Testing Tools

by

Dr. Günter Kolousek

JUnit 4

- `http://www.junit.org`
- Prinzip
  - verwendet Annotations
  - nimmt an, dass Methoden in beliebiger Reihenfolge aufgerufen werden können
    - $\leadsto$ Tests sollten nicht von anderen Tests abhängen

Beispiel 1

```java
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

public class TestCalculations {
    private Calculator calc;

    @Before
    public void setUp() {
        calc = new Calculator();
    }

    @Test
    public void test_empty() {
        assertEquals(calc.add(1, 1), 2);
        // wenn nicht gleich, dann -> AssertionError
    }
```

org.junit.Assert.*

- `assertEquals(long expected, long actual)`
  - `assertEquals(String message, long expected, long actual)`
  - ... alle anderen ebenfalls mit "message"
- `assertEquals(double expected, double actual, double delta)`
- `assertEquals(Object expected, Object actual)`
- `assertSame(Object expected, Object actual)`
- `assertNull(Object obj)` bzw. `assertNotNull(Object obj)`
- `assertTrue(boolean condition)` bzw. `assertFalse(boolean condition)`
- `assertArrayEquals(.,.)` mit verschiedenen Arten von Arrays
- `fail()`

Beispiel 2

```java
import static org.junit.Assert.*;
import org.junit.*;
import java.io.*;  // Pfui

public class TestFile {
    @Before
    public void setUp() {
        // create file
    }
    @After
    public void tearDown() {
        // remove file
    }
    // Test schlaegt fehl, wenn IOException geworfen wir
    @Test
    public void test_read() throws IOException {
        // read the contents of the file
    }
```

Beispiel 3

```java
import static org.junit.Assert.*;
import org.junit.*;
import java.io.*;  // Pfui

public class TestFile {
    @BeforeClass
    public void setUpOnce() throws Exception {
        // connect to database
    }
    @AfterClass
    public void tearDownOnce() {
        // close connection to database
    }
    @Test
    public void test_select() {
        // query the database
    }
```

Organisation der Tests

- wohin mit den Testklassen im Dateisystem?
  - gemeinsam mit den zu testenden Klassen
  - getrennt (empfohlen), z.B.:

```
/myproject/
    src/
        calculator/
            UPNCalculator.java
    tests/
        calculator/
            TestUPNCalculator.java
```

- CLASSPATH setzen

```
export CLASSPATH=.../junit.jar:~/myproject/classes
```

- Testrunner ausführen:

```
java org.junit.runner.JUnitCore TestCalculations
```

pytest – Überblick

- `unittest` analog zu JUnit
- aber `py.test` ist ”more pythonic”
- keine Klassen notwendig
  - aber möglich
  - $\leadsto$ Funktionen
- `py.test` startet im aktuellen Verzeichnis
  - sucht nach Modulen, die mit `test_` beginnen.
  - jede Funktion, die mit `test` beginnt

pytest – Beispiel

```
def test_zero():
    assert 0 == 0.0
```

- Keine Markierung notwendig
- Keine speziellen Methoden nötig

Loggen mit Java

- Anwendung erzeugt einen Logger, z.B.

```
import java.util.logging.Logger;
class Test {
  private final static Logger LOGGER =
    Logger.getLogger(Test.class.getName());
```

- Logger hat einen Namen in Punktschreibweise
  - oft vollständiger Name, z.B.: ”at.ac.htlwrn.Manager”
  - formen Hierarchie: default-mäßig an Eltern-Logger
    - siehe später

Level

- `java.util.logging.Level`
  - gibt die Bedeutung der Lognachricht an
  - Hierarchie der Level
    1. SEVERE (höchster Level)
    2. WARNING
    3. INFO
    4. CONFIG
    5. FINE
    6. FINER
    7. FINEST (niedrigster Level)
- ”Level” einer `Logger` - Instanz kann gesetzt werden
  - `LOGGER.setLevel(Level.INFO)` $\rightarrow$ Nachrichten mit Level SEVERE, WARNING und INFO geloggt
- Loggen einer Nachricht mit einem bestimmten Level
  - `LOGGER.info(”das ist eine info”);`

Methoden zum Loggen

- Je Level eine Methode (siehe vorhergehende Folie)
- `log(Level level, String msg)`
- `log(Level level, String msg, Object param)`
  - `log.log(Level.INFO, ”hello {0}!”, ”World”);`
- `log(Level level, String msg, Object[] params)`
  - mit mehreren Parametern
- `logp(Level level, String sourceClass, String method, String msg)`
  - auch mit einem oder mehreren Parametern
- `entering(String sourceClass, String method)`
- `leaving(String sourceClass, String method)`

