



# TDD. Testy jednostkowe

Patryk Fiałkowski  
Szymon Głodziński  
Bartosz Floryan



## Plan prezentacji

1. Czym jest testowanie oprogramowania? Testy automatyczne.
2. Metodyki wytwarzania oprogramowania. TDD.
3. Przykład TDD w praktyce.

## Czym jest testowanie oprogramowania?

**Testowanie oprogramowania** to proces związany z wytwarzaniem oprogramowania. Testowanie jest częścią procesów zarządzania jakością oprogramowania. Ma na celu weryfikację oraz walidację oprogramowania. Weryfikacja pozwala skontrolować, czy wytwarzane oprogramowanie jest zgodne z podaną specyfikacją, z kolei walidacja sprawdza, czy oprogramowanie jest zgodne z oczekiwaniami klienta.

Testowanie oprogramowania może być wdrożone w dowolnym momencie jednakże rozpoczyna się zazwyczaj już w momencie powstawania specyfikacji wymagań, które określają to w jaki sposób ma działać aplikacja. Możemy wyróżnić kilka rodzajów testów jak chociażby:

- Testowanie **negatywne** i **pozytywne**, zależnie od tego, czy chcemy udowodnić że program działa lub też nie,
- Testowanie **manualne** i **automatyczne** (kolejno uruchomienie testów przez testera lub skryptów testowych),



## Rola testera oprogramowania.



W procesie tworzenia aplikacji bierze udział wiele osób, w tym między innymi osoba zwana testerem oprogramowania. Rolą testera oprogramowania jest zweryfikowanie, czy stworzony program odpowiada wymaganiom postawionym przez osobę zamawiającą go (klienta).

W tym wypadku praca testera polega na znalezieniu sprzeczności w wymaganiach, potencjalnych luk oraz na szukaniu efektywniejszych rozwiązań, które mogą pomóc użytkownikowi w korzystaniu z programu. Testerzy często analizują to, w jaki sposób ma działać aplikacja, zanim jeszcze zostanie napisany kod. W prostych słowach można stwierdzić, że zadaniem testera jest przewidzenie jak największej liczby potencjalnych scenariuszy korzystania z aplikacji i wyeliminowanie potencjalnych błędów.



## Testy manualne.

W **Testach manualnych** analiza i sprawdzanie jakości kodu jest przeprowadzana ręcznie przez testera oprogramowania. Tester sprawdza wszystkie istotne cechy danego oprogramowania, przygotowuje scenariusze testowe i na ich podstawie generuje raporty.

Zalety:

- dokładne informacje zwrotne,
- testy manualne można wdrożyć szybciej niż testy automatyczne,

Wady:

- podatność na błędy ludzkie,
- w niektórych przypadkach uzyskanie wyników może być czasochłonne.



## Testy automatyczne.

**Testy automatyczne** w przeciwieństwie do testów manualnych nie są przeprowadzane ręcznie przez testera, zamiast tego tester korzysta ze skryptów przygotowanych specjalnie na potrzeby danego scenariusza oraz wspomaga się odpowiednim oprogramowaniem (np. Sahi dla aplikacji internetowych). Takie testy opierają się na wstępnie zaprojektowanym scenariuszu, który automatycznie porównuje otrzymane wyniki z wynikami spodziewanymi.

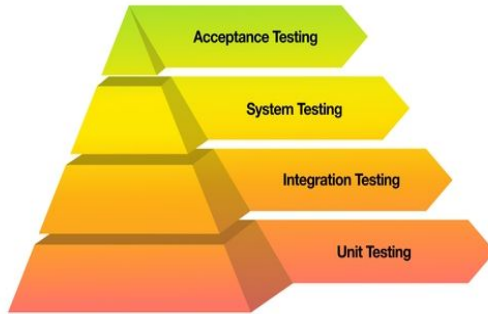
Zalety:

- łatwe monitorowanie procesu testowania,
- przy odpowiednim przygotowaniu całego procesu, można szybko otrzymywać dokładne wyniki,

Wady:

- wysoki koszt oprogramowania do testów,
- ograniczenia narzędzi (np. podatność skryptów na błędy).

