

# Upiększ swoje testy!

## Testowanie jednostkowe dla średnio zaawansowanych

Marcin Stachniuk

[mstachniuk@gmail.com](mailto:mstachniuk@gmail.com)

<http://mstachniuk.blogspot.com>

26 września 2014



# Agenda

- 1 Wstęp
- 2 Sekcja given
- 3 Sekcja when
- 4 Sekcja then
- 5 Podsumowanie



# Agenda

- 1 Wstęp
- 2 Sekcja given
- 3 Sekcja when
- 4 Sekcja then
- 5 Podsumowanie



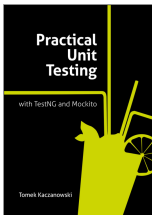
Marcin Stachniuk

Kontakt: [mstachniuk@gmail.com](mailto:mstachniuk@gmail.com)

Blog: [mstachniuk.blogspot.com](http://mstachniuk.blogspot.com)

Twitter: [@MarcinStachniuk](https://twitter.com/MarcinStachniuk)

Recenzent: [practicalunittesting.com](http://practicalunittesting.com)



Kod z zadaniami dostępny tutaj:  
[github.com/mstachniuk/SolarSystem](https://github.com/mstachniuk/SolarSystem)

Wymagania systemowe

- JDK 1.8
- Maven 3.1.1
- Git
- IntelliJ Idea



# Ultimate Test Template

*//given //when //then forever*

```
1: @Test
2: public void shouldDoSomethingCool() throws Exception {
3:     // given
4:
5:     // when
6:
7:     // then
8: }
```



## Settings — > Live Templates

```
1: @org.junit.Test
2: public void should$NAME$() {
3:     // given
4:     $END$
5:
6:     // when
7:
8:     // then
9: }
```



# Agenda

- 1 Wstęp
- 2 Sekcja given
- 3 Sekcja when
- 4 Sekcja then
- 5 Podsumowanie





# Pierwszy test

```
1: @Test
2: public void shouldCalculateOrbitalCircumferenceForMercury() {
3:     // given
4:     OrbitalCircumferenceCalculator calculator =
5:         new OrbitalCircumferenceCalculator();
6:     Planet mercury = new Planet("Mercury", RotationDirection.LEFT,
7:         Distance.createFromMeter(new BigDecimal("4879400")),
8:         new SiderealYear(new BigDecimal("87.96935")));
9:     mercury.setAvgOrbitalSpeed(Speed.createKmPerSecond("47.362"));
10:    // ...
11:
12:    // when
13:    Distance result = calculator.calculate(mercury);
14:
15:    // then
16:    assertEquals(Distance.createFromKM(new BigDecimal("359977336.24608")),
17:        result);
18: }
```



# Pierwszy test

```
1: @Test
2: public void shouldCalculateOrbitalCircumferenceForMercury() {
3:     // given
4:     OrbitalCircumferenceCalculator calculator =
5:         new OrbitalCircumferenceCalculator();
6:     Planet mercury = new Planet("Mercury", RotationDirection.LEFT,
7:         Distance.createFromMeter(new BigDecimal("4879400")),
8:         new SiderealYear(new BigDecimal("87.96935")));
9:     mercury.setAvgOrbitalSpeed(Speed.createKmPerSecond("47.362"));
10:    // ...
11:
12:    // when
13:    Distance result = calculator.calculate(mercury);
14:
15:    // then
16:    assertEquals(Distance.createFromKM(new BigDecimal("359977336.24608")),
17:        result);
18: }
```

Co jest brzydkie w tym teście?



Co jest brzydkie w tym teście?

- Skomplikowany konstruktor
- Niska czytelność
- Wartości niemające znaczenia dla testu
- Problemy przy refaktoryzacji



# Builder Pattern

```
1: public class PlanetBuilder {
2:
3:     private String name;
4:     private RotationDirection rotationDirection;
5:     // ...
6:
7:     public PlanetBuilder name(String name) {
8:         this.name = name;
9:         return this;
10:    }
11:
12:    public PlanetBuilder rotation(RotationDirection direction) {
13:        this.rotationDirection = direction;
14:        return this;
15:    }
16:
17:    public Planet build() {
18:        Planet planet = new Planet(...);
19:        // ...
20:        return planet;
21:    }
22: }
```



# Builder Pattern

```
1: Planet mercury = new PlanteBuilder()  
2:     .name("Mercury")  
3:     .rotationDirection(LEFT)  
4:     // ...  
5:     .build();
```



## Zadanie 01

Uprość sekcję given za pomocą wzorca budowniczy.



