

Maven, Testy Jednostkowe

Artur Panek



maven

TestNG

JUnit



AGENDA

- Maven – instalacja, tworzenie projektu, dodawanie zależności
- JUnit – testy jednostkowe, asercje i adnotacje
- TestNG – testy, adnotacje, zbiory testów i dostarczanie danych
- Unitils – Reflection Assert - porównywanie złożonych obiektów



maven

wprowadzenie

- Maven to narzędzie, które pozwala na zarządzanie projektem w sposób zorganizowany
- Wprowadza specyficzną strukturę dla projektu
- Pozwala na łatwe zarządzanie zależnościami/bibliotekami używanymi w projekcie
- Jest dobrze zintegrowany z wieloma narzędziami IDE
- Jego funkcjonalność może być rozszerzana poprzez dodawane pluginy

Budowanie oprogramowania ma zakończyć się osiągnięciem wybranego przez budującego **celu**.

Powodzenie każdego kolejnego celu uzależnione jest od pomyślnej realizacji celów znajdujących się wcześniej w cyklu.

Dostępnych celów jest wiele:

validate - sprawdzenie, czy projekt jest poprawny i czy wszystkie niezbędne informacje zostały określone

compile - kod źródłowy jest kompilowany

test - przeprowadzane są testy jednostkowe

package - budowana jest paczka dystrybucyjna

integration-test - zbudowany projekt umieszczany jest w środowisku testowym, gdzie przeprowadzane są testy integracyjne

verify - sprawdzenie, czy paczka jest poprawna

install - paczka umieszczana jest w repozytorium lokalnym - może być używana przez inne projekty jako zależność

deploy - paczka umieszczana jest w repozytorium zdalnym (opublikowana)



maven

instalacja

W pierwszej kolejności należy zainstalować SDK Javy ze strony Oracle i ustawić zmienne środowiskowe:

```
JAVA_HOME=C:\Program Files\Java\jdk1.8.0_60  
PATH=C:\Program Files\Java\jdk1.8.0_60\bin
```

Poprawność instalacji Javy możemy sprawdzić z konsoli:

```
java -version
```

A screenshot of a Windows Command Prompt window. The title bar says "Command Prompt". The command prompt shows the user's location as "C:\Users\apanek" and the command "java -version" has been entered. The output is displayed on the next lines: "java version \"1.8.0_60\"", "Java(TM) SE Runtime Environment (build 1.8.0_60-b27)", and "Java HotSpot(TM) 64-Bit Server VM (build 25.60-b23, mixed mode)". The prompt ends with "C:\Users\apanek>_".

```
C:\Users\apanek>java -version  
java version "1.8.0_60"  
Java(TM) SE Runtime Environment (build 1.8.0_60-b27)  
Java HotSpot(TM) 64-Bit Server VM (build 25.60-b23, mixed mode)  
C:\Users\apanek>_
```



maven

instalacja

Instalacja Maven'a w środowisku Windows:

Ściągamy ze strony <http://maven.apache.org/download.cgi> odpowiednią paczkę dla naszego systemu i rozpakowujemy. Następnie ustawiamy odpowiednio do wybranej lokalizacji zmienne środowiskowe:

```
M2_HOME=C:\apache-maven-3.2.5  
PATH=C:\apache-maven-3.2.5\bin
```

Poprawność instalacji możemy sprawdzić z konsoli:

```
mvn -version
```

```
Command Prompt  
C:\Users\apanek>mvn -version  
Apache Maven 3.2.5 (12a6b3acb947671f09b81f49094c53f426d8cea1;  
3+01:00)  
Maven home: C:\apache-maven-3.2.5  
Java version: 1.8.0_60, vendor: Oracle Corporation  
Java home: C:\Program Files\Java\jdk1.8.0_60\jre  
Default locale: en_US, platform encoding: Cp1250  
OS name: "windows 8.1", version: "6.3", arch: "amd64", family:  
C:\Users\apanek>_
```



maven

instalacja

Instalacja Maven'a w środowisku Linux:

```
// Instalacja Javy
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update

sudo apt-get install oracle-java8-installer

// Ustawienie zmiennych środowiskowych
sudo update-alternatives --config java // sprawdzenie ścieżki
sudo nano /etc/environment           // ustawienie zmiennej

JAVA_HOME="YOUR_PATH"

source /etc/environment           // przeładowanie pliku
echo $JAVA_HOME                  // weryfikacja zmiennej
java -version                    // weryfikacja javy

// Zawartość pliku /etc/environment
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/share/maven/bin"
JAVA_HOME=/usr/lib/jvm/java-8-oracle
M2_HOME=/usr/share/maven
```

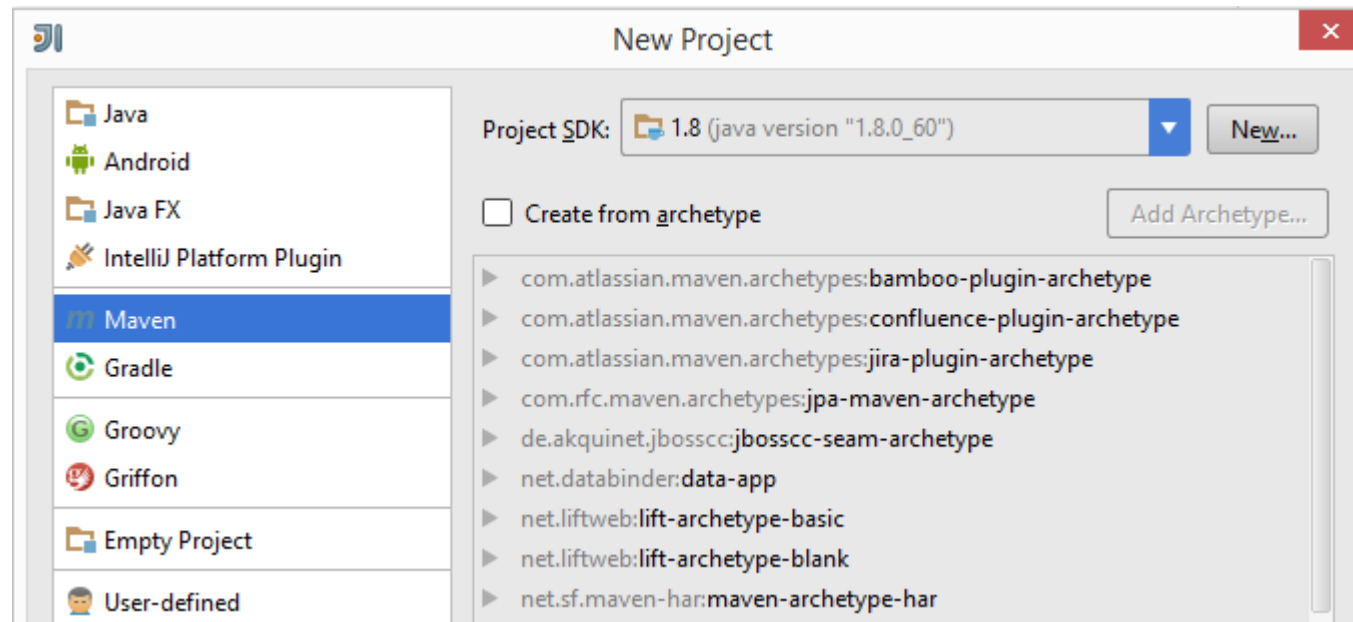
```
// Instalacja maven'a
sudo apt-get update
sudo apt-get install maven

// Lokalizacja maven'a:
/usr/share/maven

// Konfiguracja maven'a:
/etc/maven

// Weryfikacja maven'a
mvn -version
```

