HELLO JUNIT5

QUICKSTART

JUnit5 with IntelliJ

Create a new Gradle project and add the dependency:

```
testCompile "org.junit.jupiter:junit-jupiter-api:5.0.0-M4"
```

https://stackoverflow.com/questions/3466850/complex-password-regular-expression /* The password must then contain characters from at least 3 of the following 4 rules: Upper case Lower case Numbers Non-alpha numeric */ ^(?:(?=.*[a-z])(?:(?=.*[A-Z])(?=.*[\d\W])|(?=.*\W)(?=.*\d))|(?=.*\W)(?=.*[A-Z])(?=.*\d)).{8,}$

NEW ANNOTATIONS

- @Test
- @BeforeEach (was @Before)
- @BeforeAll (was @BeforeClass)
- @Disabled (was @Ignore)

No more properties on annotations

```
@Test(expect=Exception.class))
```

READABILITY

Tests don't have to be public

```
class PasswordPolicyTest {

    PasswordPolicy passwordPolicy;

    @BeforeEach
    void setUp() {
        passwordPolicy = new PasswordPolicy();
    }

    @Test
    void shouldMatchPasswordPolicy() {
        assertTrue(passwordPolicy.validatePassword("ABCdef9!"));
    }

    @Test
    void shouldNotMatchPasswordPolicy() {
        assertFalse(passwordPolicy.validatePassword("123456"));
    }
}
```

JAVA 8 LAMBDA ASSERTIONS

```
@Test
void shouldMatchPasswordPolicy() {
    assertAll(
            () -> assertTrue(passwordPolicy.validatePassword("ABCdef9!")),
            () -> assertTrue(passwordPolicy.validatePassword("VeryLongPasswordWithNumb
            () -> assertTrue(passwordPolicy.validatePassword("jdhkfgjhdf/()!!888AAA"))
    );
}
```

PARAMETERIZED TESTS

```
testCompile "org.junit.jupiter:junit-jupiter-params:5.0.0-M4"
```

```
@ParameterizedTest
@MethodSource(names = "passwordsToTest")
void shouldMatchPasswordPolicy(String password, Matcher matcher) {
    assertThat(passwordPolicy.matchPasswordPolicy(password), matcher);
}

static Stream<Arguments> passwordsToTest() {
    return Stream.of(
            ObjectArrayArguments.create("ABde0eee!", is(true)),
            ObjectArrayArguments.create("dfnfg", is(false))
    );
}
```

EXCEPTION TESTING

```
@Test
void shouldThrowExceptionWhenNull() {
    Throwable exception = assertThrows(IllegalArgumentException.class,
            () -> passwordPolicy.validatePassword(null));

    assertEquals("password may not be null", exception.getMessage());
}
```

NESTED TESTS

BDD style for JUnit. Finally :-)

```
@Nested
class UserControllerNestedTest {

    UserController userController;

    @BeforeEach
    void setUp() {
        userController = new UserController();
    }

    @Nested
    @DisplayName("An anonymous user")
    class anonymousUser {

        @Nested
        @DisplayName("creates a user account")
        class createsAUserAccount {

            @Test
            void returnsCreated() {
                //...
            }

            @Test
```

ASSUMPTIONS

Assumptions can be used to conditionally run a test:

```
@Test
void thatEncodedPasswordsAreDifferent() {
    assumingThat("CI".equals(System.getenv("ENV")),
    () -> {
        String encoded1 = encoder.encodePassword("some-password");
        String encoded2 = encoder.encodePassword("some-password");
        assertNotEquals(encoded1, encoded2, "passwords should not match");
    });
}
```

compile('org.springframework.boot:spring-boot-starter-security')

TAGS

```
@Test
@Tag("slow")
void thatEncodedPasswordsAreDifferent() {
    String encoded1 = encoder.encodePassword("some-password");
    String encoded2 = encoder.encodePassword("some-password");
    assertNotEquals(encoded1, encoded2, "passwords should not match");
}
```

