METODY TESTOWANIA OPROGRAMOWANIA

LABORATORIUM 5

Izolacja testu - wątki, testy integracyjne

wersja 1.0

przygotował:
dr inż. Radosław Adamus

Efekty:

Po ukończeniu laboratorium będziesz:
1. Potrafił określić przyczynę niedeterministycznego wykonania kodu testu jednostkowego.
2. Potrafił dokonać refaktoryzacji kodu wykonywanego wielowątkowo do postaci możliwej do przetestowania jednostkowo.
3. Potrafił wykorzystać reguły JUnit do określania liczby powtórzeń wykonania testu.
4. Potrafił skonfigurować środowisko dla wykonania prostych testów integracyjnych w warstwie dostępu do danych z wykorzystaniem Spring Framework.
5. Potrafił zaprojektować testy integracyjne, których wykonanie nie zależy od kolejności wykonania poszczególnych testów.
6. Potrafił napisać prosty test parametryczny

Wymagania wstępne:

1. Posiadanie konta na platformie Github.

Narzędzia:

1. Eclipse IDE
2. Git
3. Spring Framework

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:

1. Testowanie jednostkowe kodu wykonywanego wielowątkowo

Sklonuj (operacja fork) repozytorium https://github.com/mto-lab/lab5_1 na swoje konto GitHub (a następnie na lokalny komputer). Zaimportuj projekt do IDE.

Projekt zawiera przykład kodu wykorzystującego wątki w środowisku Java. Zaimplementowany test sprawdza czy radar w chwili zaobserwowania nieprzyjacielskiej rakiety zleci wystrzelenie w jej kierunku jednej rakiety systemu Patriot. Jakim rezultatem powinien zakończyć się test? Uruchom test kilkukrotnie. Jakie są rezultaty wykonania testu? Dlaczego?

Zaimplementuj klasę BetterRadar w taki sposób aby logika była odizolowana od sposobu wykonania (wystrzelenie rakiety w tym samym wątku/wystrzelenie rakiety w odrębnym wątku). Jako podstawę wykorzystaj mechanizm executor'ów w Java (interfejs java.util.concurrent.Executor).
Zaimplementuj test (w osobnej klasie) dla nowej implementacji w taki sposób aby kod wykonał się w wątku w którym uruchomiony jest JUnitRunner.
Dodaj funkcjonalność umożliwiającą wielokrotne wykonywanie wybranych testów.
Wskazówka:
http://www.codeaffine.com/2013/04/10/running-junit-tests-repeatedly-without-loops/

2. Testy integracyjne w warstwie dostępu do danych

Opis ćwiczenia znajduje się w pliku Lab5_2.pdf

3. Testy parametryczne

Repozytorium, które poznaliśmy w trakcie zajęć nt. TDD posiada gałąź o nazwie "parametrizedtests" (https://github.com/mto-lab/loadBalancerKataMto/tree/parametrizedtests).
Rozbuduj klasę ServerLoadBalancerParametrizedTest w taki sposób aby istniejący test był wykonywany wielokrotnie dla różnych parametrów pojemności serwera oraz rozmiaru wirtualnej maszyny.
Wskazówki dot. testów parametrycznych w JUnit:
https://github.com/junit-team/junit/wiki/Parameterized-tests
http://examples.javacodegeeks.com/core-java/junit/junit-parameterized-test-example/

Zadanie dodatkowe (na dużego plusa do zaliczenia). Opracuj parametryczny test dla ostatniego, siódmego testu w ćwiczeniu Kata (balance_serversAndVms - wiele serwerów i wiele wirtualnych maszyn).

Przypisy

1. http://jokerconf.com/presentations/frankel.pdf