JUnit 5

Patryk Chamera
Karol Bystrek

Czym są testy jednostkowe?

JUnit 5

Testy jednostkowe to testy najniższego poziomu, które sprawdzają poprawność działania najmniejszych, odrębnych fragmentów kodu, takich jak pojedyncze metody lub funkcje. Są łatwe do automatyzacji, co umożliwia ich regularne wykonywanie i ciągłą weryfikację jakości kodu.

Co to jest JUnit?

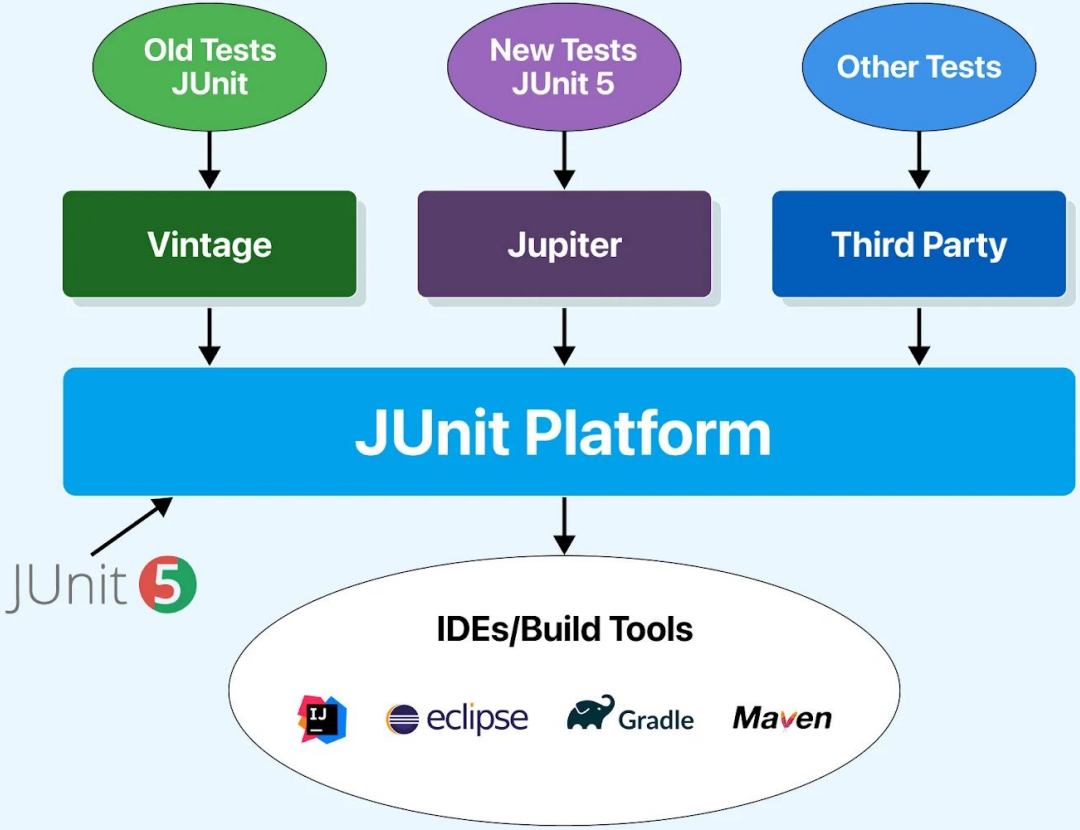
JUnit to popularny framework do testów jednostkowych w Javie. Pozwala programistom na szybkie i efektywne sprawdzanie poprawności fragmentów kodu. Dzięki automatyzacji testów JUnit:

- ułatwia identyfikację błędów na wczesnym etapie
- przyspiesza proces testowania

Z czego składa się JUnit 5?

JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage

- **JUnit Platform to baza całego frameworku, która definiuje sposób uruchamiania testów, integrację z IDE czy systemami budowania (Maven, Gradle)**
- **JUnit Jupiter to główny moduł do pisania testów, wprowadza adnotacje takie jak @Test, @BeforeEach itp.**
- **JUnit Vintage zapewnia kompatybilność wsteczną, która umożliwia uruchamianie testów napisanych w starszych wersjach**


Old Tests
JUnit
New Tests
JUnit 5
Other Tests
Vintage
Jupiter
Third Party
JUnit Platform
JUnit 5
IDEs/Build Tools
eclipse
Gradle
Maven

Uwagi dotyczące testów jednostkowych w JUnit!