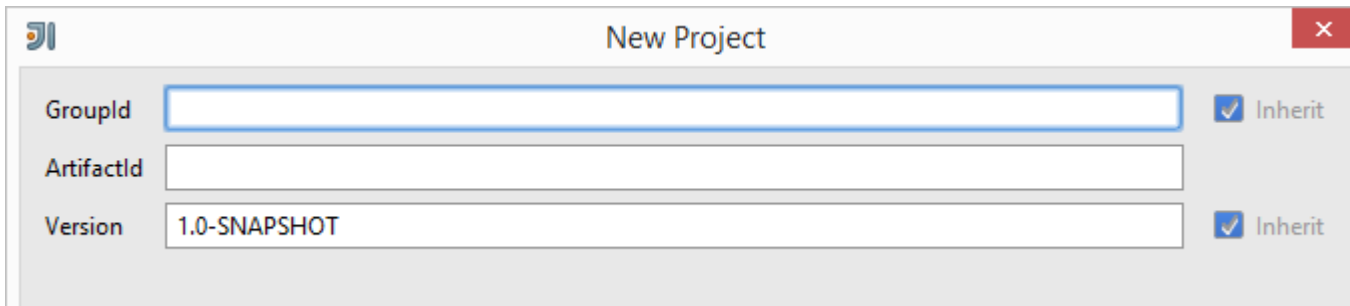
1) Z menu głównego wybieramy: File/New/Project:



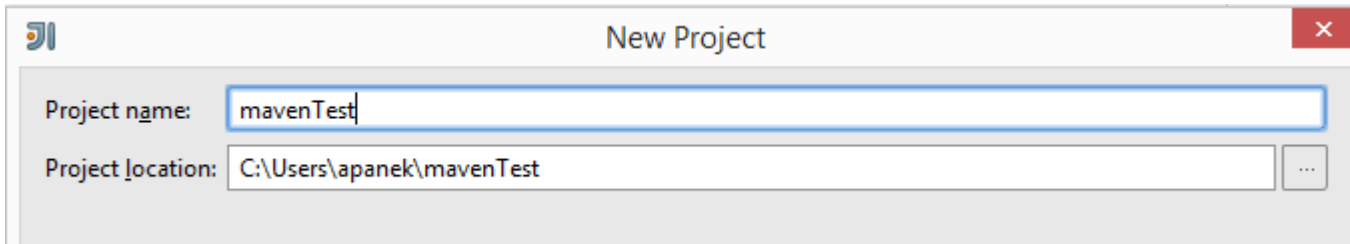
2) Zaznaczamy „Maven” i klikamy „next”

3) Wpisujemy „GroupId” – twórca np: „com.pgs-soft”

4) Wpisujemy „ArtifactId” – nazwa aplikacji np: „testAutomatyczny”

A screenshot of the "New Project" dialog box in an IDE. It has three input fields: "GroupId", "ArtifactId", and "Version". The "Version" field is pre-filled with "1.0-SNAPSHOT". To the right of each field is a checkbox labeled "Inherit", which is checked for all three. The dialog has a title bar with the IDE logo and a close button (X).

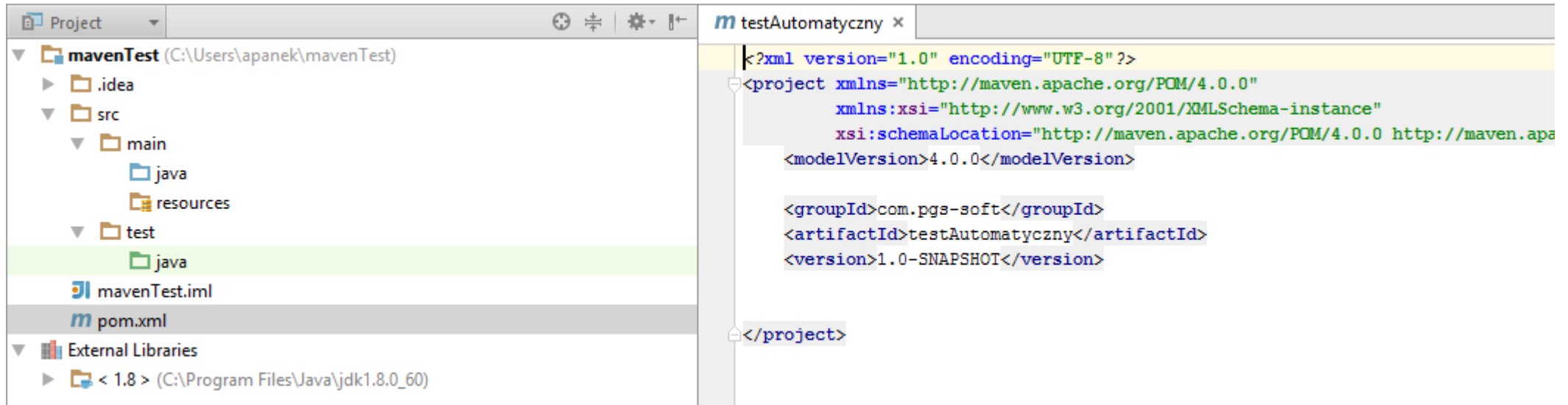
5) Klikamy „next”

A screenshot of the "New Project" dialog box in an IDE, showing the second step. It has two input fields: "Project name:" and "Project location:". The "Project name:" field is pre-filled with "mavenTest". The "Project location:" field is pre-filled with "C:\Users\apanek\mavenTest". There is a button with three dots ("...") to the right of the "Project location:" field. The dialog has a title bar with the IDE logo and a close button (X).

6) Wpisujemy nazwę projektu np: „mavenTest” i klikamy „finish”

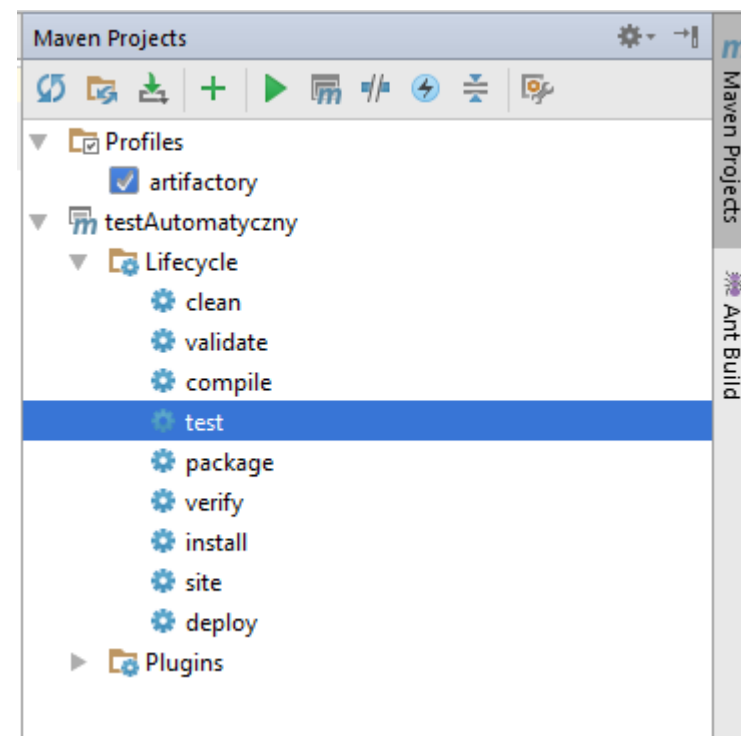
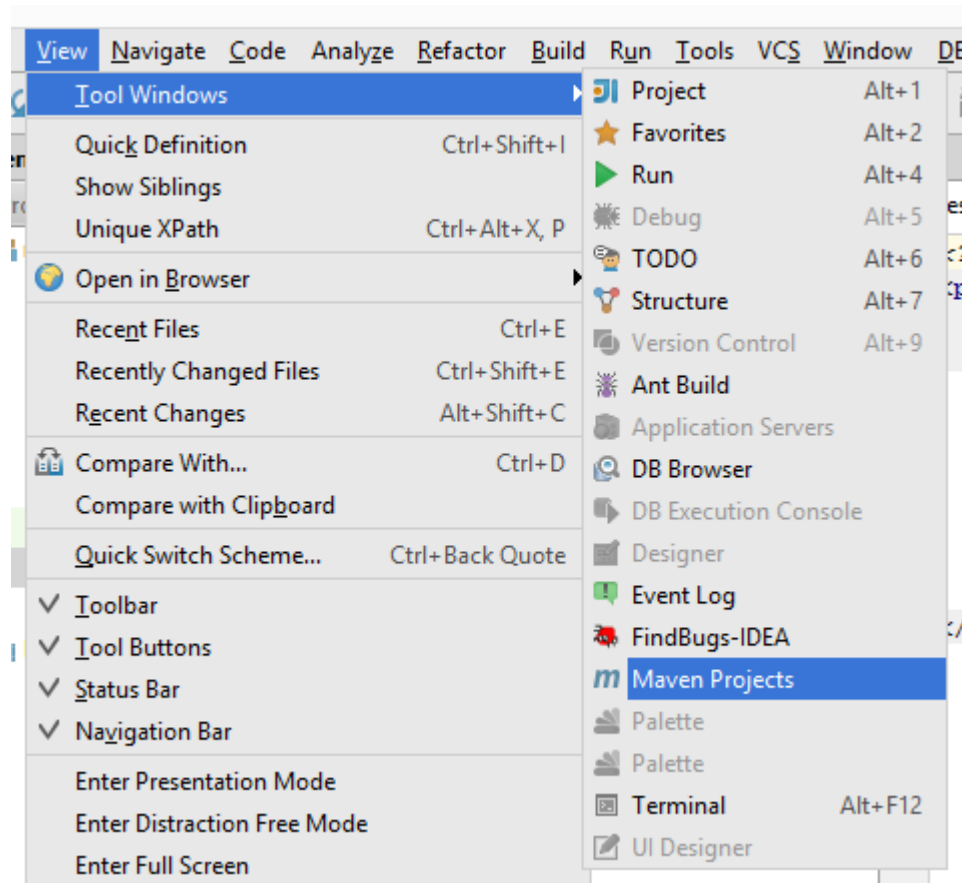


