Testy jednostkowe infoShare ACADEMY

Hello

Michał Samujło

engineering manager @ Nordea Trading Technology

Dlaczego warto pisać testy?

… skąd wiesz, że twój kod działa prawidłowo?

Jak przetestować automat z napojami?

Jak przetestować automat z napojami?

- test suite
  - *zestaw testów*
- test case
  - *przypadek testowy*
- assertion
  - *założenie*

Rodzaje testów

- akceptacyjne
- integracyjne
- jednostkowe

JUnit

```
git clone https://github.com/infoshareacademy/jjdd8-materialy-junit.git
git checkout before
```

Uruchamianie testów

- `com.infoshare.junit5.simple.PurchaseTests`

- maven i plugin surefire

```
/> mvn test
```

- IntelliJ
  - **Ctrl+Shift+F10** - w zależności od umiejscowienia kursora uruchomi wybrany test lub wszystkie testy
  - **Ctrl+F5** - uruchom ponownie
  - **Ctrl+Shift+T** - utwórz nowy test

Jak napisać przypadek testowy?

- `@Test` - adnotacja przypadku testowego
- `assertTrue(), assertFalse(), assertEquals(), assertSame()` - podstawowe asercje
- `fail()` - powoduje porażkę przypadku testowego

Reguły weryfikacji testów

Exactly one reason to fail

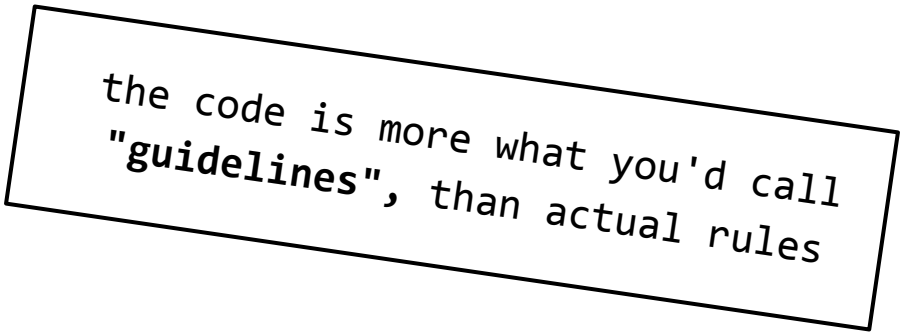
Nie używaj wielu asercji w jednej metodzie

Zero, one, many rule

Sprawdź zachowanie obiektu dla zera, dla jednej i dla wielu wartości

No logic in test

Nie używaj if, for, switch



Cechu dobrego testu jednostkowego

- automatyczny
- powtarzalny
- łatwy do napisania
- tani w utrzymaniu
- szybki

Cykl życia zestawu testów

- **`@BeforeAll`** - uruchom metodę przed zestawem testów
- **`@BeforeEach`** - uruchom metodę przed przypadkiem testowym
- **`@AfterEach`** - uruchom metodę po przypadku testowym
- **`@AfterAll`** - uruchom metodę po zestawie testów
- **`@Disabled`** - zignoruj przypadek testowy

Konwencja nazw

Klasy nazywamy zgodnie z konwencją maven-surefire-plugin

```
**/Test*.java, **/*Test.java, **/*Tests.java, **/*TestCase.java

  class TestDispenser { //... }

  class CalculatorTest { //... }

  class DispanserTests { //... }

  class VendingMachineTestCase { //... }
```