## Poziomy testowania.



Testy dzieli się na cztery poziomy:

- **testy jednostkowe** (testowanie pojedynczych elementów programu np. metod),
- **testy integracyjne** (wykrywanie defektów w interfejsach i interakcjach pomiędzy modułami lub systemami),
- **testy systemowe** (konkretny system/oprogramowanie jest testowane - ocena zgodności systemu z określonymi wymaganiami),
- **testy akceptacyjne** (celem jest uzyskanie formalnego potwierdzenia wykonania oprogramowania o odpowiedniej jakości, a nie wykrycie błędów)

## Testy jednostkowe.



**Test jednostkowy** to metoda testowania tworzonego oprogramowania poprzez wykonywanie testów weryfikujących poprawność działania pojedynczych elementów (jednostek) programu – np. metod lub obiektów w programowaniu obiektowym lub procedur w programowaniu proceduralnym. Fragment programu jest wtedy poddawany testowi, który wykonuje go i porównuje wynik (np. stan obiektu, zwrócone wartości) z oczekiwanymi wynikami.

Zaletą takich testów jest możliwość wykonywania na bieżąco zautomatyzowanych testów na modyfikowanych elementach programu, dzięki czemu szybko można wychwycić błąd po jego pojawieniu się.





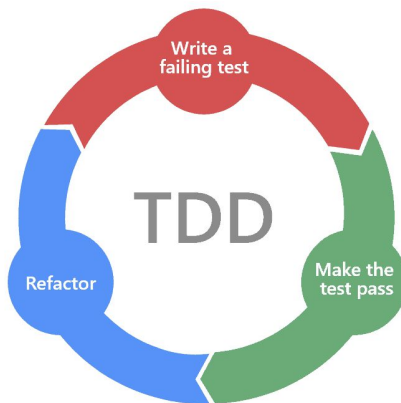
## Podział testów jednostkowych.

Testy jednostkowe dzielone są na cztery rodzaje:

- **Analiza ścieżek** – określamy punkt początkowy oraz końcowy dla przeprowadzenia testów i badamy przebieg możliwych ścieżek pomiędzy nimi.
- **Użycie klas równoważności** – wykorzystujemy klasę równoważności, która jest zbiorem danych o podobnym sposobie przetwarzania, używanych do przeprowadzenia testu. Wykonanie testu z użyciem kilku elementów zbioru powoduje uznanie całej klasy za poprawną i nie jest testowana dalsza jej część.
  - np. długość wiadomości od 1 do 20 znaków – testowane wartości (1,5,8,13,18).
- **Testowanie wartości brzegowych** – rozwinięcie testów z użyciem klas równoważności. Sprawdzanie wartości wewnątrz, pomiędzy lub przy granicy danej klasy równoważności.
  - np. długość wiadomości od 1 do 20 znaków – testowane wartości (0,1,2,19,20,21).
- **Testowanie składniowe** – sprawdzanie poprawności danych wprowadzanych do systemu.
  - np. autokorekty w MS Office.

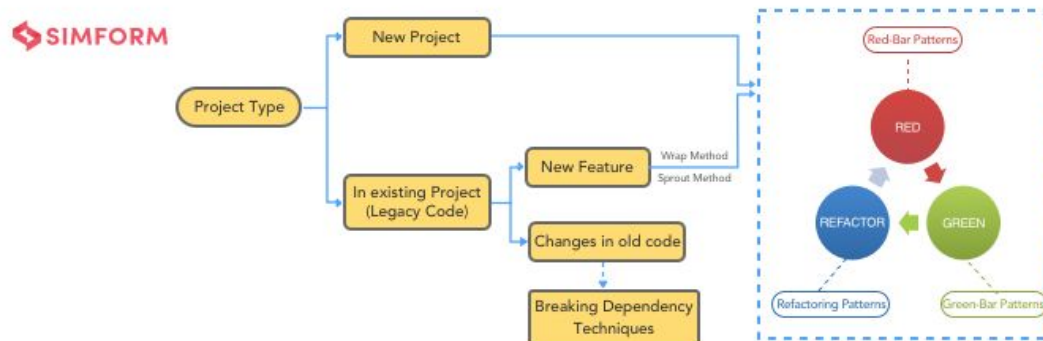
## Metodyki wytwarzania oprogramowania. TDD.

TDD (Test-Driven Development) jest techniką tworzenia oprogramowania (nie jest to technika pisania testów), w której główną ideą jest w pierwszej kolejności pisanie testów do nieistniejącej funkcjonalności, a dopiero potem napisanie kodu implementującego tę funkcjonalność.



