



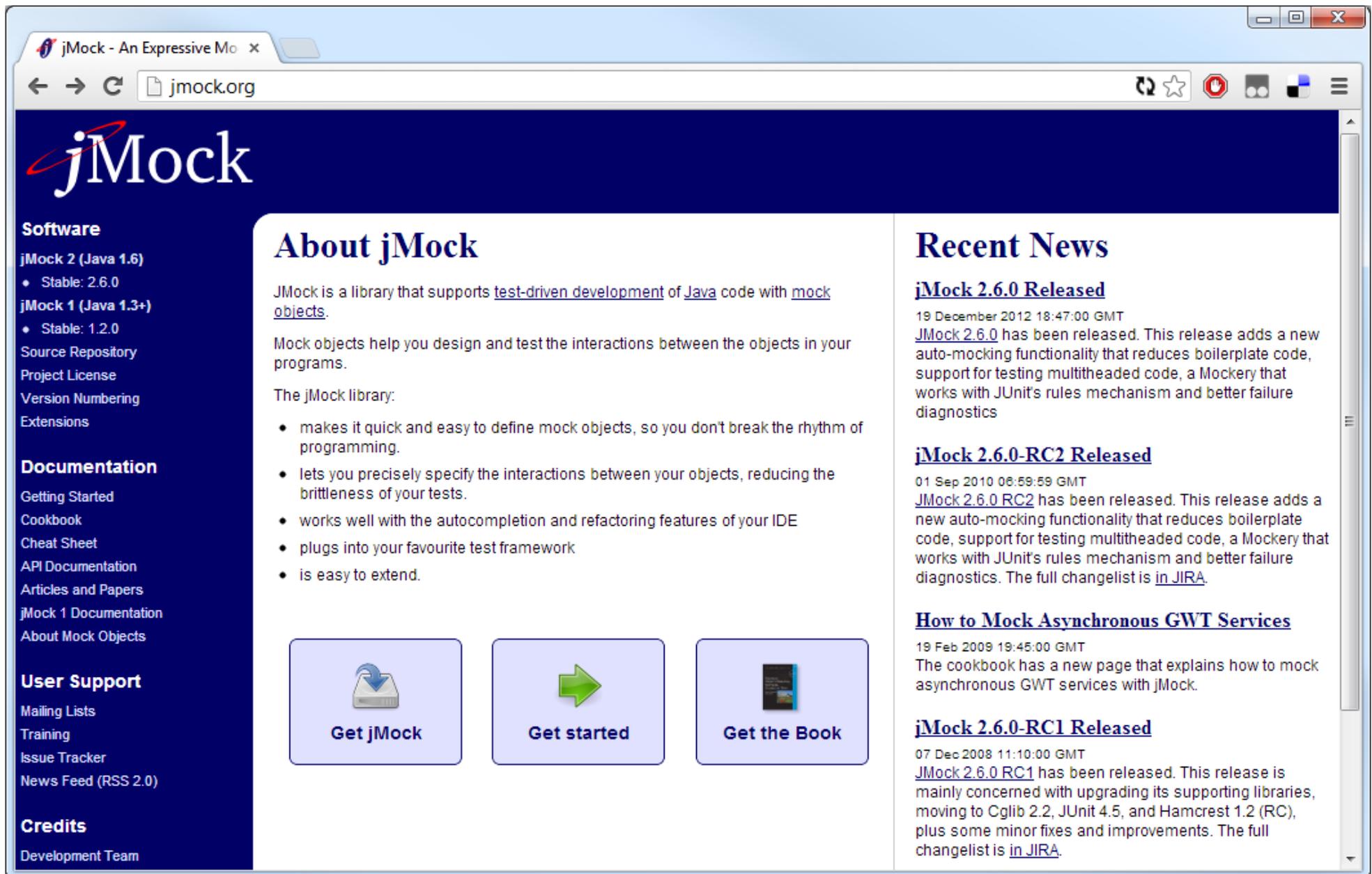
# **Mocking Libraries Shootout**

# **Warum ist Mocking interessant?**

- **Ermöglicht Tests ohne echte Testumgebung**
- **Schnelle Tests durch gute Test-Isolation**
- **Kontrolle des indirekten Inputs**
- **Simulation von Fehlern**
- **Überprüfung des indirekten Outputs/Verhaltens**
- **Überprüfung der Aufrufreihenfolge**

# **Die Kandidaten**

# Die Kandidaten



The image shows a screenshot of a web browser displaying the jMock website. The browser's address bar shows 'jmock.org'. The website has a dark blue header with the 'jMock' logo. The main content area is white and features a navigation sidebar on the left, a central 'About jMock' section, and a 'Recent News' section on the right. The 'About jMock' section includes a description of the library, its benefits, and three call-to-action buttons: 'Get jMock', 'Get started', and 'Get the Book'. The 'Recent News' section lists three recent releases: 'jMock 2.6.0 Released', 'jMock 2.6.0-RC2 Released', and 'jMock 2.6.0-RC1 Released', each with a date and a brief description of the changes.

**Software**

- jMock 2 (Java 1.6)
  - Stable: 2.6.0
- jMock 1 (Java 1.3+)
  - Stable: 1.2.0

Source Repository  
Project License  
Version Numbering  
Extensions

**Documentation**

Getting Started  
Cookbook  
Cheat Sheet  
API Documentation  
Articles and Papers  
jMock 1 Documentation  
About Mock Objects

**User Support**

Mailing Lists  
Training  
Issue Tracker  
News Feed (RSS 2.0)

**Credits**

Development Team

## About jMock

JMock is a library that supports [test-driven development](#) of Java code with [mock objects](#).

Mock objects help you design and test the interactions between the objects in your programs.

The jMock library:

- makes it quick and easy to define mock objects, so you don't break the rhythm of programming.
- lets you precisely specify the interactions between your objects, reducing the brittleness of your tests.
- works well with the autocompletion and refactoring features of your IDE
- plugs into your favourite test framework
- is easy to extend.

[Get jMock](#)   [Get started](#)   [Get the Book](#)

## Recent News

### [jMock 2.6.0 Released](#)

19 December 2012 18:47:00 GMT  
[JMock 2.6.0](#) has been released. This release adds a new auto-mocking functionality that reduces boilerplate code, support for testing multithreaded code, a Mockery that works with JUnit's rules mechanism and better failure diagnostics

### [jMock 2.6.0-RC2 Released](#)

01 Sep 2010 06:59:59 GMT  
[JMock 2.6.0 RC2](#) has been released. This release adds a new auto-mocking functionality that reduces boilerplate code, support for testing multithreaded code, a Mockery that works with JUnit's rules mechanism and better failure diagnostics. The full changelist is [in JIRA](#).

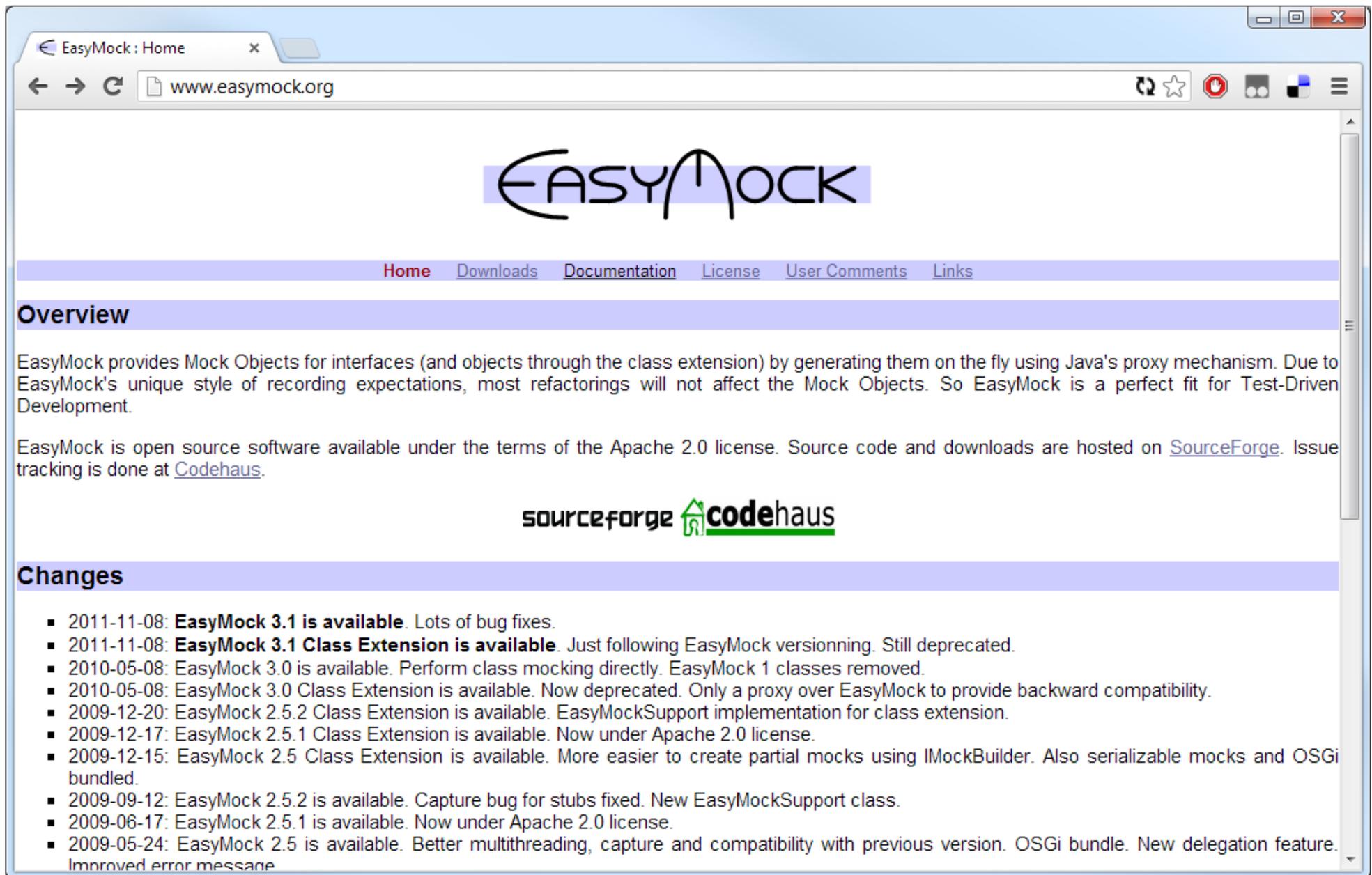
### [How to Mock Asynchronous GWT Services](#)

19 Feb 2009 19:45:00 GMT  
The cookbook has a new page that explains how to mock asynchronous GWT services with jMock.

### [jMock 2.6.0-RC1 Released](#)

07 Dec 2008 11:10:00 GMT  
[JMock 2.6.0 RC1](#) has been released. This release is mainly concerned with upgrading its supporting libraries, moving to Cglib 2.2, JUnit 4.5, and Hamcrest 1.2 (RC), plus some minor fixes and improvements. The full changelist is [in JIRA](#).

