Carlos Alexandro Becker Status Stats Talks Archive RSS Contact About

# Using JUnit Rules to simplify your tests

Nov 18, 2014 5 minutes read

Have you ever wrote JUnit tests extending a class that does some before and after work, so you didn't have to repeat that code in various test classes? Well, I will not say that you have been doing it wrong, but, sure enough, you could do it better. How? Using JUnit Rules!

## The Basics

Well, before we learn all that, let's start with the basics, shall we?

#### **Timeouts**

Let's take a simple example: Suppose that you want to set a timeout for all test methods in a given class, an easy way to do that is like this:

```
// ...
}
// ...
```

Besides that you repeated yourself tons of times, if you want to change this timeout, you will have to change it in all methods. There is no need to do that. Just use the Timeout Rule:

```
public class BlahTest {
        @Rule
        public Timeout timeout = new Timeout(2000);
        @Test
        public void testA() throws Exception {
                // ...
        public void testB() throws Exception {
                // ...
        public void testC() throws Exception {
                // ...
        @Test
        public void testD() throws Exception {
                // ...
        public void testE() throws Exception {
                // ...
        // ...
}
```

### **Temporary Folder**

Have you ever needed to do some test that uses File and/or needed a temporary file/folder? TemporaryFolder to the rescue:

```
public class BlahTest {
     @Rule
     public TemporaryFolder tempFolder = new TemporaryFolder();
```

```
@Test
public void testIcon() throws Exception {
    File icon = tempFolder.newFile("icon.png");
    // do something else...
}
```

## **Expected Exceptions**

Ever needed more control on exceptions? Try the ExpectedException rule:

## **Custom Rules**

That's neat, but... what if you need something else... something more "custom"? Well, you can implement your own rules by implementing the Testrule interface, for example, a Rule that init Mockito mocks (not very useful):

```
@RequiredArgsConstructor
public class MockRule implements TestRule {
   private final Object target;

public Statement apply(Statement base, Description description) {
    return new Statement() {
        @Override
        public void evaluate() throws Throwable {
            MockitoAnnotations.initMocks(target);
            base.evaluate();
        }
    };
}
```

To use it, you just need to declare that rule in your test class:

#### **External Resources**

Returning to the example of this post's first paragraph, you can also have custom external resources rules by extending the ExternalResource class:

```
public class MyServer extends ExternalResource {
    @Override
    protected void before() throws Throwable {
        // start the server
    }
    @Override
    protected void after() {
        // stop the server
        }
}
```

I believe that this makes more sense with Integration Tests, though. Also, in this case, you probably would not want/need to start and stop the server before and after each test method, right? So, you can use the @classRule annotation:

**Attention**: Note that when you use @classRule, your rule instance should be static, just like @Beforeclass and @Afterclass methods.

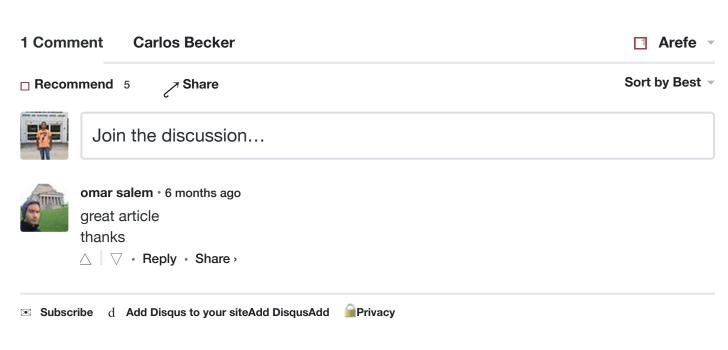
## **Going Further**

That's the basics that will save you tons of abstract classes and ugly code. I would also recommend you to take a good read at the junit wiki. If you have any question,

don't exitate to comment bellow, I will surely try to answer them. in



#### Your ad here.



#### **Related Posts**

**A Repository Graveyard** 

**Charting Repository Stars** 

**GKE** in production

Stuff written by Carlos Alexandro Becker.

</> available on Github.