

•
•
•
•
•
•
•
•
•
•
•

JUnit Extensions

Version 1.0
(Juni 2001)

präsentiert von
Manfred Duchrow

Daedalos International AG, Copyright © 2001





Unit Testing mit JUnit

Was ist eigentlich so gut am Unit-Testing ?

- Vertrauen in den entwickelten Code
- Automatisierte Tests
- Wiederholbare Tests

Was ist eigentlich so gut an JUnit ?

- Leichtgewichtig
- Schnell zu verstehen
- Defacto Standard für Unit-Tests



-
-
-
-
-
-
-

Erweiterungen zu JUnit

Warum Erweiterungen ?

- Zusätzliche Test-Bereiche bei Client/Server-Anwendungen (HTTP, Servlets, JSPs, EJBs, ...)
- Einbeziehung externer Ressourcen (Datenbank, LDAP, Dateien, Programme, ...)
- Integration externer Test- und QA Werkzeuge

Ergänzende Frameworks

- HttpUnit
- Cactus
- JXUnit
- Daedalus JUnit Extensions
- ...

-
-
-
-
-
-
-
-



Daedalos JUnit Extensions

- Test-Ressourcen
- Einbindung externer Programme
- Vereinfachte Datenbank-Abfragen





Test-Ressourcen...

- Test-Ressourcen sind Komponenten, die für die Ausführbarkeit von Testfällen für bestimmte Klassen/Methoden notwendig sind
- Test-Ressourcen werden nicht nur von einem einzelnen Testfall genutzt, sondern von einer ganzen Gruppe von Testfällen
- Test-Ressourcen stellen in der Regel Verbindungen zu externen System-Komponenten dar, deren Initialisierung möglichst nur ein Mal durchgeführt werden sollte





...Test-Ressourcen

- Ein wichtiger Vorteil von TestRessourcen ist, dass sie über den gesamten Testzeitraum zur Verfügung stehen (solange der TestRunner aktiv ist), nicht nur für einen Testdurchlauf. So kann ein- und derselbe Unit-Test z.B. nach Korrekturen am Source-Code x-mal wiederholt werden, ohne dass jedesmal die benötigten TestRessourcen neu initialisiert werden müssen.
- Beispiele für Test-Ressourcen
 - Datenbanken
 - LDAP
 - P2P Socket-Connection





Erstellen von Test-Ressourcen

- Eine Unterklasse von ***com.daedalos.junit.TestResource*** erstellen
z.B.: *com.xyz.supplier.SupplierDBTestResource*
- Dort die Methoden implementieren
start()
stop()
- In der Datei TestResources.cfg die neue Test-Ressource bekannt machen (kann auch über den TestResourceBrowser geschehen)
z.B.:
SupplierDatabase=com.xyz.supplier.SupplierDBTestResource



Verwenden von Test-Ressourcen...

- In einem TestCase kann über die Test-Resource-Factory auf alle (in TestResource.cfg) definierten Ressourcen zugegriffen werden.

Beispiel:

...

```
SupplierDBTestResource supplierDB = null ;
```

```
supplierDB =
```

```
    (SupplierDBTestResource)TestResourceFactory.getTestResource("SupplierDatabase");
```

...

- Zur Ausführung der Tests muss statt dem *junit.swingui.TestRunner* der *com.daedalos.junit.gui.TRTestRunner* verwendet werden.



...Verwenden von Test-Ressourcen

- Auch mit den Original-TestRunner-Klassen können TestRessourcen eingesetzt werden. Über die folgenden Methoden in TestResourceFactory hat man Kontrolle über das Starten und Stoppen derselben:

static boolean startTestResource(String name)