# Builder Pattern – przykładowe rozwiązanie

```
1: public class PlanetBuilder {
2:     private Speed avgOrbitalSpeed;
3:     private SiderealYear siderealYear;
4:     // ...
5:
6:     private PlanetBuilder() { }
7:
8:     public static PlanetBuilder aPlanet() { return new PlanetBuilder(); }
9:
10:    public PlanetBuilder avgOrbitalSpeedInKmPerSecond(String avgOrbitalSpeed) {
11:        this.avgOrbitalSpeed = Speed.createKmPerSecond(avgOrbitalSpeed);
12:        return this;
13:    }
14:
15:    public PlanetBuilder siderealYearInEarthDays(String siderealYear) {
16:        this.siderealYear = new SiderealYear(new BigDecimal(siderealYear));
17:        return this;
18:    }
19:    // ...
20: }
```



# Builder Pattern – przykładowe rozwiązanie

```
1: @Test
2: public void shouldCalculateOrbitalCircumferenceForMercury() {
3:     // given
4:     OrbitalCircumferenceCalculator calculator =
5:         new OrbitalCircumferenceCalculator();
6:
7:     Planet mercury = aPlanet()
8:         .avgOrbitalSpeedInKmPerSecond("47.362")
9:         .siderealYearInEarthDays("87.96935")
10:        .build();
11:
12:    // when
13:    Distance result = calculator.calculate(mercury);
14:
15:    // then
16:    assertEquals(Distance.createFromKM(new BigDecimal("359977336.24608")),
17:        result);
18: }
```





Kwestie do rozwiązania:

- Osobna klasa, czy jako statyczna wewnętrzna?
- Tylko testy, czy kod produkcyjny?
- Wartości domyślne
- Spójne nazewnictwo
- Pisane ręcznie, czy generowane?



[projectlombok.org](http://projectlombok.org)

- @Data – generuje gettery / settery, toString, equals, hashCode, konstruktory
- @Cleanup – automatycznie wywołuje metodę close() (np. dla strumieni plików)
- @Log4j – generuje loggera w klasie
- @Builder – tworzy buildera jako statyczną wewnętrzną klasę



## Zadanie 02

Uprość sekcję `given` za pomocą budowniczego dostarczonego przez Lombok'a.



# Builder Pattern, Lombok – rozwiązanie

w kodzie produkcyjnym:

```
1: @Builder
2: @AllArgsConstructor // dla Compilation Error: Planet cannot be applied to ...
3: public class Planet {
4:     // ...
```

w teście:

```
1: @Test
2: public void shouldCalculateOrbitalCircumferenceForMercury() {
3:     // given
4:     Planet mercury = Planet.builder()
5:         .avgOrbitalSpeed(Speed.createKmPerSecond("47.362"))
6:         .siderealYear(new SiderealYear(new BigDecimal("87.96935")))
7:         .build();
8:     // ...
```



Wady projektu Lombok:

- Gdzie jest mój kod?
- Niemożliwość ułatwienia, np:

```
.siderealYear("87.96935")
```

zamiast

```
.siderealYear(new SiderealYear(new BigDecimal("87.96935")))
```

- nulluje wartości domyślne



# Agenda

- 1 Wstęp
- 2 Sekcja given
- 3 Sekcja when**
- 4 Sekcja then
- 5 Podsumowanie



# Testowanie metody rzucającej wyjątek

```
1: @Test
2: public void shouldThrowSomeException() {
3:     // given
4:     SomeClass someClass = new SomeClass();
5:
6:     try {
7:         // when
8:         someClass.doSomething();
9:         fail("This Method should throw SomeException");
10:    } catch (SomeException e) {
11:        // then
12:        assertTrue(true);
13:    }
14: }
```



# Testowanie metody rzucającej wyjątek

```
1: @Test
2: public void shouldThrowSomeException() {
3:     // given
4:     SomeClass someClass = new SomeClass();
5:
6:     // when
7:     try {
8:         someClass.doSomething();
9:         fail("This Method should throw SomeException");
10:    } catch (SomeException e) {
11:        // then
12:        assertTrue(true);
13:    }
14: }
```





