About Rules and Runners

@StefanBirkner

Rule

- Code Duplication
- Before/After
- Separation of Concerns

RuleIntroTest

```
public class RuleIntroTest {
    @Rule
    public final TestRule timeout = new Timeout(1_000);

    @Test
    public void successfulTest() throws Exception {
        //nothing to do
    }

    @Test
    public void failingTest() throws Exception {
        Thread.sleep(2_000);
    }
}
```

Example – TemporaryFolder

```
public class TemporaryFolderTest {
    @Rule
    public final TemporaryFolder folder = new TemporaryFolder();

    @Test
    public void writesToFile() throws Exception {
        File file = folder.newFile();
        Files.write(file.toPath(), asList("dummy text"));
    }
}
```

Example - ProvideSystemProperty

```
public class SystemPropertyTest {
    @Rule
    public final ProvideSystemProperty myPropertyHasMyValue
            = new ProvideSystemProperty("MyProperty", "MyValue");

    @Test
```

```
    public void overrideProperty() {
        assertEquals("MyValue", System.getProperty("MyProperty"));
    }
}
```

Example - HttpServer

```
public class ServerTest {
    @Rule
    public final HttpServer server = new HttpServer();

    @Test
    public void checksStatusCodeOkForOurFile() throws Exception {
        URL url = new URL("http://localhost:8080/test.html");
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.connect();
        assertEquals(200, connection.getResponseCode());
    }
}
```

Statement

```
void evaluate() throws Throwable
```

TestClass

```
public class ATest {
    @Test
    public void firstTest() {
    }

    @Test
    public void secondTest() {
    }
}
```

Test as Statement

```
Statement firstTest = () -> {
    ATest aTest = new ATest();
    aTest.firstTest();
}
```

Test Result

```
public class TestResultTest {
```

```
    @Test
    public void success() {
        //does not throw an exception
    }

    @Test
    public void failure()  {
        assertTrue(false);//throws AssertionError
    }

    @Test
    public void ignored() {
        assumeTrue(false);//throws AssumptionViolatedException
        assertTrue(false);
    }

    @Test
    public void error() throws Exception {
        throw new Exception();
    }
}
```

Rule

```
                  +------+
{ Statement } --> | Rule | --> { Statement }
                  +------+
```

TestRule interface

```
Statement apply(Statement base, Description description)
```

Wrapping a Statement

```
@Rule
public final TestRule rule = (statement, description) -> new Statement() {
    @Override
    public void evaluate() throws Throwable {
        statement.evaluate();
    }
};
```

Verify Exception

```
public static class ExpectedException implements TestRule {
    private final Class<?> type;

    public ExpectedException(Class<?> type) {
```

```
        this.type = type;
    }

    @Override
    public Statement apply(Statement base, Description description) {
        return new Statement() {
            @Override
            public void evaluate() throws Throwable {
                try {
                    base.evaluate();
                } catch (Throwable t) {
                    if (!type.isAssignableFrom(t.getClass())) {
                        throw t;
                    }
                }
            }
        };
    }
}
```

Verify System.out

```
public class LogStdOutTest {
    @Rule
    public final StandardOutputStreamLog log = new StandardOutputStreamLog();

    @Test
    public void test() {
        System.out.println("hello world");
        assertEquals("hello world\n", log.getLog());
    }

    public static class StandardOutputStreamLog implements TestRule {
        private final ByteArrayOutputStream log = new ByteArrayOutputStream();
        private PrintStream originalStream;

        @Override
        public Statement apply(Statement base, Description description) {
            return new Statement() {
                @Override
                public void evaluate() throws Throwable {
                    originalStream = System.out;
                    try {
                        System.setOut(new PrintStream(log));
                        base.evaluate();
                    } finally {
                        System.setOut(originalStream);
                    }
                }
            };
        }

        public String getLog() {
            try {
                return log.toString("UTF-8");
```

```
            } catch (UnsupportedEncodingException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

Chaining Rules

```
public class IdealoRule implements TestRule {
    private final TestRule chain = RuleChain.outerRule(new Timeout(1_000))
        .around(new DoNotWriteToStdOutRule());

    @Override
    public Statement apply(Statement base, Description description) {
        return chain.apply(base, description);
    }
}
```

Runner

Executes the tests of a single test class.

Example Test

```
public class RunnerDemoTest {
    @Rule
    public ...

    @Before
    public ...

    @Test
    public successfulTest() {
    }

    @Test
    public failingTest() {
        assertTrue(false);
    }

    @Test
    @Ignore
    public ...

    @After ...
}
```

Create a Runner - The Test

```
@RunWith(Java8Runner.class)
public class Java8Test {
  public static List<Test> tests = asList(
    test("true is not false", () -> Assert.assertNotEquals(true, false)),
    test("sets value", () -> {
      int a = 1;
      Assert.assertEquals(a, 1);
    }));
}
```

Create a Runner - Test Model

```
public interface TestStatement {
    void evaluate() throws Throwable;
}

public class Test {
    public final String name;
    public final TestStatement statement;

    public static Test test(String name, TestStatement statement) {
        return new Test(name, statement);
    }

    public Test(String name, TestStatement statement) {
        this.name = name;
        this.statement = statement;
    }
}
```

Create a Runner - The Runner

```
public class Java8Runner extends ParentRunner<Test> {
    public Java8Runner(Class<?> testClass) throws InitializationError {
        super(testClass);
    }

    @Override
    protected List<Test> getChildren() {
        try {
            return (List<Test>)
getTestClass().getJavaClass().getField("tests").get(null);
        } catch (IllegalAccessException | NoSuchFieldException e) {
            throw new RuntimeException(e);
        }
    }

    @Override
    protected Description describeChild(Test child) {
        return Description.createTestDescription(getTestClass().getJavaClass(),
child.name);
    }
```

```
    @Override
    protected void runChild(Test child, RunNotifier notifier) {
        Description description = describeChild(child);
        EachTestNotifier eachTestNotifier = new EachTestNotifier(
                notifier, description);
        eachTestNotifier.fireTestStarted();
        try {
            child.statement.evaluate();
        } catch (Throwable t) {
            eachTestNotifier.addFailure(t);
        } finally {
            eachTestNotifier.fireTestFinished();
        }
    }
}
```

And now?

- Use Rules
- Write Rules
- Write Runners

Anything else?

Create (micro-)libraries!

Join the idealo team

- Import and process 1 billion records.
- Alexa Rank #32 in Germany
- http://jobs.idealo.de

Sources

Most code that is used in this talk is a narrow version of existing code. Please visit the following projects for the complete code.

- http://junit.org/
- https://github.com/stefanbirkner/system-rules
- https://github.com/stefanbirkner/server-rule
- https://github.com/stefanbirkner/talkative-junit-tests