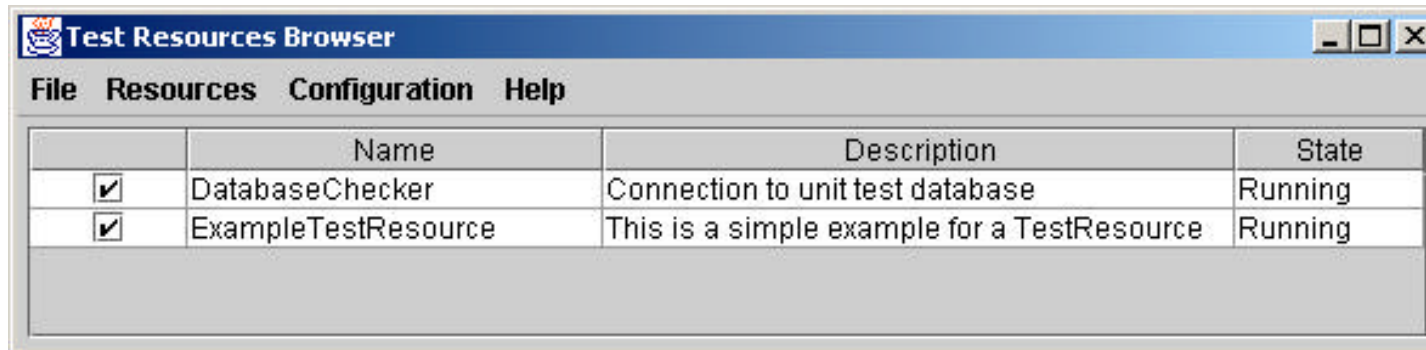
static boolean stopTestResource(String name)

- Alle selbst entwickelten Test-Resource-Klassen müssen in der Datei 'junit/runner/excluded.properties' eingetragen werden, damit sie bei einem erneuten Start der Tests nicht durch den JUnit eigenen ClassLoader nochmals geladen werden!
- Über diverse Konfigurationsdateien können die Test-Ressourcen und der Test-Ressourcen-Browser in ihrem Verhalten angepasst werden. Z.B. kann eingestellt werden, ob beim Start des Test-Runners automatisch auch gleich die Test-Ressourcen gestartet werden sollen.



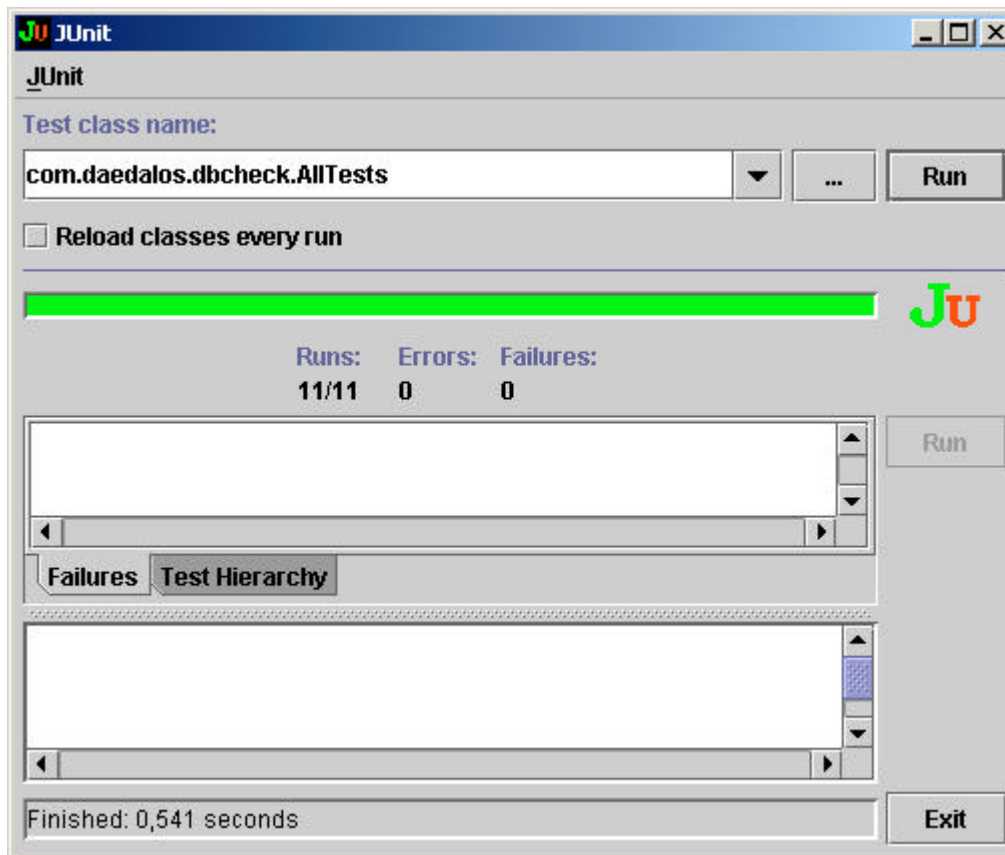
Test-Ressourcen-Browser

- Der Test-Ressourcen-Browser dient der Verwaltung der Test-Ressourcen
- Er kann sowohl aus dem JUnit-GUI als auch Standalone aufgerufen werden.