Konwencja nazw

Metody nazywamy tak, żeby można było zrozumieć co testują

- `should_ExpectedBehavior_When_StateUnderTest`

  `public void` **`should_dispenseProduct_when_givenExactAmount`**`(){...}`

- `when_StateUnderTest_Expect_ExpectedBehavior`

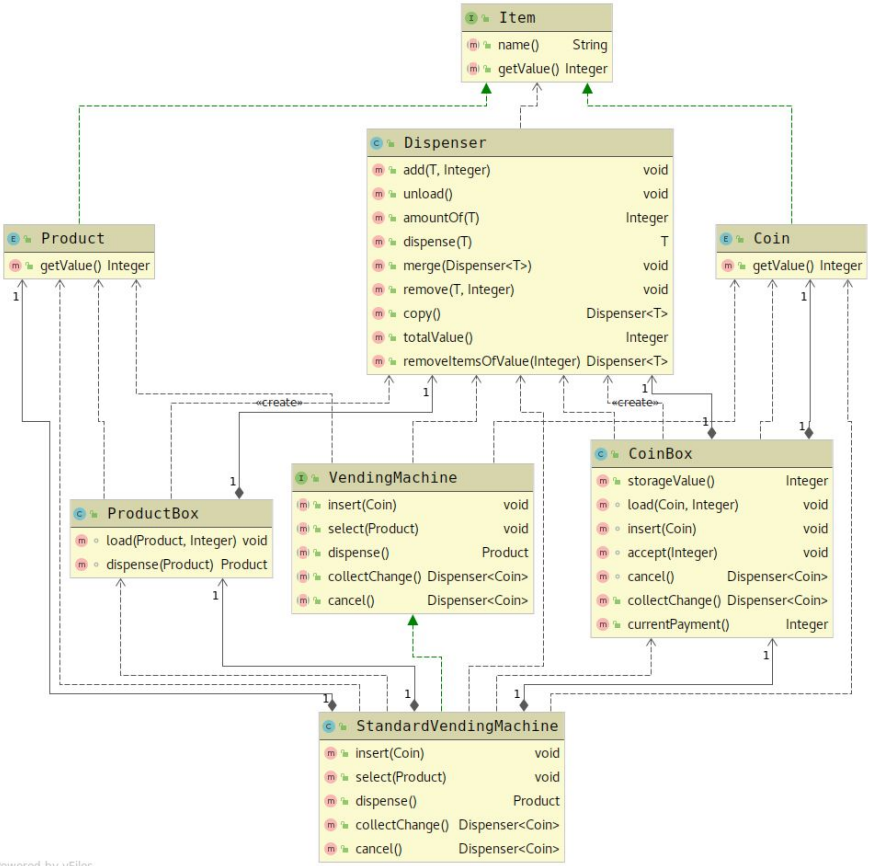
  `public void` **`when_exactAmountInserted_expect_productDispensed`**`() {...}`

- **`@DisplayName, @Nested`** - umożliwiają użycie dowolnych opisów

Ćwiczenia

Object oriented vending machine

- `StandardVendingMachine`
- `CoinBox`
- `ProductBox`
- `Dispenser`
- `Item`



JUnit5 ćwiczenia

- `com.infoshare.junit5.PurchaseTests`

  - zaimplementuj wszystkie TODO używając podstawowych assercji

    ```
    // TODO
    fail();
    ```

  - https://junit.org/junit5/docs/current/user-guide/#writing-tests-assertions

    `assertNotNull, assertTrue, assertFalse, assertEquals, assertThrows`

AssertJ

- prosty w użyciu, ze względu na assercje dostosowane do typu obiektu

```
assertThat(actual).is(expected);
```

- IDE podpowiada metody
    - lepsze opisy błędów
- tylko jeden import
    - import static org.assertj.core.api.Assertions.*;

AssertJ ćwiczenia

- `com.infoshare.junit5.domain.DispenserTests`
- zaimplementuj wszystkie TODO używając podstawowych assercji

```
// TODO
fail();
```

- https://assertj.github.io/doc/

AssertJ ćwiczenia

- `com.infoshare.junit5.domain.BookTests`
- zaimplementuj wszystkie TODO używając AssertJ

```
// TODO
fail();
```

- https://assertj.github.io/doc/

Testy parametryzowane

@ParametrizedTest, @EnumSource, @ValueSource, assertAll()

- com.infoshare.junit5.CoinBoxTests
- com.infoshare.junit5.domain.DispenserLoadingTests
  - TODO should_load_and_count_value_of_all_products

@CsvSource

- com.infoshare.junit5.domain.DispenserTests
  - Should_calculate_total_value_of_ones_and_twos
  - TODO should_calculate_total_value_of_fives_and_tens

@MethodSource, Arguments

- com.infoshare.junit5.domain.DispenserTests
  - Should_calculate_change
  - TODO should_calculate_total_value_of_five_and_tens

JUnit5 inne annotacje

Co jeżeli chcemy uruchomić testy w innej kolejności?

- `@TestMethodOrder, @Order`

Co zrobić, jeżeli wykonanie metody powinno się skończyć w określonym czasie?

- `@Timeout(value = 100, unit = TimeUnit.MILLISECONDS)`

Dzieki

You can find me at
michal.samujlo@gmail.com