# Die Kandidaten



The screenshot shows a web browser window with the address bar displaying "www.easymock.org". The page features the "EASYMOCK" logo in a stylized font. Below the logo is a navigation menu with links for "Home", "Downloads", "Documentation", "License", "User Comments", and "Links". The "Overview" section describes EasyMock's functionality and its open-source status under the Apache 2.0 license. It also mentions that source code and downloads are hosted on SourceForge and issue tracking is done at Codehaus. The "Changes" section lists several updates and releases, including EasyMock 3.1, 3.0, 2.5.2, and 2.5.1, with details on bug fixes, licensing, and new features.

EasyMock : Home x

www.easymock.org

## EASYMOCK

[Home](#) [Downloads](#) [Documentation](#) [License](#) [User Comments](#) [Links](#)

### Overview

EasyMock provides Mock Objects for interfaces (and objects through the class extension) by generating them on the fly using Java's proxy mechanism. Due to EasyMock's unique style of recording expectations, most refactorings will not affect the Mock Objects. So EasyMock is a perfect fit for Test-Driven Development.

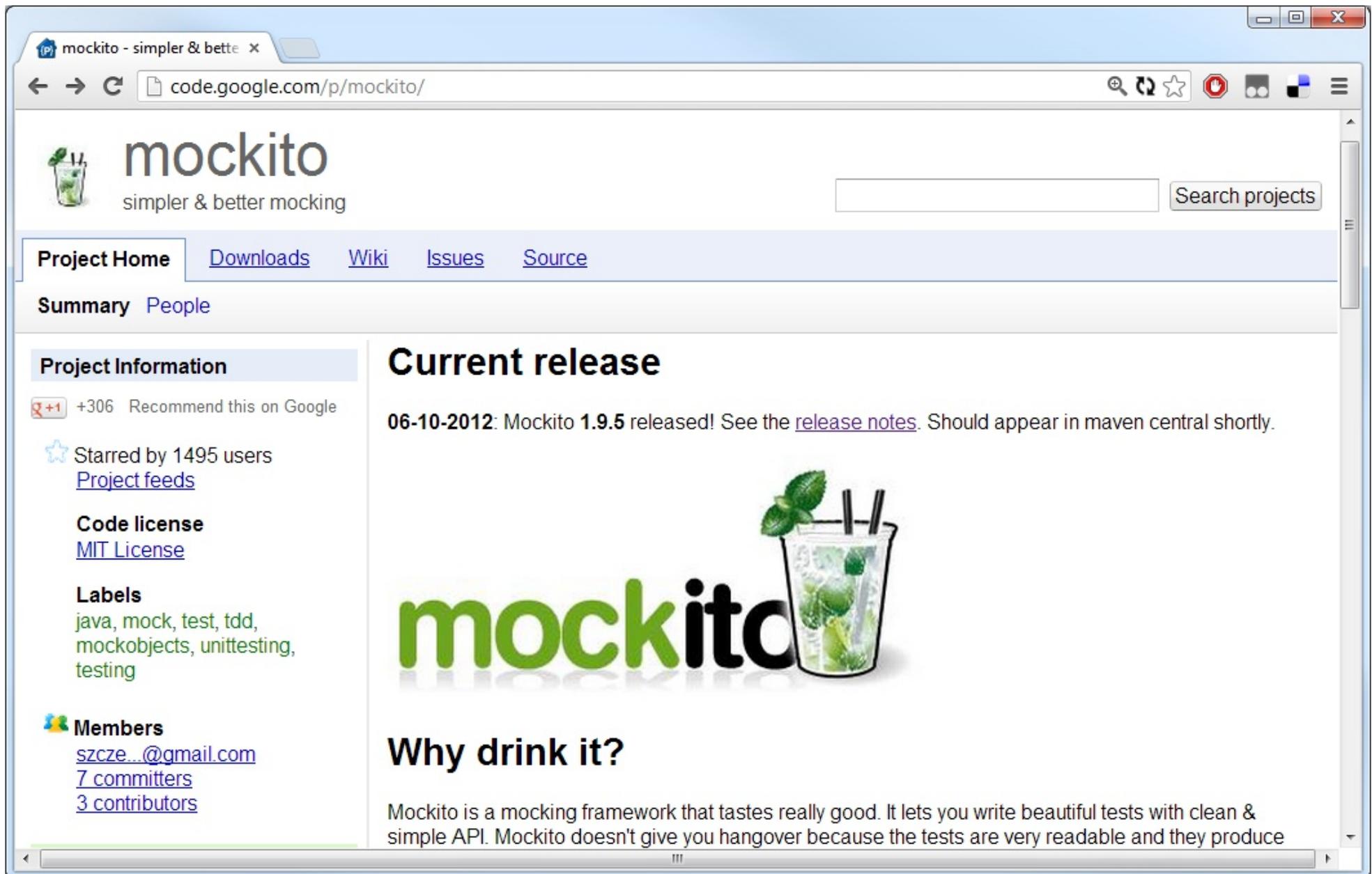
EasyMock is open source software available under the terms of the Apache 2.0 license. Source code and downloads are hosted on [SourceForge](#). Issue tracking is done at [Codehaus](#).

sourceforge  codehaus

### Changes

- 2011-11-08: **EasyMock 3.1 is available**. Lots of bug fixes.
- 2011-11-08: **EasyMock 3.1 Class Extension is available**. Just following EasyMock versioning. Still deprecated.
- 2010-05-08: EasyMock 3.0 is available. Perform class mocking directly. EasyMock 1 classes removed.
- 2010-05-08: EasyMock 3.0 Class Extension is available. Now deprecated. Only a proxy over EasyMock to provide backward compatibility.
- 2009-12-20: EasyMock 2.5.2 Class Extension is available. EasyMockSupport implementation for class extension.
- 2009-12-17: EasyMock 2.5.1 Class Extension is available. Now under Apache 2.0 license.
- 2009-12-15: EasyMock 2.5 Class Extension is available. More easier to create partial mocks using IMockBuilder. Also serializable mocks and OSGi bundled.
- 2009-09-12: EasyMock 2.5.2 is available. Capture bug for stubs fixed. New EasyMockSupport class.
- 2009-06-17: EasyMock 2.5.1 is available. Now under Apache 2.0 license.
- 2009-05-24: EasyMock 2.5 is available. Better multithreading, capture and compatibility with previous version. OSGi bundle. New delegation feature. Improved error message

# Die Kandidaten



The image shows a screenshot of a web browser displaying the Mockito project page on code.google.com. The browser's address bar shows the URL 'code.google.com/p/mockito/'. The page features the Mockito logo, which is a glass of mojito with a lime wedge and a straw, next to the text 'mockito' and the tagline 'simpler & better mocking'. Below the logo, there are navigation links for 'Project Home', 'Downloads', 'Wiki', 'Issues', and 'Source'. The 'Project Home' link is selected. The page is divided into two main sections: 'Project Information' on the left and 'Current release' on the right. The 'Project Information' section includes a 'Recommend this on Google' button, a star icon indicating it is starred by 1495 users, and links to 'Project feeds', 'Code license' (MIT License), and 'Labels' (java, mock, test, tdd, mockobjects, unittesting, testing). The 'Members' section lists '7 committers' and '3 contributors'. The 'Current release' section features a large heading, a date '06-10-2012', and a message that Mockito 1.9.5 has been released, with a link to 'release notes'. Below this, there is a large graphic of the Mockito logo and a glass of mojito. The 'Why drink it?' section contains a paragraph describing Mockito as a mocking framework that is easy to use and produces readable tests.

mockito - simpler & better x

code.google.com/p/mockito/

 **mockito**  
simpler & better mocking

Search projects

**Project Home** [Downloads](#) [Wiki](#) [Issues](#) [Source](#)

**Summary** [People](#)

**Project Information**

+306 Recommend this on Google

★ Starred by 1495 users  
[Project feeds](#)

**Code license**  
[MIT License](#)

**Labels**  
java, mock, test, tdd,  
mockobjects, unittesting,  
testing

**Members**  
[szcze...@gmail.com](#)  
[7 committers](#)  
[3 contributors](#)

**Current release**

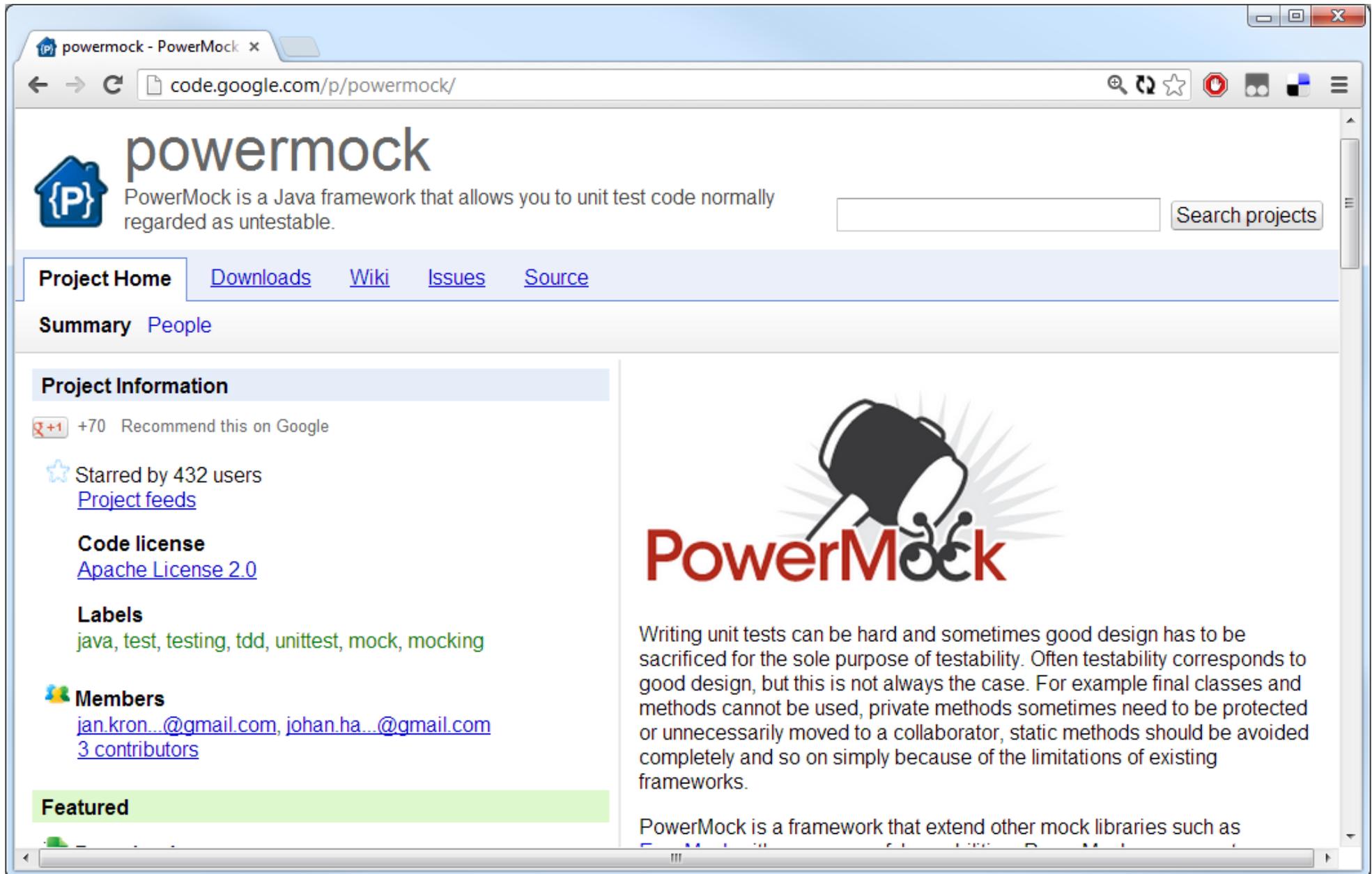
**06-10-2012:** Mockito 1.9.5 released! See the [release notes](#). Should appear in maven central shortly.



