



Testy jednostkowe

jUnit, AssertJ, Mockito

Gdańsk, 27 luty 2018

Plan szkolenia

- 1) Testy automatyczne – co, jak i dlaczego
- 2) *jUnit* - pierwsze starcie
- 3) *JUnit* - przygotowanie danych testowych
- 4) Podstawy weryfikacji działania testu
- 5) Weryfikacje z użyciem biblioteki *AssertJ*
- 6) Budowa namiastek z użyciem biblioteki *Mockito*
- 7) Przykład testów serwisu bankowego

Dlaczego piszemy testy?



Rodzaje testów

- akceptacyjne
- integracyjne
- jednostkowe



Zalety testów jednostkowych

- weryfikują prawidłowość działania kodu
- zmuszają do myślenia o tym jak kod jest używany
- pozwalają bezpiecznie wprowadzać zmiany

Cechu dobrego testu jednostkowego

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

Narzędzia do testów

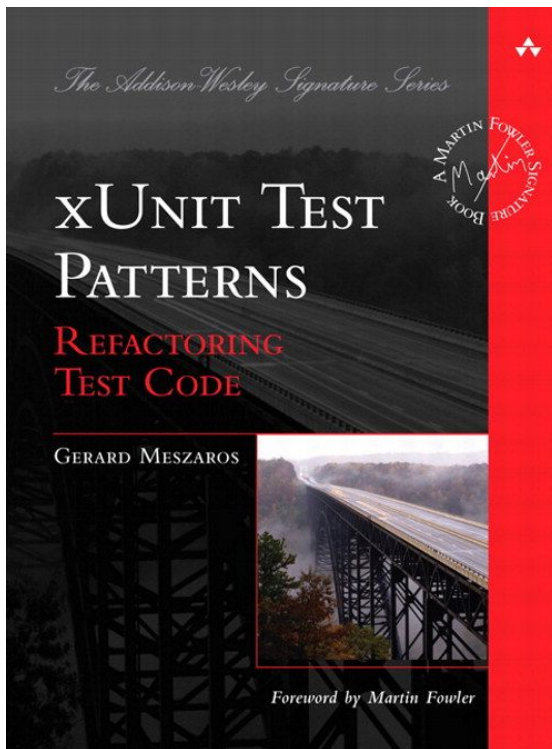
```
/> git checkout before  
> mvn clean  
> mvn compile  
> mvn exec:exec
```

```
com.infoshare.junit.automotive.CarTesterApp
```

Problemy

- czytelność?
- jak uruchomić wiele testów?
- jak przekazać wynik do innego programu n.p. maven?

XUnit – biblioteki do testów



jUnit – pierwszy test

```
/> git checkout before  
> mvn clean  
> mvn test
```

com/infoshare/junit/\$1_first_test/CarUnitTest.java

Ćwiczenie:

- uruchom test z IntelliJ Idea
- Ctrl + Shift + F10 – uruchom test
- Ctrl + F5 – powtórz ostatnie testy
- Ctrl + Shift + T – nowy test

jUnit – @Test i Assert

```
import org.junit.Assert
import org.junit.Test;

public class UnitTest {

    @Test
    public void description_of_test() {
        boolean condition = true;
        assertTrue("description of the assertion", condition);
    }

}
```

jUnit – pierwszy test

```
@Test
public void toyota_engine_should_be_running_after_ignition() throws Exception {
    // given
    Car sut = new CarFactory().forBrand(Brand.TOYOTA).build();
    // when
    sut.ignite();
    // then
    assertTrue("OMG! Car is not running after ignition", sut.isRunning());
}
```

Ćwiczenie:

- napisz test sprawdzający, że w Hondzie startuje silnik
- napisz test sprawdzający, że Toyota nie przekracza norm zanieczyszczeń
- napraw test sprawdzający poziom emisji spalin Volkswagena

jUnit – podstawowe adnotacje

```
@Test(timeout = 1000)
@Test(expected = Exception.class)
@Ignore
```

```
com/infoshare/junit/$1_first_test/CarUnitTest.java
```

Ćwiczenie:

