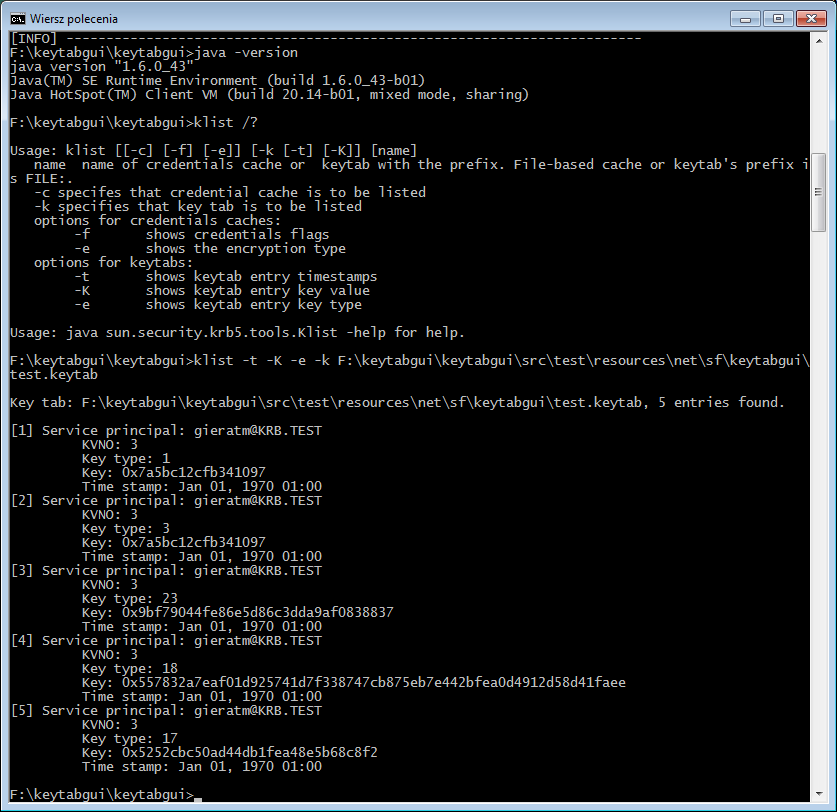
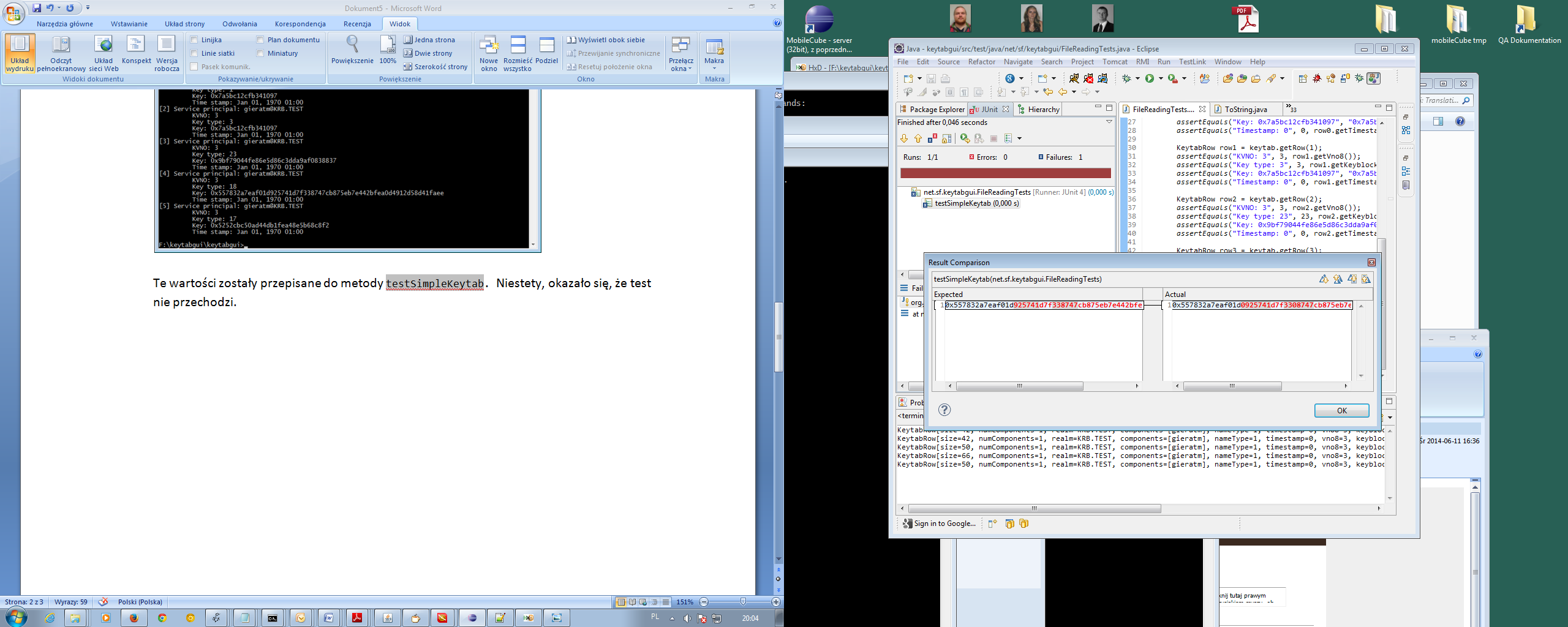
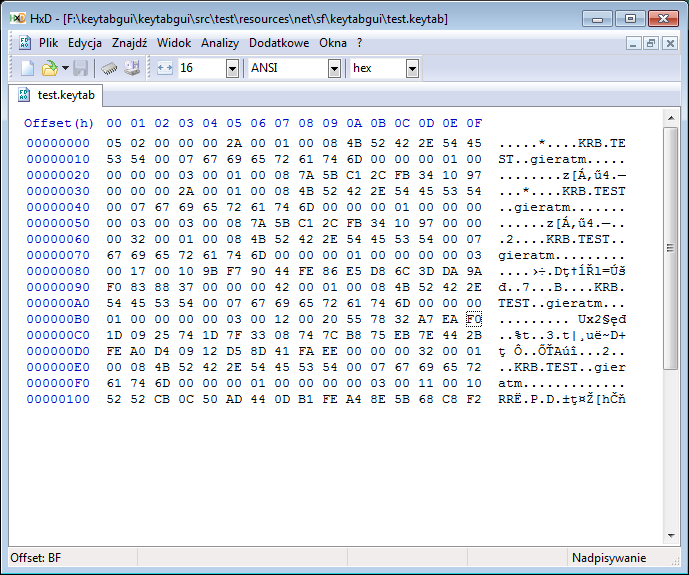
