Projektowanie przypadków testowych

[Scenariusz, przypadek] testowy

- Scenariusz testowy
  - Reprezentuje to co ma być przetestowane
    - Np. sprawdzenie działania funkcjonalności Loadbalancer'a
- Przypadek testowy
  - Możliwa do zaimplementowania minimalna jednostka kodu testująca dedykowana dla pokrycia wybranych założeń poprawności działania.
    - Np. brak obciążenia serwera, obciążenie pasuje do serwera, obciążenie przekracza pojemność serwera, itd..
- Można przyjąć, że w JUnit
  - klasa testu – scenariusz
  - metody testowe – przypadki

Implementacja przypadków testowych

- Każdy przypadek osobną metodą w klasie testu
  - Metoda publiczna bez parametrów i wartości zwracanej
  - W JUnit adnotowana @Test

```
@Test
public void balancingAServer_noVms_serverStaysEmpty() {
    Server theServer = a(server().withCapacity(1));

    balance(aListOfServersWith(theServer), anEmptyListOfVms());

    assertThat(theServer, hasLoadPercentageOf(0.0d));
}
```

- Nazwa testu odzwierciedlająca przypadek testowy, np.:

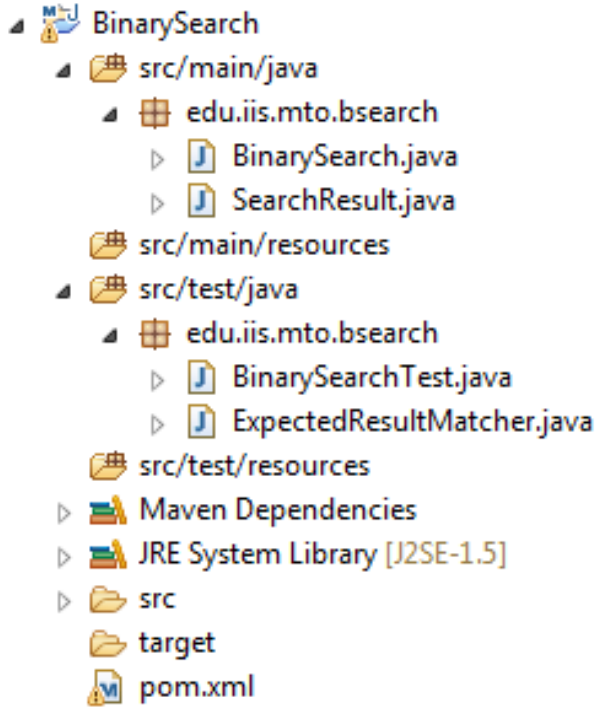
```
@Test public void
searchForExistingElem_singleElemSequence_findsItsPosition() { //… }
```

Implementacja przypadków testowych

- Każdy przypadek testowy (metoda testująca) powinna mieć przygotowane nowe środowisko (ang. test fixture)
- Kolejność metod testujących nie może mieć znaczenia
- Unikanie duplikacji kodu testującego
  - Wiele przypadków testowych dla których inicjalizujemy środowisko łatwo może doprowadzić do naruszania zasady DRY
  - JUnit - można wykorzystać metody adnotowane @Before, @BeforeClass
    - (lepsze metody jeszcze poznamy ☺)

Testy a struktura projektu

- Fizyczne wydzielanie
  - W strukturze projektu, w osobnym projekcie



Testowanie wyjątków

```
@Test(expected = IllegalArgumentException.class)
public void testForExceptions1() {
    MyUnit myUnit = new MyUnit();

    myUnit.throwIllegalArgumentException();
}
```

```
@Test
public void testForExceptions2() {
    MyUnit myUnit = new MyUnit();

    try{
        myUnit.throwIllegalArgumentException();

        fail("expected IllegalArgumentException");

    } catch(IllegalArgumentException e){
        //ignore, this exception is expected.
    }
```

http://tutorials.jenkov.com/java-unit-testing/exceptions.html

Strategie projektowania przypadków testowych

- Identyfikacja grup danych wejściowych posiadających tę samą charakterystykę, które powinny być przetwarzane w taki sam sposób
- Dobór przypadków testowych na podstawie doświadczenia

Klasy równoważności

- Dane wejściowe oraz rezultaty można często podzielić na klasy (dziedziny).
  - Liczby dodatnie, liczby ujemne, napisy bez białych znaków
- W ramach wartości należących do danej klasy program zachowuje się w porównywalny sposób.
  - Przypadki testowe powinny być określane ta, aby uwzględniały wszystkie klasy.

Pomysły na testy - Burza mózgów

- Pole przyjmuje wartości całkowite z przedziału <20,50>
- Jakie testy powinniśmy przeprowadzić?

Typowe przypadki testowe:

Test	Dlaczego?	Oczekiwany rezultat
20	Najmniejsza poprawna wartość	Akceptuj
19	Najmniejsza -1	Odrzuć, komunikat o błędzie
0	0 jest zawsze interesujące	Odrzuć, komunikat o błędzie
Blank	Puste pole, co powoduje?	Odrzuć? Ignoruj?
30	Poprawna wartość	Akceptuj
50	Największa poprawna wartość	Akceptuj
51	Największa +1	Odrzuć, komunikat o błędzie
-1	Wartość ujemna	Odrzuć, komunikat o błędzie
4294967296	2^32, integer overflow	Odrzuć, komunikat o błędzie

Dodatkowe założenia – projektowanie testów na podstawie doświadczenia

- Sekwencje danych wejściowych
  - Testuj na sekwencjach, które składają się tylko z jednej wartości
  - Użyj sekwencji rozmaitych rozmiarów w różnych testach
  - Opracuj testy, aby można było odczytać pierwszy, środkowy i ostatni element sekwencji
  - Testuj z wykorzystaniem sekwencji o zerowej długości
- Ogólne heurystyki
  - Wybierz takie dane wejściowe, które spowodują wygenerowanie wszystkich komunikatów o błędach
  - Zaprojektuj wejścia w taki sposób, aby spowodowały przepełnienie bufora.
  - Wielokrotnie powtarzaj te same dane wejściowe lub ich serie.
  - Wymuś wygenerowanie niepoprawnych danych wyjściowych.
  - Wymuś wykonanie obliczeń, których rezultat będzie zbyt mały lub zbyt duży.

Testy strukturalne

- Nazywane również testami białej skrzynki (ang. white-box testing)
- Opracowywane na podstawie wiedzy o strukturze i implementacji oprogramowania
- Celem jest zapewnienie aby każde instrukcja programu była wykonana co najmniej raz (nie wszystkie możliwe ścieżki programu).
- Możliwe do opracowania dla stosunkowo niewielkich komponentów

Testowanie ścieżek

- Odmiana testowania strukturalnego, której celem jest zbadanie każdej niezależnej ścieżki wykonania programu
- Punktem startu jest opracowanie grafu strumieni (grafu przepływu) programu
  - Węzły reprezentują decyzje (instrukcje decyzyjne)
  - Krawędzie reprezentują przepływ sterowania
- Metoda wykorzystywana przede wszystkim w ramach testów jednostkowych