AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

NIEZAWODNOŚĆ OPROGRAMOWANIA

Aplikacja do tworzenia renowacji testowanie TDD za pomocą Junit

Autor:
Jakub Sarota
Grupa P2

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2023

Spis treści

1. Ogólne określenie wymagań

Celem niniejszej pracy jest zapewnienie poprawności i niezawodności Aplikacja do tworzenia renowacji, napisanej w języku Java.

1.1. Test when_calculatingVolumeWithValidDimensions_then_returnCo

Ten test sprawdza poprawność działania funkcji która oblicza objętość na podstawie przekazanych wymiarów w postaci stringa.

- Sprawdzamy, czy wynik obliczeń nie jest nullem, czyli czy funkcja zwraca jakąś wartość.
- Sprawdzamy, czy wynik obliczeń zgadza się z oczekiwaną wartością.

1.2. Test when_calculatingFieldWithValidDimensions_then_returnCorr

Ten test sprawdza poprawność działania funkcji która oblicza pole na podstawie przekazanych wymiarów w postaci stringa.

- Sprawdzamy, czy wynik obliczeń nie jest nullem, czyli czy funkcja zwraca jakąś wartość.
- Sprawdzamy, czy wynik obliczeń zgadza się z oczekiwaną wartością.

2. Analiza problemu

W analizie problemu związanej z aplikacją do tworzenia renowacji dążymy do pełnego zrozumienia przyczyn oraz ustalenia niezbędnych funkcji, które powinny być uwzględnione w aplikacji w konsolowej wersji. Naszym celem jest skupienie się na identyfikacji kluczowych aspektów, obejmujących zarówno funkcje, jak i aspekty niefunkcjonalne. Chcemy precyzyjnie określić zakres projektu, aby móc skutecznie i kompleksowo rozwiązać problem związany z tworzeniem renowacji w wersji konsolowej aplikacji.

2.1. Wyjaśnienie podstawowych pojęć

Test-Driven Development (TDD) to metoda programowania, w której proces tworzenia oprogramowania rozpoczyna się od napisania testów automatycznych, a następnie implementacji kodu, który spełnia te testy. Cykl TDD składa się z trzech kroków:

- Napisz test: Programista rozpoczyna od napisania testu jednostkowego, który opisuje jedną konkretną funkcję lub cechę oprogramowania, której jeszcze nie ma. Test ten powinien być prosty i minimalny, skupiając się na jednym aspekcie kodu.

- Uruchom testy: Następnie programista uruchamia wszystkie testy, włącznie z tymi, które zostały już napisane wcześniej. Nowo dodany test oraz wcześniejsze testy powinny zakończyć się niepowodzeniem, ponieważ funkcjonalność nie została jeszcze zaimplementowana.

- Napisz kod: W tym kroku programista implementuje kod, aby spełnić warunki postawione przez testy. Celem jest napisanie minimalnej ilości kodu, która pozwoli na przejście wszystkich testów.

Metoda testów Test-Driven Development (TDD) opiera się na cyklu Red-Green-Refactor, składającym się z trzech kluczowych kroków:

- Red (Czerwony): W pierwszym etapie programista tworzy test jednostkowy opisujący funkcję lub cechę oprogramowania, która jeszcze nie istnieje lub wymaga zmian. Nazwa "Czerwony"odzwierciedla fakt, że na tym etapie testy zazwyczaj kończą się niepowodzeniem, ponieważ funkcjonalność nie została jeszcze zaimplementowana.

- Green (Zielony): Następnie celem jest napisanie kodu, który spełni warunki postawione przez testy. Nowa funkcjonalność jest implementowana w taki sposób, aby wszystkie testy przechodziły pomyślnie, zmieniając ich status z "czerwonych"na "zielone"(powodzenie).

