Developing GUIs in HIPE

Jaime Saiz Santos

HIPE Forum 2011

Garmisch-Partenkirchen, Germany

esa

European Space Agency

ROADMAP

- General guidelines
- GUI development
- GUI testing
- Other test utilities

ROADMAP

- ➔ **General guidelines**
- GUI development
- GUI testing
- Other test utilities

GENERAL GUIDELINES

esa

- Know the ways things are being done
  - Read documentation...
  - ... or ask for help
  - Commonalities: don't reinvent the wheel
- Keep it simple
  - Avoid copy & paste
  - Put common code in abstract or utility classes
- Make it maintenable
  - Write test harnesses for your code
  - Comment code, document it

ROADMAP

- ✔ General guidelines
- ➔ **GUI development**
- GUI testing
- Other test utilities

esa

- Beware of the EDT
  - GUI code must be executed in the Event Dispatching Thread (EDT)
  - Some events do happen outside the EDT (like `CommandExecutedEvent`)
  - Use `SwingUtilities.invokeLater` in such cases
  - Be careful with `SwingUtilities.invokeAndWait`
    - Can produce deadlocks
    - Some call it `invokeAndDie`

GUI DEVELOPMENT

esa

- Make the application responsive (1)
  - If a task or job is taking too long, there should be a way to cancel it
  - Using a **`BusyJob`** lets you cancel it with the *Stop* button if you override `cancel()` properly (refer to its *Javadoc*)
  - Long lasting running code in HIPE happens mainly within **Jython code**
  - Normal commands can be cancelled with *Stop*. E.g.:

```
while 1: print 1  # infinite loop
```

esa

- Make the application responsive (2)
  - If you write **tasks** that might take long to execute, they shall be **cancellable**
  - To let a task be cancelled in the middle of its execution, call **`Task.checkInterrupted`** from time to time within the execution.
  - Example:

```
// In the execute method of a task
for (int i = 0; i < nSteps; i++) {
    checkInterrupted("Interrupted in step " + i);
    doStep(i);
}
```

esa

- Make the application responsive (3)
  - Every **selection** in HIPE triggers a call to a bunch of **listeners**, which are run sequentially

    ⇒ `selectionChanged(SiteEvent event)` must be quick
  - (Hundreds of) **tasks** are checked for **applicability** for each variable selection

    ⇒ `validate` of `ParameterValidators` must be quick
  - *Quick* implies that any access to disk is forbidden

esa

- Make the application responsive (4)
  - **Code** executed **in the EDT** should be fast enough
  - If it takes too long to execute, the GUI gets frozen
  - It is not possible to cancel it with the *Stop* button, because the click is handled in the EDT, which is busy!
  - Execute long lasting code in the background
    - `javax.swing.SwingWorker`
    - `herschel.ia.gui.apps.components.util.Dereferencer`
    - `java.util.concurrent.Executors`
    - `java.lang.Thread`

They call a method in the EDT when finished (SwingWorker, Dereferencer)

You may call `invokeLater` when finished (Executors, Thread)

ROADMAP

- ✔ General guidelines
- ✔ GUI development
- ➔ **GUI testing**
- Other test utilities

- Testing GUIs may be tricky
  - Testing GUIs in HIPE may be trickier
  - Sometimes you need a *mock Site*. Here it is:

```
Site site = SiteUtil.getSite();
if (site == null) {
    site = new AbstractSite() { // registers itself
        @Override public Image  getLogo    () { return null; }
        @Override public String getName    () { return "testName"; }
        @Override public String getProject() { return "testProject"; }
        @Override public String getTitle   () { return "testTitle"; }
    };
}
```

Obsolete

GUI TESTING

- Two helper classes for testing GUIs in HIPE
  - `GuiFrameTest`
  - `GuiSiteTest`
  - Both reside in `herschel.ia.gui.kernel.util.test`
  - They are designed for extension, though they can be used with composition as well

GUI TESTING

- `GuiFrameTest`
  - Meant for testing GUI components in isolation, by showing them in a `JFrame`
  - No Site, Editor area, Views, etc. are required
  - The setup method creates a frame (but doesn't show it)
  - Extending classes can:
    - `getFrame()`, populate and show it manually, or
    - `showFrameWith(component)`, opening the frame with the given content
  - The tear down method takes care of closing the frame

