Jemmy tutorial

Introduction to Jemmy testing framework

Pawel Prokop

pawel.prokop@adfinem.net

http://prokop.uek.krakow.pl

March 14, 2012

Manualy testing

- error prone
- slow and not efficient
- repeating test costs a lot
- no place for TDD
- boring for developers

  (dinner syndrome)
- flexible
- look & feel fast feedback

Recording user actions to exercise the application

- efficient
- fast preparation
- fast execution
- cheap
- hard to modify during application development

  common functionalities
- no place for TDD

Available tools for test recording

- **SWTBot** is an open-source Java based functional testing tool for testing SWT and Eclipse based applications.
  `http://www.eclipse.org/swtbot/`
- **Marathon** allows to record the script that can be modified later manualy. Supports assertions.
  `www.marathontesting.com/`
- **Jacareto** is a tool that allows capture actions on applications and replay them later.
  `http://jacareto.sourceforge.net/wiki`

Write code that simulates user actions

- efficient
- moderate slow preparation
- parts of code may be reused
- fast execution
- easy to modify during application development
- can be used with TDD

Testing frameworks

- **UISpec4J** is an open source framework to test Swing applications.
  `http://www.uispec4j.org/`
- **JFCUnit** is an extension of JUnit that allows to execute unit tests against Swing based interface.
  `http://jfcunit.sourceforge.net/`
- **Jemmy** is an open source framework that allows to simulate user interactions with Swing applications.
  `https://svn.java.net/svn/jemmy~svn`

Summary

	Manual testing	Recording	Framework
Efficiency	-	+	+
Preparation time	+	+	+/-
Execution time	-	+	+
Price	-	+	+
Reusability	-	-	+
TDD	-	-	+
Flexibility	+	-	-

Jemmy introduction

Jemmy documentation

- not so many tutorials
- not so many presentations and documents
- good javadoc is enough to start
- few samples
- low Jemmy vitality

Jemmy framework

How Jemmy works?

- the same JVM as tested application
- simulates user operations on the components by calling events

  events are stored on the QueueTool class and then provided to AWT EventQueue
- search components recursively by given criteria (caption, name)
- criteria defined as implementation of ComponentChooser interface