**Why drink it?**

Mockito is a mocking framework that tastes really good. It lets you write beautiful tests with clean & simple API. Mockito doesn't give you hangover because the tests are very readable and they produce

# Die Kandidaten



The screenshot shows a web browser window with the address bar containing `code.google.com/p/powermock/`. The page title is "powermock". Below the title, there is a description: "PowerMock is a Java framework that allows you to unit test code normally regarded as untestable." To the right of the description is a search box labeled "Search projects".

Navigation links include "Project Home", "Downloads", "Wiki", "Issues", and "Source". Below these are "Summary" and "People".

The "Project Information" section includes:

- +70 Recommend this on Google
- Stared by 432 users, with a link to "Project feeds"
- Code license: Apache License 2.0
- Labels: java, test, testing, tdd, unittest, mock, mocking
- Members: jan.kron...@gmail.com, johan.ha...@gmail.com, 3 contributors

The "Featured" section is partially visible at the bottom.

On the right side of the page, there is a large logo for "PowerMock" featuring a megaphone. Below the logo, there is a paragraph of text:

Writing unit tests can be hard and sometimes good design has to be sacrificed for the sole purpose of testability. Often testability corresponds to good design, but this is not always the case. For example final classes and methods cannot be used, private methods sometimes need to be protected or unnecessarily moved to a collaborator, static methods should be avoided completely and so on simply because of the limitations of existing frameworks.

Below this paragraph, the text "PowerMock is a framework that extend other mock libraries such as" is visible, followed by a list of other mock libraries.

# Aufgabe 0: Project Setup

jmack maven

**Web**

Bilder

Maps

Shopping

Mehr ▾

Suchoptionen

Ungefähr 103.000 Ergebnisse (0,26 Sekunden)

[jMock - Maven Configuration](#)

[jmack.org/maven.html](http://jmack.org/maven.html) - Diese Seite übersetzen

The **jMock** 2 jars are accessible via **Maven** 2 by declaring the following dependencies in your POM. All the required dependencies on **jMock** core and Hamcrest ...

# Aufgabe 0: Project Setup



## Software

jMock 2 (Java 1.6)

- Stable: 2.6.0

jMock 1 (Java 1.3+)

- Stable: 1.2.0

Source Repository

Project License

Version Numbering

Extensions

## Documentation

Getting Started

Cookbook

Cheat Sheet

API Documentation

Articles and Papers

jMock 1 Documentation

About Mock Objects

## User Support

Mailing Lists

Training

Issue Tracker

News Feed (RSS 2.0)

## Credits

Development Team

Project hosted by Github

Icons by the Tango Project

## Using jMock from Maven Builds

### jMock 2

The jMock 2 jars are accessible via Maven 2 by declaring the following dependencies in your POM. All the required dependencies on jMock core and Hamcrest will be included.

To use jMock 2.6.0 with JUnit 3:

```
<dependency>
  <groupId>org.jmock</groupId>
  <artifactId>jmock-junit3</artifactId>
  <version>2.6.0</version>
</dependency>
```

To use jMock 2.6.0 with JUnit 4:

```
<dependency>
  <groupId>org.jmock</groupId>
  <artifactId>jmock-junit4</artifactId>
  <version>2.6.0</version>
</dependency>
```

**Note:** From jMock 2.4.0 onwards, the default JUnit 4 Maven dependency is junit:junit-dep. This requires Maven's Surefire to be configured to use it instead of the standard junit:junit4. Surefire 2.3.1 and 2.4 support the optional `<junitArtifactName>junit:junit-dep</junitArtifactName>` configuration element.

To use jMock 2.6.0 to mock classes:

```
<dependency>
  <groupId>org.jmock</groupId>
  <artifactId>jmock-legacy</artifactId>
  <version>2.6.0</version>
</dependency>
```

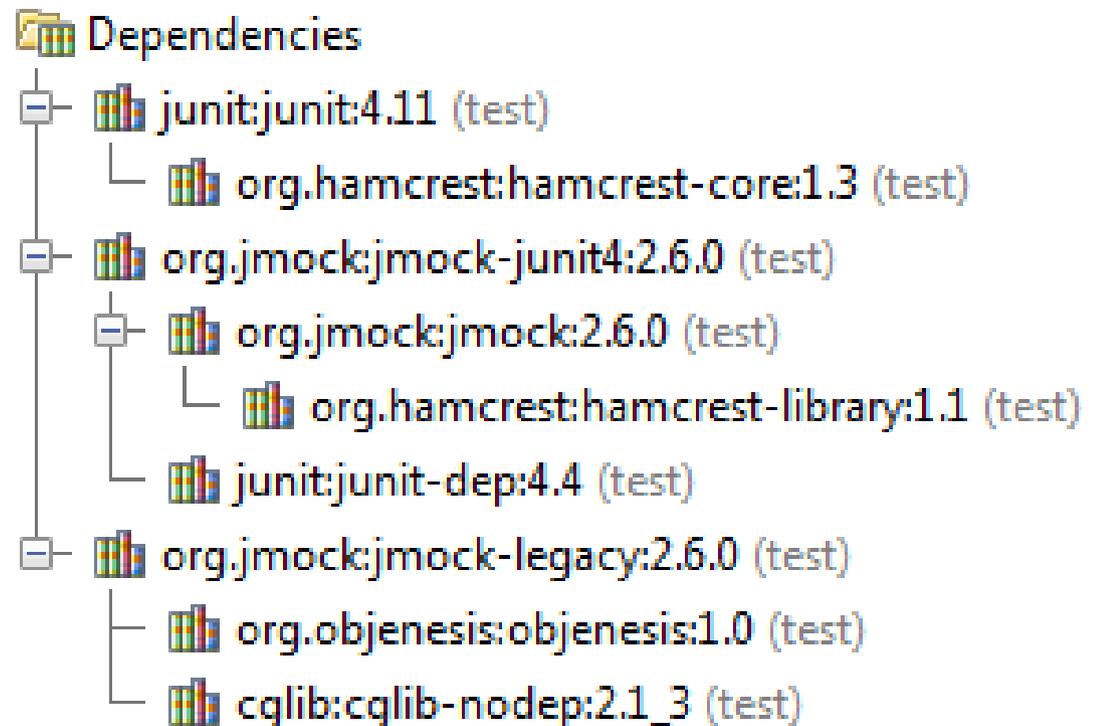
To use other versions of jMock, replace the contents of the version tags.

# Aufgabe 0: Project Setup

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.jmock</groupId>
  <artifactId>jmock-junit4</artifactId>
  <version>2.6.0</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.jmock</groupId>
  <artifactId>jmock-legacy</artifactId>
  <version>2.6.0</version>
  <scope>test</scope>
</dependency>
```

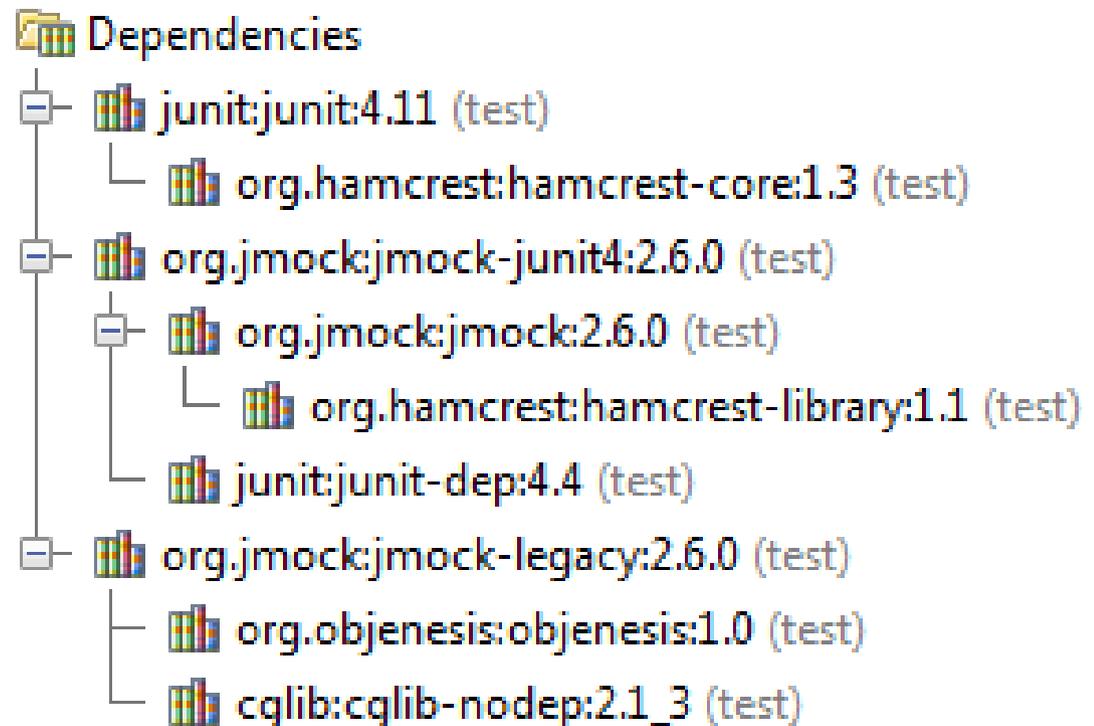
# Aufgabe 0: Project Setup

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.jmock</groupId>
  <artifactId>jmock-junit4</artifactId>
  <version>2.6.0</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.jmock</groupId>
  <artifactId>jmock-legacy</artifactId>
  <version>2.6.0</version>
  <scope>test</scope>
</dependency>
```