1. **Izolacja testów** (testy powinny być niezależne od siebie)
2. **Zrozumiałe nazewnictwo metod testowych**
3. **Asercje jako weryfikacja testów**
4. **Testowanie przypadków pozytywnych i negatywnych**
5. **Wydajność testów**
6. **Dobrą praktyką jest pisanie testu od razu po napisaniu nowej funkcji**

Jak dodać JUnit do projektu Maven

Aby korzystać z testów jednostkowych JUnit 5 w Javie, wystarczy dodać dwie zależności w pliku pom.xml:

```xml
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.8.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>3.5.1</version>
        </plugin>
    </plugins>
</build>
```

Zależność junit-jupiter umożliwia pisanie testów jednostkowych w JUnit 5, dostarczając narzędzia i adnotacje, takie jak @Test czy @BeforeEach, które są potrzebne do tworzenia i organizowania testów.

Wtyczka maven-surefire-plugin pozwala Mavenowi uruchamiać te testy.

Jak testować kod? Przykładowy test

```java
public class Calculator {  3 usages
    public int add(int a, int b) {
        return a + b;
    }
}
```

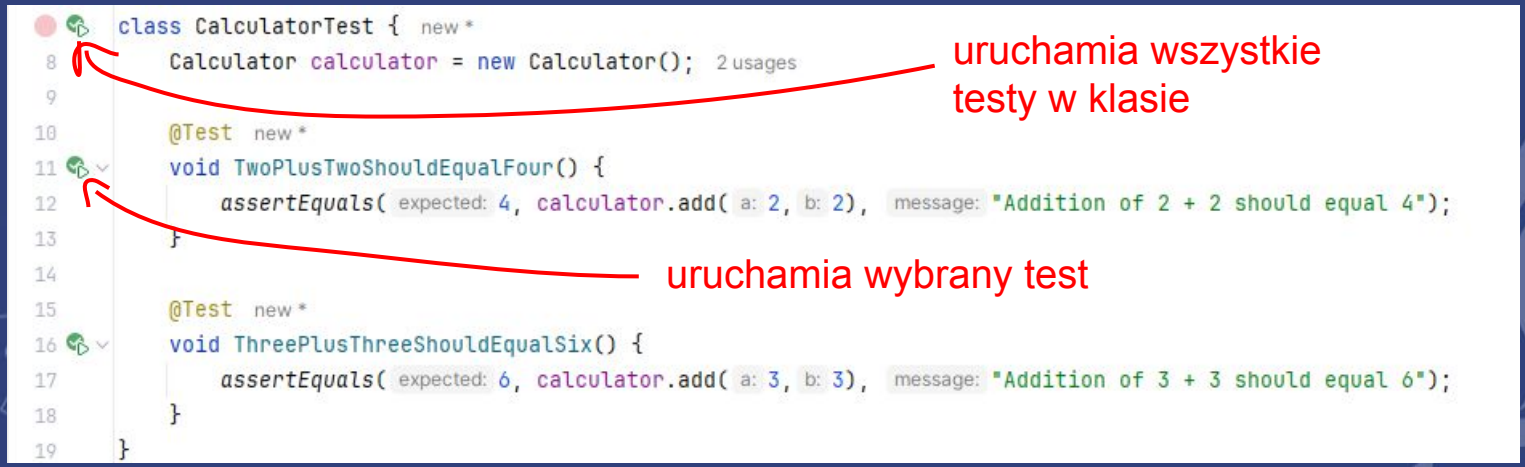
```java
class CalculatorTest {
    private final Calculator calculator = new Calculator();  1 usage
    @Test
    void twoPlusTwoShouldEqualFour() {
        assertEquals( expected: 4, calculator.add( a: 2,  b: 2),  message: "Addition of 2 + 2
         should equal 4");
    }
}
```

adnotacja @Test lub
@org.junit.Test

weryfikacja założeń

Jak uruchamiać testy

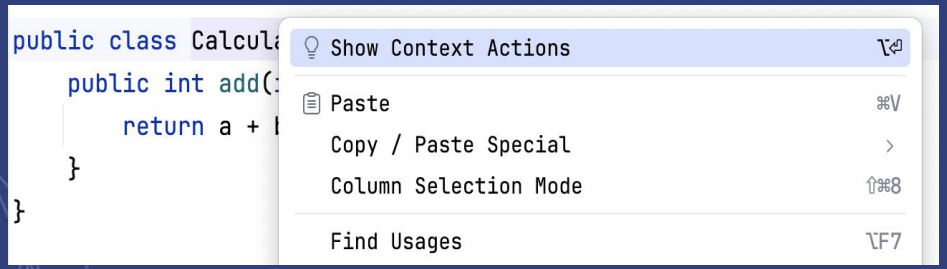
- **IDE:** klikając w małe zielone trójkąciki obok nazwy metody lub klasy