GUI TESTING

- `GuiSiteTest`
  - Meant for testing GUI components with a `Site` defined, and optionally a perspective with the editor area, views...
  - Every test is run in *isolation*
    - Doesn't depend on whether there is a previously defined `Site`, the state of the `ExtensionRegistry`, etc.
    - Doesn't affect subsequent tests: everything is restored to its previous state
  - Stub site and perspective are provided
  - Register in the `ExtensionRegistry` the items you need
  - Optionally, show the built frame (which is the default)

GUI TESTING

- GuiSiteTest can be extended directly, but there are specializations that allow writing less code:
  - EditorComponentTest for testing an EditorComponent
    - FileEditorComponentTest for testing an EditorComponent that handles a FileSelection
    - VariableEditorComponentTest for testing an EditorComponent that handles a VariableSelection
  - ViewableTest for testing a Viewable

GUI TESTING

- If possible, avoid implementing interfaces of `herschel.ia.gui.kernel.parts` in tests
  - These interfaces are meant to be implemented only in `herschel.ia.gui.kernel.parts.impl`
  - Creating test implementations makes more difficult their evolution (add or change methods, etc.)
  - With `GuiSiteTest`, there is no need to create mock `Site`, `EditorArea`, `EditorPart`...
    The required framework is already provided
  - In case of necessity, use `GuiSiteTest.StubSite`

- Respect the EDT when testing GUIs
  - Main test thread in JUnit is *not* the EDT

```
@Test
public void testSomeGui() {
    MyComponent myComponent = new MyComponent();
    myComponent.doSomething();
}
```

```
@Test
public void testSomeGui() throws Exception {
    SwingUtilities.invokeAndWait(new Runnable() {
        @Override public void run() {
            MyComponent myComponent = new MyComponent();
            myComponent.doSomething();
        }
    });
}
```

GUI TESTING

esa

- Respect the EDT when testing GUIs
  - `UI.runInEDT` ≈ `SwingUtilities.invokeAndWait`
    - More concise
    - Unchecked exceptions (OK for testing)
    - Waits for all events queued in the EDT before returning
    - Optional argument for pausing after its execution

```
@Test
public void testSomeGui() {
    UI.runInEDT(new Runnable() { @Override public void run() {
         MyComponent myComponent = new MyComponent();
         myComponent.doSomething();
    }});
}
```

GUI TESTING

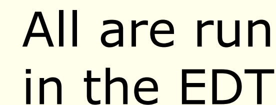
- Simulate GUI events when testing
  - Call listener methods with fake events when possible
  - Generate OS events if not
    - `java.awt.Robot`
    - `herschel.ia.gui.kernel.util.test.Cyborg`
  - Beware that the conditions in the CIB machine may differ yours (OS event handler, etc.) so, in few cases, using Robot & Cyborg may succeed in your computer but fail in the CIB
    - Working to alleviate this: INFR-566

A handy wrapper of Robot

Not done

GUI TESTING

- Simulate GUI events when testing
  - Call listener methods with fake events
  - `UI` class comes in handy here as well:
    - `click`, `leftClick`, `middleClick`, `rightClick`
    - `clickButton`
    - `clickButtonWhenShown`
    - `inputWhenAsked`
    - `mouseEvent`
    - `keyEvent`
    - `press`, `release`, `type`



GUI TESTING

- How to get the target component to check?
  - UI class comes in handy here as well:
    - `findNamedComponent`, `findNamedDialog`, `findNamedFrame`
    - `findLabelWithText`, `findButtonWithText`
    - `findDialogWithTitle`, `findFrameWithTitle`
    - `findFileChooserCurrentlyShowing, findFileChooser`
    - `findProgressBarCurrentlyShowing`
    - `findComponent` (many variants)
  - `SiteUtil.getDescendant`, `SiteUtil.getAncestor`
  - `SwingUtilities.getWindowAncestor`

