Testowanie – Testy jednostkowe

Wstęp

W trakcie zajęć zapoznasz się z koncepcją testów jednostkowych, narzędziem JUnit przeznaczonym do ich tworzenia oraz zastosujesz kilka podstawowych podejść doboru odpowiednich przypadków testowych.

JUnit to narzędzie umożliwiające automatyczne testowanie jednostkowe kodu napisanego w Javie (istnieją również odpowiedniki dla innych języków, np. Microsoft Tests i NUnit dla C#, czy CppUnit dla C++). Sposób pracy z JUnitem będzie pokazany na przykładzie środowiska programistycznego IntelliJ, z którym JUnit bardzo dobrze się integruje.

Zadanie 1. Mój pierwszy test jednostkowy

Klasa `Calculator` zawiera m.in. metody `add` i `multiply`, które służą do obliczania sumy i iloczynu dwóch liczb całkowitych. Twoim pierwszym zadaniem będzie napisanie testów dla tych metod.

1. Ściągnij i rozpakuj archiwum *Projekt.zip*
2. Otwórz w IntelliJ projekt *UnitTests*
3. Dodaj klasę testującą:
    i. Otwórz klasę `put.io.testing.junit.Calculator`
    ii. Kliknij nazwę klasy w edytorze kodu i zaczekaj aż się pojawi na ekranie żarówka
    iii. Kliknij żarówkę (lub wciśnij kombinację klawiszy *Alt+Enter*) i wybierz opcję menu *Create test*
    iv. Pozostaw wszystkie domyślne ustawienia i kliknij OK
4. Napisz metody testujące dla metody `add` i `multiply`. W każdej z nich zweryfikuj przy pomocy asercji wyniki przynajmniej dwóch wywołań danej metody, podając różne zestawy argumentów. Pamiętaj o konwencji: nazwa metody testującej powinna zaczynać się od `test`. Nie zapomnij również o dopisaniu adnotacji `@Test` – bez niej metoda nie będzie uruchomiona w trakcie testów.

5. Uruchom przed chwilą napisane testy: kliknij zieloną ikonkę po lewej stronie edytora przy definicji klasy `CalculatorTest`

Czy Twoje testy wykryły jakieś błędy w kodzie klasy `Calculator`? Jeśli nie, postaraj się bardziej i dodaj kolejne asercje – te błędy naprawdę tam są.

6. Napraw kod błędnej metody
7. Uruchom testy ponownie

Wszystko świeci się na zielono? Brawo! Możesz przejść do kolejnego zadania.

Zadanie 2. Oczekiwanie na wyjątek

Trzecia metoda klasy `Calculator`, `addPositiveNumbers`, przyjmuje jako argumenty dwie liczby i zwraca ich sumę, ale tylko wtedy, kiedy ich wartości są większe lub równe 0. W przeciwnym wypadku metoda rzuca wyjątek `IllegalArgumentException`. W trakcie tego zadania nauczysz się jak wyrazić w teście, że oczekujesz od testowanego kodu by w konkretnej sytuacji rzucił właściwy wyjątek.

1. Dodaj w odpowiedniej metodzie testującej wywołanie metody `addPositiveNumbers` dla argumentów, takich że x < 0 i y > 0.
2. Uruchom testy. Jaki jest wynik dla testu dodanego przed chwilą?
3. Dodaj wywołanie metody `assertThrows`, która będzie oczekiwać na wyjątek `IllegalArgumentException` po uruchomieniu metody `addPositiveNumbers` (przykład użycia)
4. Uruchom ponownie wszystkie testy

Jeżeli wszystkie testy przechodzą – udało Ci się!

Zadanie 3. Przygotowanie do testów i sprzątanie po

Poprzednie testy były dość proste – w każdym teście tworzyłeś obiekt testowany, uruchamiałeś testowaną metodę i przy pomocy asercji sprawdzałeś czy zwraca odpowiedni wynik. Może się jednak okazać, że proces tworzenia i konfiguracji obiektu jest bardziej skomplikowany, ale za to identyczny dla

każdego przypadku testowego. Zamiast powielać kod w każdej metodzie lepiej "wypchnąć" go do osobnej metody, która będzie uruchamiana przed każdym wywołaniem metody testującej.