# Cykl TDD

1. Napisanie przypadku testowego - Dobrym pomysłem jest pisanie testu, kierując się „use case’ami” i „user stories”.
2. Uruchomienie napisanego testu
3. Napisanie minimalnego kodu do przejścia testu
4. Refaktoryzacja kodu
5. Powtarzamy cykl





# Praktyki TDD

- Krótkie cykle
- Duża ilość testów
- Testy zrozumiałe dla innych
- Utrzymanie jakości kodu



## Kiedy stosujemy TDD

- Większe, bardziej skomplikowane projekty
- Dokładnie znamy efekt końcowy jaki chcemy osiągnąć
- Testujemy kod z konkretnymi funkcjami, wprowadzając logikę biznesową
- Testy nie są bardzo czasochłonne



## Zalety i wady

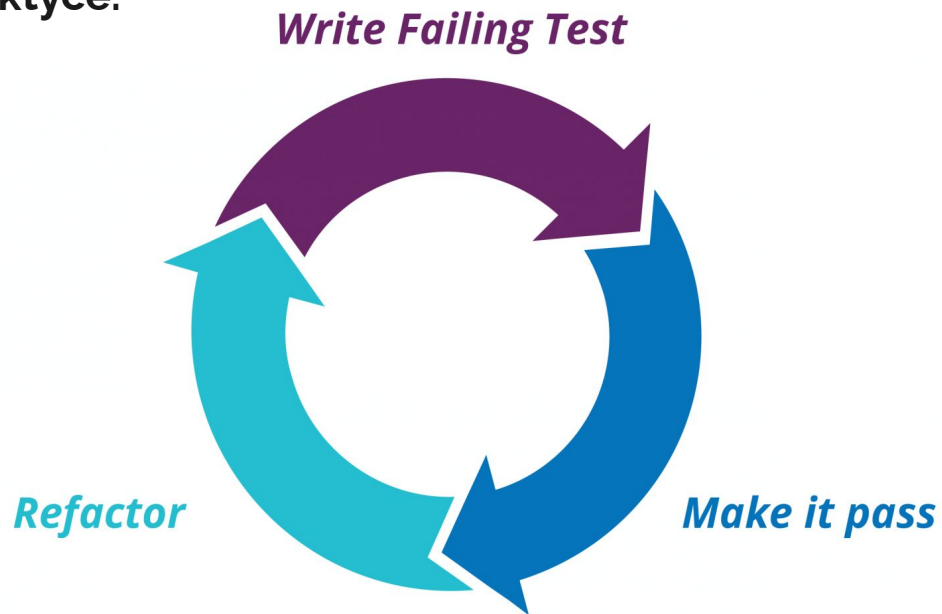


- Pozwala wykryć bugi w konkretnym kodzie przed wdrożeniem kolejnych funkcjonalności lub przed wdrożeniem na produkcję
- Możemy angażować mniej osób w poprawki kodu

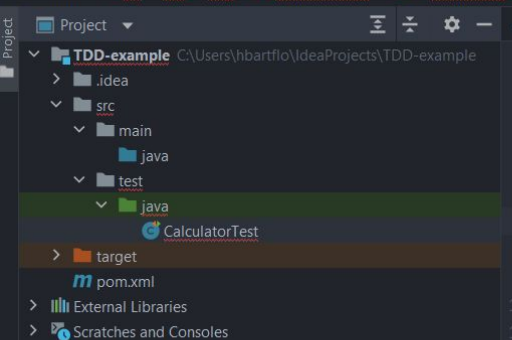


- Tworzenie testów może być czasochłonne
- Musimy bardzo konkretnie znać wszelkie zagadnienia procesu, przypadki graniczne itd.

Przykład TDD w praktyce.



TDD-example > src > test > java > CalculatorTest > AddingTest



```
1 import org.junit.Test;
2
3 public class CalculatorTest {
4
5     @Test
6     public void AddingTest(){
7         Calculator calculator = new Calculator();
8     }
9 }
10
11
12
13
```

