Mock Me If You Can

An Introduction to the Mocking Framework Mockito

Symposium on Software Performance 2014

Christian Wulf — 26.11.2014

Software Engineering Group
Kiel University, Germany

Motivation

- How to check that the underlying input stream is called only once?

```
BufferedInputStream bufferedStream = new BufferedInputStream(inputStream);
try {
    bufferedStream.read(); // delegates to the input stream
    bufferedStream.read(); // uses the buffered result
} finally {
    bufferedStream.close();
}
```

Motivation

- How to test a TcpReader‘s functionality without establishing a real TCP connection?

- `serverSocket`
- `socketChannel`

```
protected void execute() {
    ServerSocketChannel serversocket = null;
    try {
        serversocket = ServerSocketChannel.open();
        serversocket.socket().bind(new InetSocketAddress(this.port));
        logger.debug("Listening on port " + this.port);

        final SocketChannel socketChannel = serversocket.accept();
        try {
            final ByteBuffer buffer = ByteBuffer.allocateDirect(bufferCapacity);
            while (socketChannel.read(buffer) != -1) {
                process(buffer);
            }
        } finally {
            socketChannel.close();
        }
    } catch (final IOException ex) {
        logger.error("Error while reading", ex);
    } finally {
        if (null != serversocket) {
            try {
                serversocket.close();
            } catch (final IOException e) {
                logger.debug("Failed to close TCP connection!", e);
            }
        }

        this.terminate();
    }
}
```

Mockito

- Mockito library enables
  - mocks creation,
  - stubbing, and
  - verification

- Started on 11.11.2007
- Current version **1.10.13**

When to Use

When the CUD requires external (I/O) resources to work properly, e.g.,

- InputStream/OutputStream
- ServerSocket/ClientSocket
- File
- Queue

```
BufferedInputStream bufferedStream = ne
try {
    bufferedStream.read(); // delegates
    bufferedStream.read(); // uses the
} finally {
    bufferedStream.close();
}
```

```
protected void execute() {
    ServerSocketChannel serversocket = null;
    try {
        serversocket = ServerSocketChannel.open();
        serversocket.socket().bind(new InetSocketAddress(this.port));
        logger.debug("Listening on port " + this.port);

        final SocketChannel socketChannel = serversocket.accept();
        try {
            final ByteBuffer buffer = ByteBuffer.allocateDirect(bufferCapacity);
            while (socketChannel.read(buffer) != -1) {
                process(buffer);
            }
        } finally {
            socketChannel.close();
        }
    } catch (final IOException ex) {
        logger.error("Error while reading", ex);
    } finally {
        if (null != serversocket) {
            try {
                serversocket.close();
            } catch (final IOException e) {
                logger.debug("Failed to close TCP connection!", e);
            }
        }

        this.terminate();
    }
}
```

Mocking Methodology


```
@Test
public void testRead() throws Exception {
    InputStream mockedInputStream = mock(InputStream.class);

    when(mockedInputStream.read(any(byte[].class), anyInt(), anyInt())).thenReturn(3);

    BufferedInputStream bufferedStream = new BufferedInputStream(mockedInputStream);
    try {
        bufferedStream.read(); // delegates to mockedInputStream
        bufferedStream.read(); // uses the buffered result
    } finally {
        bufferedStream.close();
    }

    verify(mockedInputStream).read(any(byte[].class), anyInt(), anyInt());
}
```

Setup

- Mocking a type instantiation
- Mocking a method call (a.k.a. stubbing)

Assertion

- Verifying a method call

Mocking


```
InputStream mockedInputStream = mock(InputStream.class);

InputStream spiedInputStream = spy(new ByteArrayInputStream(new byte[] { 1, 2 }));
```

Necessary for verification

- Mock => types, i.e., interfaces, abstract classes, classes
- Spy => instances

Stubbing

- `doReturn(Object)`
- `doThrow(Throwable)`
  - `doThrow(new RuntimeException()).when(mockedList).clear();`
- `doThrow(Class)`
- `doAnswer(Answer)`
- `doNothing()`
- `doCallRealMethod()`

Verifying


```
verify(mockedList).add("one");
verify(mockedList).get(anyInt());
verify(mockedList, times(3)).add("three times");
verify(mockedList, atMost(5)).add("three times");
InOrder inOrder = inOrder(firstMock, secondMock);
verify(mockOne, never()).add("two");
verifyZeroInteractions(mockTwo, mockThree);
verify(mock, timeout(100).times(1)).someMethod();
```

Annotation-based Usage


```
public class BufferedStream4Test {

    @Mock
    private ByteArrayInputStream mockedInputStream;

    @Spy
    private final ByteArrayInputStream spiedInputStream = new ByteArrayInputStream(new byte[] { 1, 2 });

    @Before
    public void init() {
        MockitoAnnotations.initMocks(this);
    }
```

- Minimizes repetitive mock creation code.
- Makes the test class more readable.
- Makes the verification error easier to read because the field name is used to identify the mock.

Limitations

- Mockito doesn't mock final methods
  - `AbstractInterruptibleChannel.close()`
- Mockito doesn't mock static methods
  - `ServerSocketChannel.open();`

Conclusions

External resources

- InputStream/OutputStream
- ServerSocket/ClientSocket
- File
- Queue

Alternative Mocking Framework for Java:

EASYMOCK

References

- Mockito:
  - http://www.mockito.org/
- PowerMock:
  - https://code.google.com/p/powermock/
- EasyMock
  - http://easymock.org/