Na koniec otrzymujemy nowy projekt o określonej strukturze wraz z wygenerowanym plikiem **pom.xml**

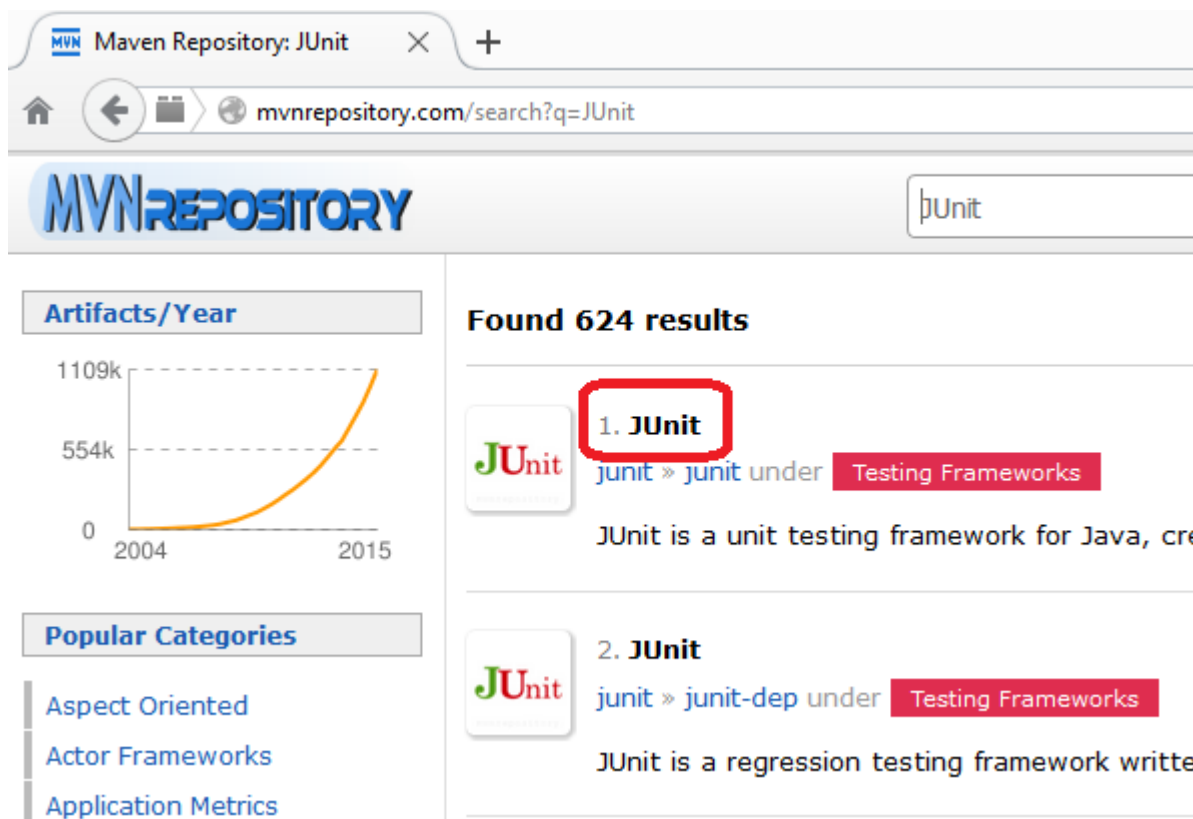


Projekt mavenowy definiuje się poprzez stworzenie i utrzymywanie pliku **pom.xml** (POM – ang. Project Object Model). pom.xml jest głównym miejscem pracy z projektem i zawiera wszystkie istotne elementy definiujące projekt, jego strukturę, sposób budowania i przede wszystkim zależności.

IntelliJ dostarcza dodatkowy panel dla realizacji celów maven'a, który możemy uruchomić z menu:



Wyszukiwanie bibliotek: <http://mvnrepository.com/>



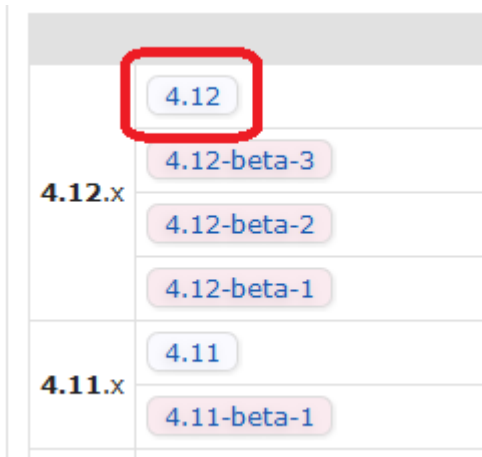
Dodawanie zależności w pliku pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