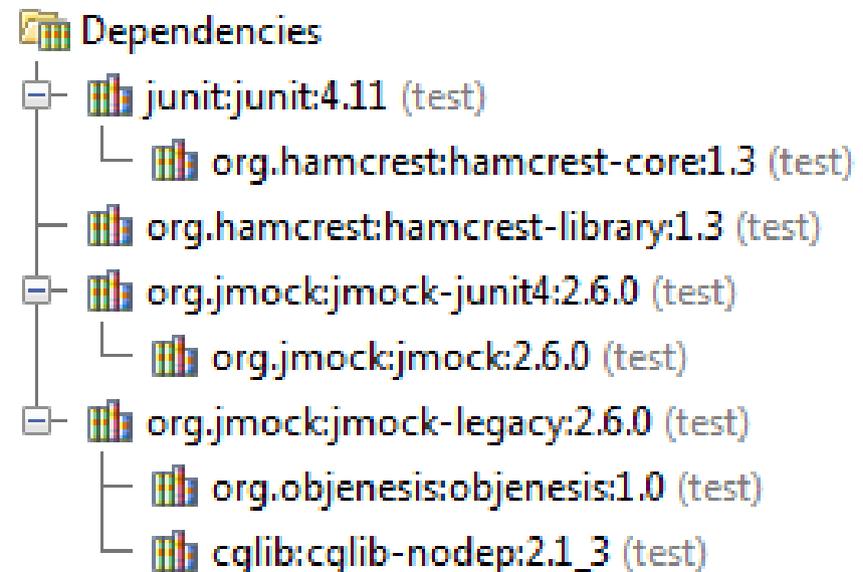
# Aufgabe 0: Project Setup

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.jmock</groupId>
  <artifactId>jmock-junit4</artifactId>
  <version>2.6.0</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.jmock</groupId>
  <artifactId>jmock-legacy</artifactId>
  <version>2.6.0</version>
  <scope>test</scope>
</dependency>
```



# Aufgabe 0: Project Setup

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.hamcrest</groupId>
  <artifactId>hamcrest-library</artifactId>
  <version>1.3</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.jmock</groupId>
  <artifactId>jmock-junit4</artifactId>
  <version>2.6.0</version>
  <scope>test</scope>
  <exclusions>
    <exclusion>
      <groupId>junit</groupId>
      <artifactId>junit-dep</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.jmock</groupId>
  <artifactId>jmock-legacy</artifactId>
  <version>2.6.0</version>
  <scope>test</scope>
</dependency>
```



# Aufgabe 0: Project Setup

easymock maven

**Web**

Bilder

Maps

Shopping

Mehr ▾

Suchoptionen

Ungefähr 176.000 Ergebnisse (0,31 Sekunden)

[EasyMock 3.0 Readme](#)

[www.easymock.org/EasyMock3\\_0\\_Documentat...](http://www.easymock.org/EasyMock3_0_Documentat...) - Diese Seite übersetzen

08.05.2010 – Installation. Using **Maven**. **EasyMock** is available in the **Maven** central repository. Just add the following dependency to your pom.xml: ...

# Aufgabe 0: Project Setup

## Installation

### Using Maven

EasyMock is available in the Maven central repository. Just add the following dependency to your pom.xml:

```
<dependency>
  <groupId>org.easymock</groupId>
  <artifactId>easymock</artifactId>
  <version>3.0</version>
  <scope>test</scope>
</dependency>
```

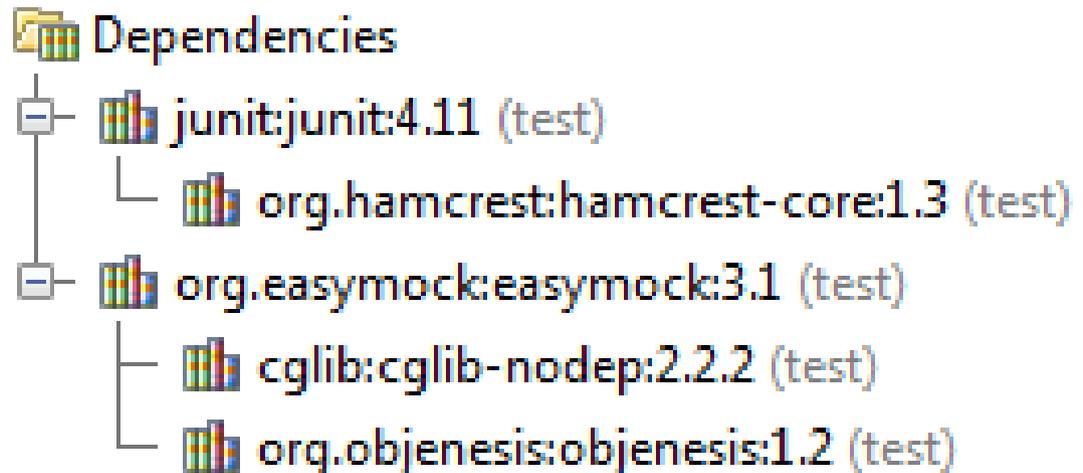
You can obviously use any other dependency tool compatible with the Maven repository.

# Aufgabe 0: Project Setup

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.easymock</groupId>
  <artifactId>easymock</artifactId>
  <version>3.1</version>
  <scope>test</scope>
</dependency>
```

# Aufgabe 0: Project Setup

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.easymock</groupId>
  <artifactId>easymock</artifactId>
  <version>3.1</version>
  <scope>test</scope>
</dependency>
```



# Aufgabe 0: Project Setup

mockito maven

Web

Bilder

Maps

Shopping

Mehr ▾

Suchoptionen

Ungefähr 198.000 Ergebnisse (0,10 Sekunden)

[Maven Repository: org.mockito » mockito-all » 1.8.4](#)

[mvnrepository.com/...mockito/mockito-all/1.8.4](#) - Diese Seite übersetzen

**Mockito**. Mock objects library for java ... HomePage, <http://www.mockito.org>. Organization. Issue Tracker. **Maven**; Ivy; Grape; Gradle; Buildr; SBT. <dependency> ...

[Maven Repository: org.mockito » mockito-all](#)

[mvnrepository.com/artifact/...mockito/mockito-a...](#) - Diese Seite übersetzen

[See All Tags]. Web site developed by @frodriguez. home » [org.mockito](#) » [mockito-all](#). **Mockito**. Mock objects library for java. tags: ...

[MavenUsers - mockito - simpler & better mocking - Google Project ...](#)

[code.google.com/p/mockito/wiki/MavenUsers](#) - Diese Seite übersetzen

How about rolling **mockito-core-1.7-sources.jar** and/or **mockito-core-1.7-javadoc.jar** and putting it in the **maven** repository next to the binaries? Those using ...

# Aufgabe 0: Project Setup



mockito

simpler & better mocking

[Project Home](#)

[Downloads](#)

[Wiki](#)

[Issues](#)

[Source](#)

Search  for

## MavenUsers

Page moved to [DeclaringMockitoDependency](#).

# Aufgabe 0: Project Setup



## mockito

simpler & better mocking

[Project Home](#)[Downloads](#)[Wiki](#)[Issues](#)[Source](#)

Search  Current pages  for

## Maven

mockito-core

```
<!-- needs extra dependencies: objenesis & hamcrest -->
<groupId>org.mockito</groupId>
<artifactId>mockito-core</artifactId>
<version>1.9.5</version>
<scope>test</scope>
```

mockito-all

```
<groupId>org.mockito</groupId>
<artifactId>mockito-all</artifactId>
<version>1.9.5</version>
<scope>test</scope>
```

## mockito-core VS mockito-all

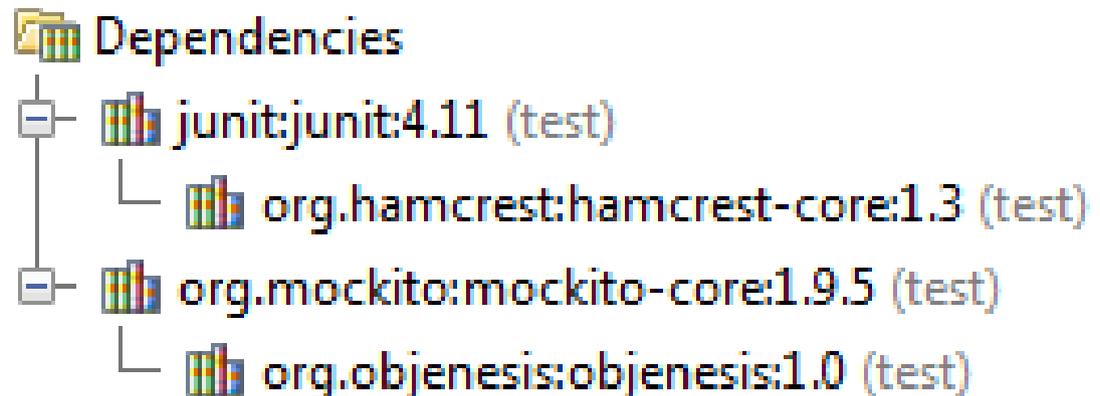
Mockito-all is a single jar with all dependencies inlined inside (that is: hamcrest and objenesis libs as of June'11). Mockito-core is just the mockito jar. Use mockito-core if you want to pull a specific version of hamcrest or objenesis. Mockito-core gives finer control on what jars end up on your classpath. Since maven deals with transitive dependencies well maven users can use mockito-core without any extra hassle.