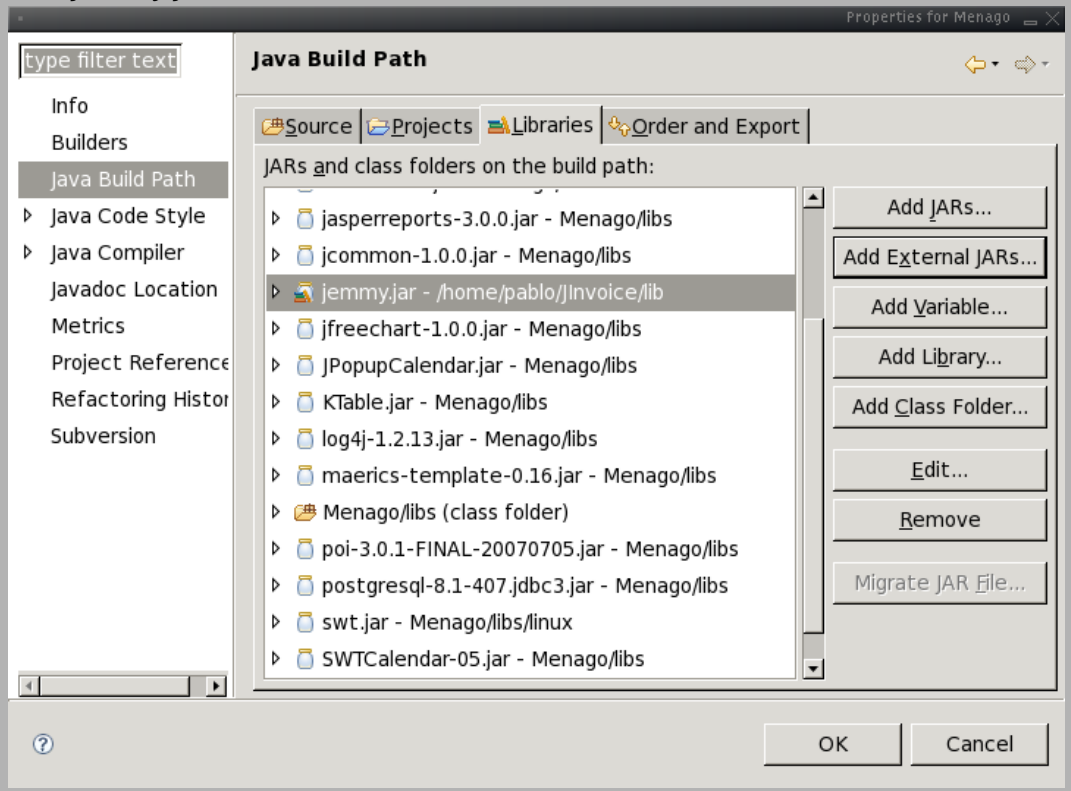
Jemmy framework

Jemmy in TDD approach

- design UI interface and the components
- implement jemmy tests to fit the interface
- run failing tests
- create implementation and re-run tests until they pass

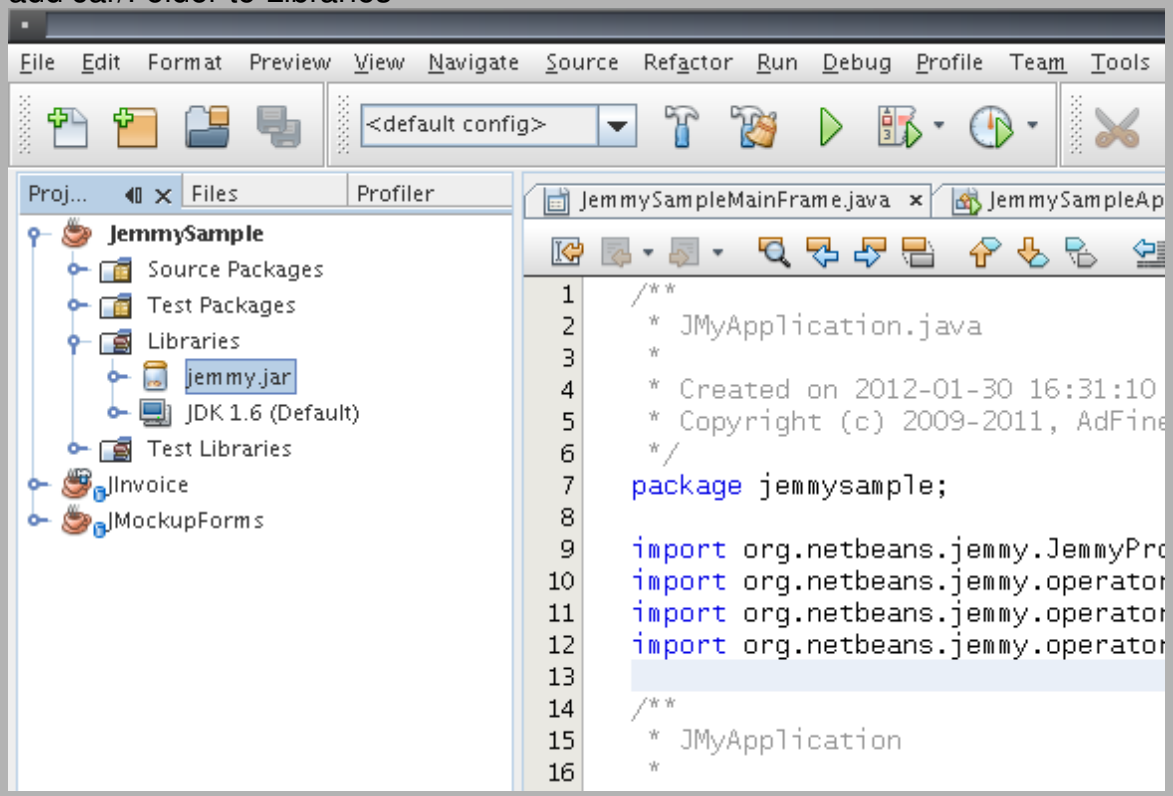
Jemmy Installation *Eclipse*

- add *jemmy.jar* to the Java Build Path



Jemmy Installation *Netbeans*

- add Jar/Folder to Libraries



Integration with JUnit

```
1  @Ignore
2  public class CommonTest {
3      protected static JMyApplication application = null;
4
5      public CommonTest() { }
6
7      @BeforeClass
8      public static void setUpClass() throws Exception { }
9
10     @AfterClass
11     public static void tearDownClass() throws Exception { }
12
13     @Before
14     public void setUp() throws Exception {
15         application = new JMyApplication();
16         application.startJMyForms();
17     }
18
19     @After
20     public void tearDown() throws Exception {
21         application.stopJMyForms();
22     }
23 }
```