- testy nie powinny trwać dłużej niż sekundę, zignoruj te które trwają dłużej
- upewnij się, że aplikacja nie zadziała, jeżeli ktoś użyje fabryki samochodów dla nie wspieranej marki

jUnit – podstawowe adnotacje

```
@Test(timeout = 1000)
@Test(expected = Exception.class)
@Ignore
```

```
com/infoshare/junit/$9_bowling/BowlingGameTest.java
```

Ćwiczenie:

- Napisz testy jednostkowe do programu obliczającego punktację w grze w kręgle

jUnit – przygotowanie testów

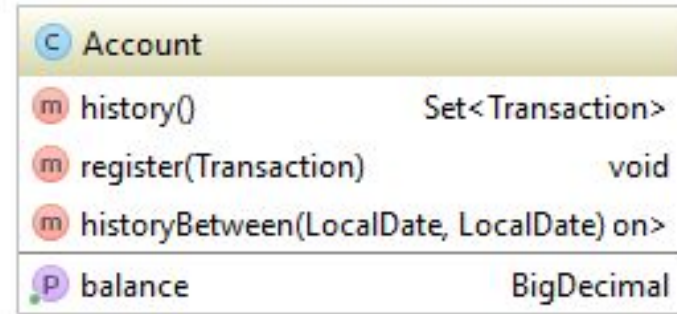
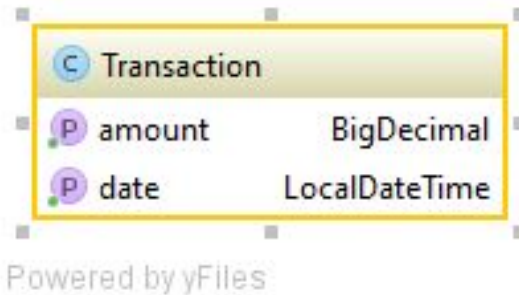
```
/> git checkout before  
/> mvn clean
```

```
com/infoshare/junit/$2_test_fixture/TransactionSetupTest.java
```

Ćwiczenie:

- uruchom test - co jest nie tak?

Testowana domena



jUnit – struktura klasy testowej

@BeforeClass

static method

@Before

method

@After

method

@Test

test method

@Before

method

@After

method

@Test

test method

@AfterClass

static method

jUnit – przygotowanie testów

@BeforeClass

@Before

@After

@AfterClass

com/infoshare/junit/\$2_test_fixture/TransactionSetupTest.java

Ćwiczenie:

- Przyspiesz test używając powyższych anotacji

jUnit – podstawowe asercje

```
import static org.junit.Assert
```

```
assertTrue
```

```
assertFalse
```

```
assertNull
```

```
assertNotNull
```

```
assertEquals
```

```
assertArrayEquals
```

```
assertSame
```

```
assertNotSame
```

```
fail
```

Reguły weryfikacji testów

- Test powinien nie przechodzić dokładnie z jednego powodu
 - nie używaj wielu asercji w jednej metodzie
- Zero, jeden, wiele
 - sprawdź zachowanie obiektu dla zera, dla jednej i dla wielu wartości
- Testy nie powinny mieć logiki
 - nie używaj if, for, switch

To tylko wytyczne, a nie żelazne reguły

Kapitan Barbossa, Piraci z Karaibów

jUnit – podstawowe asercje

`com/infoshare/junit/$3_basic_asserts/NewAccountTest.java`

`com/infoshare/junit/$3_basic_asserts/TransactionTest.java`

Ćwiczenie:

- zaimplementuj test “empty account should not have any transactions”
- sprawdź jak wyglądają komunikaty błędów dla różnych asercji

Jak przygotować dane?