GUI TESTING

esa

- To wait for conditions, like a job to be finished
  - Know utility classes
    - `UI.awaitEDT()`
    - `java.util.concurrent.Semaphore`
    - `herschel.ia.gui.kernel.util.test.Waiting`
  - Annotate your test with a timeout, to avoid the build hanging (and be killed later without exact knowledge of what happened)

```
@Test(timeout = 60000)  // in milliseconds
testSomeGui() {
    ...
}
```

esa

- Example:

```
// Auxiliary variables
Semaphore cancelClicked = new Semaphore(0);
MenuManager menuManager = getSite().getMenuManager();
final SiteAction exportAction = menuManager.getAction("Export");
final String exportTitle = "Export session";

// Run the export action and press Cancel
UI.clickButtonWhenShown("Cancel", cancelClicked);
UI.runInEDT(new Runnable() { @Override public void run() {
    exportAction.actionPerformed(null);                 // opens a dialog
    assertNotNull(UI.findDialogWithTitle(exportTitle)); // dialog shown
}});
cancelClicked.acquire();                                // wait for Cancel
assertNull(UI.findDialogWithTitle(exportTitle));        // dialog closed
```

- Example (cont.):

```
// Select a file and export the session to it
final JFileChooser fileChooser = UI.findFileChooser(); // previously opened
final File exportFile = new File("export.zip");
assertFalse(exportFile.exists());
UI.runInEDT(new Runnable() { @Override public void run() {
    fileChooser.setSelectedFile(exportFile);
    fileChooser.approveSelection();
}});

// Ensure the export file has been created
assertTrue(Waiting.waitFor(new Condition() {
    @Override public boolean isSatisfied() { return exportFile.exists(); }
}, 5000));
UI.clickButton("OK");   // acknowledge info message
```

ROADMAP

- ✔ General guidelines
- ✔ GUI development
- ✔ GUI testing
- ➔ **Other test utilities**

OTHER TEST UTILITIES

- We've seen some classes meant for testing GUIs:
  - `GuiFrameTest`, `GuiSiteTest`
  - `UI`, `Waiting`
- Package `herschel.ia.gui.kernel.util.test` contains few more classes to help testing in general. Just will mention 2 here:
  - `AllTestsBase`
  - `Reflection`

OTHER TEST UTILITIES

esa

- Have you found yourself coding test classes like these?

```
public class SomeClassTest {
    @Test
    public void testSomeThing() {
        ...
    }
    public static junit.framework.Test suite() {
        return new JUnit4TestAdapter(SomeClassTest.class);
    }
}

public class OtherClassTest {
    @Test
    public void testOtherThing() {
        ...
    }
    public static junit.framework.Test suite() {
        return new JUnit4TestAdapter(OtherClassTest.class);
    }
}
```

OTHER TEST UTILITIES

- And the `AllTests` class:

```
public class AllTests extends TestCase {

    public static Test suite() {
        TestSuite suite = new TestSuite("All tests");
        suite.addTest(SomeClassTest.suite());
        suite.addTest(OtherClassTest.suite());
        ...
        return suite;
    }
}
```

OTHER TEST UTILITIES

- With `AllTestsBase`:

```
public class AllTests extends AllTestsBase {

    public static Test suite() {
        return suite(SomeClassTest.class, OtherClassTest.class, ...);
    }
}

public class SomeClassTest {
    @Test
    public void testSomeThing() {
        ...
    }
}

public class OtherClassTest {
    @Test
    public void testOtherThing() {
        ...
    }
}
```

More concise

No need to define the static `suite()` methods here

OTHER TEST UTILITIES

- When increasing test coverage, we sometimes confront with the situation that there are many private methods difficult to test directly
  - One solution is to make them package private
    - So we can test them from the same package
    - And external code cannot access it
  - Other solution is to keep them private and use utility methods of the `Reflection` class
    - Easier than using `java.lang.reflect` directly

OTHER TEST UTILITIES

esa

- Main Reflection methods

Method name	Description
`make`	Creates an object through its constructor (like in `AbstractUtilTest`)
`call`	Calls a non-public method by name
`getFieldValue`	Gets the value of a non-public field
`setFieldValue`	Sets the value of a non-public field

  - Plus short-cuts (without checked exceptions) to get or call `Class`, `Constructor` and `Method` objects:
    - `getClass`
    - `getConstructor`
    - `getMethod`
    - `callMethod`

See *Javadoc* for details

THANK YOU

Questions?

Thank you