- Refactor (Refaktor): Po upewnieniu się, że testy przechodzą, programista przystępuje do poprawy jakości kodu, bez zmiany funkcjonalności. Proces ten obejmuje dostosowywanie struktury kodu, eliminowanie duplikatów i ogólną optymalizację. Wartość testów jednostkowych polega na tym, że pozostają "zielone"nawet po wprowadzeniu zmian, co świadczy o tym, że funkcjonalność nie uległa pogorszeniu.

Cykl Red-Green-Refactor jest powtarzany wielokrotnie podczas procesu TDD. Każda iteracja dodaje nową funkcję lub poprawia istniejącą, przy zachowaniu wysokiej jakości kodu i pełnej pokrycia testami. Po zaimplementowaniu kodu, programista ponownie uruchamia wszystkie testy, aby upewnić się, że nowa funkcjonalność działa poprawnie, a także, że wcześniej napisane funkcje nie zostały naruszone. Proces ten powtarza się cyklicznie podczas całego procesu tworzenia oprogramowania. TDD ma na celu poprawę jakości kodu poprzez zapewnienie, że każda część oprogramowania jest pokryta testami automatycznymi, co ułatwia utrzymanie i rozwijanie kodu w przyszłości.

2.2. Opis aplikacji

Aplikacja do zarządzania renowacjami obiektów budowlanych została stworzona w języku Java. Jej celem jest umożliwienie użytkownikowi prostego zarządzania informacjami dotyczącymi renowacji różnych obiektów budowlanych. Aplikacja oferuje trzy główne funkcje:
1. Dodawanie Renowacji: Użytkownik może dodawać nowe renowacje, podając informacje takie jak nazwa, wymiary, czy renowacja jest aktywna, itp. Wprowadzone dane są następnie przetwarzane, a aplikacja oblicza objętość i pole renowacji, dodając je do bazy danych.
2. Wyszukiwanie Renowacji: Użytkownik ma możliwość wyszukiwania renowacji na podstawie ich identyfikatorów. Po znalezieniu renowacji, aplikacja prezentuje szczegółowe informacje o danej renowacji, takie jak wymiary, status aktywności, objętość, pole, data ostatniej edycji itp.
3.Wyświetlanie Identfikatorów: Aplikacja umożliwia użytkownikowi wyświetlanie dostępnych identyfikatorów renowacji, co pozwala na łatwe odnalezienie konkretnych obiektów w bazie danych.
Interfejs użytkownika został zaimplementowany w formie prostego menu w konsoli, gdzie użytkownik może wybierać odpowiednie opcje, realizując różne operacje na renowacjach. Aplikacja wykorzystuje koncepcję obiektowości, umożliwiając tworzenie, przeglądanie i zarządzanie obiektami renowacji w prosty sposób. Wzorce projektowe, takie jak Builder, zostały użyte do tworzenia obiektów w bardziej czytelny i elastyczny sposób.

2.3. Wymaganie funkcjonalne

- Użytkownik ma możliwość dodawania nowych renowacji, wprowadzając dane takie jak nazwa, wymiary, status aktywności, itp.
- Aplikacja powinna obliczać objętość i pole renowacji na podstawie wprowadzonych wymiarów.
- Użytkownik może wyszukiwać renowacje na podstawie identyfikatorów.
- Znalezione renowacje powinny być prezentowane wraz z pełnymi informacjami dotyczącymi wymiarów, statusu aktywności, objętości, pola, daty ostatniej edycji itp.
- Aplikacja umożliwia użytkownikowi wyświetlanie identyfikatorów dostępnych renowacji w celu łatwiejszego odnajdywania konkretnych obiektów w bazie danych.

2.4. Wymaganie niefunkcjonalne

- Interfejs użytkownika w konsoli powinien być responsywny i łatwy w obsłudze, zapewniając czytelność i intuicyjność.
- Operacje na bazie danych, takie jak dodawanie i pobieranie renowacji, powinny być optymalne pod względem czasu wykonania.
- Aplikacja powinna zawierać mechanizmy walidacji danych wejściowych, zwłaszcza podczas dodawania nowych renowacji, aby zapobiec błędom i zachować integralność danych.
- Obliczenia związane z objętością i polem renowacji powinny być efektywne i dokładne, zapewniając poprawność danych w bazie.
- Aplikacja powinna uwzględniać podstawowe zabezpieczenia danych, takie jak bezpieczne połączenie z bazą danych i ograniczanie dostępu do pewnych funkcji.