Build: Sync x Build Output x

▼ TDD-example: build failed At 14/05/202, 5 sec, 366 ms

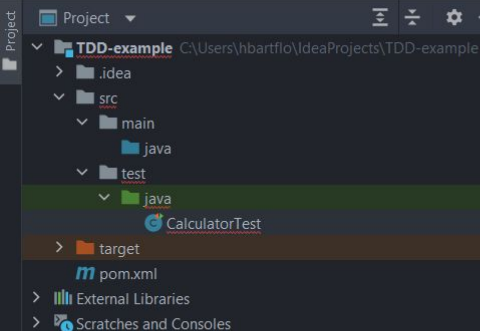
- CalculatorTest.java src\test\java 2 errors
  - cannot find symbol class Calculator:7
  - cannot find symbol class Calculator:7

C:\Users\hbartflo\IdeaProjects\TDD-example\src\test\java\CalculatorTest.java:7:9

```
java: cannot find symbol
symbol:   class Calculator
location: class CalculatorTest
```



TDD-example > src > test > java > CalculatorTest > AddingTest



```
1 import org.junit.Test;
2
3 public class CalculatorTest {
4
5     @Test
6     public void AddingTest(){
7         Calculator calculator = new Calculator();
8
9
10
11
12
13
```

- Create class 'Calculator'
- Create enum 'Calculator'
- Create inner class 'Calculator'
- Create interface 'Calculator'
- Create type parameter 'Calculator'
- Add Maven dependency...
- Search for dependency...
- Replace explicit type with 'var'
- Split into declaration and assignment

Press Ctrl+Shift+I to open preview

Build: Sync x Build Output x

- TDD-example: build failed At 14/05/202, 5 sec, 366 ms
  - CalculatorTest.java src\test\java 2 errors
    - cannot find symbol class Calculator:7
    - cannot find symbol class Calculator:7

```
C:\Users\hbartflo\IdeaProjects\TDD-example\src\test\java\CalculatorTest.java:7:9
java: cannot find symbol
symbol:   class Calculator
location: class CalculatorTest
```

TDD-example > src > test > java > CalculatorTest

Project

- TDD-example C:\Users\hbartfio\IdeaProjects\TDD-example
  - .idea
  - src
    - main
      - java
        - Calculator
    - test
      - java
        - CalculatorTest
    - target
  - External Libraries
    - < 11 > C:\Program Files\AdoptOpen\DK\jdk-11.0.8.10-hotspot
    - Maven: junit:junit:4.13
    - Maven: org.hamcrest:hamcrest-core:1.3
  - Scratches and Consoles

CalculatorTest.java

```
1 import org.junit.Test;
2
3 public class CalculatorTest {
4
5     @Test
6     public void AddingTest(){
7         Calculator calculator = new Calculator();
8     }
9 }
10
11
12
13 }
```

Run: CalculatorTest

Tests passed: 1 of 1 test – 5 ms

- CalculatorTest 5 ms
  - AddingTest 5 ms

TDD-example > src > test > java > CalculatorTest > AddingTest

Project

- TDD-example
- idea
- src
  - main
    - java
      - Calculator
  - test
    - java
      - CalculatorTest
  - target
  - pom.xml
  - TDD-example.iml- External Libraries
  - < 11 > C:\Program Files\AdoptOpenJDK\jdk-11.0.8.10-hotspot
  - Maven: junit:junit:4.13
  - Maven: org.hamcrest:hamcrest-core:1.3
- Scratches and Consoles

```
1 import org.junit.Assert;
2 import org.junit.Test;
3
4 public class CalculatorTest {
5
6     @Test
7     public void AddingTest(){
8         Calculator calculator = new Calculator();
9         calculator.add(2,2);
10
11     }
12
13
14 }
```

Create method 'add' in 'Calculator'

Press Ctrl+Shift+I to open preview

Run: CalculatorTest

Tests passed: 1 of 1 test - 6 ms

CalculatorTest 6 ms

AddingTest 6 ms

"C:\Program Files\AdoptOpenJDK\jdk-11.0.8.10-hotspot\bin\java.exe" -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.3\lib\idea\_rt.jar=51465:C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.3\bin" -Dfile.encoding=UTF-8 -classpath "C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.3\lib\idea\_rt.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.3\plugins\junit\lib\junit5-rt.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.3\plugins\junit\lib\junit-rt.jar;C:\Users\hbartflo\IdeaProjects\TDD-example\target\test-classes;C:\Users\hbartflo\IdeaProjects\TDD-example\target\classes;C:\Users\hbartflo\IdeaProjects\TDD-example\external\junit\junit-4.13.1\junit-4.13.1.jar;C:\Users\hbartflo\IdeaProjects\TDD-example\external\hamcrest\hamcrest-core-1.3.jar"