# Aufgabe 0: Project Setup

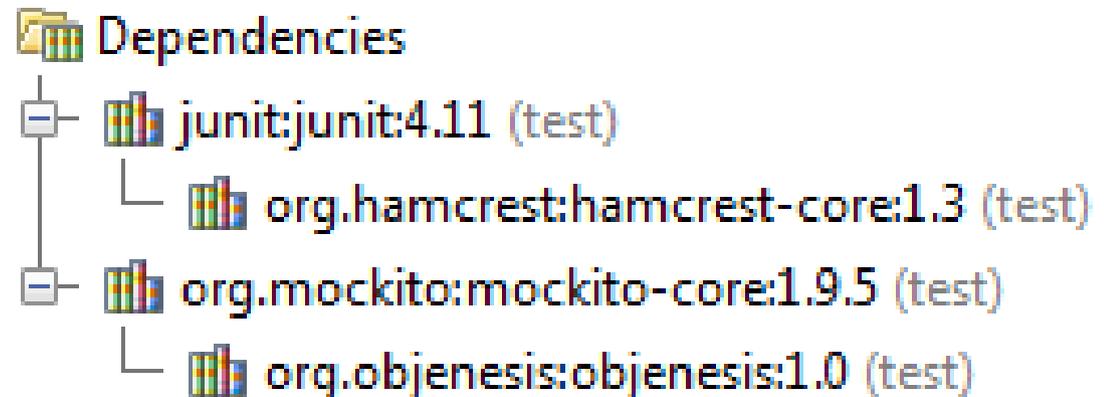
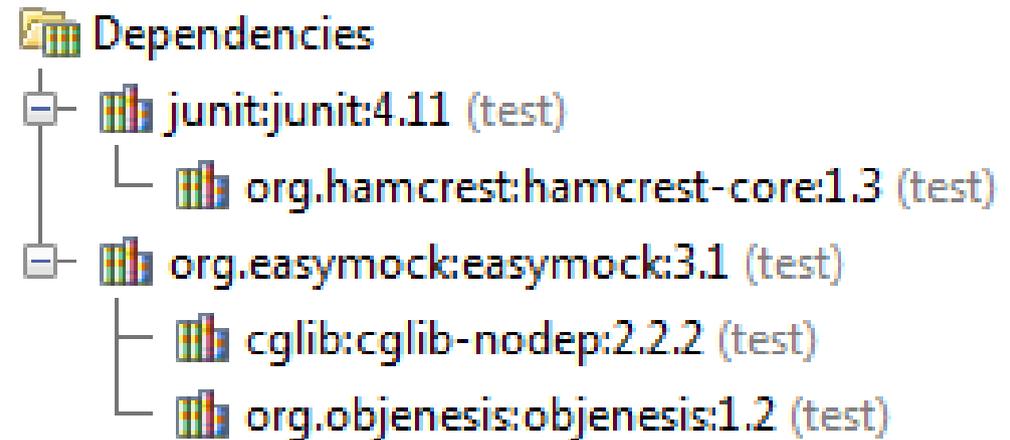
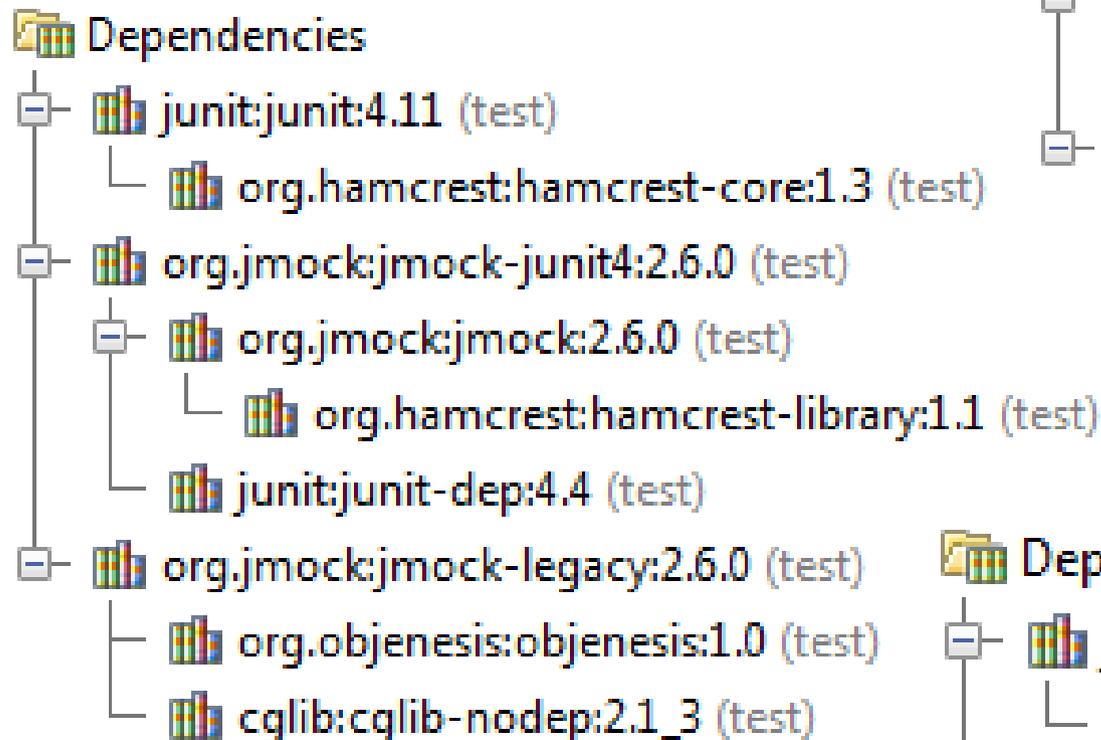
```
<dependency>  
  <groupId>junit</groupId>  
  <artifactId>junit</artifactId>  
  <version>4.11</version>  
  <scope>test</scope>  
</dependency>  
<dependency>  
  <groupId>org.mockito</groupId>  
  <artifactId>mockito-core</artifactId>  
  <version>1.9.5</version>  
  <scope>test</scope>  
</dependency>
```

# Aufgabe 0: Project Setup

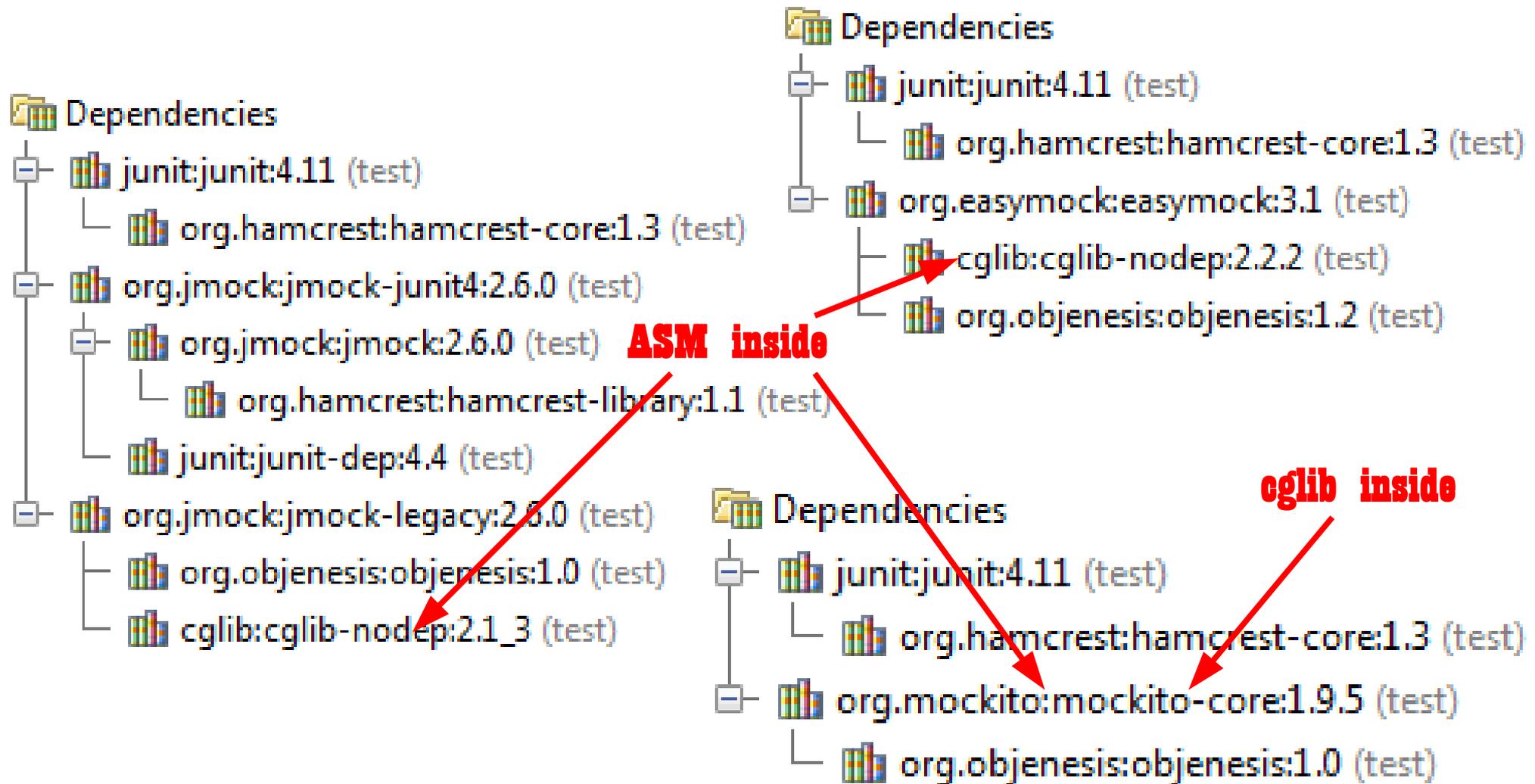
```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-core</artifactId>
  <version>1.9.5</version>
  <scope>test</scope>
</dependency>
```



