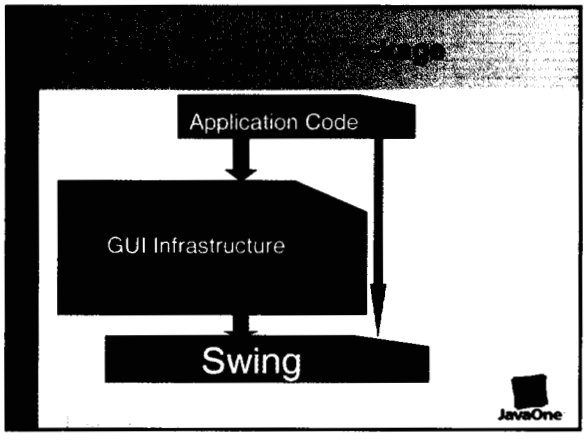
Create the Dialog

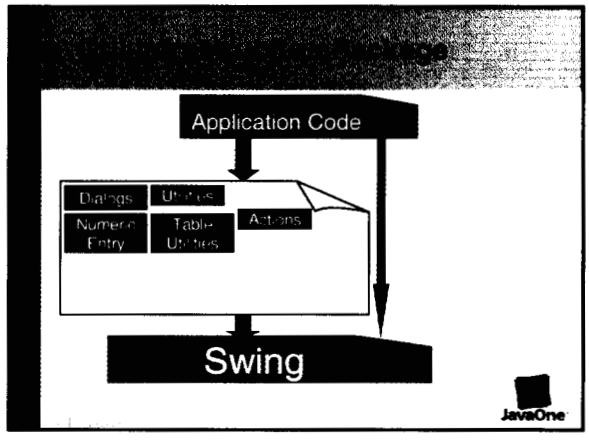
JavaOne

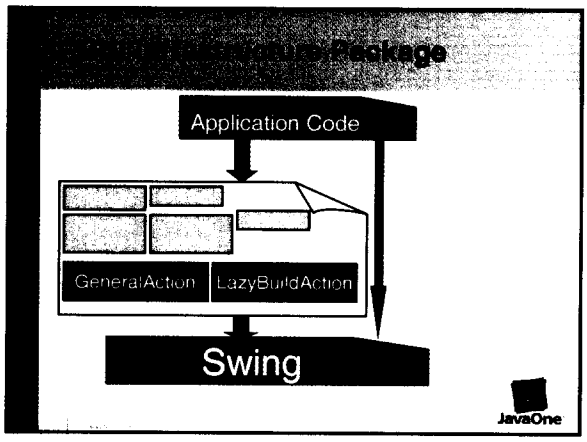
```
public class MyDialogAction extends LazyBuildAction {

 private MyDialog _dialog;
 private JFrame   _f;

 public MyDialogAction(JFrame f){
     super("myDialog");
     _f= f;
 }

 protected void build(){
     _dialog= new MyDialog(_f,data);
 }

 protected void activate(){
     _dialog.setVisible(true);
 }
}
```

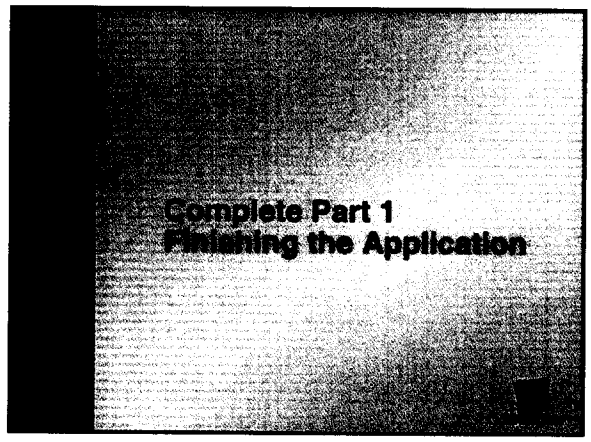
Set the dialog visible

JavaOne









Polishing Touches

- Toolbars
- Splash screens with progress bars
- Tool Tips
- Help
- Effective use of color
- Complete & informative error messages
- Drag & Drop

Impressions Are Everything

- 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

Small Features Are Important

- 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

Speed Is Key

- 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

Simply Remember

A good program is like a sky-scraper...

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

JavaOne

Q&A

JavaOne

Sun's 2002 Worldwide Java Developer Conference

BEYOND BOUNDARIES