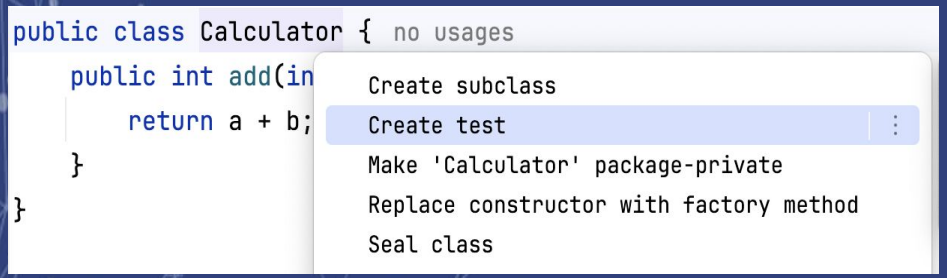


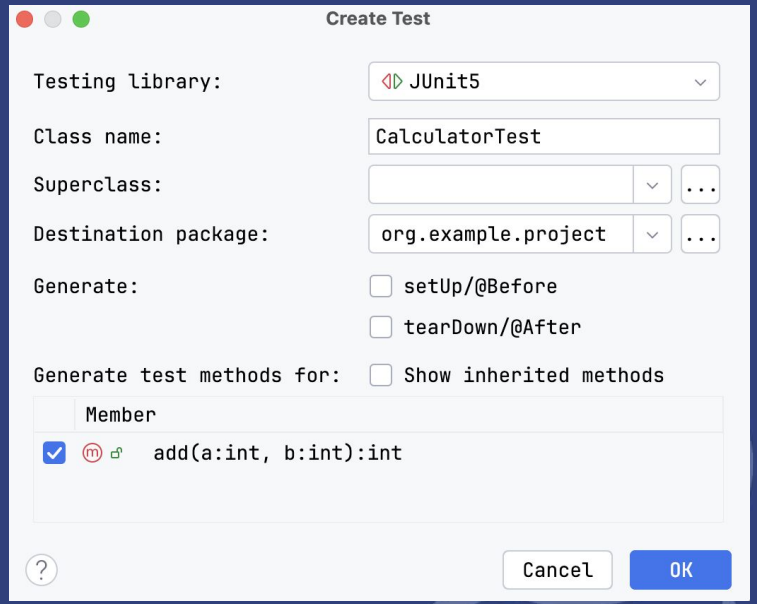
- **Linia komend:** `mvn test` ← uruchamia wszystkie testy

Aby uruchomić wybrany test należy wpisać: mvn test -Dtest=NazwaKlasyTestowej

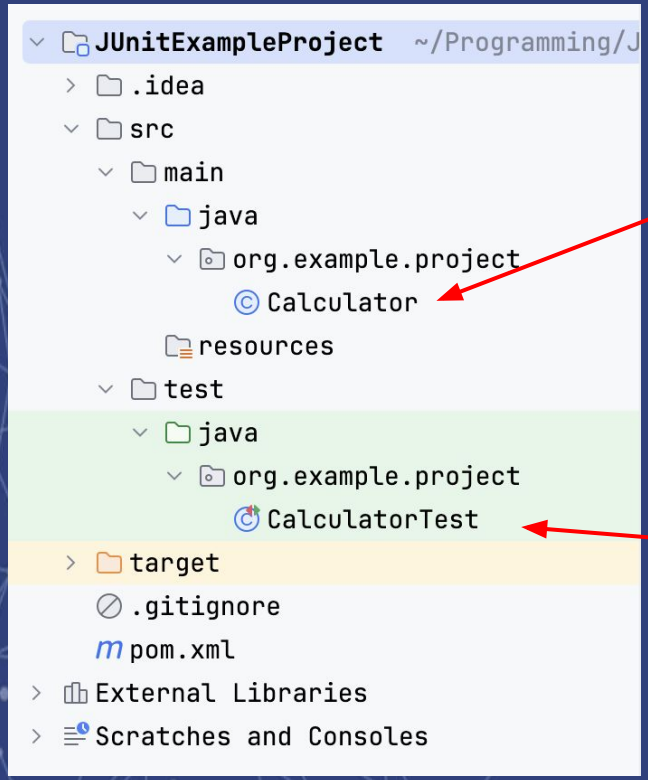
Jak automatycznie dodawać testy?