Hierarchie der Logger

- Hierarchie gegeben durch Punkt-notierten Namen
- Beispiel: `Logger.getLogger("com.uberdev.db")`
  - `Logger.getLogger("")` erzeugt und liefert die Wurzel
  - `Logger.getLogger("com")` ... das Kind der Wurzel
  - `Logger.getLogger("com.uberdev")` ...
  - `Logger.getLogger("com.uberdev.db")` liefert das Blatt
- Aber:
  - diese werden nicht automatisch angelegt:

    ```
    Logger log = Logger.getLogger("com.uberdev.db");
    Logger parent = log.getParent(); // liefert Logge
    ```

  - D.h. u.U. diese vorerst mittels `getLogger` anlegen!
- Sinn dieser Hierarchie?
  - Log-Nachrichten werden prinzipiell nach oben weitergereicht.
  - "com.uberdev.db" $\rightsquigarrow$ "com.uberdev" $\rightsquigarrow$ "com" $\rightsquigarrow$ ""

Hierarchie und Level

```
Logger log1 = Logger.getLogger(”com”);
Logger log2 = Logger.getLogger(”com.uberdev”);
Logger log3 = Logger.getLogger(”com.uberdev.db”);

log2.setLevel(Level.WARNING);

log1.info(”hallo log1”);  // wird geloggt
log2.info(”hallo log2”);  // NICHT (auch nicht an log1 w
// auch NICHT (hat keinen Level -> erbt von log2)
log3.info(”hallo log3”);

log3.setLevel(Level.INFO);  // Level setzen
log3.info(”hallo hallo log3”);  // wird jetzt geloggt!

//log3.warning(”warning”);
```

Handler

- Jedem Logger kann ein oder mehrere Handler zugewiesen sein
  - kein Handler $\rightsquigarrow$ keine Ausgabe von diesem Logger
- Nur Root-Logger hat default-mäßig eine Instanz der Klasse `ConsoleHandler`!
  - vorhergehendes Bsp.: ”hallo hallo log3” wird Handler von Root-Logger verwendet!
- Wird eine Nachricht von einem Logger akzeptiert, dann wird die Nachricht an seine Handler-Objekte und alle obenliegende Handler-Objekte zur Behandlung weitergegeben.
  - Filter und Level werden nicht mehr überprüft!!!
- Level werden aber ”vererbt”!
- Filter werden nicht ”vererbt”!

Handler – 2

- `ConsoleHandler`: Ausgabe auf `stderr`
- `StreamHandler`
  - Basisklasse von `ConsoleHandler`
  - z.B. `new StreamHandler(outputStream, formatter);`
- `FileHandler`
  - Unterklasse von `StreamHandler`
  - z.B. `new FileHandler("out.log", true)`
  - zweiter Parameter optional, `true` ... append
  - konkreter Filename oder Pattern (siehe Doku)
- `SocketHandler`
  - Unterklasse von `StreamHandler`
  - z.B.: `new SocketHandler("log.htlwrn.ac.at", 9999);`

Filter

- jeder Logger kann einen Filter zugeordnet haben
- passiert die Log-Nachricht den Filter $\leadsto$ Verarbeitung
  - kein Filter $\leadsto$ ebenfalls Verarbeitung
- wenn Nachricht an Eltern-Logger, dann kein Durchlauf des Filters des Eltern-Logger (einmal passiert, dann ok)

Filter – 2

```
Logger log1 = Logger.getLogger(”com.ueberdev”);
Logger log2 = Logger.getLogger(”com.ueberdev.db”);
log1.setFilter(new Filter() {
        public boolean isLoggable(LogRecord rec) {
            return false; }
    });
log2.setFilter(new Filter() {
        public boolean isLoggable(LogRecord rec) {
            return true; }
    });
log1.info(”hallo log1”);  // verschwindet im Nirvana
// log2, dann log1, dann root-Logger!
// keine Handler fuer log2 und log1!
log2.info(”hallo log2”);
// nur eine Ausgabe!
```

Formatter

- Jeder Handler hat einen Formatter
- `handler.setFormatter(...)`
- `LogRecord` wird mittels Formatter auf Ausgabe geschrieben
- `Formatter`
  - `SimpleFormatter`
  - mittels Format-String
- `XMLFormatter`

Formatter – Beispiel

```
import java.text.MessageFormat;
import java.util.Date;
import java.util.logging.Formatter;
import java.util.logging.LogRecord;

public class MessageFormatFormatter extends Formatter {
  private static final MessageFormat messageFormat =
      new MessageFormat("{0}[{1}|{2}|{3,date,H:mm:ss}]:
  public MessageFormatFormatter() {
      super(); }
  @Override public String format(LogRecord record) {
      Object[] arguments = new Object[6];
        arguments[0] = record.getLoggerName();
        arguments[1] = record.getLevel();
        arguments[2] = Thread.currentThread().getName();
        arguments[3] = new Date(record.getMillis());
        arguments[4] = record.getMessage();
        return messageFormat.format(arguments);
  } }
```