GRADLE PLUGIN

The JUnit Platform plugin has to be enabled
and the Jupiter Engine must be available as testRuntime:

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies { classpath 'org.junit.platform:junit-platform-gradle-plugin:1.0.0-M4
}
apply plugin: 'org.junit.platform.gradle.plugin'

dependencies {
    testCompile "org.junit.jupiter:junit-jupiter-api:5.0.0-M4"
    testRuntime "org.junit.jupiter:junit-jupiter-engine:5.0.0-M4"
}

junitPlatform {
    filters {
        tags { exclude 'slow' }
    }
}
```

TESTFACTORY

Tests can be created at runtime (!)
Use-cases: TCK, real test data, ...

```
@TestFactory
Stream<DynamicTest> dynamicTests() {
    return IntStream.of(1, 2, 3, 4, 6).mapToObj(
        i -> dynamicTest(i + " should be equal", () -> assertTrue(i % 2 == 0))
    );
}
```

EXTENSIONS

Extensions replace *Runners* and *Rules*.

```
public class TimingExtension implements BeforeTestExecutionCallback, AfterTestExecutio

    private static final Logger LOG = Logger.getLogger(TimingExtension.class.getName()

    @Override
    public void beforeTestExecution(TestExtensionContext context) throws Exception {
        getStore(context).put(context.getTestMethod().get(), System.currentTimeMillis(
    }

    @Override
    public void afterTestExecution(TestExtensionContext context) throws Exception {
        Method testMethod = context.getTestMethod().get();
        long start = getStore(context).remove(testMethod, long.class);
        long duration = System.currentTimeMillis() - start;

        LOG.info(() -> String.format("Method [%s] took %s ms.", testMethod.getName(),
    }

    private Store getStore(TestExtensionContext context) {
```

SCENARIO TESTS

(Planned for M5)

```
@ScenarioTest
class WebSecurityScenarioTest {

    @Step(next = "login")
    void visitPageRequiringAuthorizationWhileNotLoggedIn() {
        // attempt to visit page which requires that a user is logged in
        // assert user is redirected to login page
    }

    @Step(next = "visitSecondPageRequiringAuthorizationWhileLoggedIn")
    void login() {
        // submit login form with valid credentials
        // assert user is redirected back to previous page requiring authorization
    }

    @Step(next = "logout")
    void visitSecondPageRequiringAuthorizationWhileLoggedIn() {
        // visit another page which requires that a user is logged in
        // assert user can access page
    }

    @Step(next = END)
    void logout() {
        // visit logout URL
```

NO HAMCREST DEPENDENCY

Use AssertJ!

```
testCompile 'org.assertj:assertj-core:3.8.0'
```

JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage

JUNIT PLATFORM

Launcher defines an API for Console, IDEs and Build Tools to find and execute tests.

TestEngine API to support different test frameworks (e. g. JUnit Jupiter, JUnit (4) Vintage, Spock, Spek ...)

SPEK

Spek is a TestEngine implementation to define Specifications in Kotlin.

```
testCompile 'org.jetbrains.spek:spek-api:1.1.2'
testRuntime 'org.jetbrains.spek:spek-junit-platform-engine:1.1.2'
```

```
object PasswordEncoderSpec : Spek({

    describe("password encoder") {
        val passwordEncoder = PasswordEncoder()

        on("encoding a password twice") {
            val password = "secret"
            val encoded1 = passwordEncoder.encodePassword(password)
            val encoded2 = passwordEncoder.encodePassword(password)

            it("should have different encoded values") {
                assertNotEquals(encoded1, encoded2)
            }
        }

        describe("encoding null") {
            it("should throw Exception") {
                assertFailsWith(IllegalArgumentException::class, {
                    passwordEncoder.encodePassword(null)
                })
            }
        }
    }

})
```