Integration with Ant

Integration with Ant

```
1  <project name="JemmySample">
2      <target name="junit">
3          <junit printsummary="yes">
4              <classpath path="libs/junit-4.5.jar" />
5              <classpath path="libs/jemmy.jar" />
6              <classpath path="build/test/classes/" />
7              <classpath path="build/classes/" />
8              <batchtest fork="yes">
9                  <fileset dir="test" includes="**/*Test.java"/>
10             </batchtest>
11         </junit>
12     </target>
13 </project>
```

```
1  ant -f junit.xml junit
```

Operators

- Communication with UI controls is realized by operators
- Jemmy defines own operators for every AWT/Swing UI component
  - JFrameOperator
  - JDialogOperator
  - JButtonOperator
  - JCheckBoxOperator
  - JComboBoxOperator
  - JFileChooserOperator
  - ...and many others
- Operators can be extended by the developer

Operators - small example

- Yes/No dialog operator

```
1  public class YesNoDialogOperator extends JDialogOperator {
2      public YesNoDialogOperator() {
3          super("Confirmation");
4      }
5
6      public void pushYes() {
7          new JButtonOperator(this, "Yes").push();
8      }
9
10     public void pushNo() {
11         new JButtonOperator(this, "No").push();
12     }
13 }
```

Operators - small example

- Usage

```
1  // ...
2  @Test
3  public void test_some_functionality() {
4      // ...
5      YesNoDialogOperator operator = new YesNoDialogOperator();
6      assertNotNull(operator);
7      operator.pushYes();
8      // ...
9  }
10 // ...
```

Load application with jemmy

- class reference

```
1  new ClassReference("jemmysample.JemmySampleApplication").startApplication();
```

- direct call

```
1  JemmySampleApplication.main(null);
```

Concept depended approach

- create classes that support some application's functionality:

```
1  class MyMainDialog extends JDialogOperator {
2      public static String dialogCaption = "My Dialog";
3
4      public MyMainDialog() { super(dialogCaption); }
5
6      public void pushButtonCancel() {
7          new JButtonOperator("Cancel").push();
8      }
9  }
```

- and then use them from test suites:

```
1  @Test
2  public void test_dialog_can_cancel() {
3      MyMainDialog dialogOperator = new MyMainDialog();
4      dialogOperator.pushButtonCancel();
5  }
```

Application Shadow

```
1  public class JMyApplication {
2      private JFrameOperator mainFrame = null;
3
4      public JFrameOperator getMainFrame() {
5          return mainFrame;
6      }
7
8      public int startJMyForms () {
9          JemmySampleApplication.main(null);
10         mainFrame = new JFrameOperator("JMyForms");
11         return 0;
12     }
13
14     public int stopJMyForms() {
15         JMenuBarOperator menuBarOp = new JMenuBarOperator(mainFrame);
16         JMenuOperator fileMenu = new JMenuOperator(menuBarOp.getMenu(0));
17         fileMenu.pushMenu("File|Exit");
18         return 0;
19     }
20 }
```

The Component Choosers

How to find component?

- using its index inside the container

  textfield next to label

- using Component Chooser implementation
  - `NameComponentChooser` uses a `name` property
  - `PropChooser` uses properties and methods to match a component
  - Developer can define custom component choosers - implementing `ComponentChooser` interface

RegexpFrameTitleChooser

```
1  public class RegexpFrameTitleChooser implements ComponentChooser {
2      Pattern pattern = null;
3
4      public RegexpFrameTitleChooser (String pattern) {
5          this.pattern = Pattern.compile(pattern);
6      }
7
8      @Override
9      public boolean checkComponent(Component component) {
10         if (component instanceof JFrame) {
11             String frameTitle = ((JFrame)component).getTitle();
12             Matcher matcher = pattern.matcher(frameTitle);
13             return matcher.matches();
14         }
15         return false;
16     }
17
18     @Override
19     public String getDescription() {
20        return pattern.pattern();
21     }
22 }
```

