Automated GUI Tests with SWTBot
Jens Kübler
Overview

Introduction
Requirements for GUI tests
Live Execution
Concepts
Code Example
Conclusion
Tradeoffs for automated GUI tests

- **Manual testing vs. automated testing**
  - Outcome: User “noise“ vs. precise results
  - Low frequency vs. daily (or more) builds
  - Error detection vs. regression

- **Time to create a test + time to maintain it**
**Tight integration**
- Use JUnit to execute
- Use Eclipse launching facilities
- Use Plugin infrastructure
- Dock to SWT

**Usability**
- Maintainable!
- Readability
- Abstractions
Requirements to GUI tests 2/2

- Extensible
  - Custom SWT Controls
  - Custom search strategies within the UI

- Continuous Integration

- I18n
Show me what you got
Commonly used functionality built-in SWTBot

- Example: Checkbox
  - checkBoxWithName(String)
  - checkBoxWithName(String, int)
  - checkBox(String)
  - checkBox(String, int)
  - checkBoxWithTooltip(String)
  - checkBoxWithTooltip(String, int)
  - checkBoxWithId(String, String)
  - checkBoxWithId(String, String, int)
  - checkBoxWithId(String)
  - checkBoxWithId(String, int)

Optional to define IDs for controls in ambiguous situations

I18N: Resource bundles
- Advanced search strategies through matchers
- Extend BaseMatcher or AbstractMatcher
- Example: WithText<T> matcher

```java
protected boolean doMatch(Object obj) {
    try {
        boolean result = false;
        if (ignoreCase)
            result = getText(obj).equalsIgnoreCase(text);
        else
            result = getText(obj).equals(text);
        return result;
    } catch (Exception e) {
        // do nothing
    }
    return false;
}
```

- Matcher quantifiers: AllOf<T>, AnyOf<T>, ...
Concepts: Test Execution Flow 1/2

- Separate launcher (vs. PDE launcher)
- Runs in a non-UI thread
  - Pros
    - Non blocking
    - Sending events to UI (i.e. close blocking dialogs)
  - Cons
    - Threading issues
    - Additional tweaks for headless testing
### Solutions to threading issues

1) Send thread to sleep an arbitrary time
   - Bad because timing is tied to the test case
   - What if the amount of time does work only for some systems?

2) Let SWTBot handle this issue
   - Defines a default search timeout
   - Central point for specifying timeout behaviour
   - Can be modified for the machine it is running on
   - Use Interface ICondition

```
Concepts: Domain & Page Objects

- **Domain Objects**: Encapsulate Domain functionality
  - Create a project
  - Compile a Java project

- **Page Objects**: Encapsulate UI functionality
  - How to click a button
  - How to navigate to a menu
  - Hold and expose the (error) state of UI elements
  - Examples
    - Menu
    - Specific View i.e. Navigator
Additionals Features and Missing Items

- **Features**
  - Screenshots in tests
  - Integration for headless build
  - Extensible for custom controls
  - Spy View for inspecting SWT Controls (Shift+CTRL)
  - Logging via Log4J

- **Missing**
  - Not all SWT controls supported yet
  - Good documentation
  - No support for native dialogs (i.e. FileDialog, Print)
Conclusion

- Promising framework for GUI testing with Eclipse
- Very intuitive
- Extensible because of open source
- Still incubation
- Some more additional libs/jars required
- SWTBot 4GEF not integral part of SWTBot, yet
Links

- SWTBot: http://www.eclipse.org/swtbot/
- SWTBot4GEF: http://code.google.com/p/swtbot4gef/
- Aquintos: www.aquintos.com
Thank you very much for your attention!

aquintos GmbH
Lammstrasse 21
76133 Karlsruhe, Germany

phone  +49 (0) 721 51638-0
fax     +49 (0) 721 51638-38

info@aquintos.com
www.aquintos.com

Dipl-Inf. Jens Kübler
Software Engineer
mail: kuebler@aquintos.com
Currently unsupported SWT Controls

Button Arrow
Browser
Canvas
Composite
CTabFolder
Link
ProgressBar
Sash
Scale
ScrolledComposite
Slider
Spinner
TabFolder