TDD-example > src > main > java > Calculator

Project

- TDD-example C:\Users\hbartflo\IdeaProjects\TDD-example
  - .idea
  - src
    - main
      - java
        - Calculator
      - test
        - java
          - CalculatorTest
      - target
    - External Libraries
      - < 11 > C:\Program Files\AdoptOpenJDK\jdk-11.0.8.10-hotspot
      - Maven: junit:junit:4.13
      - Maven: org.hamcrest:hamcrest-core:1.3
    - Scratches and Consoles

m pom.xml (TDD-example) x CalculatorTest.java x Calculator.java x

```
1 public class Calculator {  
2  
3     public void add(int a, int b) {  
4  
5     }  
6 }  
7
```

2 ^ v

String Manipulation

Database

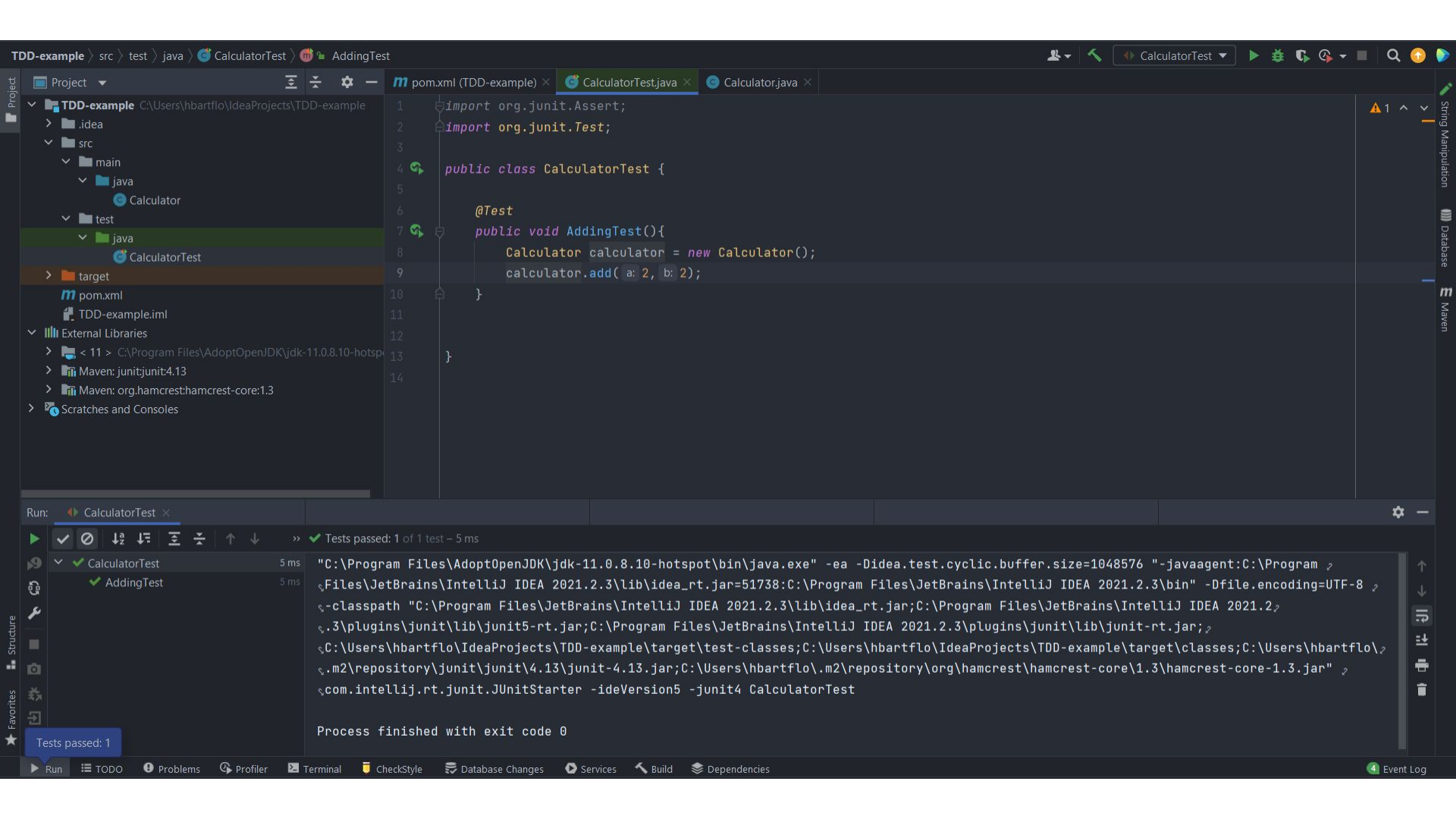
Maven

Build: Sync x Build Output x

TDD-example: build failed At 14/05/202, 2 sec, 184 ms

- CalculatorTest.java src\test\java 1 error
  - 'void' type not allowed here :9

C:\Users\hbartflo\IdeaProjects\TDD-example\src\test\java\CalculatorTest.java:9:43  
java: 'void' type not allowed here



TDD-example > src > test > java > CalculatorTest > AddingTest

Project

- TDD-example C:\Users\hbartflo\IdeaProjects\TDD-example
  - .idea
  - src
    - main
      - java
        - Calculator
    - test
      - java
        - CalculatorTest
    - target
      - pom.xml
      - TDD-example.iml
  - External Libraries
    - < 11 > C:\Program Files\AdoptOpenJDK\jdk-11.0.8.10-hotspot
    - Maven: junit:junit:4.13
    - Maven: org.hamcrest:hamcrest-core:1.3
  - Scratches and Consoles

```
1 import org.junit.Assert;
2 import org.junit.Test;
3
4 public class CalculatorTest {
5
6     @Test
7     public void AddingTest(){
8         Calculator calculator = new Calculator();
9         Assert.assertEquals(calculator.add(2,2), actual: 4);
10    }
11
12
13 }
14
```

Build: Sync × Build Output ×

❗ TDD-example: build failed At 14/05/202, 2 sec, 169 ms

- CalculatorTest.java src\test\java 1 error
  - cannot find symbol method add(int,int):9

C:\Users\hbartflo\IdeaProjects\TDD-example\src\test\java\CalculatorTest.java:9:39  
java: cannot find symbol  
symbol: method add(int,int)  
location: variable calculator of type Calculator