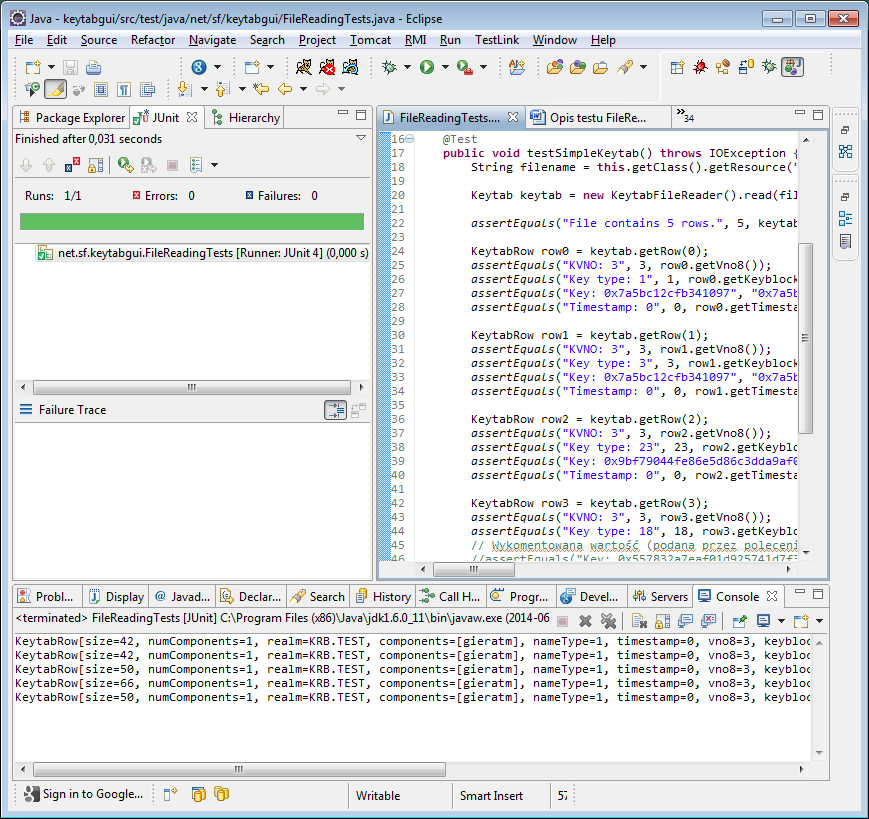
# Testy jednostkowe FileReadingTest.java