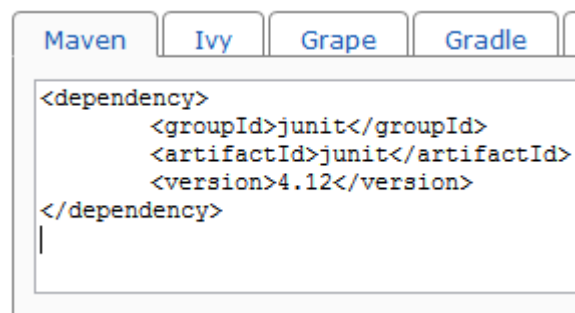
  <groupId>com.pgs-soft</groupId>
  <artifactId>testAutomatyczny</artifactId>
  <version>1.0-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
    </dependency>
  </dependencies>
</project>
```

Wybieramy wersję dla
Dla wyszukiwanej biblioteki:



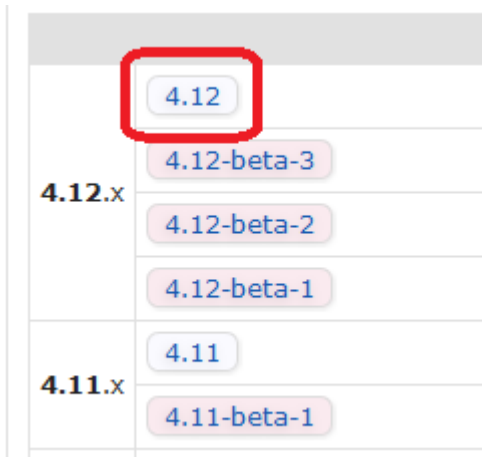
Kopiujemy blok zależności
przygotowany dla Maven'a:



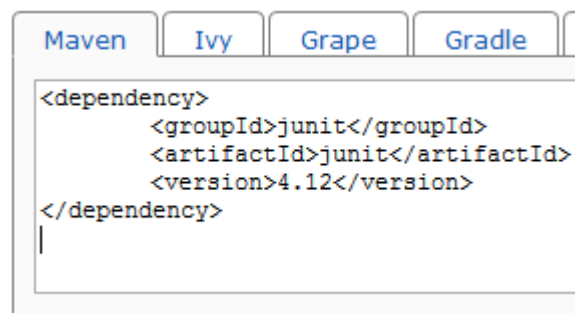
Dodajemy wyszukaną zależność
do bloku dependencies:



Wybieramy wersję dla
Dla wyszukiwanej biblioteki:



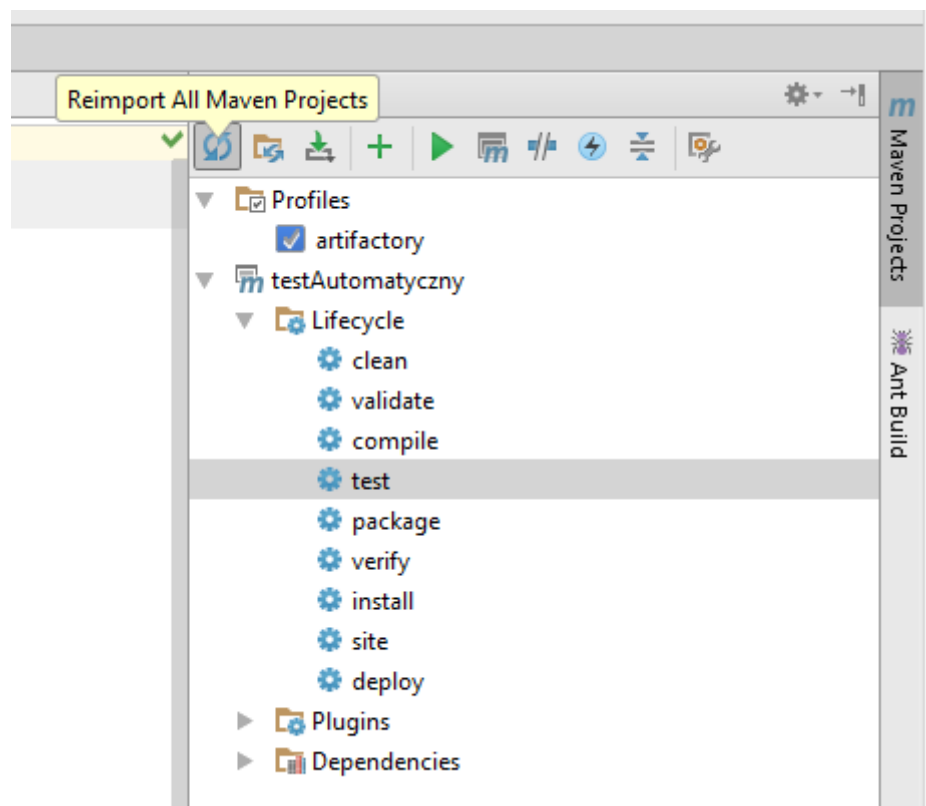
Kopiujemy blok zależności
przygotowany dla Maven'a:



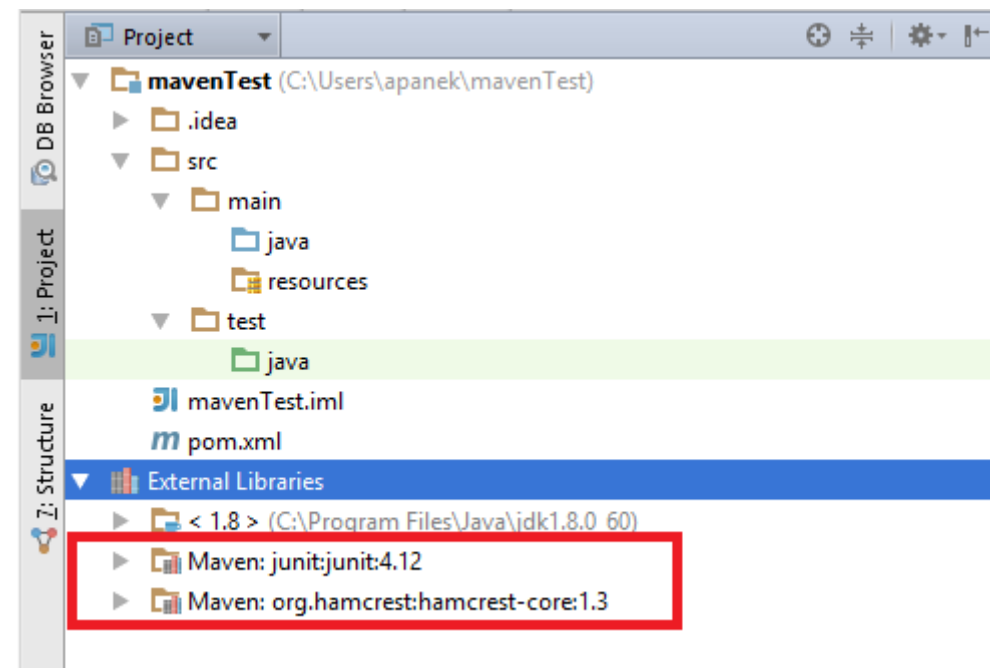
Dodajemy wyszukaną zależność
do bloku dependencies:



Odświeżanie bibliotek:



Na koniec w strukturze projektu pojawiają się pożądane biblioteki:





Testy jednostkowe

JUnit – narzędzie służące do tworzenia powtarzalnych testów jednostkowych oprogramowania pisanego w języku Java.

Testy jednostkowe są niezwykle ważnym narzędziem programisty. Przeprowadzane podczas pisania aplikacji pozwalają na sprawdzenie poprawności kodu, wyłapanie błędów i szybkie usunięcie ich.

W nowoczesnych metodykach wytwarzania oprogramowania testy jednostkowe są jednymi z najważniejszych elementów procesu. Tworzenie systemów bez korzystania z testów jednostkowych często porównywane jest do pisania programów na kartce.

Nowoczesne narzędzia takie, jak JUnit bardzo ułatwiają przeprowadzanie testów jednostkowych, integrując się ze środowiskami programistycznymi.

Testy jednostkowe to nic innego jak zbiór testów, które weryfikują, czy jednostka kodu (np. Klasa, serwis itp.) działa zgodnie z oczekiwaniami. Jeśli używamy Mavena, testy jednostkowe uruchamiane są automatycznie przy każdym budowaniu projektu, jeśli któryś z testów nie zadziała, cała procedura jest przerywana.





Testy jednostkowe - Adnotacje

JUnit wykorzystuje adnotacje w celu określania metod o specjalnym przeznaczeniu.

@Test – adnotacja oznaczająca daną metodę jako test. Metoda będąca testem musi być publiczna, nie może posiadać żadnych argumentów oraz nie może zwracać żadnych wartości.

JUnit pozwala nam na wykonanie jakiegoś kodu przed i po pojedynczym teście jak i po całym zestawie testów. Służą do tego adnotacje @Before, @BeforeClass, @After oraz @AfterClass. Przykładowa klasa może wyglądać następująco:

```
public class KlasaTestowa {  
    @BeforeClass  
    public static void setUpBeforeClass() throws Exception {  
        //ta metoda będzie wywołana raz, przed wszystkimi testami  
    }  
  
    @AfterClass  
    public static void tearDownAfterClass() throws Exception {  
        //ta metoda będzie wywołana raz, po wszystkich testach  
    }  
  
    @Before  
    public void setUp() throws Exception {  
        //ta metoda będzie wywołana przed każdym testem  
    }  
  
    @After  
    public void tearDown() throws Exception {  
        //ta metoda będzie wywołana po każdym teście  
    }  
  
    @Test  
    public void testujCos() {  
        //...  
    }  
}
```




Testy jednostkowe - Asercje

JUnit definiuje wiele asercji, które są dostarczone z klasą `Assert` i używane w celu sprawdzenia określonego warunku:

`fail([String message])`

`assertEquals([String message], expected, actual)`
`assertEquals([String message], expected, actual, tolerance)`
`assertNull([String message], java.lang.Object object)`
`assertNotNull([String message], java.lang.Object object)`
`assertSame([String message], expected, actual)`
`assertNotSame([String message], expected, actual)`
`assertTrue([String message], boolean condition)`
`assertFalse([String message], boolean condition)`

...