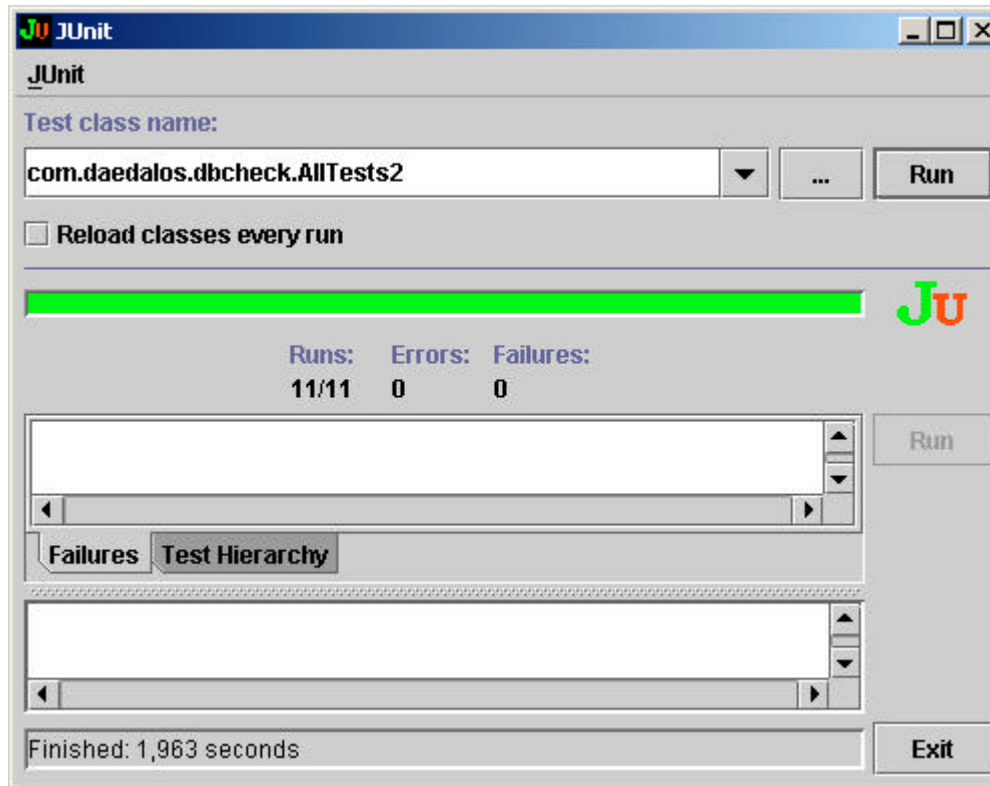
- Anlegen / Löschen von Test-Ressourcen
- Starten / Stoppen von Test-Ressourcen

Beispiel mit Test-Ressource



Dauer: 0,54 Sekunden

Beispiel ohne Test-Ressource



Dauer: 1,96 Sekunden



Einbindung externer Programme

- Für bestimmte Tests kann der Aufruf anderer Programme und die Auswertung deren Return-Codes sehr hilfreich sein.

Beispiel:

Mit **grep** überprüfen, ob eine Datei eine bestimmte Zeichenkette enthält, die durch den Test erstellt wurde

- Spezielle Unterstützung für die Integration von externen Test- und QA-Tools wie TestMentor, jLint, RecJava etc.