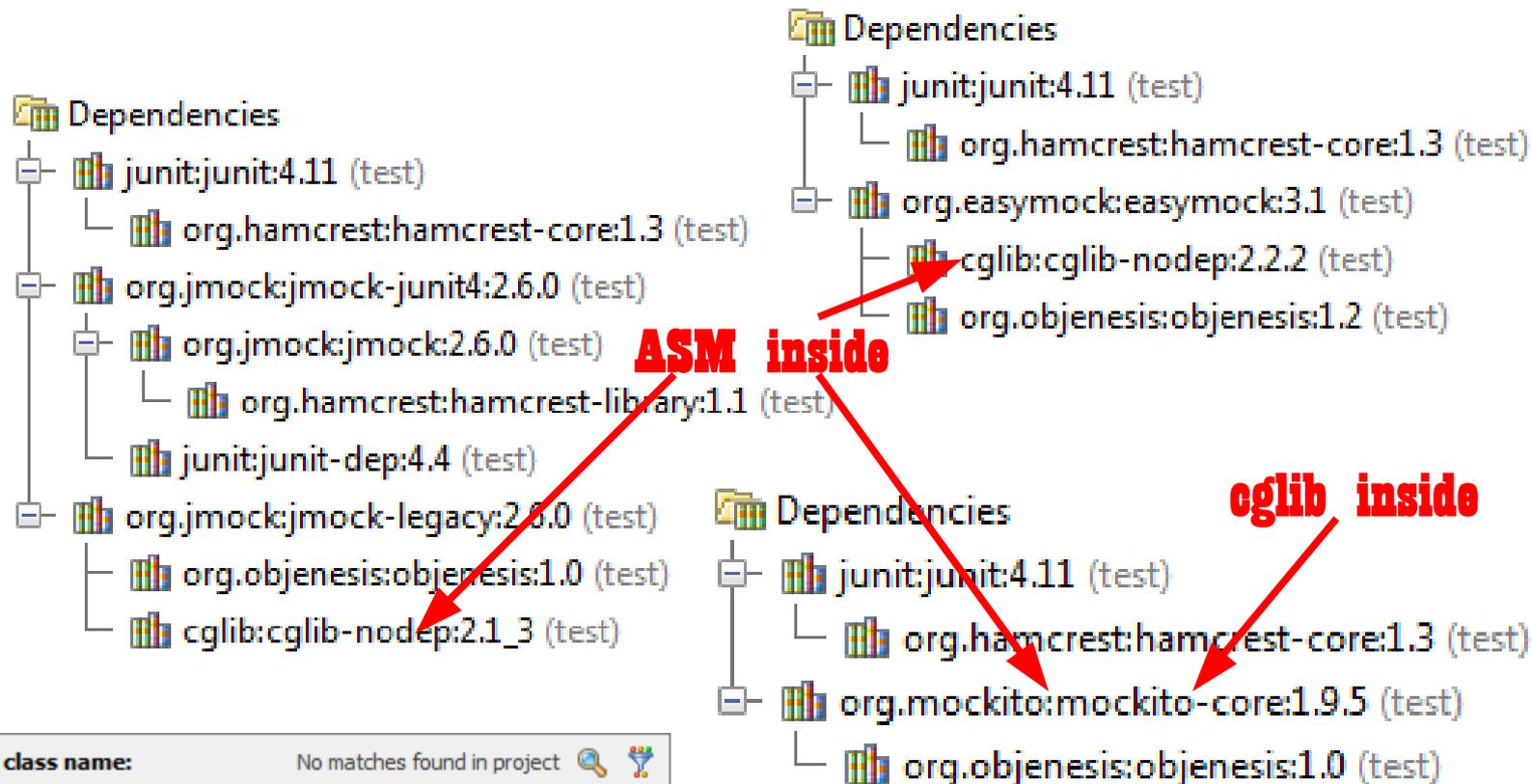
# Aufgabe 0: Project Setup



# Aufgabe 0: Project Setup



# Aufgabe 0: Project Setup



Enter class name: No matches found in project

ClassVisitor (net.sf.cglib.asm)	Maven: cglib:cglib-nodep:2.2.2 (cglib-nodep-2.2.2.jar)
ClassVisitor (org.mockito.asm)	Maven: org.mockito:mockito-core:1.9.5 (mockito-core-1.9.5.jar)
ClassVisitor (org.objectweb.asm)	Maven: org.ow2.asm:asm:4.1 (asm-4.1.jar)
ClassVisitor (org.springframework.asm)	Maven: org.springframework:spring-core:3.2.2.RELEASE (spring-core-3.2.2.RELEASE.jar)
ClassVisitor (com.sun.xml.internal.ws.org.objectweb.asm)	< 1.7 > (rt.jar)
ClassVisitorTee (net.sf.cglib.transform)	Maven: cglib:cglib-nodep:2.2.2 (cglib-nodep-2.2.2.jar)
ClassVisitorTee (org.mockito.cglib.transform)	Maven: org.mockito:mockito-core:1.9.5 (mockito-core-1.9.5.jar)
ClassVisitorTee (org.springframework.cglib.transform)	Maven: org.springframework:spring-core:3.2.2.RELEASE (spring-core-3.2.2.RELEASE.jar)

# Aufgabe 1: Test Isolation

```
public class LoginController implements Controller {

    private final UserRepository userRepository;

    @Autowired
    public LoginController(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        User user = userRepository.load(username, password);
        if (user == null) {
            return loginFailed();
        } else {
            request.getSession().setAttribute("currentUser", user);
            return loginSucceeded();
        }
    }

    private ModelAndView loginSucceeded() {...}

    private ModelAndView loginFailed() {...}
}
```

# Aufgabe 1: Test Isolation

```
import static aufgabe1.UserBuilder.anUser;
import static org.fest.assertions.api.Assertions.assertThat;

public class LoginControllerTest_jMock2 {

    @Rule
    public JUnitRuleMockery mockery = new JUnitRuleMockery() {{
        setImposteriser(ClassImposteriser.INSTANCE);
    }};

    @Test
    public void testHappyPath() throws Exception {
        // Arrange ...
        final UserRepository userRepository = mockery.mock(UserRepository.class);
        final User user = anUser().withUsername("foo").withPassword("bar").build();
        mockery.checking(new Expectations() {{
            oneOf(userRepository).load("foo", "bar"); will(returnValue(user));
        }});
        LoginController loginController = new LoginController(userRepository);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Act ...
        loginController.handleRequest(request, null);
        // Assert ...
        assertThat(request.getSession().getAttribute("currentUser")).isSameAs(user);
    }
}
```

# Aufgabe 1: Test Isolation

```
import static aufgabe1.UserBuilder.anUser;
import static org.easymock.EasyMock.*;
import static org.fest.assertions.api.Assertions.assertThat;

public class LoginControllerTest_EasyMock extends EasyMockSupport {

    @Test
    public void testHappyPath() throws Exception {
        // Arrange & Record ...
        UserRepository userRepository = createMock(UserRepository.class);
        User user = anUser().withUsername("foo").withPassword("bar").build();
        expect(userRepository.load("foo", "bar")).andReturn(user);
        LoginController loginController = new LoginController(userRepository);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Switch to replay mode ...
        replayAll();
        // Act ...
        loginController.handleRequest(request, null);
        // Assert ...
        assertThat(request.getSession().getAttribute("currentUser")).isSameAs(user);
    }
}
```

# Aufgabe 1: Test Isolation

```
import static aufgabe1.UserBuilder.anUser;
import static org.mockito.Mockito.*;
import static org.fest.assertions.api.Assertions.assertThat;

public class LoginControllerTest_Mockito {

    @Test
    public void testHappyPath() throws Exception {
        // Arrange ...
        UserRepository userRepository = mock(UserRepository.class);
        User user = anUser().withUsername("foo").withPassword("bar").build();
        when(userRepository.load("foo", "bar")).thenReturn(user);
        LoginController loginController = new LoginController(userRepository);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Act ...
        loginController.handleRequest(request, null);
        // Assert ...
        assertThat(request.getSession().getAttribute("currentUser")).isSameAs(user);
    }
}
```

# Aufgabe 2: Behavior Verification

```
public class LoginController implements Controller {

    private final UserRepository userRepository;
    private final EventBus eventBus;

    @Autowired
    public LoginController(UserRepository userRepository, EventBus eventBus) {
        this.userRepository = userRepository;
        this.eventBus = eventBus;
    }

    @Override
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        eventBus.publish(Event.LOGIN_ATTEMPT, username);
        User user = userRepository.load(username, password);
        if (user == null) {
            eventBus.publish(Event.LOGIN_FAILED, username);
            return loginFailed();
        } else {
            request.getSession().setAttribute("currentUser", user);
            eventBus.publish(Event.LOGIN_SUCCESS, user);
            return loginSucceeded();
        }
    }

    private ModelAndView loginSucceeded() {...}

    private ModelAndView loginFailed() {...}
}
```

# Aufgabe 2: Behavior Verification

```
import static aufgabe2.UserBuilder.anUser;

public class LoginControllerTest_jMock2 {

    @Rule
    public JUnitRuleMockery mockery = new JUnitRuleMockery() {{
        setImposteriser(ClassImposteriser.INSTANCE);
    }};

    @Test
    public void testHappyPath() throws Exception {
        // Arrange ...
        final UserRepository userRepository = mockery.mock(UserRepository.class);
        final EventBus eventBus = mockery.mock(EventBus.class);
        final User user = anUser().withUsername("foo").withPassword("bar").build();
        mockery.checking(new Expectations() {{
            oneOf(eventBus).publish(Event.LOGIN_ATTEMPT, "foo");
            oneOf(userRepository).load("foo", "bar"); will(returnValue(user));
            oneOf(eventBus).publish(Event.LOGIN_SUCCESS, user);
        }});
        LoginController loginController = new LoginController(userRepository, eventBus);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Act ...
        loginController.handleRequest(request, null);
        // Verify is done automatically by JUnitRuleMockery ...
    }
}
```

# Aufgabe 2: Behavior Verification

```
import static aufgabe2.UserBuilder.anUser;
import static org.easymock.EasyMock.*;

public class LoginControllerTest_EasyMock extends EasyMockSupport {

    @Test
    public void testHappyPath() throws Exception {
        // Arrange ...
        UserRepository userRepository = createMock(UserRepository.class);
        EventBus eventBus = createMock(EventBus.class);
        User user = anUser().withUsername("foo").withPassword("bar").build();
        LoginController loginController = new LoginController(userRepository, eventBus);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Record ...
        eventBus.publish(Event.LOGIN_ATTEMPT, "foo");
        expect(userRepository.load("foo", "bar")).andReturn(user);
        eventBus.publish(Event.LOGIN_SUCCESS, user);
        // Switch to replay mode ...
        replayAll();
        // Act ...
        loginController.handleRequest(request, null);
        // Verify ...
        verifyAll();
    }
}
```

# Aufgabe 2: Behavior Verification

```
import static aufgabe2.UserBuilder.anUser;
import static org.mockito.Mockito.*;

public class LoginControllerTest_Mockito {

    @Test
    public void testHappyPath() throws Exception {
        // Arrange ...
        final UserRepository userRepository = mock(UserRepository.class);
        final User user = anUser().withUsername("foo").withPassword("bar").build();
        when(userRepository.load("foo", "bar")).thenReturn(user);
        final EventBus eventBus = mock(EventBus.class);
        LoginController loginController = new LoginController(userRepository, eventBus);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Act ...
        loginController.handleRequest(request, null);
        // Verify ...
        verify(eventBus).publish(Event.LOGIN_ATTEMPT, "foo");
        verify(userRepository).load("foo", "bar");
        verify(eventBus).publish(Event.LOGIN_SUCCESS, user);
        verifyNoMoreInteractions(userRepository, eventBus);
    }
}
```

**Demo**

# Aufgabe 3: Call Order Verification

```
public class LoginControllerTest_jMock2 {

    @Rule
    public JUnitRuleMockery mockery = new JUnitRuleMockery() {{
        setImposteriser(ClassImposteriser.INSTANCE);
    }};

    @Test
    public void testHappyPath() throws Exception {
        // Arrange ...
        final UserRepository userRepository = mockery.mock(UserRepository.class);
        final EventBus eventBus = mockery.mock(EventBus.class);
        final User user = anUser().withUsername("foo").withPassword("bar").build();
        final Sequence expectedCallOrder = mockery.sequence("expectedCallOrder");
        mockery.checking(new Expectations() {{
            oneOf(eventBus).publish(Event.LOGIN_ATTEMPT, "foo"); inSequence(expectedCallOrder);
            oneOf(userRepository).load("foo", "bar"); will(returnValue(user)); inSequence(expectedCallOrder);
            oneOf(eventBus).publish(Event.LOGIN_SUCCESS, user); inSequence(expectedCallOrder);
        }});
        LoginController loginController = new LoginController(userRepository, eventBus);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Act ...
        loginController.handleRequest(request, null);
        // Verify is done automatically by JUnitRuleMockery ...
    }
}
```

# Aufgabe 3: Call Order Verification

```
public class LoginControllerTest_EasyMock extends EasyMockSupport {

    @Test
    public void testHappyPath() throws Exception {
        // Arrange ...
        IMocksControl strictControl = createStrictControl();
        UserRepository userRepository = strictControl.createMock(UserRepository.class);
        EventBus eventBus = strictControl.createMock(EventBus.class);
        User user = anUser().withUsername("foo").withPassword("bar").build();
        LoginController loginController = new LoginController(userRepository, eventBus);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Record ...
        eventBus.publish(Event.LOGIN_ATTEMPT, "foo");
        expect(userRepository.load("foo", "bar")).andReturn(user);
        eventBus.publish(Event.LOGIN_SUCCESS, user);
        // Switch to replay mode ...
        replayAll();
        // Act ...
        loginController.handleRequest(request, null);
        // Verify ...
        verifyAll();
    }
}
```