```
@Test
public void subtractTest() {
    Assert.assertEquals(-10, mathProvider.subtract(10, 20));
}

@After
public void Sy
    assertEquals(double expected, double actual, double delta)
    assertEquals(long expected, long actual)
    assertTrue(boolean condition)
    assertEquals(boolean[] expecteds, boolean[] actuals)
    assertEquals(byte[] expecteds, byte[] actuals)
    assertEquals(char[] expecteds, char[] actuals)
    assertEquals(double[] expecteds, double[] actuals, double delta)
    assertEquals(float[] expecteds, float[] actuals, float delta)
    assertEquals(int[] expecteds, int[] actuals)
    assertEquals(long[] expecteds, long[] actuals)
    assertEquals(Object[] expecteds, Object[] actuals)
    assertEquals(short[] expecteds, short[] actuals)
    assertEquals(String message, boolean[] expecteds, boolean[] actuals)
    assertEquals(String message, byte[] expecteds, byte[] actuals)
    assertEquals(String message, char[] expecteds, char[] actuals)
    assertEquals(String message, double[] expecteds, double[] actuals, double delta)
    assertEquals(String message, float[] expecteds, float[] actuals, float delta)
    assertEquals(String message, int[] expecteds, int[] actuals)
    assertEquals(String message, long[] expecteds, long[] actuals)
    assertEquals(String message, Object[] expecteds, Object[] actuals)
    assertEquals(String message, short[] expecteds, short[] actuals)
    assertEquals(float expected, float actual, float delta)
    assertEquals(Object expected, Object actual)
    assertEquals(String message, double expected, double actual, double delta)
    assertEquals(String message, float expected, float actual, float delta)
```



Testy jednostkowe - przykład

The screenshot shows an IDE with the following components:

- Project Explorer (Left):** Displays the project structure for `mavenTest` (C:\Users\apanek\mavenTest). The `test` directory is highlighted, showing the `application` package containing the `MathTest` class.
- Code Editor (Middle):** Displays the `MathProvider.java` file. The code is as follows:

```
package application;

public class MathProvider {

    public int subtract(int firstNumber, int secondNumber) {
        return (firstNumber - secondNumber);
    }

    public int add(int firstNumber, int secondNumber) {
        return (firstNumber + secondNumber);
    }

    public int multiply(int firstNumber, int secondNumber) {
        return (firstNumber * secondNumber);
    }
}
```
- Code Editor (Right):** Displays the `MathTest.java` file. The code is as follows:

```
package application;

import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.BeforeClass;
import org.junit.Test;

public class MathTest {

    private static MathProvider mathProvider;

    @BeforeClass
    public static void setUp() {
        mathProvider = new MathProvider();
        System.out.println("Testing started");
    }

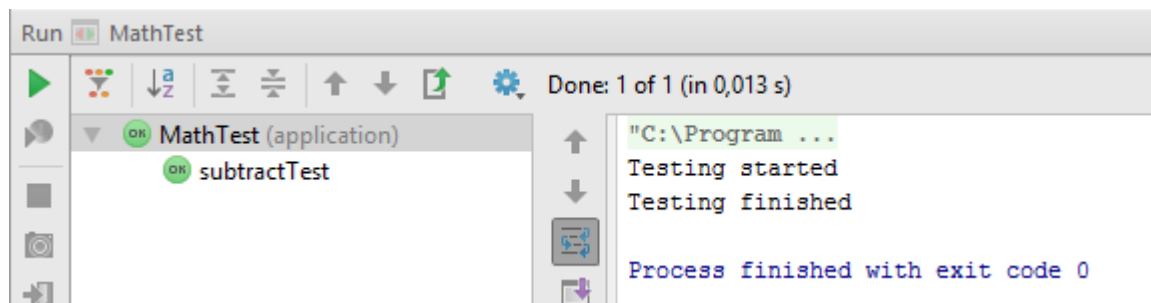
    @Test
    public void subtractTest() {
        Assert.assertEquals(-10, mathProvider.subtract(10, 20));
    }

    @AfterClass
    public static void tearDown() {
        System.out.println("Testing finished");
    }
}
```

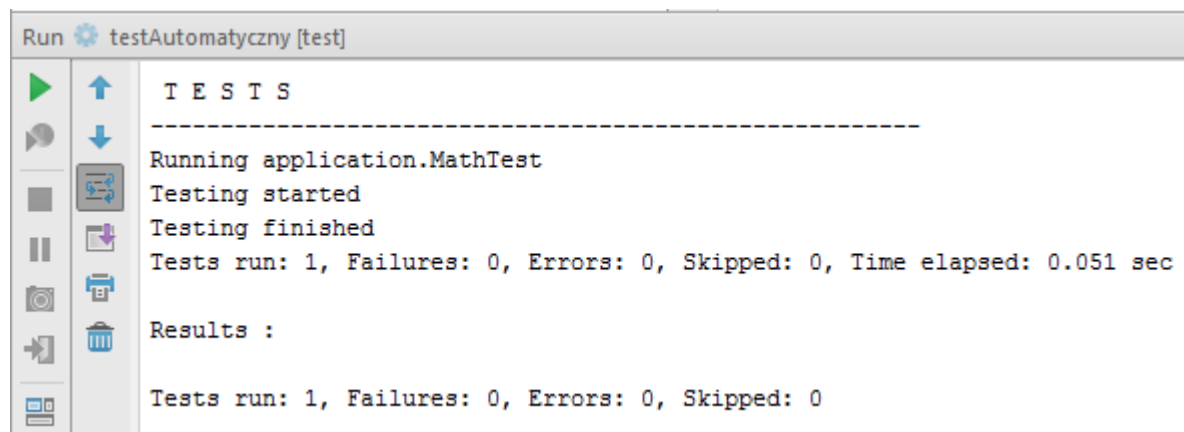
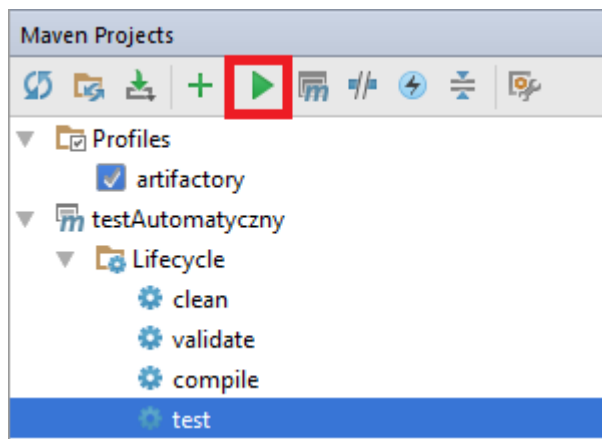


Testy jednostkowe - uruchamianie

1) Run „MathTest” z menu kontekstowego:



2) Uruchamianie z panelu Maven'a:



3) Uruchamianie z terminala:

```
mvn test
```



Testy jednostkowe - ćwiczenie

The screenshot shows an IDE with the following components:

- Project Explorer (Left):** Shows the project structure for `mavenTest` (C:\Users\apanek\mavenTest). The `test` directory is expanded, showing `java` and `application` subdirectories. The `MathTest` class is highlighted in the `application` directory.
- Code Editor (Middle):** Displays the `MathProvider.java` file. The code is as follows:

```
package application;

public class MathProvider {

    public int subtract(int firstNumber, int secondNumber) {
        return (firstNumber - secondNumber);
    }

    public int add(int firstNumber, int secondNumber) {
        return (firstNumber + secondNumber);
    }

    public int multiply(int firstNumber, int secondNumber) {
        return (firstNumber * secondNumber);
    }
}
```
- Code Editor (Right):** Displays the `MathTest.java` file. The code is as follows:

```
package application;

import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.BeforeClass;
import org.junit.Test;

public class MathTest {

    private static MathProvider mathProvider;

    @BeforeClass
    public static void setUp() {
        mathProvider = new MathProvider();
        System.out.println("Testing started");
    }