Test jednostkowy JUnit4

Przebieg pisania testu:

1. Plik test.keytab został utworzony na kontrolerze domeny Windows Server 2008 w standardowy sposób przy pomocy polecenia ktpass.   
   
2. Jego zawartość została wypisana na konsoli przy użyciu polecenia klist (znadującego się w katalogu %JDK\_HOME%/bin)   
     
     
     
   Te wartości zostały przepisane do metody testSimpleKeytab. Niestety, okazało się, że test nie przechodzi:  
     
   Wartość Expected została przepisana z wyniku polecenia klist,  
   wartość Actual została odczytana przez testowaną klasę KeytabFileReader.
3. Aby znaleźć przyczynę różnicy, plik został wyświetlony w edytorze hexadecymalnym.  
     
   Po odszukaniu klucza (na rysunku wyróżniony jest w ramce) okazało się, że w tekście wyróżnionym przez JUnita, przed 9 powinno być 0. Wartość Expected jest błędna (błąd jest w teście, tzn. zła jest wartość podana przez klist).   
     
   Wartość klucza widać również na pierwszym screenshot’cie, przedstawiającym wynik wykonania polecenia ktpass. Widzimy że w czwartym kluczu także przed 9 jest 0.
4. Po przepisaniu do testu wartości podanych przez polecenie Microsoft ktpass po utworzeniu pliku test.keytab, test przechodzi:  
     
   

## Test Driven Development

Zgodnie z zasadą TDD, najpierw piszemy test, następnie implementację.

Przechodzimy przez trzy fazy:

1. fazę czerwoną (mamy test, metody bez implementacji, test nie przechodzi),
2. następnie piszemy implementację (faza zielona),
3. a na końcu wykonujemy re factoring. Powinniśmy utrzymać tę samą funkcjonalność (test powinien przechodzić) przy lepszej organizacji kodu.

Testy podnoszą pewność. Po wprowadzeniu zmian w innym fragmencie kodu (np. dodaniu następnej funkcjonalności) można uruchomić automatyczne testy (np. w nocy lub po każdym commit’cie do repozytorium) by zobaczyć, czy wprowadzone zmiany nie wprowadziły błędów. Osoba, która sprawiła, że testy nie przechodzą, jest odpowiedzialna za naprawienie szkód. Takie podejście nazywamy **Continuous Integration**. Do budowania oraz automatycznego wykonywania testów stosuje się m.in. Jenkinsa.

## Design by Contract

Przy okazji pisania testu trzeba się najpierw zastanowić co chce się uzyskać. Zwykle prowadzi to do prostszego i lepszego API.