# Testowanie metody rzucającej wyjątek

```
1: @Test
2: public void shouldThrowSomeException() {
3:     // given
4:     SomeClass someClass = new SomeClass();
5:
6:     // when
7:     try {
8:         someClass.doSomething();
9:         // then
10:         fail("This Method should throw SomeException");
11:     } catch (SomeException e) {
12:         assertTrue(true);
13:     }
14: }
```



```
1: @Test
2: public void shouldThrowSomeException() {
3:     // given
4:     SomeClass someClass = new SomeClass();
5:
6:     try {
7:         // when
8:         someClass.doSomething();
9:         // then
10:         fail("This Method should throw SomeException");
11:     } catch (SomeException e) {
12:         assertTrue(true);
13:     }
14: }
```

## Propozycje?



## Zadanie 03

Zastosuj expected w teście



# @Test expected – rozwiązanie

@Test expected:

```
1: @Test(expected = InvalidPlanetSpeed.class)
2: public void shouldThrowExceptionWhenAvgOrbitalSpeedIsGreaterThanLightSpeed()
3:     throws InvalidPlanetSpeed {
4:
5:     // given
6:     PlanetLifeValidator validator = new PlanetLifeValidator();
7:     Planet planet = examplePlanet();
8:     planet.setAvgOrbitalSpeed(Speed.createKmPerSecond("310000")); // too big
9:
10:    // when
11:    validator.canBeLife(planet);
12:
13:    // then
14:    fail("It should throw Exception, because planet orbital speed can't be "
15:        + "greater than light speed");
16: }
```



Wady @Test expected:

- Nie widać w sekcji then, że jest spodziewany wyjątek
- Brak możliwości sprawdzenia wiadomości wyjątku lub innych atrybutów
- Niebezpieczeństwo łapania wszystkiego, np. Exception
- Wyjątek może lecieć z innej linii niż się spodziewamy



## Zadanie 04

Zastosuj `@Rule` i `ExpectedException` w teście



# @Rule i ExpectedException – rozwiązanie

```
1: public class PlanetLifeValidatorTest {
2:
3:     @Rule
4:     public ExpectedException thrown = ExpectedException.none();
5:
6:     @Test
7:     public void shouldThrowExceptionWhen...() throws InvalidPlanetSpeed {
8:         // given
9:         PlanetLifeValidator validator = new PlanetLifeValidator();
10:        Planet planet = examplePlanet();
11:        planet.setAvgOrbitalSpeed(Speed.createKmPerSecond("310000"));
12:        thrown.expect(InvalidPlanetSpeed.class);
13:
14:        // when
15:        validator.canBeLife(planet);
16:
17:        // then
18:        fail("It should throw Exception, because planet orbital speed "
19:            + "can't be greater than light speed");
20:    }
21: }
```



Wady JUnit @Rule:

- Nie widać w sekcji then, że jest spodziewany wyjątek
- Wyjątek może lecieć z innej linii niż się spodziewamy
- Rule musi być publiczny (Checkstyle)





Inne rozwiązanie:

## catch-exception

```
1: @Test
2: public void shouldThrowSomeException() throws SomeException {
3:     // given
4:     SomeClass someClass = new SomeClass();
5:
6:     // when
7:     catchException(someClass).doSomething();
8:
9:     // then
10:    assertEquals(SomeException.class, caughtException().getClass());
11:    assertEquals("Some message", caughtException().getMessage());
12: }
```



## Zadanie 05

Zastosuj catch–exception w teście



# catch-exception – przykładowe rozwiązanie

catch-exception:

```
1: @Test
2: public void shouldThrowExceptionWhen...() throws InvalidPlanetSpeed {
3:     // given
4:     PlanetLifeValidator validator = new PlanetLifeValidator();
5:     Planet planet = examplePlanet();
6:     planet.setAvgOrbitalSpeed(Speed.createKmPerSecond("310000"));
7:
8:     // when
9:     catchException(validator).canBeLife(planet);
10:
11:    // then
12:    assertEquals(InvalidPlanetSpeed.class, caughtException().getClass());
13: }
```



Wady catch-exception:

- Problem gdy metoda zwraca void
- Inny import dla checked i unchecked Exceptions
- Nie wspierane więcej bo...



