Mock framework Mockito

Pieter van den Hombergh
Stefan Sobek

Fontys Hogeschool voor Techniek en Logistiek

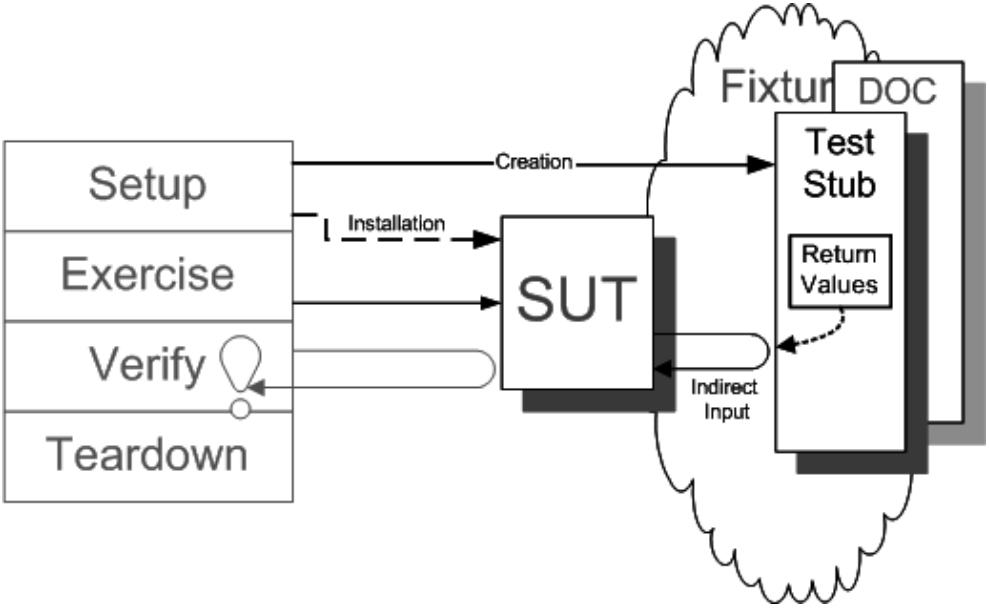
April 18, 2018

To mock or not to mock

- In many cases, a class (system under test or **SUT**) does not work in isolation. It needs **collaborators**.
- The SUT is dependent on these collaborators.
  - We call these collaborators Dependent On Components[1] or **DOC**.
- Often these collaborators are not yet available or do not provide the required functionality for the test.
  - Think of testing an exception handling for an exception that is hard to trigger in the real class.

[1] it is Okay to remember Collaborator

Test stub



source: `http://xunitpatterns.com/Using%20Test%20Doubles.html`

Download using maven

Maven is build to download dependencies it selves. So you can let maven do the heavy lifting. You can even forego the download of the dependencies, and create the netbeans library afterwards. Create a simple **maven -Java** project (in netbeans) and add the following as dependency. Build the project and maven will download mockito and its dependencies.

Mockito maven dependency, Version March 2017

```
<dependencies>
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-core</artifactId>
        <version>2.7.19</version>
        <scope>test</scope>
    </dependency>
    <dependency>
```

Maven saves the downloaded files in a so called repository, organised by the maven coordinates. You find the repository under:

Linux,OSX `/.m2/repository`

Windows `C:/Documents and Settings<windows username>/.m2/repository`

Download mockito “by hand”

- download mockito-core-2.7.19-beta.zip using google.
- download objenesis-2.2-bin.zip using google
- download byte-buddy-agent-1.6.11-bin.zip using google
- download byte-buddy-1.6.11-bin.zip using google
- Unpack (unzip) the downloaded in a convenient place.
  My **convention** is to put the unpacked files under `~/Library/mockito`.
  This works for OSX as well as under Linux.
  In Windows (assuming the username is hom) this would be
  `C:/users/hom/Library/mockito`

Create the Netbeans Library

The easiest way to use mockito with netbeans is by creating a **Library**. It then works in standard netbeans projects, which is just fine.
In NetBeans, go to Tools.Libraries and add a new Library, name it **Mockito**.

ClassPath Add the code jars (mockito-core-2.7.19-beta.jar, and the objenesis and the byte-buddy jars) from the unpacked zips in the class path tab

Sources Add the sources jars to the Sources tab.

Javadoc Add the javadoc jars in the Javadoc tab.

Now you can use Mockito in your default netbeans projects.

First mock

Test doubles or Mock objects are collaborators with various roles

Dummy Just needs to be present and are typically passed around.

Stub Provides indirect inputs.

Spy Help observe (indirect) data output from the SUT.

Mock Can be stub and spy, but has some programmed (recorded) behavior for a specific test case.

Fake Is neither controlled or observed by the test software. Has functionality similar to the real object, but is

- available now (easy to create).
- Easier to set up.
- Faster then the real thing.

DEMO

Coupling is a problem to be avoided

Not just in testing.
Steps in avoiding dependencies:

- Program to interfaces, not to implementations.
- Avoid `new` in code, as this would still bind to an implementation.
- Receive the dependencies by *inserting* via setter or constructor or by using **field injection**.
- This implies that not the class itself but the context (world, it's creator) is responsible of the creation and insertion.
- That also helps tests, because testing is a *different* Context.

The technology is often called **Context and dependency injection**. The service of injecting is provided by a "container". Think of it as the world the classes live in. The aquarium metaphor also works.
It is available in JEE (glassfish) since JEE6 (Glassfish 3).

Sounds difficult

Not really. You still ~~cook with water~~ program in java.
But more modern is to use annotations.
These are also provided in Glassfish[2]

```
class SimpleClass {

    @Inject
    private Oracle oracle;

    public void doSomeThing(){
    System.out.println(oracle.giveMeWisdom());
    }
}
```

[2] and any other JEE6 or higher container, payara or wildfly

Nice, nice

But I want to write a test here, now who does the injecting in this case?

Who does the field injecting in the test?

Answer You, who else is there?

Answer Write your own injector.

Fine Now I have yet another problem  .

But it's simple See next slide

Injector helper

Just one method that can inject **@EJB**, **@Inject** and **@Resource**

```
public static void injectField( Object target,
        String fieldName,
        Object dep ) throws Exception {
    String cn = target.getClass().getCanonicalName();
    Field field = target.getClass().getDeclaredField( fieldName );
    Annotation ejb = field.getAnnotation( EJB.class );
    Annotation inject = field.getAnnotation( Inject.class );
    Annotation resource = field.getAnnotation( Resource.class );
    boolean annotationFound = ( null != ejb || null != inject
            || null != resource );
    assertTrue(
            "Missing annotation, none of "
            + "(EJB,Inject,Resource) found on  "
            + cn + "." + fieldName, annotationFound );
    field.setAccessible( true );
    field.set( target, dep );
}
```

The **@Resource** is available in the standard JDK/JRE. But there you will have to find a "container". Maybe google-guice is a good choice. It appears to be a reference implementation for use outside a container.

Injection with mockito

Dependency injection is a trick that mockito can also play.
You only need to tell Mockito to do it, e.g. in a **@Before** method.

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

Mockito can use field injection even without annotations.

Mockito field injection

Setup mock and inject

```
Calendar mc; // Mocked Calender

@InjectMocks
Cal tcal; // SUT

@Before
public void setUp() {
    mc = mock( Calendar.class );
    when( mc.get( Calendar.YEAR ) ).thenReturn( 2020 );
    //tcal = new Cal(); Look Ma, no ctor
    MockitoAnnotations.initMocks( this );
}
```