

# jfcUnit – Ein Überblick

Von Knut Lorenz

02.06.04

# Gliederung

- **1. Motivation**
  - Wozu jfcUnit?
- **2. Kenndaten**
  - Was ist jfcUnit?
- **3. Fähigkeiten**
  - Was bietet jfcUnit?
- **4. Verwendung**
  - Wie benutzt man jfcUnit?
- **5. Beispiele**
  - Wie sieht jfcUnit in Aktion aus?
- **6. Fazit / Grenzen**
  - Was kann jfcUnit nicht?

# 1.1 Motivation

Wozu jfcUnit?

- Applikation besteht aus vielen Schichten
- Jede Schicht implementiert verschiedene Funktionalitäten als Methoden
- Diese Methoden können durch einen „fingierten“ Aufruf getestet werden
- Problem: Test der UI-Schicht da diese Interaktion voraussetzt

# 1.2 Motivation

Wozu jfcUnit?

- Wie testet man Interaktionen mit dem Nutzer? (manuelle Tests wären unpraktisch)
- Wie stelle ich bei automatischen Tests sicher, dass die JVM genug Zeit hatte um UI zu rendern?
- Lösung: jfcUnit
- => Bietet Framework zum Test JFC/Swing basierter UIs

# 2. Kenndaten

Was ist jfcUnit?

- SourceForge Projekt
- <http://jfcunit.sourceforge.net/>
- Unter GNU-Lizenz in Java-implementiert
- Aktuelle Version 2.06
- Erweiterung zu JUnit
- Setzt jUnit ab Version 3.7 und Apache Jakarta regexp ab Version 1.2 voraus

# 3. Fähigkeiten

Was bietet jfcUnit?

- „Threadsafer“ Test
- Zugriff auf Java Container
- Finden von Komponenten in Containerhierarchie
- Auslösen von Events auf diesen Komponenten
- Interaktionssimulation durch Event-Firing  
und/oder `java.awt.robot` (simuliert Eingaben von Tastatur und Mouse)
- Aufnahme / Abspielen von Tests (XML)
- Plugins für Eclipse und JBuilder verfügbar

# 4. Verwendung

Wie benutzt man jfcUnit?

- jfcUnit-Klasse umschliesst zutestende Klasse
- Erzeugte UI-Elementen können manipuliert werden
- Zusicherungen ermöglichen Steuerung
- Komplexe Testsuites lassen sich als XML speichern und abspielen

# 5.1 Beispiele

Wie sieht jfcUnit in Aktion aus?



- Test einer Textbox
- 1. Testfälle festlegen
- 2. Rahmen erstellen, setUp(), tearDown() implementieren
- 3. Testfälle implementieren
- 4. Test ausführen

# 5.2 Beispiele

Wie sieht jfcUnit in Aktion aus?

- Erstellen des Rahmens:

```
import junit.extensions.jfunit.*;
import junit.extensions.jfunit.finder.*;
import junit.extensions.jfunit.eventdata.*;

public LoginScreenTest extends JFCTestCase {
    private LoginScreen loginScreen = null;

    public LoginScreenTest( String name ) {
        super( name );
    }

    protected void setUp() throws Exception {
        super.setUp();

        // Choose the text helper
        setHelper( new JFCTestHelper( ) ); // Uses the AWT Event Queue.
        // setHelper( new RobotTestHelper( ) ); // Uses the OS Event Queue.

        loginScreen = new LoginScreen( 'LoginScreenTest: ' + getName( ) );
        loginScreen.setVisible( true );
    }

    protected void tearDown() throws Exception {
        loginScreen = null;
        getHelper().cleanup( this );
        super.tearDown();
    }

    ... TESTS ...
}
```

# 5.3 Beispiele

Wie sieht jfcUnit in Aktion aus?

- Implementierung eines Testfalls

```
public void testuserAndPasswordEmpty() {  
    1:    JDialog dialog;  
  
    2:    NamedComponentFinder finder = new NamedComponentFinder(JComponent.class, "Exitbutton" );  
    3:    JButton exitButton = ( JButton ) finder.find( loginScreen, 0 );  
    4:    assertNotNull( "Could not find the Exit button", exitButton );  
  
    5:    finder.setName( "EnterButton" );  
    6:    JButton enterButton = ( JButton ) finder.find( loginScreen, 0 );  
    7:    assertNotNull( "Could not find the Enter button", enterButton );  
  
    8:    finder.setName( "LoginNameTextField" );  
    9:    JTextField userNameField = ( JTextField ) finder.find( loginScreen, 0 );  
    10:    assertNotNull( "Could not find the userNameField", userNameField );  
    11:    assertEquals( "Username field is empty", "", userNameField.getText() );  
  
    12:    finder.setName( "PasswordTextField" );  
    13:    JTextField passwordField = ( JTextField ) finder.find( loginScreen, 0 );  
    14:    assertNotNull( "Could not find the passwordField", passwordField );  
    15:    assertEquals( "password field is empty", "", passwordField.getText() );  
  
    16:    getHelper().enterClickAndLeave( new MouseEventData( this, enterButton ) );  
    17:    DialogFinder dFinder = new DialogFinder( loginScreen );  
    18:    showingDialogs = dFinder.findAll();  
    19:    assertEquals( "Number of dialogs showing is wrong", 1, showingDialogs.size() );  
    20:    dialog = ( JDialog ) showingDialogs.get( 0 );  
    21:    assertEquals( "wrong dialog showing up", "Login Error", dialog.getTitle() );  
    22:    getHelper().disposeWindow( dialog, this );  
}
```



# 6. Fazit / Grenzen

Was kann jfcUnit nicht?

- jfcUnit ist gut zum Test von Swing-Funktionalitäten die auf Nutzerinteraktion beruhen
- Es erweitert JUnit um Möglichkeit die Oberfläche zu testen
- => Testüberdeckung einer Applikation wird besser
- Nachteil: relativ geringe Geschwindigkeit

# Quellen

- <http://sourceforge.net/projects/jfcunit/>
- <http://jfcunit.sourceforge.net/>
- [www.dpunkt.de/ leseproben/3-89864-150-3/korrigiertes\\_Kapitel\\_13\\_des\\_Nachdrucks.pdf](http://www.dpunkt.de/leseproben/3-89864-150-3/korrigiertes_Kapitel_13_des_Nachdrucks.pdf)
- Mitgelieferte Beispiele und Dokumentationen zu jfcUnit 2.05