Raport

Cel:

Porównanie kosztów obsługi wyjątków ze względu na rodzaj obsługiwanego wyjątku, a także ze względu na metodykę jego obsługi (wewnątrz funkcji, w której został wygenerowany, bądź pochwycenie w funkcji zewnętrznej)

Metodyka:

Utworzenie par prostych funkcji, gdzie jedna jest zakodowana w sposób błędny – generujący wyjątki, a druga wywołuję ową funkcję podaną ilość razy, mierząc czasy egzekucji i uśredniając je, a następnie zwraca uśredniony wynik w 10-krotnej pętli w funkcji main, gdzie owe wyniki są zapisywane na ekranie. Zaproponowałem wygenerowanie następujących wyjątków – ArrayIndexOutOfBoundsException, ArithmeticException (Unchecked Exceptions), oraz IOException i NotSerializableException (Checked Exceptions). Każdy został obsłużony w dwa sposoby, tzn poprzez lokalne przechwycenie wyjątku try-catch, a także przekazanie go do metody wyższej, odpowiedzialnej za iterowanie funkcji generującej wyjątek i łapiącej go przy każdej iteracji. Koszta czasu obsługi wyjątków zostały porównane używając różnicy wartości System.nanoTime() po i przed wywołaniem metody generującej wyjątek. Dla uzyskania dokładności pomiarów, każdą z 4 opcji iteracji (1, 100, 1000 i 10000-krotnie) powtórzyłem 10 razy.

1. IOException

Kod dwóch metod, z czego jedna generuje i wyrzuca wyjątek IOException związany z brakiem możliwości wczytania pliku, druga metoda natomiast wywołuje tą pierwszą i łapie przekazany w niej wyjątek. Cały proces odbywa się n razy, a następnie zwracany jest uśredniony wynik. czasu obsługi omówionego procesu

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Poniżej natomiast są przedstawione wyniki pomiarów

Obraz zawierający stół

Opis wygenerowany automatycznie

Obraz zawierający stół

Opis wygenerowany automatycznie

Następnym przetestowanym w ten sam sposób wyjątkiem jest NotSerializableException, wygenerowany w metodzie próbującej zzerializować obiekt klasy, nie implementującej interfejsu Serializable. Pojawia się tutaj także IOException, jednak ten wyjątek został zadeklarowany tylko z powodu kompilatora, który go wymaga. IOException w tym przykładzie nie jest ani generowany ani obsługiwany, a więc nie ma wpływu na ogólny wynik.

Obraz zawierający tekst

Opis wygenerowany automatycznieObraz zawierający tekst

Opis wygenerowany automatycznie

Wyniki:

Obraz zawierający stół

Opis wygenerowany automatycznieObraz zawierający stół

Opis wygenerowany automatycznie

Kolejnym testowanym wyjątkiem jest ArithmeticException wywołane poprzez prosty podział dwóch zmiennych typu int, z czego dzielna ma przypisaną wartość 0.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Wyniki:

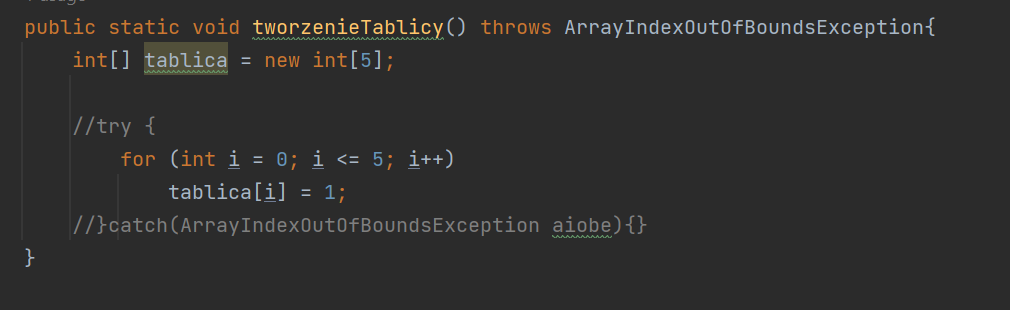
Obraz zawierający stół

Opis wygenerowany automatycznie

Obraz zawierający stół

Opis wygenerowany automatycznie

Ostatnią omawianą przeze mnie metodą jest taka, która generuje wyjątek ArrayIndexOutOfBondsException, dzięki prostej iteracji dodającej 6 elementów do 5 elementowej tablicy.



Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający stół

Opis wygenerowany automatycznie

Obraz zawierający stół

Opis wygenerowany automatycznie

Wnioski:

Porównując poszczególne wykresy, możemy łatwo zauważyć, iż wraz z ilością iteracji drastycznie spada czas egzekucji, wynika to z buforu, pierwsza iteracja jest zazwyczaj wielokrotnie bardziej kosztowna niż kolejne. Jeżeli zaś chodzi o różnice kosztów łapania wyjątków wewnątrz metody, a używania instrukcji throws do wyrzucania jej w wyższych instancjach – różnice owe są znikome i techniki te powinno się stosować w zależności od kontekstu, nie zaś martwiąc się o różnice wydajnościowe.