    @Test
    public void subtractTest() {
        Assert.assertEquals(-10, mathProvider.subtract(10, 20));
    }

    @AfterClass
    public static void tearDown() {
        System.out.println("Testing finished");
    }
}
```

Napisz testy do wszystkich pozostałych metod z klasy MathProvider



Testy jednostkowe @Ignore, timeout

```
MathTest.java x
public class MathTest {

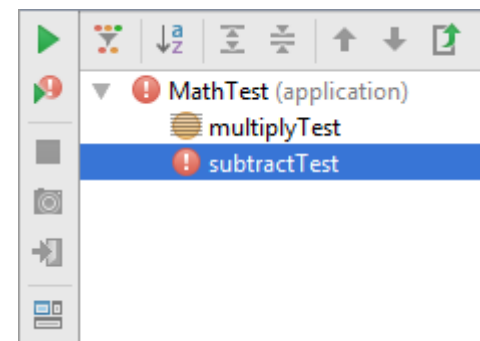
    private static MathProvider mathProvider;

    @BeforeClass
    public static void setUp() {
        mathProvider = new MathProvider();
        System.out.println("Testing started");
    }

    @Test(timeout = 1000)
    public void subtractTest() throws InterruptedException {
        Thread.sleep(1001);
        Assert.assertEquals(-10, mathProvider.subtract(11, 20));
    }

    @Ignore
    @Test
    public void multiplyTest() {
        Assert.assertEquals("Should be zero: ", 0, mathProvider.multiply(0, 1));
    }

    @AfterClass
    public static void tearDown() {
        System.out.println("Testing finished");
    }
}
```



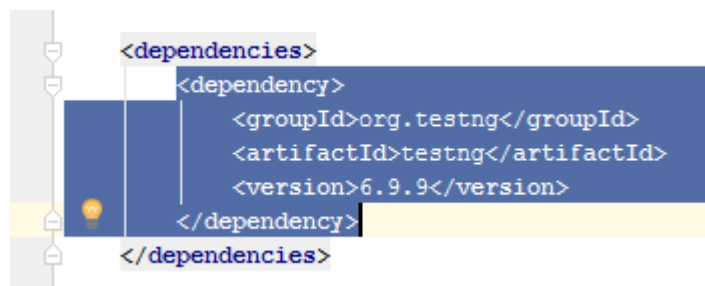
```
org.junit.runners.model.TestTimedOutException: test timed out after 1000 milliseconds
    at java.lang.Thread.sleep(Native Method)
    at application.MathTest.subtractTest(MathTest.java:19) <9 internal calls>
    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
    at java.lang.Thread.run(Thread.java:745)
```

Adnotacja @Ignore może przyjmować parametr określający powód ignorowania testu. @Ignore("Test needs to be updated")

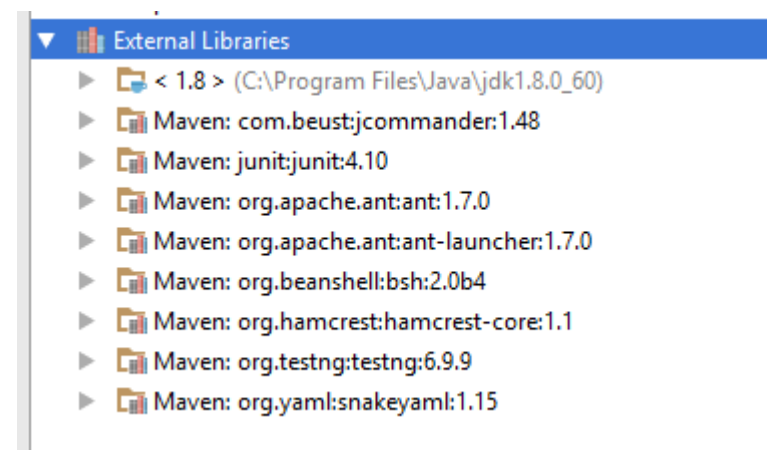
TestNG – framework, który został utworzony w oparciu o JUnit oraz NUnit wprowadzając usprawnienia pozwalające na lepszą organizację pracy z testami „more powerful and easier to use” ☺

IntelliJ domyślnie ma zainstalowany plugin do obsługi TestNG

Dodajemy TestNG do naszego projektu zamiast JUnit:



```
<dependencies>
  <dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>6.9.9</version>
  </dependency>
</dependencies>
```



- < 1.8 > (C:\Program Files\Java\jdk1.8.0_60)
- Maven: com.beust:jcommander:1.48
- Maven: junit:junit:4.10
- Maven: org.apache.ant:ant:1.7.0
- Maven: org.apache.ant:ant-launcher:1.7.0
- Maven: org.beanshell:bsh:2.0b4
- Maven: org.hamcrest:hamcrest-core:1.1
- Maven: org.testng:testng:6.9.9
- Maven: org.yaml:snakeyaml:1.15

@BeforeSuite: The annotated method will be run before all tests in this suite have run.

@AfterSuite: The annotated method will be run after all tests in this suite have run.

@BeforeTest: The annotated method will be run before any test method belonging to the classes inside the tag is run.

@AfterTest: The annotated method will be run after all the test methods belonging to the classes inside the tag have run.

@BeforeGroups: The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.

@AfterGroups: The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

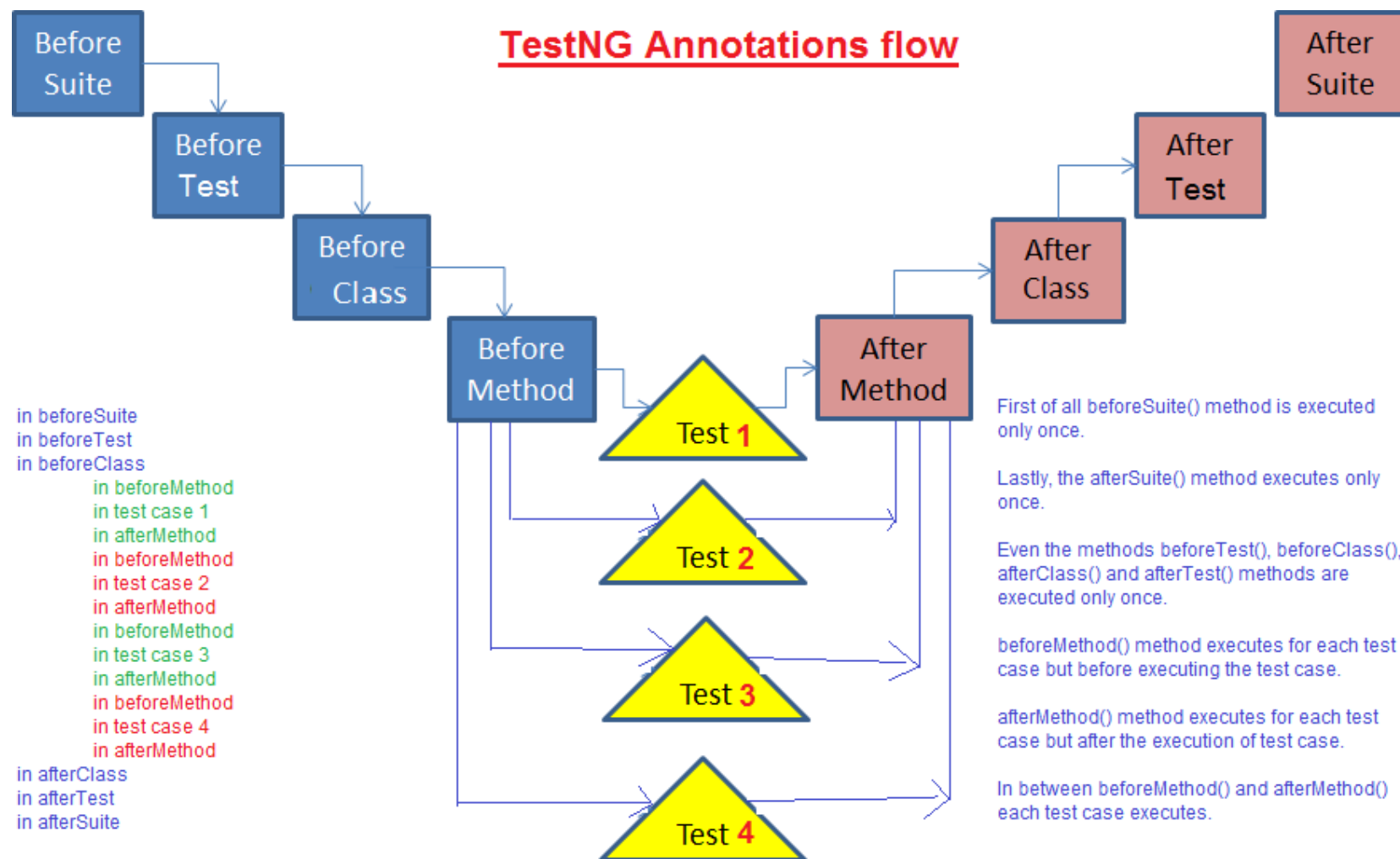
@BeforeClass: The annotated method will be run before the first test method in the current class is invoked.

@AfterClass: The annotated method will be run after all the test methods in the current class have been run.

@BeforeMethod: The annotated method will be run before each test method.

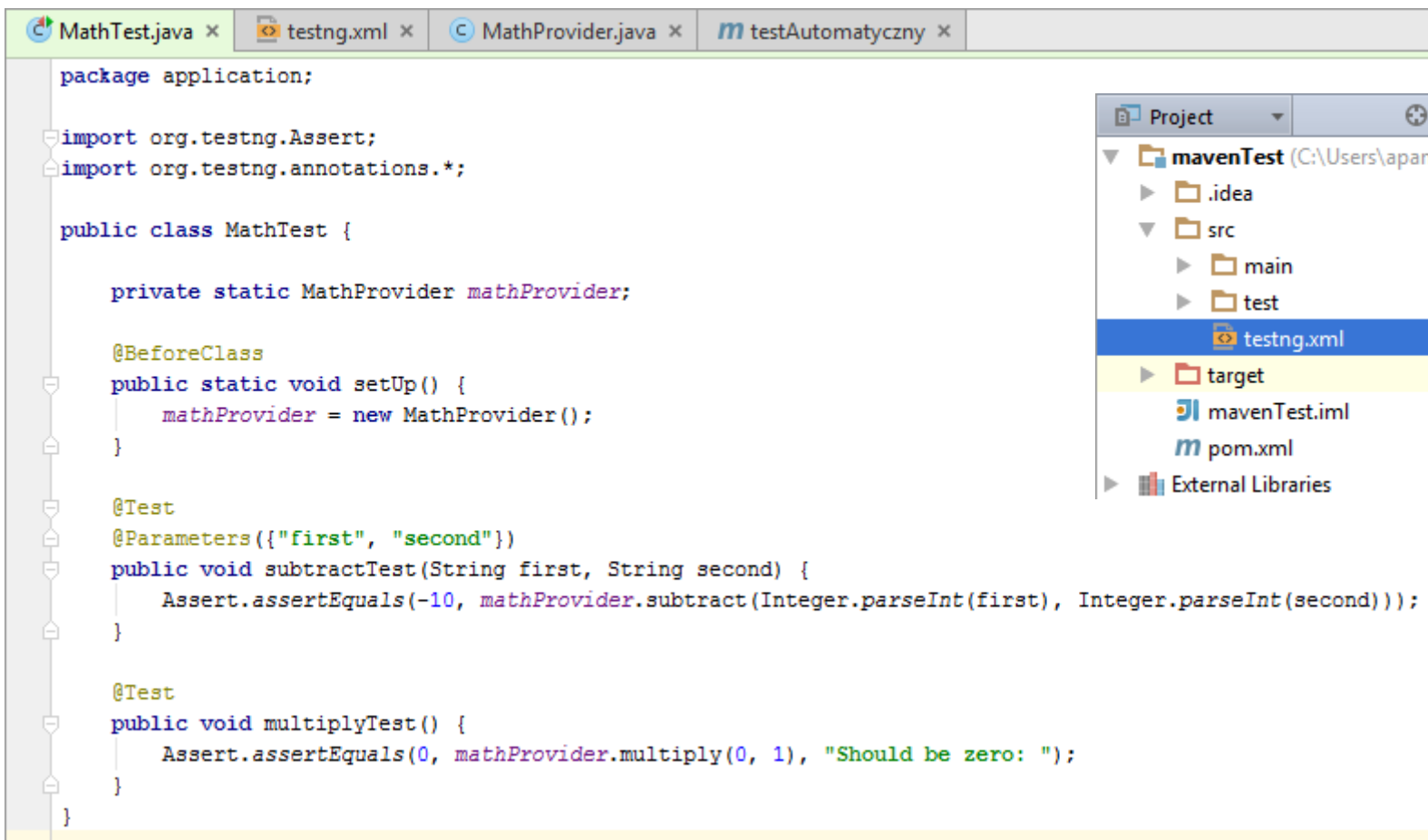
@AfterMethod: The annotated method will be run after each test method.

@Test: The annotated method is a part of a test case.




```
MathTest.java x testAutomatyczny x testng.xml x MathProvider.  
package application;  
  
import org.testng.Assert;  
import org.testng.annotations.*;  
import org.testng.annotations.Test;  
  
public class MathTest {  
  
    private static MathProvider mathProvider;  
  
    @BeforeClass  
    public static void setUp() {  
        mathProvider = new MathProvider();  
        System.out.println("Testing started");  
    }  
  
    @Test  
    public void subtractTest() {  
        Assert.assertEquals(-10, mathProvider.subtract(10, 20));  
    }  
  
    @AfterClass  
    public static void tearDown() {  
        System.out.println("Testing finished");  
    }  
}
```

```
Done: 1 of 1 (0,348 s)  
"C:\Program ...  
[TestNG] Running:  
  C:\Users\apanek\.IdeaIC14\system\temp-testng-customsuite.xml  
  
Testing started  
Testing finished  
  
=====  
Custom suite  
Total tests run: 1, Failures: 0, Skips: 0  
=====  
  
Process finished with exit code 0
```



```
package application;

import org.testng.Assert;
import org.testng.annotations.*;

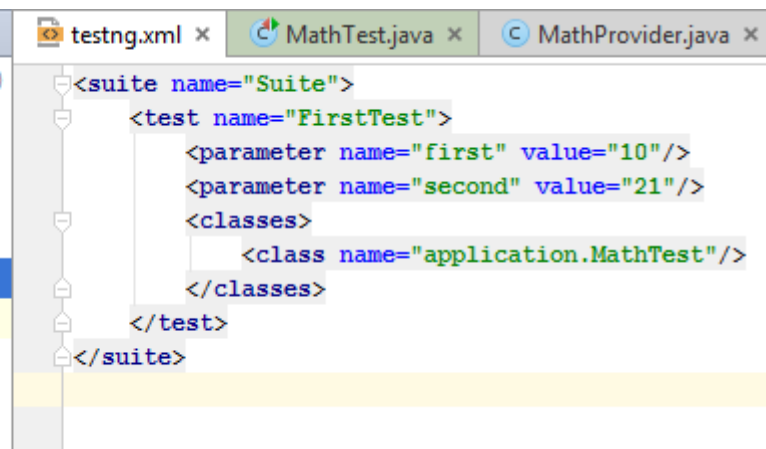
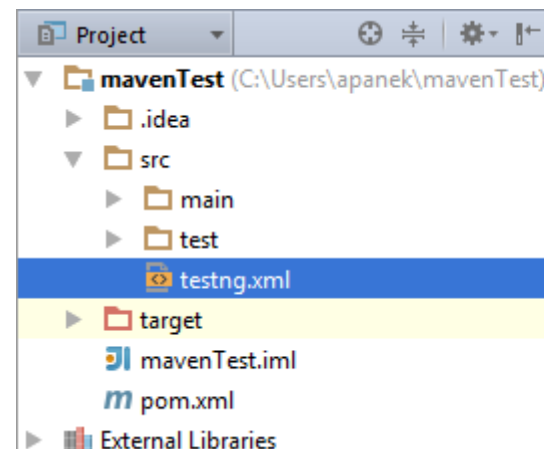
public class MathTest {

    private static MathProvider mathProvider;