2.5. Zagrożenia i ograniczenia

W czasie przeprowadzania analizy problemu, zidentyfikowano następujące potencjalne zagrożenia:

- Aplikacja może być podatna na ataki typu SQL Injection lub inne próby nieautoryzowanego dostępu do bazy danych, co może prowadzić do utraty lub modyfikacji danych.
- Brak mechanizmów autoryzacji i uwierzytelniania może umożliwić nieautoryzowanym użytkownikom dostęp do funkcji, które są zarezerwowane dla administratorów.
- Aplikacja może nie obsługiwać różnych jednostek miary, co może ograniczać elastyczność dla użytkowników pracujących w różnych jednostkach.

3. Projektowanie

Projektowanie aplikacji do zarządzania renowacjami obiektów budowlanych obejmuje kilka kluczowych aspektów, takich jak struktura klas, wzorce projektowe i organizacja komponentów.

3.1. Struktura

Projektowanie: Projektowanie aplikacji do zarządzania renowacjami obiektów budowlanych obejmuje kilka kluczowych aspektów, takich jak struktura klas, wzorce projektowe i organizacja komponentów. Oto główne założenia projektowe:
1. Wzorzec Builder: Zastosowanie wzorca Builder dla klasy Renovation umożliwia konstrukcję obiektów renowacji w bardziej czytelny sposób, szczególnie gdy obiekty te posiadają wiele opcjonalnych atrybutów.
2. Oddzielenie Warstw: Aplikacja powinna składać się z logicznie oddzielonych warstw, takich jak warstwa modelu (Renovation), warstwa dostępu do danych (RenovationRepository), warstwa usług (RenovationService), oraz warstwa prezentacji (Dashboard).
3. Zastosowanie Koncepcji Jednolitego Źródła Prawdy: Klasy reprezentujące obiekty renowacji (takie jak Renovation i RenovationDTO) powinny być jedynym źródłem prawdy dla informacji o renowacjach, co pomaga uniknąć inkonsystencji danych.
4. Obsługa Wyjątków: W kodzie powinna być odpowiednio zaimplementowana obsługa wyjątków, zwłaszcza podczas komunikacji z bazą danych, aby zapewnić niezawodność i zminimalizować ryzyko utraty danych.

3.2. Struktura

Poniżej znajduje się ogólna struktura klas w aplikacji:
1. Pakiet org.example.renovation.model:
Renovation: Klasa reprezentująca obiekty renowacji. Prywatne atrybuty: id, nazwa, wymiary, status aktywności, daty utworzenia i ostatniej edycji, objętość, pole. Zastosowanie wzorca Builder do tworzenia obiektów.
2. Pakiet org.example.renovation.repository:
RenovationRepository: Klasa obsługująca komunikację z bazą danych. Metody do pobierania identyfikatorów renowacji, pobierania danych renowacji po id oraz dodawania nowych renowacji do bazy. Metody obliczające objętość i pole renowacji.
3. Pakiet org.example.renovation.service
RenovationService: Klasa dostarczająca funkcje obsługujące operacje na renowa-

cjach. Metody do prezentacji danych, dodawania nowych renowacji i sprawdzania poprawności danych wejściowych.
4. Pakiet org.example.rest.DTO:
RenovationDTO: Klasa przenosząca dane pomiędzy warstwą prezentacji a warstwą biznesową. Zastosowanie wzorca Builder do tworzenia obiektów DTO.
5. Pakiet org.example.util:
Dashboard: Klasa dostarczająca interfejs użytkownika w konsoli. ShapeCalculator: Klasa zajmująca się obliczeniami związanymi z objętością i polem renowacji. ValidateDimension: Klasa dostarczająca funkcję walidacji wartości wymiaru.
6. Pakiet org.example:
Main: Główna klasa uruchomieniowa aplikacji. Odpowiada za obsługę prostego menu w konsoli, umożliwiając interakcję użytkownika z funkcjonalnościami aplikacji.