# Aufgabe 3: Call Order Verification

```
public class LoginControllerTest_Mockito {

    @Test
    public void testHappyPath() throws Exception {
        // Arrange ...
        final UserRepository userRepository = mock(UserRepository.class);
        final User user = anUser().withUsername("foo").withPassword("bar").build();
        when(userRepository.load("foo", "bar")).thenReturn(user);
        final EventBus eventBus = mock(EventBus.class);
        LoginController loginController = new LoginController(userRepository, eventBus);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Act ...
        loginController.handleRequest(request, null);
        // Verify ...
        InOrder inOrder = inOrder(userRepository, eventBus);
        inOrder.verify(eventBus).publish(Event.LOGIN_ATTEMPT, "foo");
        inOrder.verify(userRepository).load("foo", "bar");
        inOrder.verify(eventBus).publish(Event.LOGIN_SUCCESS, user);
        verifyNoMoreInteractions(userRepository, eventBus);
    }
}
```

**Demo**

# Aufgabe 4: Ignore Method Calls

```
public class LoginControllerTest_jMock2 {

    @Rule
    public JUnitRuleMockery mockery = new JUnitRuleMockery() {{
        setImposteriser(ClassImposteriser.INSTANCE);
    }};

    @Test
    public void testHappyPath() throws Exception {
        // Arrange ...
        final UserRepository userRepository = mockery.mock(UserRepository.class);
        final EventBus eventBus = mockery.mock(EventBus.class);
        final User user = anUser().withUsername("foo").withPassword("bar").build();
        mockery.checking(new Expectations() {{
            oneOf(userRepository).load("foo", "bar"); will(returnValue(user));
        }});
        LoginController loginController = new LoginController(userRepository, eventBus);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Act ...
        loginController.handleRequest(request, null);
        // Assert ...
        assertThat(request.getSession().getAttribute("currentUser")).isSameAs(user);
    }
}
```

# Aufgabe 4: Ignore Method Calls



```
java.lang.AssertionError: unexpected invocation: EventBus.publish(<LOGIN_ATTEMPT>, "foo")
expectations:
  expected once, never invoked: userRepository.load("foo", "bar"); returns <User foo>
what happened before this: nothing!
  at org.jmock.api.ExpectationError.unexpected(ExpectationError.java:23)
  at org.jmock.internal.InvocationDispatcher.dispatch(InvocationDispatcher.java:85)
  at org.jmock.Mockery.dispatch(Mockery.java:231)
  at org.jmock.Mockery.access$100(Mockery.java:29)
  at org.jmock.Mockery$MockObject.invoke(Mockery.java:271)
  at org.jmock.internal.InvocationDiverter.invoke(InvocationDiverter.java:27)
  at org.jmock.internal.FakeObjectMethods.invoke(FakeObjectMethods.java:38)
  at org.jmock.internal.SingleThreadedPolicy$1.invoke(SingleThreadedPolicy.java:21)
  at org.jmock.lib.legacy.ClassImposteriser$4.invoke(ClassImposteriser.java:129)
  at aufgabe2.EventBus$$EnhancerByCGLIB$$906ee3da.publish(<generated>)
  at aufgabe2.LoginController.handleRequest(LoginController.java:25)
  at aufgabe4.LoginControllerTest_jMock2.testHappyPath(LoginControllerTest_jMock2.java:38) <7 internal calls>
  at org.jmock.integration.junit4.JUnitRuleMockery$1.evaluate(JUnitRuleMockery.java:49) <16 internal calls>
```

# Aufgabe 4: Ignore Method Calls

```
public class LoginControllerTest_jMock2 {

    @Rule
    public JUnitRuleMockery mockery = new JUnitRuleMockery() {{
        setImposteriser(ClassImposteriser.INSTANCE);
    }};

    @Test
    public void testHappyPath() throws Exception {
        // Arrange ...
        final UserRepository userRepository = mockery.mock(UserRepository.class);
        final EventBus eventBus = mockery.mock(EventBus.class);
        final User user = anUser().withUsername("foo").withPassword("bar").build();
        mockery.checking(new Expectations() {{
            oneOf(userRepository).load("foo", "bar"); will(returnValue(user));
            ignoring(eventBus);
        }});
        LoginController loginController = new LoginController(userRepository, eventBus);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Act ...
        loginController.handleRequest(request, null);
        // Assert ...
        assertThat(request.getSession().getAttribute("currentUser")).isSameAs(user);
    }
}
```

# Aufgabe 4: Ignore Method Calls

```
public class LoginControllerTest_EasyMock extends EasyMockSupport {

    @Test
    public void testHappyPath() throws Exception {
        // Arrange & Record ...
        UserRepository userRepository = createMock(UserRepository.class);
        EventBus eventBus = createMock(EventBus.class);
        User user = anUser().withUsername("foo").withPassword("bar").build();
        expect(userRepository.load("foo", "bar")).andReturn(user);
        LoginController loginController = new LoginController(userRepository, eventBus);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Switch to replay mode ...
        replayAll();
        // Act ...
        loginController.handleRequest(request, null);
        // Assert ...
        assertThat(request.getSession().getAttribute("currentUser")).isSameAs(user);
    }
}
```

# Aufgabe 4: Ignore Method Calls



```
java.lang.AssertionError:
  Unexpected method call EventBus.publish(LOGIN_ATTEMPT, "foo"):
    at org.easymock.internal.MockInvocationHandler.invoke(MockInvocationHandler.java:44)
    at org.easymock.internal.ObjectMethodsFilter.invoke(ObjectMethodsFilter.java:85)
    at sun.proxy.$Proxy5.publish(Unknown Source)
    at aufgabe2.LoginController.handleRequest(LoginController.java:25)
    at aufgabe4.LoginControllerTest_EasyMock.testHappyPath(LoginControllerTest\_EasyMock.java:31) <23 internal calls>
```

# Aufgabe 4: Ignore Method Calls

```
public class LoginControllerTest_EasyMock extends EasyMockSupport {

    @Test
    public void testHappyPath() throws Exception {
        // Arrange & Record ...
        UserRepository userRepository = createMock(UserRepository.class);
        EventBus eventBus = createNiceMock(EventBus.class);
        User user = anUser().withUsername("foo").withPassword("bar").build();
        expect(userRepository.load("foo", "bar")).andReturn(user);
        LoginController loginController = new LoginController(userRepository, eventBus);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Switch to replay mode ...
        replayAll();
        // Act ...
        loginController.handleRequest(request, null);
        // Assert ...
        assertThat(request.getSession().getAttribute("currentUser")).isSameAs(user);
    }
}
```

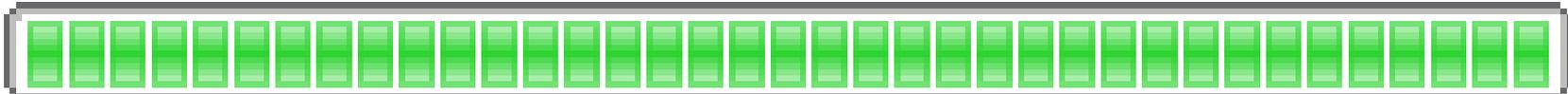
# Aufgabe 4: Ignore Method Calls

```
public class LoginControllerTest_Mockito {

    @Test
    public void testHappyPath() throws Exception {
        // Arrange ...
        UserRepository userRepository = mock(UserRepository.class);
        User user = anUser().withUsername("foo").withPassword("bar").build();
        when(userRepository.load("foo", "bar")).thenReturn(user);
        EventBus eventBus = mock(EventBus.class);
        LoginController loginController = new LoginController(userRepository, eventBus);
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.addParameter("username", "foo");
        request.addParameter("password", "bar");
        // Act ...
        loginController.handleRequest(request, null);
        // Assert ...
        assertThat(request.getSession().getAttribute("currentUser")).isSameAs(user);
    }
}
```

# Aufgabe 4: Ignore Method Calls

```
public class LoginControllerTest_Mockito {  
  
    @Test  
    public void testHappyPath() throws Exception {  
        // Arrange ...  
        UserRepository userRepository = mock(UserRepository.class);  
        User user = anUser().withUsername("foo").withPassword("bar").build();  
        when(userRepository.load("foo", "bar")).thenReturn(user);  
        EventBus eventBus = mock(EventBus.class);  
        LoginController loginController = new LoginController(userRepository, eventBus);  
        MockHttpServletRequest request = new MockHttpServletRequest();  
        request.addParameter("username", "foo");  
        request.addParameter("password", "bar");  
        // Act ...  
        loginController.handleRequest(request, null);  
        // Assert ...  
        assertThat(request.getSession().getAttribute("currentUser")).isSameAs(user);  
    }  
}
```



# Die alte Mocking Philosophie

**1. Erwartungen und Antworten aufzeichnen**

**2. Produktionscode ausführen**



**3. Erwartungen überprüfen**



# Die alte Mocking Philosophie

1. **Erwartungen und Antworten aufzeichnen**

2. **Produktionscode ausführen**



3. **Erwartungen überprüfen**



- **viel Setup-Code**
- **viel Ausprobieren beim Schreiben von Tests**
- **Tests brechen öfters**
- **Erwartungen stehen vor der getesteten Aktion**

# Eine neue Mocking Philosophie

**1. Antworten aufzeichnen (Stubbing)**

**2. Produktionscode ausführen**

**3. Erwartungen überprüfen (Verification)**



# Eine neue Mocking Philosophie

**1. Antworten aufzeichnen (Stubbing)**

**2. Produktionscode ausführen**

**3. Erwartungen überprüfen (Verification)**

- + **weniger Setup-Code**
- + **robustere Tests**
- + **besser verständliche Tests**



# Eine neue Mocking Philosophie

**1. Antworten aufzeichnen (Stubbing)**

**2. Produktionscode ausführen**

**3. Erwartungen überprüfen (Verification)**

- + **weniger Setup-Code**
- + **robustere Tests**
- + **besser verständliche Tests**



**Aber ...**

# Zusatzaufgabe für Mockito:

## Do not ignore unexpected calls

```
public class LoginControllerTest_Mockito {  
  
    @Test  
    public void testHappyPath() throws Exception {  
        // Arrange ...  
        UserRepository userRepository = mock(UserRepository.class);  
        User user = anUser().withUsername("foo").withPassword("bar").build();  
        when(userRepository.load("foo", "bar")).thenReturn(user);  
        EventBus eventBus = pseudo(EventBus.class);  
        LoginController loginController = new LoginController(userRepository, eventBus);  
        MockHttpServletRequest request = new MockHttpServletRequest();  
        request.addParameter("username", "foo");  
        request.addParameter("password", "bar");  
        // Act ...  
        loginController.handleRequest(request, null);  
        // Assert ...  
        assertThat(request.getSession().getAttribute("currentUser")).isSameAs(user);  
    }  
  
    public static <T> T pseudo(Class<T> classToMock) {  
        return mock(classToMock, new Answer<Object>() {  
            @Override  
            public Object answer(InvocationOnMock invocation) throws Throwable {  
                throw new AssertionError("Unexpected method call " + invocation);  
            }  
        });  
    }  
}
```

