Unit testing

Typy testů

- https://blog.frankel.ch/different-kinds-testing/

Základní knihovny

- JUnit – aktuálně verze 5, ale nejvíc se stále používá verze 4.
- AssertJ – fluent assertions (lepší než Hamcrest)
- Mockito – mocking framework
- PowerMock – když Mockito nestačí
  - Mockování statických metod, konstruktorů atd ... brrrr

- Anebo: Spock http://spockframework.org/
  - https://speakerdeck.com/szpak/is-spock-still-needed-in-the-time-of-junit-5

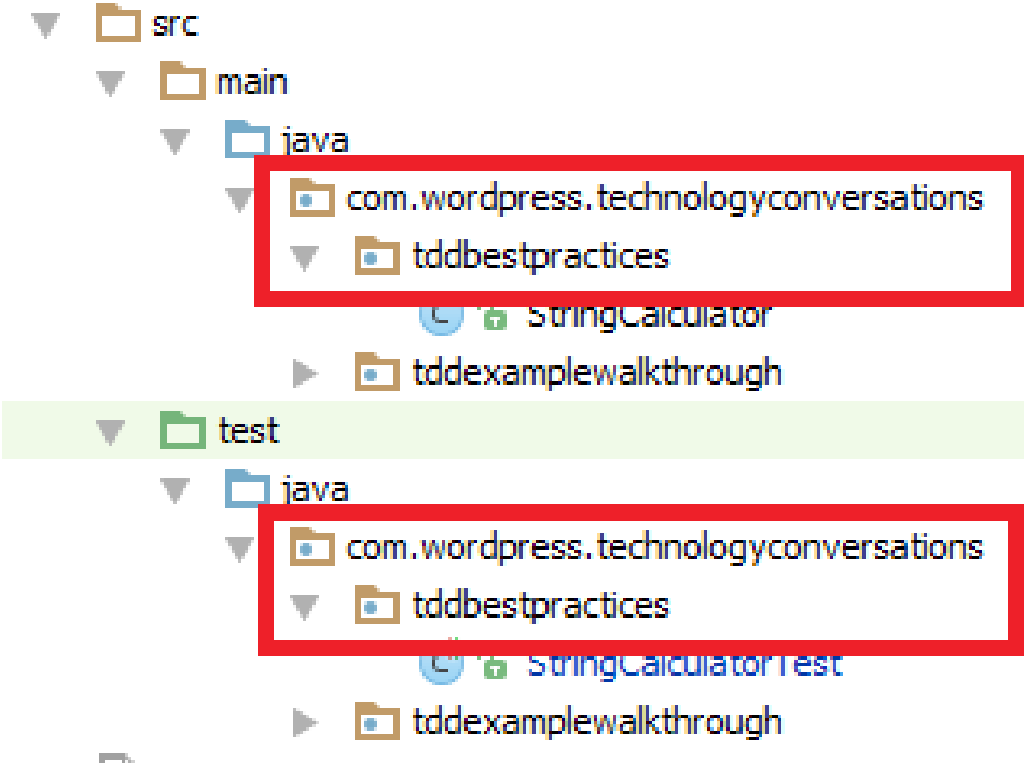
- Další:
  - https://github.com/akullpp/awesome-java#testing

Podpora v IDE

- IntelliJ Idea:
  - CTRL + SHIFT + T
- Eclipse:
  - https://moreunit.github.io/MoreUnit-Eclipse/

Best Practices

- Test class naming conventions:
  - Testovaná třída je například ItemService, název test třídy bude ItemServiceTest, navíc je dobrý nápad, aby test byl ve stejném balíčku, abychom pomocí něj mohli lehce testovat package-private a protected metody.
  - Je best practice, aby se testy "nepřibalovaly" k výslednému JAR / WAR souboru. K tomu je best practice například u Mavenu dávat třídy do adresáře src/main/java a testy do src/test/java a mít testovanou třídu i test samotný "ve stejném balíčku":



Best Practices

- Test method naming conventions:
  - Je dobrý nápad mít v názvu testu co má test vlastně dělat:

```
@Test
public final void whenSemicolonDelimiterIsSpecifiedThenItIsUsedToSeparateNumbers() {
    Assert.assertEquals(3+6+15, StringCalculator.add("//;n3;6;15"));
}
```

  - Poznámka: taky jsem viděl konvenci pojmenování:
    - `when_semicolon_delimiter_is_specified_then_it_is_used_to_separate_numbers`
  - Vypadá to sice blbě, ale když pak člověk vidí že nějaké testy skončily chybou, tak na první pohled vidí o jaký test se skutečně jedná.
  - V JUnit 5 je anotace `@DisplayName("název testu")`, se kterou už toto nemá význam řešit. :-)

Best Practices

- TDD flow:
  - Napsat test předtím, než se napíše implementace metody.
  - Psát nový kód pouze když test selže (čili nejprve jsme napsali test a poté dopsali implementaci).
  - Pokaždé když se změní kód, tak spustit všechny testy.
  - Všechny testy by měly projít předtím, než se napíše nový test.
  - Refactoring kódu provádět pouze, když všechny testy procházejí.
- https://technologyconversations.com/2013/12/24/test-driven-development-tdd-best-practices-using-java-examples-2/