    @BeforeClass
    public static void setUp() {
        mathProvider = new MathProvider();
    }

    @Test
    @Parameters({"first", "second"})
    public void subtractTest(String first, String second) {
        Assert.assertEquals(-10, mathProvider.subtract(Integer.parseInt(first), Integer.parseInt(second)));
    }

    @Test
    public void multiplyTest() {
        Assert.assertEquals(0, mathProvider.multiply(0, 1), "Should be zero: ");
    }
}
```



```
<suite name="Suite">
  <test name="FirstTest">
    <parameter name="first" value="10"/>
    <parameter name="second" value="21"/>
    <classes>
      <class name="application.MathTest"/>
    </classes>
  </test>
</suite>
```

Plik testng.xml umożliwia definiowanie zbiorów testów, które mają być uruchamiane poprzez dodawanie odpowiednich klas lub poszczególnych metod

TestNG Uruchamianie testów z Maven'a

Bezpośrednie uruchamianie testów zdefiniowanych w pliku testng.xml wymaga wskazania jego lokalizacji w pliku pom.xml za pomocą pluginu:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>2.19</version>
      <configuration>
        <suiteXmlFiles>
          <suiteXmlFile>src/testng.xml</suiteXmlFile>
        </suiteXmlFiles>
      </configuration>
    </plugin>
  </plugins>
</build>
```

T E S T S

Running TestSuite

Tests run: 2, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.388 sec <<< FAILURE! - in TestSuite
subtractTest(application.MathTest) Time elapsed: 0.005 sec <<< FAILURE!
java.lang.AssertionError: expected [-11] but found [-10]
at application.MathTest.subtractTest(MathTest.java:18)

Results :

Failed tests:

MathTest.subtractTest:18 expected [-11] but found [-10]

Tests run: 2, Failures: 1, Errors: 0, Skipped: 0



Unitils – Reflection Assert

<http://unitils.org/summary.html>

```
<dependency>
  <groupId>org.unitils</groupId>
  <artifactId>unitils-core</artifactId>
  <version>3.4.2</version>
</dependency>
```

```
MathTest.java × Person.java × index.html × testAutomatyc
package application;

import org.unitils.reflectionassert.ReflectionAssert;

public class Person {

    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public boolean isOld() {
        return age > 65 ? true : false;
    }

    public static void main(String args[]) {
        Person ala = new Person("Ala", 18);
        Person ola = new Person("Ola", 18);

        ReflectionAssert.assertReflectionEquals(ala, ola);
    }
}
```

"C:\Program ...

Exception in thread "main" junit.framework.AssertionFailedError:

Expected: Person<name="Ala", age=18>

Actual: Person<name="Ola", age=18>

--- Found following differences ---

name: expected: "Ala", actual: "Ola"

--- Difference detail tree ---

expected: Person<name="Ala", age=18>

actual: Person<name="Ola", age=18>

name expected: "Ala"

name actual: "Ola"

at junit.framework.Assert.fail(Assert.java:50)

at org.unitils.reflectionassert.ReflectionAssert.assertReflectionEquals

at org.unitils.reflectionassert.ReflectionAssert.assertReflectionEquals

at application.Person.main(Person.java:23) <5 internal calls>

Process finished with exit code 1



Unitils – Reflection Assert

```
MathTest.java × Person.java × index.html × testAutomatyczny × MathProvider.java × mavenTest.i

package application;

import org.unitils.reflectionassert.ReflectionAssert;
import org.unitils.reflectionassert.ReflectionComparatorMode;

public class Person {

    private String name;
    private int age;
    private String phone;

    public Person(String name, int age, String phone) {
        this.name = name;
        this.age = age;
        this.phone = phone;
    }

    public static void main(String args[]) {
        Person ala = new Person("Ala", 18, "123");
        Person ola = new Person("Ala", 18, null);

        ReflectionAssert.assertReflectionEquals(ola, ala, ReflectionComparatorMode.IGNORE_DEFAULTS);
    }
}
```

Run: testAutomatyczny [package] Person

"C:\Program ...

Process finished with exit code 0



Unitils – Reflection Assert

```
MathTest.java × Numbers.java × testAutomatyczny × MathProvider.java × mavenTest.iml ×  
  
package application;  
  
import org.unitils.reflectionassert.ReflectionAssert;  
import org.unitils.reflectionassert.ReflectionComparatorMode;  
  
import java.util.LinkedList;  
import java.util.List;  
  
public class Numbers {  
  
    public static void main(String args[]) {  
        List<String> list1 = new LinkedList<String>();  
        List<String> list2 = new LinkedList<String>();  
  
        list1.add("1");  
        list1.add("2");  
  
        list2.add("2");  
        list2.add("1");  
  
        ReflectionAssert.assertReflectionEquals(list1, list2, ReflectionComparatorMode.LENIENT_ORDER);  
    }  
}
```

Run: testAutomatyczny [package] Numbers

"C:\Program ...

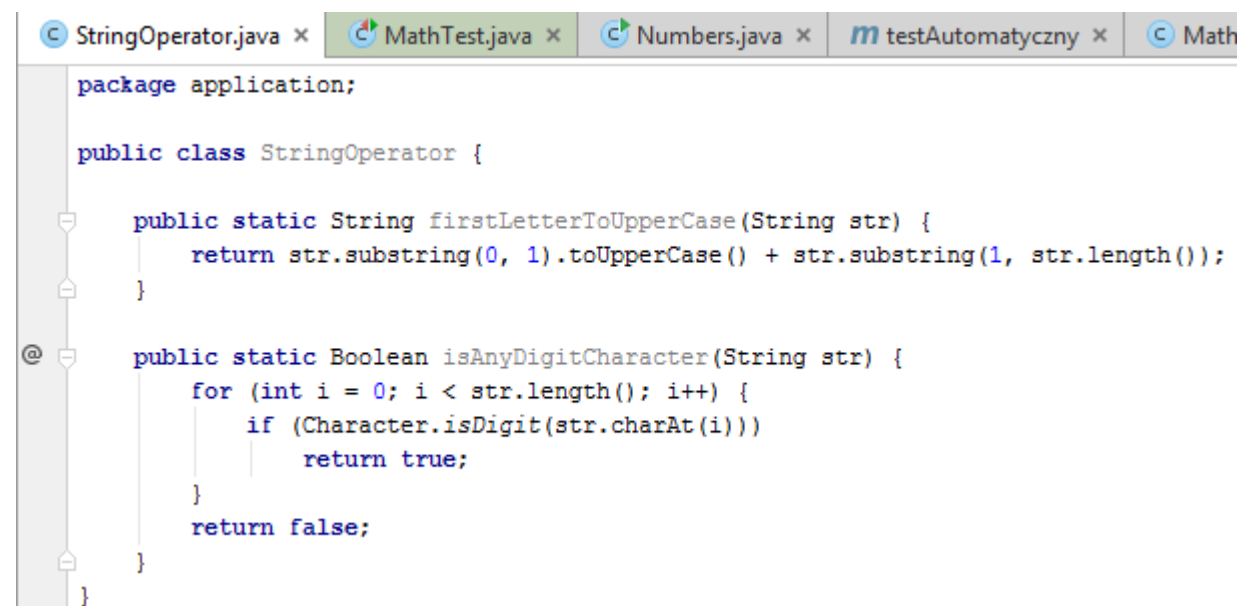
Process finished with exit code 0

Zadanie

Utwórz zestaw testów jednostkowych wykorzystując wybrany framework (JUnit lub TestNG).

Testy powinny weryfikować poprawność działania dwóch dostępnych metod:

- a) firstLetterToUpperCase
- b) isAnyDigitCharacter



```
StringOperator.java x MathTest.java x Numbers.java x m testAutomatyczny x Math
package application;

public class StringOperator {

    public static String firstLetterToUpperCase(String str) {
        return str.substring(0, 1).toUpperCase() + str.substring(1, str.length());
    }

    @
    public static Boolean isAnyDigitCharacter(String str) {
        for (int i = 0; i < str.length(); i++) {
            if (Character.isDigit(str.charAt(i)))
                return true;
        }
        return false;
    }
}
```




To wszystko!
Dzięki! 😊