`com/infoshare/junit/$2_test_fixture/TransactionSetupTest.java`

`com/infoshare/junit/$2_test_fixture/TestTransactions.java`

`com/infoshare/junit/$2_test_fixture/TransactionsBuilder.java`

Wzorce

- Object Mother - przygotowuje gotowy zestaw danych testowych
- Test Data Builder - pozwala łatwo tworzyć dane do testów

Biblioteka AssertJ

- Czytelniejszy kod
- Lepsze komunikaty błędów
- Definiowanie własnych weryfikacji
- Integracja z popularnymi bibliotekami - JodaTime, Guava

AssertJ – podstawowe asercje

```
import static org.assertj.core.api.Assertions.assertThat;  
  
assertThat(Object object)  
  
assertThat(Integer integer)  
  
assertThat(String string)  
  
assertThat(Collection collection)  
  
...
```

AssertJ – podstawowe asercje

```
import static org.assertj.core.api.Assertions.assertThat;
```

```
assertThat(game.score()).isGreaterThan(10);
```

```
assertThat(game.score()).isEqualTo(10);
```

```
assertThat(transactions)  
    .filteredOn(t->t.getAmount()>10)  
    .hasSize(10);
```

```
assertThat(transactions).  
    extracting(Transaction::getStatus).  
    containsOnly(TransactionStatus.SUCCESS);
```


AssertJ - weryfikacja wyjątków

@Test

```
public void testException() {  
    assertThatThrownBy(() -> { throw new Exception("boom!"); })  
        .assertInstanceOf(Exception.class)  
        .hasMessageContaining("boom");  
}
```

AssertJ - “miękkie” weryfikacje

@Test

```
public void host_dinner_party_where_nobody_dies() {  
    Mansion mansion = new Mansion();  
    mansion.hostPotentiallyMurderousDinnerParty();  
    // use SoftAssertions instead of direct assertThat methods  
    SoftAssertions softly = new SoftAssertions();  
    softly.assertThat(mansion.guests()).as("Living Guests").isEqualTo(7);  
    softly.assertThat(mansion.kitchen()).as("Kitchen").isEqualTo("clean");  
    softly.assertThat(mansion.library()).as("Library").isEqualTo("clean");  
    softly.assertThat(mansion.revolverAmmo()).as("Revolver Ammo").isEqualTo(6);  
    softly.assertThat(mansion.candlestick()).as("Candlestick").isEqualTo("pristine");  
    softly.assertThat(mansion.colonel()).as("Colonel").isEqualTo("well kempt");  
    softly.assertThat(mansion.professor()).as("Professor").isEqualTo("well kempt");  
    // Don't forget to call SoftAssertions global verification !  
    softly.assertAll();  
}
```

Biblioteka AssertJ

`com/infoshare/junit/$5_assertj/NewAccountMatchersTest.java`

`com/infoshare/junit/$5_assertj/TransactionMatchersTest.java`

Ćwiczenie:

- Przepisz testy z użyciem biblioteki AssertJ

AssertJ - Condition

```
static Set<String> JEDIS = newLinkedHashSet("Luke", "Yoda", "Obiwan");

Condition<String> jedi = new Condition<String>("jedi") {
    @Override
    public boolean matches(String value) {
        return JEDIS.contains(value);
    }
};

@Test
public void jedi() {
    assertThat(newLinkedHashSet("Luke", "Yoda", "Leia")).areAtLeast(2, jedi);
}
```

Zależności i dublerzy

THE GORILLA / BANANA PROBLEM

"The problem with object-oriented languages is they've got all this implicit environment that they carry around with them. You wanted a banana, but what you got was a gorilla holding the banana and the entire jungle."

~ Joe Armstrong, "Coders at Work"

Rodzaje dublerów

- fake - uproszczona implementacja zastępowanego obiektu n.p.: in-memory database
- stub - fake, ale pozwalający definiować predefiniowane zachowania
- mock - stub, ale pozwala weryfikować które zachowania zostały wywołane
- spy - jak mock, ale potrafi przekierować zachowania do prawdziwej implementacji

Biblioteka Mockito

Udawanie obiektu zwracanego z metody

```
@Test
public void reporter_should_contain_total_value() {
    // given
    OrderService srv = new OrderService(orderDatabase, warehouseConnector);

    OrderReporter reporter = new OrderReporter(srv);

    // when
    Summary summary = reporter.summarize();

    // then
    assertThat(summary.total).isEqualTo(200);
}
```