ShowingComponentChooser

```
1  public class ShowingNameComponentChooser extends NameComponentChooser {
2      public ShowingNameComponentChooser (String name) {
3          super(name);
4      }
5
6      @Override
7      public boolean checkComponent(Component component) {
8          if (super.checkComponent(component) && component.isShowing()) {
9              return true;
10         }
11         return false;
12     }
13
14
15
16 }
```

Timeouts

- time values (millisec-onds) to wait for something to be done by the application

Timeouts

- to delay simulated user behaviour
  - kept by `org.netbeans.jemmy.Timeouts` class

```
1  // Set delay before key is pressed during typing
2  JemmyProperties.getCurrentTimeouts().setTimeout
3      ("JTextComponentOperator.PushKeyTimeout", 50);
```

  - each instance can have its own timeouts set

```
1  // Set delay between button pressing and releasing.
2  JButtonOperator btnOperator = new JButtonOperator("Process");
3  btnOperator.getTimeouts()
4      .setTimeout("AbstractButtonOperator.PushButtonTimeout", 500);
```

Timeouts cont.

- to wait until timeout exception is raised

```
1  // wait up to 5 seconds for JDialog
2  JemmyProperties.getCurrentTimeouts().setTimeout
3      ("DialogWaiter.WaitDialogTimeout", 5000);
4  JDialogOperator albums = new JDialogOperator("Dialog Caption");
```

More timeouts

- `JComboBoxOperator.BeforeSelectingTimeout` - time to sleep after list opened and before item selected
- `JComboBoxOperator.WaitListTimeout` - time to wait list opened
- `ComponentOperator.WaitStateTimeout` - time to wait for item to be selected
- `JTextComponentOperator.PushKeyTimeout` - time between key pressing and releasing during text typing
- `JTextComponentOperator.BetweenKeysTimeout` - time to sleep between two chars typing

find method vs operator constructor

- static find method returns the `operator` or `null` immediately:

```
1  @Test
2  public void test_some_functionality() {
3      // ...
4      JDialogOperator myDialog = JDialogOperator.findDialog("Error", true, true);
5      if (null == myDialog)   {
6          // no Error dialog is currently displayed
7      }
8      // ...
9  }
```

- constructor blocks until component appears on the screen or timeout.

```
1  public void test_some_functionality() {
2      // ...
3      JDialogOperator myDialog = new JDialogOperator("Error");
4      // ...
5  }
```

Screenshot feature

- Capture screen shot

```
1  // ...
2  @Test
3  public void testSomeFunctionality() {
4      // ...
5      org.netbeans.jemmy.util.PNGEncoder
6          .captureScreen("testSomeFunctionality.png");
7      // ...
8  }
9  // ...
```

- Image modes
  - black and white
  - grayscale
  - colour

Dump components in JVM

Dump components in JVM

- Dump all components

```
1  // ...
2      @Test
3      public void testSomeFunctionality() {
4          // ...
5          org.netbeans.jemmy.util.Dumper.dumpAll("jemmy_example.xml");
6          // ...
7      }
8  // ...
```

- Dump component

```
1  // ...
2  public void testSomeFunctionality() {
3      // ...
4      Component comp = application.getMainFrame().getComponent(0);
5          org.netbeans.jemmy.util.Dumper.dumpComponent( comp,
6              "jemmy_example_component.xml");
7      // ...
8  }
9  // ...
```

GUIBrowser tool

- shows all GUI objects in current JVM
- shows components in hierarchy
- allows to take snapshot in defined time delay
- displays component members (name, caption, size...)

Launch GUIBrowser tool

```
1    new ClassReference("org.netbeans.jemmy.explorer.GUIBrowser").startApplication();
```

Questions?

Questions?

Links

1. Operators:
   `http://wiki.netbeans.org/Jemmy_Operators_Environment`
2. Jemmy repository: `https://svn.java.net/svn/jemmy~svn`
3. Jemmy tutorial: `http://wiki.netbeans.org/Jemmy_Tutorial`
4. this presentation: `http://prokop.ae.krakow.pl/projects/download/jemmy_introduction.pdf`

Credits

1. flickr/ant/binux
2. flickr/clock/kobiz7