4. Implementacja

W tym rozdziale przedstawione zostały szczegóły implementacji, skupiające się na kluczowych fragmentach kodu oraz opisach funkcjonalności. Aplikacja do tworzenia renowacji napisana w języku Java z myślą o elastyczność, wydajność i łatwość utrzymania.

4.1. Testy Test-driven development (TDD)

W tym rozdziale przedstawione zostaną testy TDD w celu implementacji projektu aplikacji do tworzenia renowacji.

4.1.1. Test when_calculatingVolumeWithValidDimensions_then_returnCorrectValue

```
1  @Test
2    void
      when_calculatingVolumeWithValidDimensions_then_returnCorrectValue
      () {
3      // Given
4      String dimensions = "10|/5|/8";
5
6      // When
7      double calculatedVolume = ShapeCalculator.shapeCalculator(
      dimensions, true);
8
9      // Then
10     assertNotNull(calculatedVolume);
11     assertEquals(400.0, calculatedVolume);
12   }
```

Listing 1. Test when_calculatingVolumeWithValidDimensions_then_returnCorrectValue

Test jednostkowy w listingu 1 sprawdza poprawność działania funkcji shapeCalculator z klasy ShapeCalculator w przypadku obliczania objętości. Oto kroki, które są realizowane w ramach tego testu:
Given: Przypisuje wartość do zmiennej dimensions, która reprezentuje wymiary obiektu renowacji w formie ciągu znaków. W tym konkretnym przypadku, wartość to ”10—/5—/8”.
When: Wywołuje funkcję shapeCalculator z przekazanymi wymiarami dimensions i flagą true oznaczającą, że obliczana jest objętość.

Then: Sprawdza, czy wynik obliczeń nie jest równy null, używając assertNotNull. Porównuje obliczoną objętość (calculatedVolume) z oczekiwaną wartością 400.0 za pomocą assertEquals.
W skrócie, test sprawdza, czy funkcja shapeCalculator poprawnie oblicza objętość na podstawie przekazanych wymiarów. Jeśli test jest zakończony sukcesem, oznacza to, że funkcja działa zgodnie z oczekiwaniami dla tego konkretnego przypadku testowego.

4.1.2. Test when_calculatingFieldWithValidDimensions_then_returnCorrectValue

```
1  @Test
2    void
       when_calculatingFieldWithValidDimensions_then_returnCorrectValue
       () {
3      // Given
4      String dimensions = "10|/5|/8";
5
6      // When
7      double calculatedField = ShapeCalculator.shapeCalculator(
       dimensions, false);
8
9      // Then
10     assertNotNull(calculatedField);
11     assertEquals(340.0, calculatedField);
12   }
```

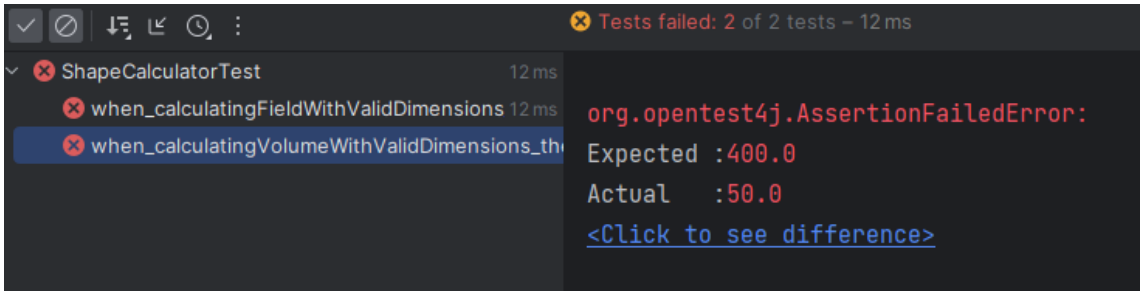
Listing 2. Test when_calculatingFieldWithValidDimensions_then_returnCorrectValue