... w Java 8 mamy Lambdy ( $\lambda$ ):

```
1: List<String> numbers = Arrays.asList("One", "Two", "Three", "Four");
2: numbers.forEach(new Consumer<String>() {
3:     @Override
4:     public void accept(String number) {
5:         System.out.println(number);
6:     }
7: });
```

```
1: numbers.forEach((number) -> System.out.println(number));
```



... w Java 8 mamy Lambdy ( $\lambda$ ):

```
1: List<String> numbers = Arrays.asList("One", "Two", "Three", "Four");
2: Stream<String> numbers2 = numbers.stream().filter(new Predicate<String>() {
3:     @Override
4:     public boolean test(String s) {
5:         return s.startsWith("T");
6:     }
7: });
```

```
1: Stream<String> numbers3 = numbers.stream().filter((s) -> s.startsWith(s));
```



## Zadanie 06

Zastosuj Lambdy w teście do łapania wyjątku



# Lambdy – przykładowe rozwiązanie

```
1: @FunctionalInterface
2: public interface ExceptionThrower {
3:     void throwException() throws Throwable;
4: }
```

```
1: public class ThrowableCaptor {
2:
3:     public static Throwable captureThrowable(ExceptionThrower thrower) {
4:         try {
5:             thrower.throwException();
6:             return null;
7:         } catch (Throwable throwable) {
8:             return throwable;
9:         }
10:    }
11: }
```





# Lambdy – przykładowe rozwiązanie

```
1: @Test
2: public void shouldThrowExceptionWhenAvgOrbitalSpeedIsGreaterThanLightSpeed() {
3:     // given
4:     PlanetLifeValidator validator = new PlanetLifeValidator();
5:     Planet planet = examplePlanet();
6:     planet.setAvgOrbitalSpeed(Speed.createKmPerSecond("310000"));
7:
8:     // when
9:     Throwable throwable = ThrowableCaptor.captureThrowable(
10:         () -> validator.canBeLife(planet));
11:
12:     // then
13:     assertEquals(InvalidPlanetSpeed.class, throwable.getClass());
14: }
```



Wady Lambdy:

- Trzeba napisać `ExceptionThrower` i `ThrowerCaptor`



# Agenda

- 1 Wstęp
- 2 Sekcja given
- 3 Sekcja when
- 4 Sekcja then**
- 5 Podsumowanie



```
1: @Test
2: public void shouldCreateInnersPlants() {
3:     // given
4:     SolarSystemFactory factory = new SolarSystemFactory();
5:
6:     // when
7:     List<Planet> innerPlanets = factory.createInnerPlanets();
8:
9:     // then
10:    Planet mercury = innerPlanets.get(0);
11:    assertEquals("Mercury", mercury.getName());
12:    assertEquals(RotationDirection.LEFT, mercury.getRotationDirection());
13:    // ...
14:
15:    Planet venus = innerPlanets.get(1);
16:    assertEquals("Venus", venus.getName());
17:    assertEquals(RotationDirection.RIGHT, venus.getRotationDirection());
18:    // ...
```



```
1: @Test
2: public void shouldCreateInnersPlants() {
3:     // given
4:     SolarSystemFactory factory = new SolarSystemFactory();
5:
6:     // when
7:     List<Planet> innerPlanets = factory.createInnerPlanets();
8:
9:     // then
10:    Planet mercury = innerPlanets.get(0);
11:    assertEquals("Mercury", mercury.getName());
12:    assertEquals(RotationDirection.LEFT, mercury.getRotationDirection());
13:    // ...
14:
15:    Planet venus = innerPlanets.get(1);
16:    assertEquals("Venus", venus.getName());
17:    assertEquals(RotationDirection.RIGHT, venus.getRotationDirection());
18:    // ...
```

Co jest brzydkiego w tym teście?



Co jest brzydkiego w tym teście?

- Wiele sprawdzanych warunków
- Potwórzania w kodzie
- Totalna nieczytelność



## Zadanie 07

Utwórz własną metodę `assertPlanet(...)`



## Własne asercje – przykładowe rozwiązanie

```
1: // then
2: Planet mercury = innerPlanets.get(0);
3: assertPlanet(mercury, "Mercury", RotationDirection.LEFT, "4879400",
4:             "87.96935", "47.362", 3.701,
5:             Gas.OXYGEN, Gas.SODIUM, Gas.HYDROGEN);
6: // ...

1: private void assertPlanet(Planet planet, String planetName,
2:                             RotationDirection direction, String diameterInMeter,
3:                             String yearInEarthDays, String avgOrbitalSpeedInKmPerSecond,
4:                             double acceleration, Gas... atmosphereGases) {
5:
6:     assertEquals(planetName, planet.getName());
7:     assertEquals(direction, planet.getRotationDirection());
8:     assertEquals(0, new BigDecimal(diameterInMeter).compareTo(
9:         planet.getDiameter().getMeter()));
10: // ...
11: }
```





