Übung 9 – Unit testing & mock objects

Methoden des Software Engineering
WS 2012/13

Christian Kroiß

Outline

Today

- **Introduction in JUnit**
- **Using EasyMock for Mock Object Testing**

JUnit

- JUnit (www.junit.org): framework for unit testing in Java.
- 4 main Annotations realize test cycle: **initialize fixture**, **test**, **clean up**
  - **@Test:** declares method as a test case
  - **@Before / @After :** declares that a method should be called **before/after each execution** of a test method
  - **@BeforeClass** / **@AfterClass**: declares that a method should be called **once before/after any** of the test methods in the class
- Set of assertion methods: assertEquals(…), assertNotNull(…)

Writing Assertions

```
if (obtainedResult == null ||
      !expectedResult.equals(obtainedResult))
      throw new MyTestThrowable("Bad output for #
            attempt");
```

With JUnit, you can code

```
import static org.junit.Assert.*;
...
assertEquals("Bad output for # attempt",
      expectedResults, obtainedResults);
```

Scaffolding

Code developed to facilitate testing is called *scaffolding*, by analogy to the temporary structures erected around a building during construction or maintenance.

- **Test drivers:** substituting for a main or calling program
- **Test stubs:** substituting for functionality called or used by the software under test
- **Test harnesses:** substituting for parts of the deployment environment

Integration Testing Strategies

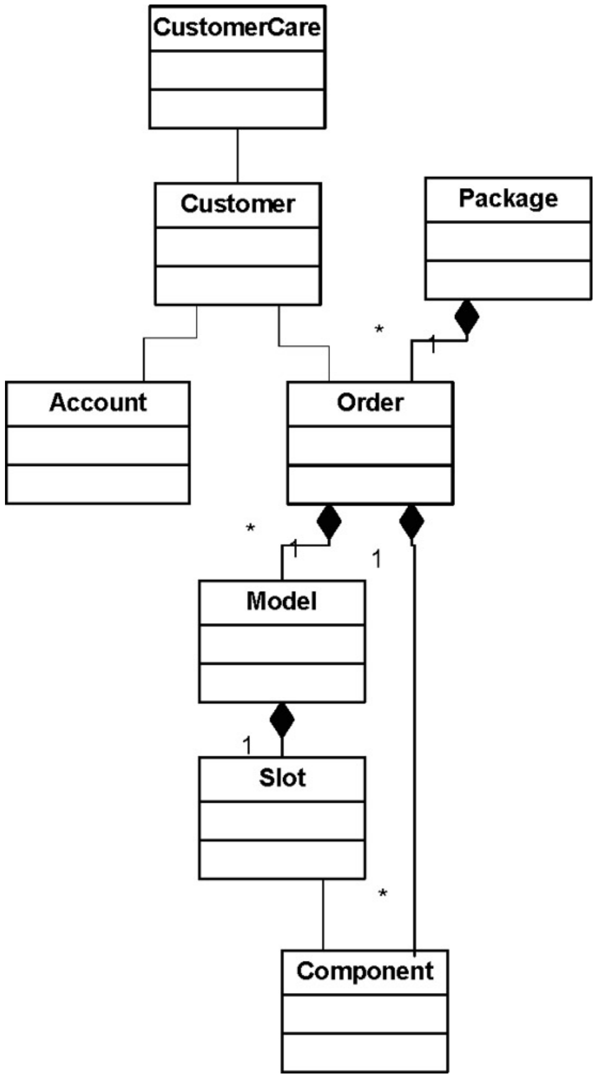
Big bang

- Avoid cost of scaffolding by waiting until all modules are integrated
- →effectively system testing
- **Problem:** losses in observability, diagnosability, and feedback
- less a rational strategy than an attempt to recover from a lack of planning

Integration Testing Strategies (2)

Structural integration: top down/bottom up

- use/include relation: The topmost modules are not used or included in any other module, while the bottom-most modules do not include or use other modules.
- **Top down:** replace ***used*** modules with ***stubs***
- **Bottom up:** replace ***using*** modules with ***drivers***



Test Doubles

Test Doubles: any kind of pretend object used in place of a real object for testing purposes

- **Dummy** objects are passed around but never actually used. Usually they are just used to fill parameter lists.
- **Fake** objects actually have working implementations, but usually take some shortcut which makes them not suitable for production (an in memory database is a good example).

LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

Test Doubles (2)

- **Stubs** provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test. Stubs may also record information about calls, such as an email gateway stub that remembers the messages it 'sent', or maybe only how many messages it 'sent'.
- **Mocks** are objects pre-programmed with expectations which form a specification of the calls they are expected to receive.

EasyMock

- The Java framework EasyMock (easymock.org) allows to use a macro-recording-like mechanism to define the expected behavior of a Mock Object.

Expects method call with specified parameters

```
expect(warehouse.hasInventory("iPad‟,25).andReturn(
      true);
warehouse.remove("iPad", 25);
replay(warehouse);
order.fill(warehouse);
verify(warehouse);
```

Defines what the mock object should return

Executes the tested functionality

Activates the “recorded” behavior

Checks if all expected method calls happened

Behavior Verification

Mocks objects support **behavior verification:**

"Each test specifies not only how the client of the SUT interacts with it during the exercise SUT phase of the test, but also how the SUT interacts with the components on which it should depend. This ensures that the SUT really is behaving as specified rather than just ending up in the correct post-exercise state." [6]

Links & Further Reading

1. JUnit: http://junit.org
2. EasyMock: http://easymock.org/
3. EasyMock Documentation: http://easymock.org/EasyMock3_0_Documentation.html
4. Tim Mackinnon, Steve Freeman, Philip Craig. Endo Testing: Unit Testing with Mock Objects. 2000. http://connextra.com/aboutUs/mockobjects.pdf
5. Tutorial: JUnit 4 & Eclipse: http://www.vogella.de/articles/JUnit/article.html
6. Gerard Meszaros. xUnit Test Patterns. Addison Wesley, 2007