Test jednostkowy z listingu 2 sprawdza poprawność działania funkcji shapeCalculator z klasy ShapeCalculator w przypadku obliczania pola powierzchni. Oto kroki, które są realizowane w ramach tego testu:
Given: Przypisuje wartość do zmiennej dimensions, która reprezentuje wymiary obiektu renowacji w formie ciągu znaków. W tym konkretnym przypadku, wartość to ”10—/5—/8”.
When: Wywołuje funkcję shapeCalculator z przekazanymi wymiarami dimensions i flagą false oznaczającą, że obliczane jest pole powierzchni.
Then: Sprawdza, czy wynik obliczeń nie jest równy null, używając assertNotNull. Porównuje obliczone pole powierzchni (calculatedField) z oczekiwaną wartością 340.0 za pomocą assertEquals.

W skrócie, test sprawdza, czy funkcja shapeCalculator poprawnie oblicza pole powierzchni na podstawie przekazanych wymiarów. Jeśli test jest zakończony sukcesem, oznacza to, że funkcja działa zgodnie z oczekiwaniami dla tego konkretnego przypadku testowego.

4.2. Wywołanie testów przed refaktoryzacją

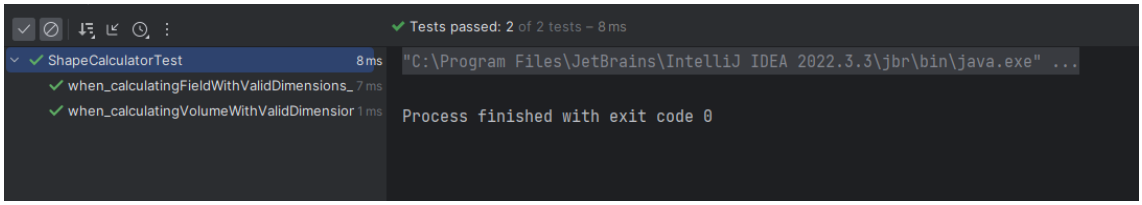
Refaktoryzacja kodu obejmowała funkcje i klasy odpowiedzialne za obliczanie pola i objętości. Przed przeprowadzeniem refaktoryzacji, podczas podawania wartości nie uwzględniano długości, co skutkowało odmiennym zapisem wyników w bazie danych. Przygotowując się do tej aktualizacji, napisano testy integracyjne, dostosowując je do nowej funkcjonalności. Wyniki tych testów przedstawiono na rysunku 4.1.



Rys. 4.1. Testy przed refaktoryzacją

4.3. Wywołanie testów po refaktoryzacji

Na rysunku 4.2 przedstawione są testy, które zostały przeprowadzone po zrefaktoryzowaniu kodu do nowych funkcji.



Rys. 4.2. Testy po refaktoryzacji

5. Wnioski

Aplikacja do zarządzania renowacjami obiektów budowlanych została skutecznie zaimplementowana, umożliwiając dodawanie, wyszukiwanie i wyświetlanie informacji o renowacjach. W trakcie procesu refaktoryzacji, funkcje i klasy odpowiedzialne za obliczanie pola i objętości zostały zoptymalizowane, co poprawiło dokładność wyników zapisywanych do bazy danych. Testy integracyjne zostały dostosowane do nowych funkcji, a proces Test-Driven Development (TDD) odegrał kluczową rolę w tworzeniu niezawodnego i dobrze udokumentowanego kodu. Dzięki zastosowaniu TDD, aplikacja jest bardziej elastyczna, łatwiejsza w zrozumieniu i zabezpieczona przed potencjalnymi błędami, co przekłada się na wysoką jakość końcowego produktu.

Spis rysunków

Spis tabel

Spis listingów