## Wady własnej assercji

- Który parametr w wywołaniu co oznacza
- Przy błędzie, nie wiadomo do końca, gdzie coś jest źle



<https://github.com/hamcrest/JavaHamcrest>

- Część JUnit'a
- Tworzy czytelniejsze assercje



```
1: public class IsNotANumber extends TypeSafeMatcher<Double> {
2:
3:     @Override
4:     public boolean matchesSafely(Double number) {
5:         return number.isNaN();
6:     }
7:
8:     public void describeTo(Description description) {
9:         description.appendText("not a number");
10:    }
11:
12:    @Factory
13:    public static <T> Matcher<Double> notANumber() {
14:        return new IsNotANumber();
15:    }
16: }

1: public void testSquareRootOfMinusOneIsNotANumber() {
2:     assertThat(Math.sqrt(-1), is(notANumber()));
3: }
```



## Zadanie 08

Zrefaktoruj metodę `assertPlanet(...)`, aby używała Hamcresta



# Hamcrest – przykładowe rozwiązanie

```
1: public class PlanetNameMatcher extends BaseMatcher<Planet> {
2:
3:     private String name;
4:
5:     private PlanetNameMatcher(String name) { this.name = name; }
6:
7:     @Override
8:     public boolean matches(Object o) {
9:         Planet p = (Planet) o;
10:        return p.getName().equals(name);
11:    }
12:
13:    @Override
14:    public void describeTo(Description description) {
15:        description.appendText("not Planet with name: " + name);
16:    }
17:
18:    @Factory
19:    public static PlanetNameMatcher name(String name) {
20:        return new PlanetNameMatcher(name);
21:    }
22: }
```



W teście:

```
1: assertThat(planet, is(name(planetName)));  
2: assertThat(planet, is(rotation(direction)));  
3: assertThat(planet, is(diameterInMeter(diameterInMeter)));  
4: // ...
```



## Wady Hamcresta:

- Trzeba wygenerować wiele klas, 1 klasa / 1 właściwość
- Nienajlepsza czytelność
- Słabe raportowanie błędów



<https://github.com/alexruiz/fest-assert-2.x>

- Fluent interface
- Przypomina Buildera





# FEST Assert 2.X – Przykład

```
1: public class HumanAssert<T extends HumanAssert<T>> extends AbstractAssert
2:     <HumanAssert<T>, Human> {
3:
4:     public HumanAssert(Human actual) {
5:         super(actual, HumanAssert.class);
6:     }
7:
8:     public T hasName(String name) {
9:         assertNotNull();
10:         assertThat(actual.name).isEqualTo(name);
11:         return (T) this;
12:     }
13:     //...
14: }
```

```
1: @Test
2: public void shouldHumanHaveNameAndSurname() {
3:     Human human = new Human("Jan", "Kowalski");
4:     assertThat(human)
5:         .hasName("Jan")
6:         .hasSurname("Kowalski");
7: }
```



## Zadanie 09

Napisz assercje za pomocą FEST Assert 2.X



# FEST Assert 2.X – przykładowe rozwiązanie

```
1: public class PlanetAssert extends AbstractAssert<PlanetAssert, Planet> {
2:
3:     protected PlanetAssert(Planet actual) {
4:         super(actual, PlanetAssert.class);
5:     }
6:
7:     public static PlanetAssert assertThat(Planet actual) {
8:         Assertions.assertThat(actual).isNotNull();
9:         return new PlanetAssert(actual);
10:    }
11:
12:    public PlanetAssert hasRotation(RotationDirection rotationDirection) {
13:        Assertions.assertThat(actual.getRotationDirection())
14:            .isEqualTo(rotationDirection);
15:        return this;
16:    }
17:
18:    public PlanetAssert hasDiameterInMeter(String diameterInMeter) {
19:        Assertions.assertThat(actual.getDiameter().getMeter())
20:            .isEqualToComparingTo(diameterInMeter);
21:        return this;
22:    }
```