TDD-example > src > main > java > Calculator

Project

TDD-example

CA\Users\hbartflo\IdeaProjects\TDD-example

>

src

>

main

>

java

>

Calculator

>

test

>

java

>

CalculatorTest

>

target

>

pom.xml

>

TDD-example.iml

>

External Libraries

>

< 11 > C:\Program Files\AdoptOpenJDK\jdk-11.0.8.10-hotspot

>

Maven: junit:junit:4.13

>

Maven: org.hamcrest:hamcrest-core:1.3

>

Scratches and Consoles

m pom.xml (TDD-example) x

CalculatorTest.java x

Calculator.java x

1

2

3

4

5

6

7

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

Build: Sync x Build Output x

TDD-example: build failed At 14/05/2022: 2 sec, 169 ms

CalculatorTest.java src\test\java 1 error

cannot find symbol method add(int,int):9

C:\Users\hbartflo\IdeaProjects\TDD-example\src\test\java\CalculatorTest.java:9:39

java: cannot find symbol

symbol: method add(int,int)

location: variable calculator of type Calculator

Structure

String Manipulation

Database

Maven






```
1  import org.junit.Assert;
```

```
2 import org.junit.Test;
```

```
3
4 public class CalculatorTest {
```



»  Tests passed: 1 of 1 test – 9 ms



>

1

```
"C:\Program Files\AdoptOpenJDK\jdk-11.0.8.10-hotspot\bin\java.exe" -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.3\lib\idea_rt.jar=51792:C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.3\bin" -Dfile.encoding=UTF-8 -classpath "C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.3\lib\idea_rt.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.3\plugins\junit\lib\junit5-rt.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.3\plugins\junit\lib\junit-rt.jar;C:\Users\hbartflo\IdeaProjects\TDD-example\target\test-classes;C:\Users\hbartflo\IdeaProjects\TDD-example\target\classes;C:\Users\hbartflo\m2\repository\junit\junit4.13\junit4.13.jar;C:\Users\hbartflo\m2\repository\org\hamcrest\hamcrest-core\1.3\hamcrest-core-1.3.jar" com.intellij.rt.junit.JUnit4TestRunner -ideVersion5 -junit4 CalculatorTest
```

```
Process finished with exit code 0
```

Run TODO Problems Profiler Terminal CheckStyle Database Changes Services Build Dependencies

#### 4 Event Log