Java

JUnit Tests

Vincent Gerber, Tilman Hinnerichs

Java Kurs

12. Juli 2018

Overview

1. Introduction to JUnit
   - Why to use JUnit?
   - What to keep in mind
   - How to use and build test classes

Why to use JUnit?

We would like to make sure that the following code is working properly:

```
1  public class Book{
2      public Book(Author author){...}
3
4      private int page = 1;
5      private Author author;
6
7      public void turnOver(int pages){
8          page+=pages;
9      }
10
11     public void setAuthor(Author author){
12         this.author = author;
13     }
14
15     public Author getAuthor(){
16         return author;
17     }
18 }
19
```

Why to use JUnit?

We modify the code as we learned last time:

```
1   public class Book{
2       public Book(Author author){...}
3
4       private int page = 0;
5       private Author author;
6
7       public void turnOver(int pages){
8           if(page+pages<1){throw new InvalidArgumentException();}
9           page+=pages;
10      }
11
12      public void setAuthor(Author author){
13          if(author==null){throw new NullPointerException();}
14          this.author = author;
15      }
16
17      public Author getAuthor(){
18          return author;
19      }
20  }
21
```

Why to use JUnit?

But how can we make sure that the implementation got all the methods and functionality we would like it to have (Thinking about tasks given by a client)?

Why to use JUnit?

But how can we make sure that the implementation got all the methods and functionality we would like it to have (Thinking about tasks given by a client)?

That is why we would like to **test** what we have got!

Why to use JUnit?

For testing we need the following:

Test classes/code	This will contain methods to test our code. It will have to be initializable by the default constructor
Test methods	Mathods which are marked as test methods. They will have to be public, should therefor not take parameters and should return void.
Test case	A test case is some given test values with which one would like
Test framework	We will be using JUnit 4 as test framework

What to keep in mind

- Use an extra test class seperated from the rest of the code
- You can name you test class how you want to, but *Test is common
- Your cases should test atomary parts of your code **AND** how those parts are working together
- Do not even try to test your compiler with this

Assert

Importing org.junit.Assert.* we can use the assert method:

assertEqual(Object exp, Object act)	Tests whether two Objects are equal
assertNotNull(Object object)	Object is not null?
assertSame(Object exp, Object act)	Objects same
assertNotSame(Object unexp, Object act)	Objects not identical
assertTrue(boolean condition)	Tests condition
assertFalse(boolean condition)	Tests condition

Annotations

We can also put several annotations in front of our tests (JUnit 4):

@Test	Mark the following method as test case
@Test(expected=...Exception.class)	Mark that exception is espected
@Before	Will be executed **before** every test method
@After	Will be executed **after** every test method

A simple test class

```
1  import java.util.*;
2  import org.junit.*;
3
4  public class BookTest{
5      //NO CONSTRUCTOR FOR TEST CLASSES!
6      @Test
7      public void turnOverTest(){
8          Author douglas = new Author("Douglas Adams");
9          Book thgttg = new Book(douglas);
10         book.turnOver(5);
11         assertEquals(6,book.getPage);
12     }
13 }
14
```

Using the magic of JUnit

```
1   public class BookTest{
2       private Book book;
3       private Author author
4       @Before
5       public void setUp(){
6           autor = new Author("Douglas Adams");
7           book = new Book(douglas);
8       }
9       @Test
10      public void turnOverTest(){
11          book.turnOver(5);
12          assertEquals(6,book.getPage);
13      }
14  }
15
```

Testing to throw exceptions

```
1   public class BookTest{
2       private Book book;
3       private Author author
4
5       @Before
6       public void setUp(){
7           autor = new Author("Douglas Adams");
8           book = new Book(douglas);
9       }
10
11      @Test (expected = InvalidArgumentException.class)
12      public void turnOverTest(){
13          book.turnOver(-5);
14      }
15  }
16
```