Biblioteka Mockito

Udawanie obiektu zwracanego z metody

```
@Test
public void reporter_should_contain_total_value() {
    // given
    ArrayList<Order> orders = OrdersMother.sampleOrders(100).ofValue(2).build();

    OrderService orderServiceMock = mock(OrderService.class);
    when(orderServiceMock.getOrders()).thenReturn(orders);

    OrderReporter reporter = new OrderReporter(orderServiceMock);

    // when
    Summary summary = reporter.summarize();

    // then
    assertThat(summary.total).isEqualTo(200);
}
```


Biblioteka Mockito

Udawanie obiektu dla wybranych argumentów

```
@Test
public void shouldMatchSimpleArgument() {
    // given
    ShippingScheduler schedulerMock = mock(ShippingScheduler.class);
    when(
        schedulerMock.getNumberOfOrdersScheduledOnDate(WANTED_DATE)
    ).thenReturn(VALUE_FOR_WANTED_ARGUMENT);

    // when
    int numberForWantedArgument =
    schedulerMock.getNumberOfOrdersScheduledOnDate(WANTED_DATE);
    int numberForAnyOtherArgument =
    schedulerMock.getNumberOfOrdersScheduledOnDate(any());

    // then
    assertThat(numberForWantedArgument).isEqualTo(VALUE_FOR_WANTED_ARGUMENT);
    assertThat(numberForAnyOtherArgument).isEqualTo(0); //default value for int
}
```

Biblioteka Mockito

Metody dopasowania argumentów

<code>any(), any(Class<T> clazz)</code>	Dowolny obiekt lub null
<code>anyBoolean(), anyString(), anyCollectionOf(Class<T> clazz), anyMapOf(Class<T> clazz)</code>	Dowolny obiekt danego typu lub null
<code>anyVararg()</code>	Dowolna lista argumentów
<code>eq(T value)</code>	Dowolny obiekt równy (equals) podanemu argumentowi
<code>isA(Class<T> clazz)</code>	Dowolny obiekt implementujący podaną klasę
<code>and(first, second), or(first, second), not(first)</code>	Operatory pozwalające łączyć inne metody dopasowania

Biblioteka Mockito

Wielokrotne wołanie udawanej metody

```
@Test
public void should_return_last_defined_value_consistently() {
    TransferBank bank = mock(TransferBank.class);

    when(bank.getCommercialAccountsCount()).thenReturn(100,200);

    assertThat(bank.getCommercialAccountsCount()).isEqualTo(100);
    assertThat(bank.getCommercialAccountsCount()).isEqualTo(200);
    assertThat(bank.getCommercialAccountsCount()).isEqualTo(200);
}
```

Biblioteka Mockito

Weryfikacja interakcji

```
@Test
public void verify_processing() throws Exception {
    TransferBank bank = new GenericBank();
    Account sourceAccount = bank.getAccountFor("Kent Beck");
    sourceAccount.register(new Transaction(BigDecimal.valueOf(10000),
LocalDateTime.now()));
    Account targetAccount = mock(Account.class);
    when(targetAccount.getBalance()).thenReturn(BigDecimal.valueOf(100000));
    TransactionsBuilder.totalOf(20).transferBetween(sourceAccount, targetAccount);

    // when
    bank.process();

    // then
    verify(targetAccount, times(20)).register(isA(Transaction.class));
}
```

Testowana domena

C Transaction	
m compareTo(Transaction)	int
m equals(Object)	boolean
m hashCode()	int
m toString()	String
P amount	BigDecimal
P target	Account
P source	Account
P status	TransactionStatus
P date	LocalDateTime

C Account	
m history()	Set<Transaction>
m register(Transaction)	void
m historyBetween(LocalDate, LocalDate)	action>
m transferTo(Account, BigDecimal, LocalDateTime)	
P balance	BigDecimal

I TransferBank	
m register(Transaction)	void
m process()	Collection<Transaction>
m getAccountFor(String)	Account

Biblioteka Mockito

```
src/test/java/com/infoshare/junit/$6_stubs  
src/test/java/com/infoshare/junit/$7_mock
```

Ćwiczenie:

- Uzupełnij testy z użyciem biblioteki mockito

Testy parametryzowane

```
src/test/java/com/infoshare/junit/$8_parametrized
```

Przykład testów produkcyjnych

```
/> git checkout after
```

```
src/main/java/com/nordea/crit/acl
```


Dziękuję za uwagę