

JUnit 5 to framework do testowania jednostkowego dla języka Java, a jego celem jest ułatwienie tworzenia, uruchamiania i organizowania testów jednostkowych. Jest to jedno z najbardziej popularnych narzędzi w ekosystemie Java do automatyzacji testów jednostkowych. Poniżej przedstawiam wprowadzenie do JUnit 5:

Cele JUnit 5:

1. Testy Jednostkowe:

- JUnit 5 umożliwia programistom pisanie testów jednostkowych, które sprawdzają pojedyncze jednostki kodu, takie jak metody czy klasy.

2. Łatwa Konfiguracja:

- Framework jest łatwy do skonfigurowania i używania. Wymaga minimalnego kodu do ustawienia testów.

3. Wsparcie dla Nowoczesnych Języków Java:

- JUnit 5 jest zbudowany z myślą o nowoczesnych możliwościach języka Java, takich jak lambda expressions, co czyni go bardziej elastycznym i wygodnym w użyciu.

4. Rozbudowane Funkcje Assert:

- Oferuje rozbudowany zestaw asercji, dzięki którym programista może sprawdzać oczekiwane wyniki w trakcie testów.

5. Rozbudowane Rozszerzalność:

- JUnit 5 wprowadza model rozszerzeń, co umożliwia dostosowywanie i rozszerzanie funkcji frameworka.

INSTALACJA

Aby zainstalować JUnit 5 i korzystać z niego w projekcie Java, można użyć narzędzi do zarządzania zależnościami, takich jak Maven, Gradle lub inne. Poniżej znajdziesz instrukcje instalacji JUnit 5 dla popularnych narzędzi budujących.

Maven:

Jeśli używasz Mavena, dodaj odpowiednie zależności do sekcji `<dependencies>` w pliku `pom.xml`. Poniżej znajdziesz przykład dla JUnit 5:

```
xml
Copy code

<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.8.1</version> <!-- Sprawdź najnowszą dostępną wersję na stronie JU
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Upewnij się, że plik **pom.xml** jest zapisany, a następnie Maven pobierze i zainstaluje odpowiednie biblioteki przy następnym budowaniu projektu.

Gradle:

Jeśli używasz Gradle, dodaj poniższe zależności do sekcji **dependencies** w pliku **build.gradle**:

```
groovy
Copy code

dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.8.1'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.8.1'
}
```

Następnie, uruchom komendę **./gradlew build** (dla systemów Unix) lub **gradlew build** (dla systemów Windows), aby Gradle pobrał i zainstalował biblioteki JUnit 5.

Konfiguracja w IDE:

Jeśli korzystasz z IntelliJ IDEA lub innego środowiska programistycznego, powinno ono automatycznie wykryć zależności JUnit 5 i umożliwić uruchamianie testów.

Upewnij się, że twoje IDE jest skonfigurowane do używania JUnit 5 jako frameworku do testów.

Teraz, gdy masz JUnit 5 zainstalowane w projekcie, możesz tworzyć testy jednostkowe za pomocą adnotacji **@Test** i korzystać z różnych funkcji dostępnych w JUnit 5 do testowania swojego kodu.

Kluczowe Koncepty JUnit 5:

1. Testy:

- Testy w JUnit 5 są oznaczane adnotacją `@Test`. Każda taka metoda jest uważana za test jednostkowy.

```
java Copy code  
  
@Test  
void testMethodName() {  
    // Kod testu  
}
```

Assert:

- Do sprawdzania warunków testowych używa się asercji (metody z rodziny `org.junit.jupiter.api.Assertions`). Przykłady asercji to `assertEquals`, `assertTrue`, `assertFalse` itp.

```
java Copy code  
  
@Test  
void testAddition() {  
    int result = Calculator.add(2, 3);  
    assertEquals(5, result);  
}
```

2. Cykle Życia Testu:

- JUnit 5 wprowadza cykle życia testu, takie jak `@BeforeEach`, `@AfterEach`, `@BeforeAll` i `@AfterAll`, które pozwalają na przygotowanie i sprzątanie po testach.

```
java Copy code  
  
@BeforeEach  
void setUp() {  
    // Inicjalizacja przed każdym testem  
}  
  
@Test  
void testMethod() {  
    // Kod testu  
}  
  
@AfterEach  
void tearDown() {  
    // Sprzątanie po każdym teście  
}
```

3. Parametry Testów:

- JUnit 5 pozwala na przekazywanie parametrów do testów za pomocą adnotacji `@ParameterizedTest` i fabryk do generowania zestawów danych testowych.