Gdzie są tworzone testy?



klasa Calculator.java,
której metody testujemy

test CalculatorTest.java

Cykl życia testów w JUnit

1. @BeforeAll - statyczna metoda wykonywana przez uruchomieniem wszystkich testów w danej klasie, jej zadaniem jest inicjalizacja zasobów współdzielonych pomiędzy testami
2. @BeforeEach - metoda wywoływana przed każdym testem, służy do ustawienia początkowego stanu każdego testu
3. @Test - test jednostkowy

Cykl życia testów w JUnit

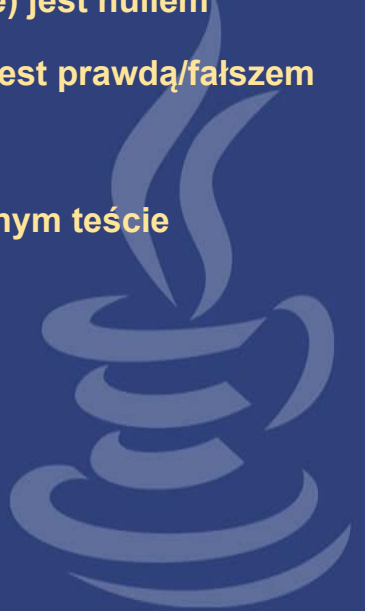
4. @AfterEach - metoda wywoływana po każdym teście, służy do zwalniania zasobów używanych przez dany test

5. @AfterAll - statyczne metoda wykonywana po uruchomieniu wszystkich testów w danej klasie, używana do zwolnienia zasobów wspólnych dla klasy testowej

Rodzaje asercji

- **assertEquals / assertNotEquals** - sprawdza czy dwie wartości są równe/różne
- **assertNull / assertNotNull** - sprawdza czy dana wartość (nie) jest nullem
- **assertTrue oraz assertFalse** - sprawdza czy podana wartość jest prawdą/fałszem
- **assertArrayEquals** - porównuje dwie tablice
- **assertAll()** - grupowanie wielu asercji w jednym teście

Rodzajów asercji jest bardzo dużo, więcej można poczytać w dokumentacji na JUnit.org

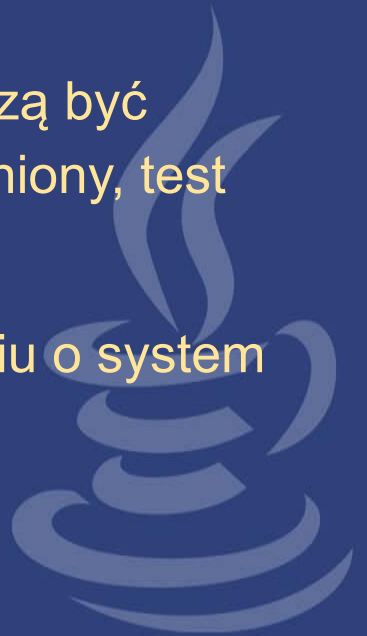


Różnica między Assert i Assume

Assert - służy do sprawdzenia poprawności wyników, jeśli nie zostanie spełniony test zakończy się błędem

Assume - używane do określenia założeń testu, które muszą być spełnione, aby test się wykonał. Jeśli warunek nie jest spełniony, test zostaje pominięty, a nie zakończy się błędem!

Przykładowe użycie Assume: uruchamianie testów w oparciu o system operacyjny użytkownika



JUnit 5

Adnotacje do zarządzania warunkami testów

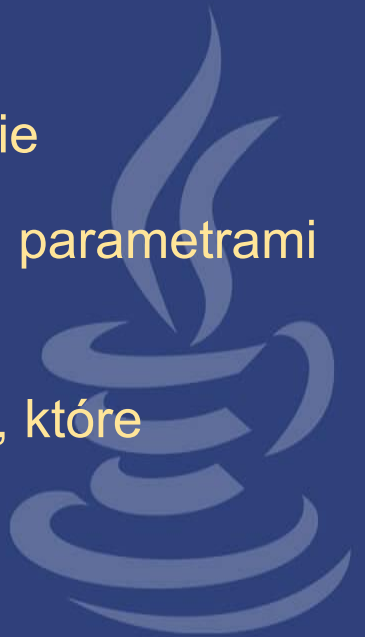
@DisplayName - ustawia niestandardową nazwę dla testu lub klasy

@Disabled - wyłącza test lub klasę testową

@RepeatedTest - umożliwia wykonywanie testu wielokrotnie

@ParameterizedTest - test zostaje uruchomiony z różnymi parametrami wejściowymi

@TestFactory - umożliwia tworzenie dynamicznych testów, które generują przypadki testowe w czasie wykonania



JUnit 5

Rodzaje parametrów do @ParametrizedTest

@ValueSource - uruchamia test dla każdej wartości

@EnumSource - uruchamia test dla każdej wartości wyliczenia

@MethodSource - używa metody jako źródła danych dla testu, metoda musi być statyczna i zwracać Stream, Iterable lub tablicę

@CsvSource - uruchamia test używając wartości w formacie CSV

@CsvFileSource - pozwala na dostarczenie danych z pliku CSV

Źródła

- https://junit.org/junit5/docs/current/user-guide/#overview
- https://www.youtube.com/watch?v=flpmSXVTqBI