Best Practices

- Struktura testu:
  - Je best practice mít co nejmenší množství assertů uvnitř testu.
- Dobře:

```
@Test
public final void whenOneNumberIsUsedThenReturnValueIsThatSameNumber() {
    Assert.assertEquals(3, StringCalculator.add("3"));
}

@Test
public final void whenTwoNumbersAreUsedThenReturnValueIsTheirSum() {
    Assert.assertEquals(3+6, StringCalculator.add("3,6"));
}
```

Best Practices

- Dobře:

```
@Test
public final void whenNegativeNumbersAreUsedThenRuntimeExceptionIsThrown() {
    RuntimeException exception = null;
    try {
        StringCalculator.add("3,-6,15,-18,46,33");
    } catch (RuntimeException e) {
        exception = e;
    }
    Assert.assertNotNull("Exception was not thrown", exception);
    Assert.assertEquals("Negatives not allowed: [-6, -18]", exception.getMessage());
}
```

Best Practices

- Špatně:

```
@Test
public final void whenAddIsUsedThenItWorks() {
    Assert.assertEquals(0, StringCalculator.add(""));
    Assert.assertEquals(3, StringCalculator.add("3"));
    Assert.assertEquals(3+6, StringCalculator.add("3,6"));
    Assert.assertEquals(3+6+15+18+46+33, StringCalculator.add("3,6,15,18,46,33"));
    Assert.assertEquals(3+6+15, StringCalculator.add("3,6n15"));
    Assert.assertEquals(3+6+15, StringCalculator.add("//;n3;6;15"));
    Assert.assertEquals(3+1000+6, StringCalculator.add("3,1000,1001,6,1234"));
}
```

Poznámka: S JUnit 5 se to dá velice lehce vyřešit pomocí metody assertAll():
https://junit.org/junit5/docs/current/user-guide/#writing-tests-assertions
(dá se také kombinovat s AssertJ)

Best Practices

- Struktura testu:
  - Ve výchozím nastavení není definované, v jakém pořadí se budou jednotlivé testy v testovací třídě vykonávat. Je tudíž dobrý nápad, aby byly jednotlivé testy na sobě nezávislé.
  - Vykonání jednoho testu by měla být záležitost maximálně jednotek milisekund – díky tomu je pak možné často spouštět celou baterii testů (všechny testy).
    - K tomu je dobré používat mocky
  - Existují metody s JUnit 4 (5) anotacemi: @Before (@BeforeEach), @After (@AfterEach), @BeforeClass (@BeforeAll), @AfterClass (@AfterAll) a je vhodné je využívat
    - https://stackoverflow.com/questions/20295578/difference-between-before-beforeclass-beforeeach-and-beforeall
  - To, že v testu dochází k duplikování kódu není na škodu, důležitější než zamezení duplikování kódu je čitelnost kódu testu. Proto se používání dědičnosti u testů moc nedoporučuje.

Best Practices

- https://phauer.com/2019/modern-best-practices-testing-java/

@RunWith, @Rule

- Když je třída oanotovaná s anotací @RunWith (nebo dědí ze třídy s touto anotací), pak bude spouštět Vaše testy tato třída a nikoli výchozí built-in JUnit runner.
  - @RunWith má základní omezení – může se specifikovat pouze jeden! Spolu s integračními testy ve Springu ale můžete chtít použít například Mockito a obojí používá @RunWith. Jak z toho ven? Použít @Rule:
    - http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#testcontext-junit4-rules
    - http://www.vogella.com/tutorials/Mockito/article.html
  - Poznámka: V JUnit 5 je anotace @ExtendWith, která může registrovat více "Extensions":
    - https://stackoverflow.com/questions/56268274/order-of-multiple-extensions-in-junit-5

JUnit 5

- Další zajímavé featury v JUnit 5:
  - https://rieckpil.de/five-junit-5-features-you-might-not-know-yet/
  - https://www.baeldung.com/parameterized-tests-junit-5

Mockito

- https://www.vogella.com/tutorials/Mockito/article.html
- https://www.baeldung.com/mockito-series
- https://github.com/mockito/mockito/wiki/How-to-write-good-tests

Mockito – rozšíření

- Mockování metody, která vrací void:
  - Pomocí Mockito.doNothing()
  - https://www.baeldung.com/mockito-void-methods
- Mocking vs. Spying:
  - https://stackoverflow.com/questions/12827580/mocking-vs-spying-in-mocking-frameworks
- Výchozí návratové hodnoty mockovaných metod:
  - Obyčejně null
  - V případě kolekcí: prázdná kolekce
  - V případě primitivních datových typů jsou hodnoty stejné jako při implicitní inicializaci

Mockito – rozšíření

- Implementace if-else:

```
Mockito.when(httpClientMock.get(Mockito.anyString()))
        .thenReturn(500);
Mockito.when(httpClientMock.get("https://google.com"))
        .thenReturn(200);
```

  - Když se zavolá metoda get() s parametrem "https://google.com", tak bude návratový kód 200, v opačném případě 500.
  - Pozor!!! Záleží na pořadí when() operací!!!

Mockito – rozšíření

- Když se má pokaždé vracet stejná hodnota (200):

```
Mockito.when(httpClientMock.get(Mockito.anyString()))
        .thenReturn(200);
```

- Když se poprvé má vrátit 200 a při dalších voláních se vrací 500:

```
Mockito.when(httpClientMock.get(Mockito.anyString()))
        .thenReturn(200);
        .thenReturn(500);
```