

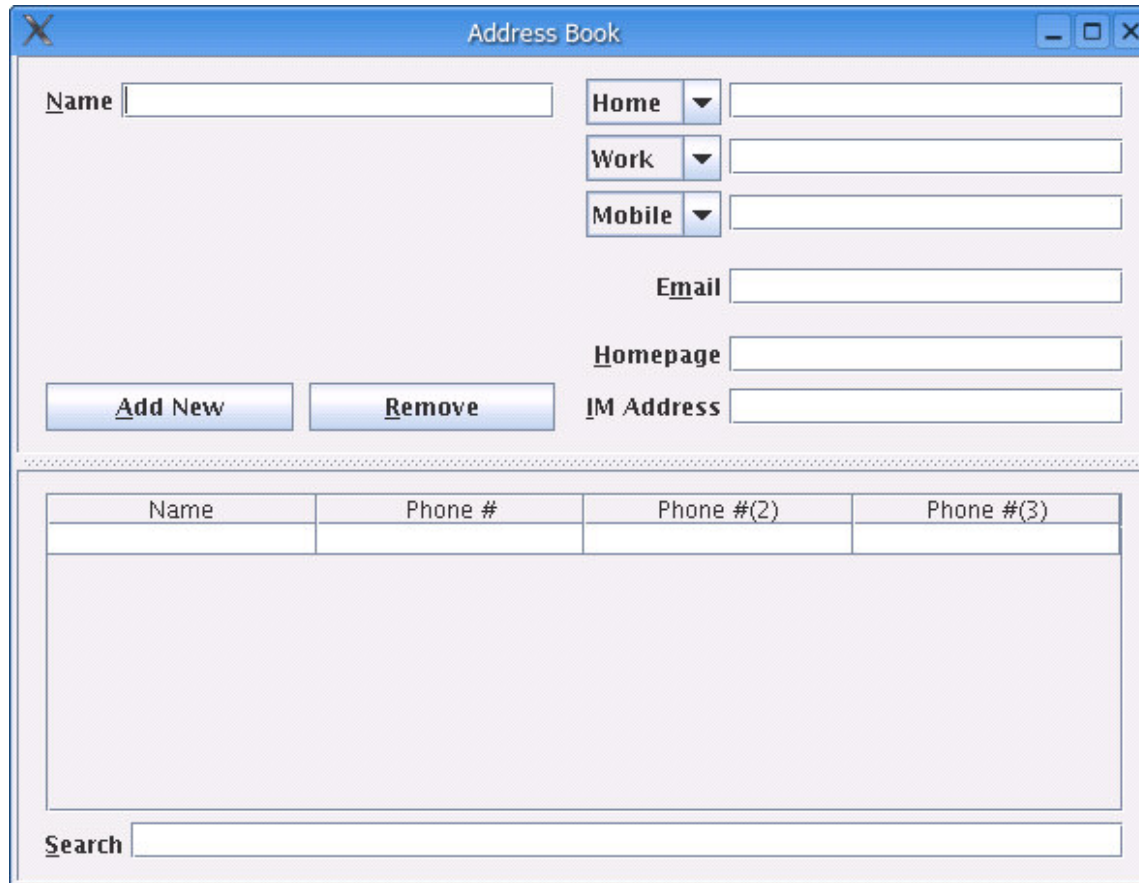


Jemmy Testing Toolkit

Mario Aquino (mario@ociweb.com)
Object Computing, Inc.



A Simple Demo...



Address Book

Name

Home

Work

Mobile

Email

Homepage

IM Address

Name	Phone #	Phone #(2)	Phone #(3)

Search



Testing

- All tests should:
 - Run automatically (without human assistance, intervention, or participation)
 - Have a pass/fail result
 - Should be able to be repeatedly run and provide consistent and indisputable results



Testing Approaches & Scopes

- Unit test(ing) should:
 - Focus on the contract of a single unit API (usually a class)
 - Not have external dependencies - should run in isolation
 - Run quickly, quietly, and all the time



Approaches & Scopes (cont.)