Aufruf externer Programme

- Grundsätzlich muss eine Unterklasse von ***com.daedalos.junit.ExternalProgramTestCase*** anstatt von ***junit.framework.TestCase*** angelegt werden.
- In der Test-Methode definiert man dann mittels der Methode ***setCommand(cmd)*** den Aufruf des externen Programms.
- Eine Überprüfung des Rückgabewertes übernimmt automatisch das Framework. Eine ***assert()***-Methode ist hier nicht mehr notwendig.
- Falls ein anderer Rückgabewert als 0 erwartet wird, kann dieser in der Test-Methode durch ***setExpectedResult()*** festgelegt werden.
- Die stdout-Ausgabe des externen Programmes kann mittels der Methode ***setOutputFilename()*** in eine Datei umgeleitet werden.

-
-
-
-
-
-
-

Beispieltest mit externem Programm

```
public void testExport()  
{  
    ImportExportProvider exporter = null ;  
    String inspectedObject       = "Whitesnake" ;  
    File filename                 = "\\test1.xml" ;  
    ObjectSpy spy                 = new ObjectSpy( inspectedObject ) ;  
  
    try { new File(filename).delete() ; } catch ( Exception ex ) {} ;  
    exporter = new XMLObjectExporter() ;  
    this.assertTrue( exporter.export( spy, filename ) ) ;  
    this.setCommand( "grep " + inspectedObject + " \"" + filename + "\"" );  
}
```



Datenbanküberprüfung

- Oft führen die zu testenden Programmteile Änderungen an einer relationalen Datenbank durch, die dann im Unit-Test verifiziert werden müssen.
- Mit Hilfe der Klasse ***com.daedalos.DatabaseChecker*** können manche der dazu notwendigen Datenbank-Abfragen vereinfacht werden
- Durch diese Klasse werden JDBC und SQL aus den TestCase-Klassen herausgehalten
- Stattdessen erfolgen die Abfragen über sehr einfache Java-Methoden
- Der DatabaseChecker ist ideal, um mittels einer Test-Resource eingebunden zu werden



API der Datenbanküberprüfung

- *Constructor*. **DatabaseChecker**(java.sql.Connection conn)
- **boolean tableExists**(String tableName)
- **long rowCount**(String tableName)
- **long rowCount**(String tableName, String whereClause)
- **boolean rowExists**(String tableName, String whereClause)
- **boolean rowsExist**(String tableName, String whereClause)
- **String getStringColumn**(String tableName, String columnName, String whereClause)
- **int getIntColumn**(String tableName, String columnName, String whereClause)

-
-
-
-
-
-
-

Beispiel der Datenbanküberprüfung

```
public void test_InsertPerson()
    throws SQLException
{
    Person person = new Person( "ID0928", "Fred", "Feuerstein" );
    databaseFrameWork.insert( person );
    assertTrue( dbChecker().rowExists( "PERSONS", "oid='ID0928'" ) );
    assertEquals( "Fred", dbChecker().getStringColumn( "PERSONS",
        "first_name", "last_name='Feuerstein'" ) );
}
```



Nutzungsrechte und Feedback

- Die Daedalos JUnit Extensions stehen kostenlos zum Download zur Verfügung
- Der Source-Code ist enthalten
- Freie Nutzung, soweit die Software nicht zu einer unmittelbaren Gewinnerzielung verwendet wird
- Anregungen zu Verbesserungen und Erweiterungen nehmen wir gerne entgegen





Referenzen

- JUnit Homepage:
<http://junit.sourceforge.net>
- HTTPUnit:
<http://httpunit.sourceforge.net>
- JXUnit:
<http://jxunit.sourceforge.net>
- Cactus:
<http://jakarta.apache.org/commons/cactus>
- Daedalus JUnit Extensions:
<http://www.daedalus.com/DE/djux>



-
-
-
-
-
-
-

Kontakt

Daedalos International AG

Ruhrtal 5

D-58456 Witten

Tel.: +49 (0)2302 979-0

Fax: +49 (0)2302 979-199

E-Mail: info@daedalos.com

Web: <http://www.daedalos.com>

Manfred Duchrow

Tel.: +49 (0)7333 210992

Fax: +49 (0)7333 210993

E-Mail:

manfred.duchrow@daedalos.com

Daedalos JUnit Extensions:

jens-uwe.pipka@daedalos.com

-
-
-
-
-
-
-
-