```
java Copy code

@ParameterizedTest
@ValueSource(ints = {1, 2, 3})
void testSquare(int value) {
    assertEquals(value * value, Calculator.square(value));
}
```

4. Testy Powtarzalne:

- o Dzięki adnotacji **@RepeatedTest** można łatwo definiować testy, które powtarzają się określoną liczbę razy.

```
java Copy code

@RepeatedTest(3)
void repeatedTest() {
    // Kod testu
}
```

5. Testy Wyjątków:

- o Adnotacja **@Test** może być używana z argumentem **expected** do testowania oczekiwanych wyjątków.

```
java Copy code

@Test(expected = ArithmeticException.class)
void testException() {
    int result = 1 / 0; // Spowoduje ArithmeticException
}
```

Przykład Prostego Testu JUnit 5:

Poniżej znajduje się przykład prostego testu jednostkowego za pomocą JUnit 5:

```
java Copy code

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class CalculatorTest {

    @Test
    void testAddition() {
        assertEquals(5, Calculator.add(2, 3));
    }

    @Test
    void testSubtraction() {
        assertEquals(2, Calculator.subtract(5, 3));
    }
}
```

W tym przykładzie, **Calculator** to klasa zawierająca metody **add** i **subtract**, a testy sprawdzają, czy te metody działają zgodnie z oczekiwaniami.

Treść ćwiczenia:

Stwórz klasę **Calculator**, która będzie zawierała metody do podstawowych operacji matematycznych (dodawanie, odejmowanie, mnożenie, dzielenie). Napisz testy jednostkowe przy użyciu JUnit 5 do sprawdzenia poprawności działania każdej z tych operacji.

Rozwiązanie:

```
java Copy code  
  
public class Calculator {  
  
    public int add(int a, int b) {  
        return a + b;  
    }  
  
    public int subtract(int a, int b) {  
        return a - b;  
    }  
  
    public int multiply(int a, int b) {  
        return a * b;  
    }  
  
    public double divide(int a, int b) {  
        if (b == 0) {  
            throw new ArithmeticException("Cannot divide by zero");  
        }  
        return (double) a / b;  
    }  
}
```

```
java Copy code

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class CalculatorTest {

    @Test
    void testAddition() {
        Calculator calculator = new Calculator();
        assertEquals(5, calculator.add(2, 3));
    }

    @Test
    void testSubtraction() {
        Calculator calculator = new Calculator();
        assertEquals(2, calculator.subtract(5, 3));
    }

    @Test
    void testMultiplication() {
        Calculator calculator = new Calculator();
        assertEquals(15, calculator.multiply(3, 5));
    }

    @Test
    void testDivision() {
        Calculator calculator = new Calculator();
        assertEquals(2.5, calculator.divide(5, 2));
    }

    @Test
    void testDivisionByZero() {
        Calculator calculator = new Calculator();
        assertThrows(ArithmeticException.class, () -> calculator.divide(5, 0));
    }
}
```

Ćwiczenie 2: Operacje na Stringach

Treść ćwiczenia:

Stwórz klasę **StringUtils** zawierającą metodę, która przyjmuje ciąg znaków i zwraca odwrócony ciąg znaków. Napisz test jednostkowy przy użyciu JUnit 5, aby sprawdzić, czy metoda działa poprawnie.

Rozwiązanie

```
java Copy code

public class StringUtils {

    public String reverseString(String input) {
        if (input == null) {
            return null;
        }
        return new StringBuilder(input).reverse().toString();
    }
}
```

```
java Copy code

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class StringUtilsTest {

    @Test
    void testReverseString() {
        StringUtils stringUtils = new StringUtils();
        assertEquals("tac", stringUtils.reverseString("cat"));
    }

    @Test
    void testReverseStringWithNull() {
        StringUtils stringUtils = new StringUtils();
        assertNull(stringUtils.reverseString(null));
    }
}
```


Ćwiczenie 3: Obsługa Wyjątków

Treść ćwiczenia:

Rozszerz klasę **Calculator** o metodę, która będzie rzucała wyjątek, jeśli podane liczby są ujemne. Napisz test jednostkowy przy użyciu JUnit 5, aby sprawdzić, czy wyjątek jest rzucany poprawnie.

Rozwiązanie:

java

 Copy code

```
public class Calculator {  
  
    public int add(int a, int b) {  
        if (a < 0 || b < 0) {  
            throw new IllegalArgumentException("Numbers must be non-negative");  
        }  
        return a + b;  
    }  
}
```

java

 Copy code

```
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.*;  
  
public class CalculatorTest {  
  
    @Test  
    void testAdditionWithNegativeNumbers() {  
        Calculator calculator = new Calculator();  
        assertThrows(IllegalArgumentException.class, () -> calculator.add(-2, 3));  
    }  
}
```