# FEST Assert 2.X – przykładowe rozwiązanie

```
1: public class PlanetSetAssert extends AbstractAssert<PlanetSetAssert,  
2:     List<Planet>> {  
3:  
4:     public PlanetAssert containsPlanetWithName3(String expectedPlanetName) {  
5:         Planet expectedPlanet = new Planet(expectedPlanetName, null, null,  
6:             null);  
7:  
8:         PlanetNameComparator comparator = new PlanetNameComparator();  
9:         Assertions.assertThat(actual)  
10:             .usingElementComparator(comparator)  
11:             .contains(expectedPlanet);  
12:  
13:         for (Planet planet : actual) {  
14:             if(comparator.compare(planet, expectedPlanet) == 0) {  
15:                 return PlanetAssert.assertThat(planet);  
16:             }  
17:         }  
18:         return null;  
19:     }
```



# FEST Assert 2.X – przykładowe rozwiązanie

```
1: // then
2: assertThat(innerPlanets)
3:     .containsPlanetWithName3("Mercury")
4:     .hasRotation(RotationDirection.LEFT)
5:     .hasDiameterInMeter("4879400")
6:     .hasYearInEarthDays("87.96935")
7:     .hasAvgOrbitalSpeedInKmPerSecond("47.362")
8:     .hasAcceleration(3.701)
9:     .containsGas(Gas.OXYGEN)
10:    .containsGas(Gas.SODIUM)
11:    .containsGas(Gas.HYDROGEN);
```



Wady FEST Assert 2.X:

- Trochę pisania, ale...



... przecież można to wygenerować za pomocą  
fest-assertion-generator:

<https://github.com/joel-costigliola/fest-assertion-generator>  
z poziomu [Maven](#)'a lub [Eclipse](#)'a



## Zadanie 10

Wygeneruj asercje za pomocą  
maven-fest-assertion-generator-plugin





Polecenie:

```
1: mvn org.easytesting:maven-fest-assertion-generator-plugin:generate-assertions
```

generuje:

```
1: /**
2:  * {@link Planet} specific assertions - Generated by CustomAssertionGenerator.
3:  */
4: public class PlanetAssert extends AbstractAssert<PlanetAssert, Planet> {
5:
6:     /**
7:      * Creates a new {@link PlanetAssert} to make assertions on ...
8:      * @param actual the Planet we want to make assertions on.
9:      */
10:    public PlanetAssert(Planet actual) {
11:        super(actual, PlanetAssert.class);
12:    }
```



Wady fest-assert-generator:

- Nie rozwijane więcej (zresztą jak FEST Assert 2.X), ale...



... w zamian mamy następcę: AssertJ

<http://assertj.org/>

- W sumie jest to kontynuacja (fork) FEST'a
- Zmieniają się tylko importy
- I parę niezrozumiałych metod wyleciało



## Zadanie 11

Napisz assercje za pomocą AssertJ



# AssertJ – przykładowe rozwiązanie

```
1: public class PlanetAssert extends AbstractAssert<PlanetAssert, Planet> {
2:
3:     protected PlanetAssert(Planet actual) {
4:         super(actual, PlanetAssert.class);
5:     }
6:
7:     public static PlanetAssert assertThat(Planet actual) {
8:         Assertions.assertThat(actual)
9:             .isNull();
10:        return new PlanetAssert(actual);
11:    }
12:
13:    public PlanetAssert hasRotation(RotationDirection rotationDirection) {
14:        Assertions.assertThat(actual.getRotationDirection())
15:            .isEqualTo(rotationDirection);
16:        return this;
17:    }
```



# AssertJ – przykładowe rozwiązanie

```
1: // then
2: assertThat(innerPlanets)
3:     .containsPlanetWithName3("Mercury")
4:     .hasRotation(RotationDirection.LEFT)
5:     .hasDiameterInMeter("4879400")
6:     .hasYearInEarthDays("87.96935")
7:     .hasAvgOrbitalSpeedInKmPerSecond("47.362")
8:     .hasAcceleration(3.701)
9:     .containsGas(Gas.OXYGEN)
10:    .containsGas(Gas.SODIUM)
11:    .containsGas(Gas.HYDROGEN);
```



Wady AssertJ:

- dotychczas nieznane :P



AssertJ posiada także (podobnie jak FEST Assert 2.X):

- [AssertJ assertions generator maven plugin](#)
- [AssertJ assertions generator](#) – Command Line
- [assertj-guava](#)
- [assertj-joda-time](#)

a także:

- [assertj-neo4j](#)
- [assertj-swing](#)





## Zadanie 12

Wygeneruj assercje za pomocą AssertJ assertions generator maven plugin



# AssertJ – przykładowe rozwiązanie

```
1: mvn assertj:generate-assertions
```

```
1: // then
2: Planet mercury = innerPlanets.get(0);
3: assertThat(mercury)
4:     .hasName("Mercury")
5:     .hasRotationDirection(RotationDirection.LEFT)
6:     .hasDiameter(Distance.createFromMeter(new BigDecimal("4879400")))
7:     .hasSiderealYear(new SiderealYear(new BigDecimal("87.96935")))
8:     .hasAvgOrbitalSpeed(Speed.createKmPerSecond("47.362"))
9:     .hasAcceleration(3.701, 0.00001)
10:    .hasAtmosphereGases(Gas.OXYGEN, Gas.SODIUM, Gas.HYDROGEN);
```



Wady AssertJ assertions generator maven plugin:

- Generować z każdym buildem?
- Nie tak super inteligentny jak byśmy chcieli :)



?



# Agenda

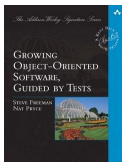
- 1 Wstęp
- 2 Sekcja given
- 3 Sekcja when
- 4 Sekcja then
- 5 Podsumowanie**



# Czego się dzisiaj nauczyliśmy?

- Tworzenie Builderów w sekcji given
- Lombok
- Testowanie wyjątków za pomocą @Test expected i JUnit @Rule
- Testowanie wyjątków za pomocą catch-exception i  $\lambda$
- Pisanie własnych assercji
- JUnit Matcher / Hamcrest
- FEST Assert 2.X i fest-assertion-generator
- AssertJ i AssertJ assertions generator maven plugin





- XUnit Test Patterns Refactoring Test Code, Gerard Meszaros  
<http://xunitpatterns.com/>
- Growing Object-Oriented Software Guided by Tests, Steve Freeman, Nat Pryce  
<http://www.growing-object-oriented-software.com/>
- Practical Unit Testing with TestNG and Mockito, Tomasz Kaczanowski <http://practicalunittesting.com/>
- Practical Unit Testing with JUnit and Mockito, Tomasz Kaczanowski <http://practicalunittesting.com/>
- Bad Tests, Good Tests, Tomasz Kaczanowski  
<http://practicalunittesting.com/>



- ❶ Marcin Stachniuk Blog: [mstachniuk.blogspot.com](http://mstachniuk.blogspot.com)
- ❷ Kod z zadaniami: [github.com/mstachniuk/SolarSystem](https://github.com/mstachniuk/SolarSystem)
- ❸ Książki Tomka Kaczanowskiego: [practicalunittesting.com](http://practicalunittesting.com)
- ❹ [monkeyisland.pl/2009/12/07/given-when-then-forever](http://monkeyisland.pl/2009/12/07/given-when-then-forever)
- ❺ [projectlombok.org](http://projectlombok.org)
- ❻ [catch-exception](http://catch-exception)
- ❼ <http://blog.codeleak.pl/2014/07/junit-testing-exception-with-java-8-and-lambda-expressions.html>
- ❽ <https://github.com/hamcrest/JavaHamcrest>
- ❾ <https://github.com/alexruiz/fest-assert-2.x>
- ❿ <https://github.com/joel-costigliola/fest-assertion-generator>
- ⓫ [FEST assertion generator plugin](#)
- ⓬ [FEST eclipse plugin](#)
- ⓭ [assertj.org](http://assertj.org)
- ⓮ [AssertJ assertions generator maven plugin](#)
- ⓯ [AssertJ assertions generator](#)
- ⓰ [assertj-guava](#)
- ⓱ [assertj-joda-time](#)
- ⓲ [assertj-neo4j](#)
- ⓳ [assertj-swing](#)
- ⓴ [xunitpatterns.com](http://xunitpatterns.com)
- ⓵ [www.growing-object-oriented-software.com](http://www.growing-object-oriented-software.com)





# Upiększ swoje testy!

## Testowanie jednostkowe dla średnio zaawansowanych

Marcin Stachniuk

[mstachniuk@gmail.com](mailto:mstachniuk@gmail.com)

<http://mstachniuk.blogspot.com>

# Dziękuję!

26 września 2014