- Integration test(ing) should:
 - Focus on specific system-supported functionality
 - Test externally observable application behaviors
 - Exercise many aspects of a system at a time
 - System components working together to provide a capability
 - Take longer to run and have a much broader scope than unit tests
 - Have a different run frequency: nightly, weekly, or as a release is approaching



Approaches & Scopes (cont.)

- Others:
 - Performance testing
 - Security testing
 - Acceptance testing
 - ... (beyond the scope of this presentation)

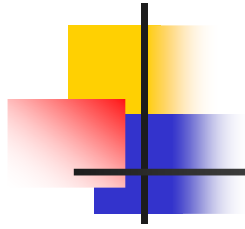


- Free, open source testing toolkit for Swing/JFC applications
- Supported by the developers of the NetBeans IDE
- <http://jemmy.netbeans.org>



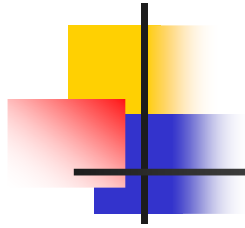
An Operator for Everything

- *Operator* classes provide interface to AWT & Swing components and containers:
JButtonOperator, JCheckBoxOperator, JColorChooserOperator,
JComboBoxOperator, JFileChooserOperator, JFrameOperator,
JInternalFrameOperator, JLabelOperator, JListOperator,
JMenuBarOperator, JMenuItemOperator, JPopupMenuOperator,
JProgressBarOperator, JScrollBarOperator,
JScrollPaneOperator, JSliderOperator, JSpinnerOperator,
JSplitPaneOperator, JTabbedPaneOperator,
JTableHeaderOperator, JTableOperator,
JTextComponentOperator, JTextFieldsOperator, JTreeOperator
... and many others



Using Operators

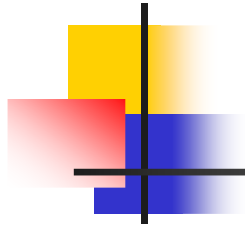
- Operators can find, read, and interact with components
- Finding components can be done several ways
 - By component name
 - By tooltip
 - By order within a container
 - Roll your own finding strategy
- Once found, component behaviors can be exercised by their corresponding operator



Decomposing a Test

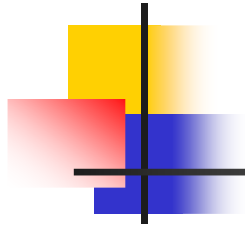
- Test sequence

1. Start your application by calling its main class
2. Find the main frame/window of your application
Everything in your app lives under this high-level container
3. Find a component somewhere inside the frame
(menu, textfield, combo box, table, etc.)
4. Use it or read its value (and make assertions)
5. Rinse and repeat...



Finding a Component

- Components can be found *quickly* or *patiently*
 - Static finder methods on Operator classes (quickly)
 - Find the component and return a reference to it or return null immediately
 - Operator class constructors (patiently)
 - Block until a timeout, waiting to find the component you are looking for

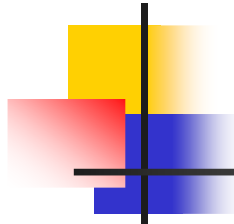


Finding a Component (cont)

- `ComponentChooser` interface used in finding components
 - Determines whether a particular component is the one you are looking for

```
boolean checkComponent(java.awt.Component comp);
```

- You can extend existing ones or create new custom choosers



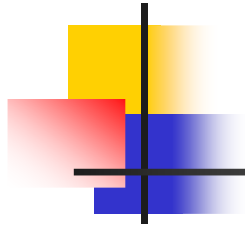
Queue Tool

- Jemmy API provides a pausing mechanism
 - QueueTool pauses execution thread until the event queue has been empty for variable period of time
 - Pausing is useful because it allows the UI to catch-up to where a test is trying to get



Testing Images

- Forms that display dynamic (generated) images can be tested using comparison tools
 - A displayed image (or portion therein) can be found (via rough or strict image finders)
 - RoughImageFinder & StrictImageFinder
 - Once found, the image can be compared (strictly or roughly) to a file image
 - RoughImageComparator & StrictImageComparator
 - Image produced during test can be stored to PNG for later visual inspection (if desired)
 - `org.netbeans.jemmy.image.PNGImageLoader`
 - `org.netbeans.jemmy.image.PNGImageSaver`



