METODY TESTOWANIA OPROGRAMOWANIA

LABORATORIUM

Behavior Driven Development

wersja 1.1

przygotował:
dr inż. Radosław Adamus

.

Efekty:

Po ukończeniu laboratorium będziesz:
1. Potrafił opracowywać proste specyfikacje w notacji Gherkin
2. Potrafił automatyzować specyfikacje z wykorzystaniem narzędzia Cucumber-JVM
3. Rozumiał różnice pomiędzy testem jednostkowym i akceptacyjnym na poziomie kodu

Wymagania wstępne:

1. Posiadanie konta na platformie Github.

Narzędzia:

1. Eclipse IDE
2. Cucumber JVM Eclipse Plugin
3. Git

Reguły wykonywania ćwiczeń laboratoryjnych:

1. Zmiany należy zatwierdzać często. Zatwierdzenie zbiorcze zmian na koniec laboratorium równoważne jest z jego niezaliczeniem.
2. Zmiany zatwierdzone w repozytorium kontroli wersji muszą posiadać znaczące komentarze.
3. Git powinien być tak skonfigurowany, aby zatwierdzane zmiany były identyfikowane danymi studenta (adres email oraz nazwisko).
4. W przypadku nieukończenia zadania w trakcie zajęć rezultat pośredni powinien być, po zatwierdzeniu w repozytorium lokalnym wypchnięte do macierzystego repozytorium na GitHub.

Opis laboratorium:

0. Wprowadzenie

Przykład, na którym opierają się zadania został zaczerpnięty z książki [1]. Opis problemu dostarczony zostanie na odrębnym wydruku w trakcie laboratorium.

1. Automatyzacja scenariuszy

1. Pobierz projekt z repozytorium https://github.com/mto-lab/labBDD. Zawiera on opisany i zautomatyzowany przykładowy scenariusz. Spróbuj wykonać ten scenariusz. W celu lepszego zarządzania automatyzacją scenariuszy w środowisku Eclipse można wykorzystać plugin “Cucumber JVM Eclipse Plugin”.
2. Utwórz w repozytorium gałąź feature2 i zatwierdzaj w niej zmiany. Dla funkcji “Informacja dla podróżnych o czasie przybycia do stacji docelowej” dopisz następujący scenariusz:
Szacowanie czasu przyjazdu
Zakładając, że chcę się dostać z Parramatta do TownHall
I następny pociąg odjeżdża o 8:02 na linii Western

Gdy zapytam o godzinę przyjazdu
Wtedy powinienem uzyskać następujący szacowany czas przyjazdu: 8:29

Zautomatyzuj scenariusz w postaci wykonywalnej specyfikacji.
Połącz zmiany z gałęzią master.

3. Utwórz w repozytorium gałąź feature2_rev1 i zatwierdzaj w niej zmiany.
Zmodyfikuj scenariusz w taki sposób aby można było uwzględnić kolejne przykłady bez duplikowania scenariuszy (wykorzystaj szablon scenariusza i przykłady).
Przykłady:

- chcę się dostać z Epping do Central, nastepny pociąg odjeżdża o 8:03 na linii Northern, powinienem uzyskać następujący szacowany czas przyjazdu: 8:48.
- chcę się dostać z Epping do Central, nastepny pociąg odjeżdża o 8:07 na linii Newcastle, powinienem uzyskać następujący szacowany czas przyjazdu: 8:37.
- chcę się dostać z Epping do Central, nastepny pociąg odjeżdża o 8:07 na linii Newcastle, powinienem uzyskać następujący szacowany czas przyjazdu: 8:37.
- chcę się dostać z Epping do Central, nastepny pociąg odjeżdża o 8:13 na linii Epping, powinienem uzyskać następujący szacowany czas przyjazdu: 8:51.

Zautomatyzuj zmodyfikowany scenariusz w postaci wykonywalnej specyfikacji
Połącz zmiany z gałęzią master.

2. Implementacja scenariuszy (kryteriów akceptacji)

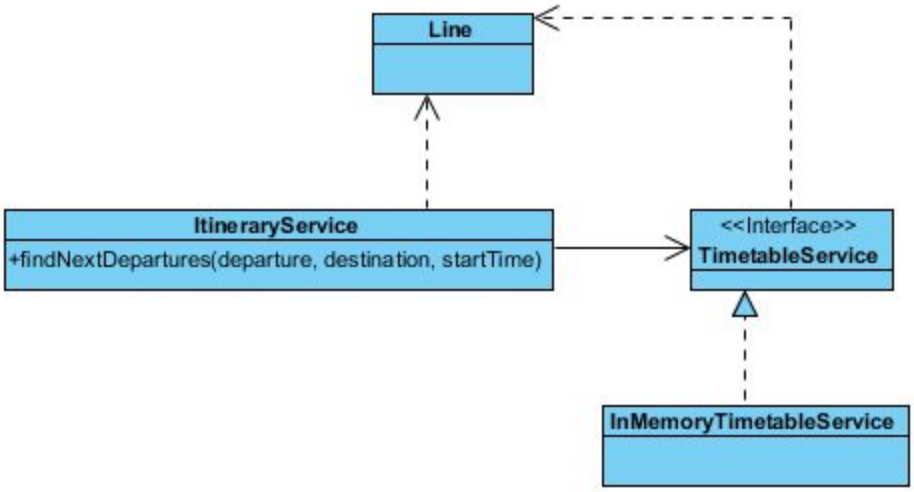
Utwórz w repozytorium gałąź feature1_impl i zatwierdzaj w niej zmiany.
Mając opisane i zautomatyzowane wybrane kryteria akceptacji można przystąpić do implementacji logiki testów.
Na podstawie scenariusza “Znajdź optymalną trasę pomiędzy stacjami na tej samej linii” spróbuj zaimplementować usługę planowania podróży implementującą wymagania, pamiętając o zatwierdzaniu etapów w repozytorium. Przyjmij następujące założenia:

1. Implementację kryteriów akceptacji rozpoczynaj od kroku @Then definiując oczekiwany rezultat. W przypadku tego scenariusza będzie to sprawdzenie czy zaproponowane (przez usługę planowania podróży) godziny są zgodne z oczekiwanymi.
2. Planowanie podróży będzie realizowane przez usługę (która ostatecznie może stać się np. Usługą Internetową dostarczającą informacji do aplikacji webowe lub mobilnej). Na obecnym etapie musimy ustalić co usługa powinna robić. Założenia te powinny zostać wyrażone w kroku @When. Na poziomie testu akceptacyjnego założymy, że usługa (nazwijmy ją ItineraryService) posiada metodę findNextDepartures. Ponieważ metoda ta nie istnieje powinniśmy ją zaimplementować. Do tego celu lepiej nadaje się test jednostkowy, koncentrujący się na komponentach w izolacji. W celu spełnienia kryteriów akceptacji wyrażonych w zautomatyzowanym scenariuszu zwykle pisze się wiele testów jednostkowych. Przyjmiemy konwencję, że testy jednostkowe będziemy umieszczać w pakiecie junit (w bazowej wersji projektu znajduje się przykładowy test dla klasy Line, kóry wykazuje błędną implementację - należy go naprawić) a nazwa klasy

reprezentującej przypadek testowy będzie wyrażała to co chcemy zaimplementować posiłkując się tym testem. W naszym przykładzie klasa może zostać nazwana: WhenCalculatingArrivalTimes.

3. Implementacja usługi InterinaryService powinna wykazać potrzebę istnienia kolejnej usługi reprezentującej rozkład jazdy. Powinna ona być zaimplementowana odrębnie, udostępniana jako interfejs i być wstrzykiwana do usługi planowania podróży (np. Poprzez parametr konstruktora). Oznacza to, że z punktu widzenia testu jednostkowego powiązanego z InterinaryService powinniśmy ją dublować (mockować). Natomiast w teście akceptacyjnym, na poziomie scenariusza wykonywalnego musimy dostarczyć realną implementację. W projekcie istnieje definicja interfejsu (TimetableService) oraz przykładowa implementacja tej usługi dla potrzeb testu akceptacyjnego (InMemoryTimetableService).

Rysunek 1 pokazuje strukturę rozwiązania.



Rysunek 1. Struktura implementacji
Na koniec połącz zmiany z gałęzią master.

3. Utrzymanie

Utwórz w repozytorium gałąź feature1_change i zatwierdzaj w niej zmiany.
Specyfikacje wykonywalne znacznie ułatwiają zespołom zajmującym się utrzymaniem wdrażanie zmian lub poprawek. Zobaczmy na prostym przykładzie, jak to działa. Załóżmy, że użytkownicy poprosili o cechę funkcjonalną wyświetlającą informacje o pociągach, które mają przybyć w ciągu najbliższych 30 minut, a nie tylko następnych 15, jak obecnie.

Oto nowy scenariusz związany z tym zmienionym wymaganiem (zwiększona liczba godzin odjazdu):

Scenariusz: Znajdź optymalną trasę pomiędzy stacjami na tej samej linii.
Zakładając pociągi linii Western z Emu Plains odjeżdżają ze stacji Parramatta do Town Hall

o 7:58, 8:00, 8:02, 8:11, 8:14, 8:21, 8:31, 8:36
Gdy chcę podróżować z Parramatta do Town Hall o 8:00
Wtedy powinienem uzyskać informację o pociągach o: 8:02, 8:11, 8:14, 8:21

Wprowadź do implementacji ten scenariusz - uruchom testy. Powinien zakończyć się on niepowodzeniem. Zaimplementuj modyfikacje.
4. Praca domowa
Zaimplementuj scenariusz cechy ‘Informacja dla podróżnych o czasie przybycia do stacji docelowej’. Zmiany zatwierdzaj w gałęzi feature2_impl.

Przypisy

[1] John Ferguson Smart, BDD w działaniu. Sterowanie zachowaniem w rozwoju aplikacji, Helion 2016.