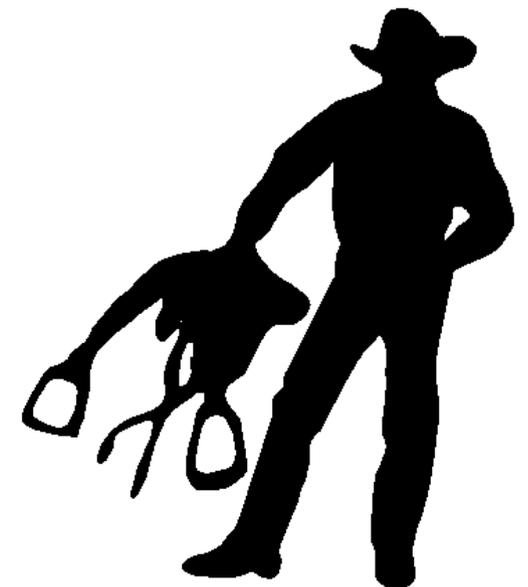
# Zusatzaufgabe für Mockito: Do not ignore unexpected calls



```
java.lang.AssertionError: Unexpected method call EventBus.publish(LOGIN_ATTEMPT, "foo");  
    at zusatzaufgabe.LoginControllerTest_Mockito$1.answer(LoginControllerTest_Mockito.java:40)  
    at org.mockito.internal.handler.MockHandlerImpl.handle(MockHandlerImpl.java:93)  
    at org.mockito.internal.handler.NullResultGuardian.handle(NullResultGuardian.java:29)  
    at org.mockito.internal.handler.InvocationNotifierHandler.handle(InvocationNotifierHandler.java:38)  
    at org.mockito.internal.creation.MethodInterceptorFilter.intercept(MethodInterceptorFilter.java:51)  
    at aufgabe2.EventBus$$EnhancerByMockitoWithCGLIB$$26b4bed.publish(<generated>)  
    at aufgabe2.LoginController.handleRequest(LoginController.java:26)  
    at zusatzaufgabe.LoginControllerTest_Mockito.testHappyPath(LoginControllerTest_Mockito.java:31) <23 internal calls>
```



**Können etwa alle  
Mocking-Bibliotheken  
das gleiche?**



# Return values for unexpected method calls

- `false` für `boolean`,
- `'\0'` für `char`
- `0` für alle anderen primitiven Typen,
- `null` in allen anderen Fällen

# Return values for unexpected method calls

- `false` für `boolean`,
- `'\0'` für `char`
- `0` für alle anderen primitiven Typen,
- den leeren String `""` für `Strings`,
- ein leeres Array für `Arrays`,
- ein Mock-Objekt, das ebenfalls ignoriert wird, wenn von der Klasse des Rückgabewerts ein Mock-Objekt erzeugt werden kann, was eigentlich immer der Fall ist, wenn Sie das Modul *jmock-legacy* benutzen, außer die Klasse ist `final`,
- `null` für alle anderen Typen

# Return values

mockito



## for unexpected method calls

- `false` für `boolean` sowie `Boolean.FALSE`, falls die Methode eine Instanz der Wrapper-Klasse `java.lang.Boolean` zurückgibt,
- `'\0'` für `char` sowie für die Wrapper-Klasse `java.lang.Character`,
- `0` für alle anderen primitiven Typen (`int`, `float`, ...) sowie ihren Wrapper-Klassen,
- eine leere `Collection` bzw. `Map`, falls der Typ des Rückgabewerts eines der in `java.util` definierten Interfaces `Collection`, `List`, `Set`, `SortedSet`, `Map` oder `SortedMap` ist, aber auch falls die Methode eine der gebräuchlichsten Implementierungen dieser Interfaces zurückgibt, und zwar: `ArrayList`, `LinkedList`, `HashSet`, `LinkedHashSet`, `TreeSet`, `HashMap`, `LinkedHashMap` oder `TreeMap`.
- `null` in allen anderen Fällen, Ausnahme: `toString()`

# Return values for unexpected method calls



ReturnsMoreEmptyValues – wie zuvor, aber zusätzlich:

- den leeren String "" für Strings,
- ein leeres Array für Arrays

ReturnsMocks – wie ReturnsMoreEmptyValues, aber zusätzlich:

- ein Mock-Objekt statt null (wenn möglich)

...

# Return values for unexpected method calls



ReturnsDeepStubs – ähnlich wie ReturnsMocks, aber merkt sich erzeugte Mock-Objekte und gibt diese erneut zurück, falls die gleiche Methode erneut mit den gleichen Parametern aufgerufen wird. Dies ermöglicht *Deep Stubbing*:

```
@Test
public void testDeepStubbing() throws SQLException {
    Statement stmt = mock(Statement.class, RETURNS_DEEP_STUBS);
    when(stmt.executeQuery(startsWith("SELECT COUNT")))
        .getInt(1).thenReturn(17);

    ResultSet rs = stmt.executeQuery("SELECT COUNT(*) FROM foo WHERE status = 0");

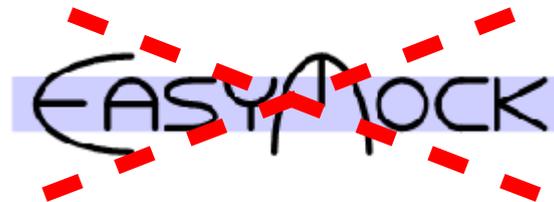
    assertThat(rs.getInt(1), is(17));
}
```

# **Spy Objects**

# Spy Objects



```
public void testCreateUsesFsArg() throws Exception {  
    FileSystem fs = FileSystem.getLocal(conf);  
    FileSystem spyFs = Mockito.spy(fs);  
    Path p = new Path(System.getProperty("test.build.data", ".")+"/testCreateUsesFSArg.seq");  
    SequenceFile.Writer writer = SequenceFile.createWriter(  
        spyFs, conf, p, NullWritable.class, NullWritable.class);  
    writer.close();  
    Mockito.verify(spyFs).getDefaultReplication(p);  
}
```



**statische Methoden mocken?**

**final Klassen/Methoden mocken?**

# statische Methoden mocken?

# final Klassen/Methoden mocken?



# statische Methoden mocken?

# final Klassen/Methoden mocken?



# statische Methoden mocken

```
@RunWith(PowerMockRunner.class)
@PrepareForTest({ Session.class, Transport.class })
public class JavaMailFacadeTest {

    @Test
    public void test_createSession() throws Exception {
        // Mock all static methods of Session ...
        mockStatic(Session.class);
        // Stub static Session.getInstance method ...
        final Session mockSession = mock(Session.class);
        when(Session.getInstance(any(Properties.class))).thenReturn(mockSession);
        JavaMailFacade sut = new JavaMailFacade();

        Session session = sut.createSession();

        assertThat(session, isSameInstanceAs(mockSession));
    }
}
```



**Das war der  
Mocking Libraries Shootout**

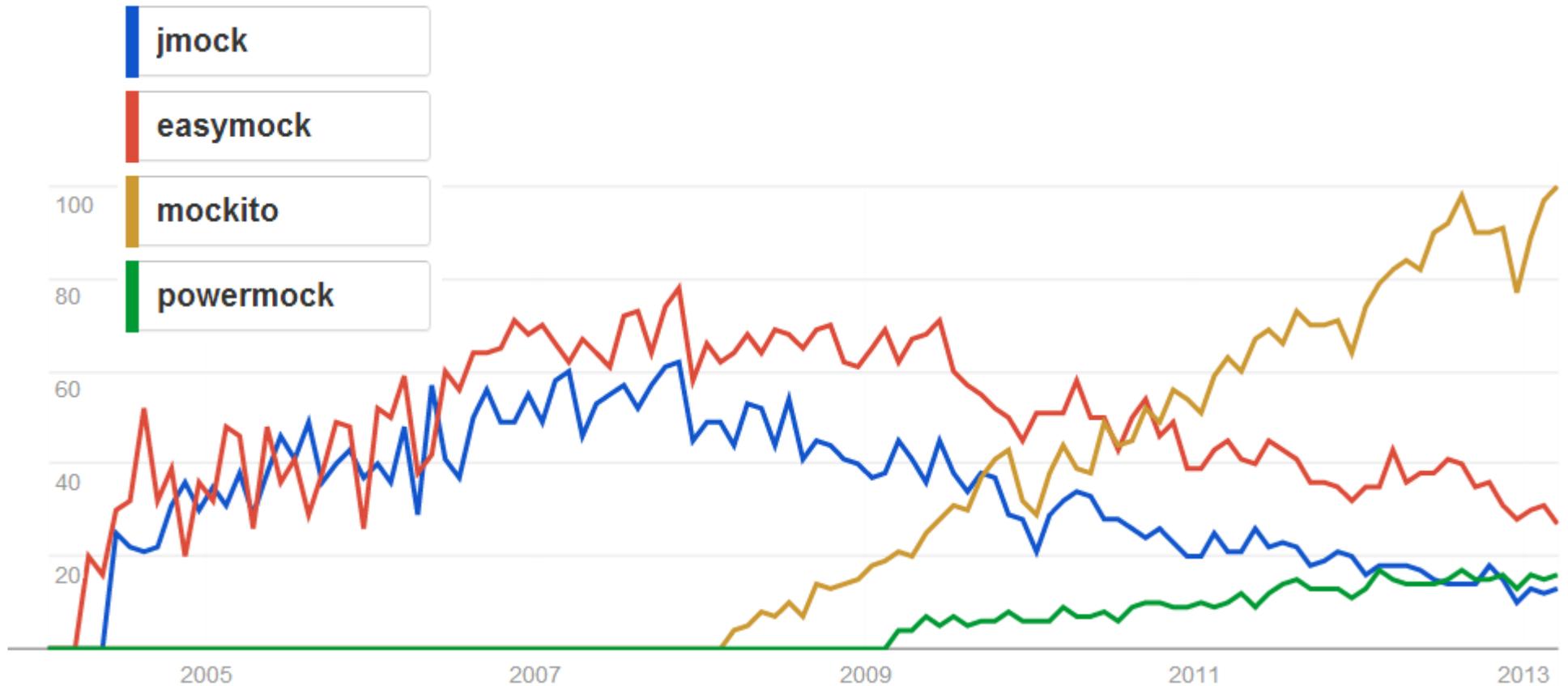


**Das war der  
Mocking Libraries Shootout  
Naja, fast ...**



**Das war der  
Mocking Libraries Shootout  
And The Winner Is ...**

# Google Trends



# Fragen?



**Vielen Dank für Ihre Aufmerksamkeit.**

**Falls Sie mehr wissen möchten:**