Scenario & Test

- The `org.netbeans.jemmy.Scenario` interface provides the actual test
 - Integration testing activity is defined within `runIt()` method
- The `org.netbeans.jemmy.Test` class executes a `Scenario`

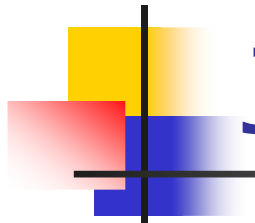
```
int runIt (Object o);
```

```
assertEquals(0, new Test(scenario).startTest(null));
```



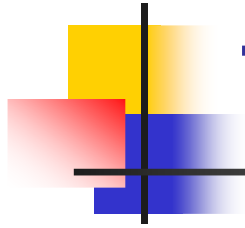
Jemmy & JUnit

- The Jemmy API does not provide for assertive evaluation
- This makes integrating Jemmy tests with JUnit a natural fit
 - The Jemmy APIs find Swing/AWT components on a form
 - The JUnit APIs perform the expected vs. actual value comparison (assertion)



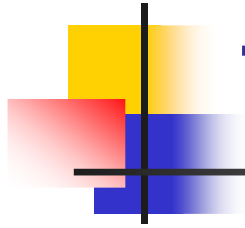
Jemmy & JUnit (cont.)

- Integrating Jemmy & JUnit gives you:
 - The facility to run Jemmy tests where ever you can run JUnit tests
 - From your IDE
 - From Ant or Maven
 - Jemmy integration tests can now be part of your continuous integration process
 - Your CruiseControl (build automation) process can now include regular execution of your integration tests



Testing the Simple Demo

- A test for adding and removing contacts
- A test for selecting contacts in the table
- A test for searching for contacts
- A test for sorting the contact table



Testing the “Interface”

- Jemmy tests should focus on the interface rather than the implementation
 - Input fields on a form are equivalent to method parameters
 - Data output on the form is the return value
 - Inputs, outputs, behaviors, and side-effects are the contract of the form
 - Just as with unit tests, focus on contract over implementation



Testing the “Interface” (cont.)

- Encourages UI design with consistent patterns:
 - All inputs and outputs have tooltips that are:
 - Immutable
 - Unique
 - Consistent
 - Initialized with the form
 - Type-safe enumerations are convenient for consolidating field metadata
 - Following these principles, tests are unaffected by rearranging the location of form inputs and outputs.



Jemmy in a TDD Context

- A typical TDD approach at the unit level:
 - Create a skeleton interface (but no implementation)
 - Write tests for the contract of interface
 - Run the tests to see that they fail
 - Start writing an implementation to fulfill the contract and pass the tests



Jemmy & TDD (cont.)

- A similar approach works for Jemmy testing as well:
 - Create a skeleton form with all the UI components
 - Write a Jemmy test to exercise functionality the form should provide
 - Run the test to confirm that it fails
 - Connect the components on the form to functionality-providing application logic



Jemmy Quirkiness

- Lots of output (if you don't turn it off)
 - JemmyProperties class controls where messages get sent during test execution
 - By default, Jemmy is verbose ☹
 - Tests should run quietly and only speak up when something is wrong!
 - `JemmyProperties.setCurrentOutput()` allows you to redirect message output



Jemmy Quirkiness (cont.)

- Exception thrown even when tests succeeds
 - TestCompletedException thrown after test finishes to signal successful completion

```
org.netbeans.jemmy.TestCompletedException: Test passed
    at org.netbeans.jemmy.Test.closeDown(Test.java:126)
    at org.netbeans.jemmy.Test.launch(Test.java:353)
    at org.netbeans.jemmy.ActionProducer.launchAction(ActionProducer.java:312)
    at org.netbeans.jemmy.ActionProducer.run(ActionProducer.java:269)
```




Jemmy Caveats

- Jemmy only supports JFC widgets
 - Custom widgets need to have custom operators designed for them
 - Widgets that extend JComponent and provide functionality not readily available in JFC widgets
- The Jemmy library doesn't have an obvious version stamp
 - Unfortunately, you need to check the website periodically for updates
 - You can look inside the Jemmy jar at a "version_info" file



Questions?

- Questions?
 - Questions?
 - Questions?