1. Dodaj do klasy testującej pole prywatne `Calculator calculator`
2. Dodaj metodę, którą możesz nazwać np. `setUp`. Oznacz ją taką adnotacją, by JUnit uruchamiał ją przed każdym testem. Dodaj kod, który przypisze polu calculator nową instancję klasy `Calculator`.
3. Zmodyfikuj wszystkie metody testujące, by zamiast samodzielnie tworzyć obiekt klasy `Calculator`, korzystały z pola `calculator`

> Pytanie 3.1. Czy testy przestałyby działać, gdyby zmieniono adnotację przy metodzie `setUp` z `BeforeEach` na `BeforeAll`? Uzasadnij swoją odpowiedź.

Zadanie 4. Failure kontra Error

JUnit zlicza niepowodzenia testów i przedstawia je jako dwie liczby: *Errors* i *Failures*. Czy potrafisz odpowiedzieć "z pamięci", kiedy niepowodzenie testu jest traktowane jako *Error*, a kiedy jako *Failure*? Zamiast zaglądać do wykładu sprawdź to w praktyce!

1. Dodaj nową klasę testującą i nazwij ją `FailureOrErrorTest`
2. Dodaj dwie metody testujące: `test1` i `test2`.
3. W metodzie `test1` wpisz asercję, dla której zdefiniowany warunek zawsze będzie fałszywy
4. W metodzie `test2` wpisz instrukcję, która rzuca dowolny wyjątek
5. Uruchom testy by znaleźć odpowiedź na niżej postawione pytanie

> Pytanie 4.1. Która metoda zostanie oznaczona jako *Failure*, a która jako *Error*?

Czy zastanawiałeś się w jaki sposób JUnit dowiaduje się o tym, że test nie przeszedł? Zobacz co by się stało jakbyś przechwycił w teście „wszystko co da się rzucać” (w tym stwierdzeniu zawarta jest podpowiedź).

1. Dodaj metodę `test3` do klasy testującej
2. Dodaj block *try-catch*. W sekcji try umieść asercję, która zawsze jest

fałszywa. W sekcji catch spróbuj wyświetlić `stackTrace` dla przechwyconego obiektu.

3. Uruchom testy i sprawdź co to za obiekt

> Pytanie 4.2. Na jaki typ obiektu rzucanego oczekuje JUnit by stwierdzić, że test się nie powiódł w sensie kategorii *Failure* (będzie oznaczony wykrzyknikiem na tle żółtego kółka).

Jak nie masz pomysłu na rozwiązanie to spróbuj prześledzić drzewo dziedziczenia dla klasy `Exception` oraz jej siostry `Error`. Obie klasy mają wspólnego przodka. Podając w bloku `catch` nazwę tej klasy, masz możliwość przechwycić każdy wyjątek jaki można wygenerować w Javie, nawet ten generowany przez asercję. Spróbuj w bloku `catch` wyświetlić nazwę klasy takiego wyjątku.

Oczywiście przechwytywanie błędów asercji w praktyce jest bardzo ryzykowne (jak się zapewne domyślasz). To zadanie miało jedynie przybliżyć Ci zasady działania JUnita i jego Test Runnera.

Zadanie 5. Analiza ścieżek

Projekt *UnitTests* zawiera klasę `AudiobookPriceCalculator`. Twoim zadaniem jest napisanie testów dla metody `calculate` wykorzystując podejście analizy ścieżek działania programu.

Zanim przejdziemy do pisania testów, proste pytanie:

> Pytanie 5.1. Jaki to typ testowania: blackbox czy whitebox?

Testowanie przy pomocy analizy ścieżek zostało krótko opisane w Wikipedii. W skrócie, powinieneś stworzyć tyle testów ile jest możliwych ścieżek w metodzie `calculate`.

> Pytanie 5.2. Ile możliwych ścieżek działania znajduje się w metodzie `calculate`, zakładając że punkt startowy to początek metody, a punkt końcowy to jej koniec? (Zanim przejdziesz do kolejnego kroku zadania, skonsultuj odpowiedź z prowadzącym.)

1. Dodaj pierwszy przypadek testowy dla klasy `AudiobookPriceCalculator` w taki sam sposób jak w *Zadaniu 1*

2. Dodaj po jednym przypadku testowym dla każdej z możliwych ścieżek metody `calculate`
3. Uruchom testy, powinny przechodzić

Jeżeli wszystkie Twoje testy kończą się sukcesem to znaczy, że udało Ci się zrealizować wszystkie zadania przewidziane na dzisiaj. Brawo!