Porównanie implementacji wielowątkowej w symulacji bankomatu

28 listopada 2024

1 Wstęp

Wielowątkowość jest kluczową techniką w programowaniu równoległym, umożliwiającą efektywne wykorzystanie zasobów systemowych i poprawę wydajności aplikacji. Klasyczne podejście do tworzenia wątków w Javie opiera się na interfejsie `Runnable`. Jednakże rozwój języka Java przyniósł bardziej zaawansowane narzędzia, takie jak `Executor`, `ExecutorService` oraz `ThreadPoolExecutor`, które upraszczają zarządzanie wątkami i umożliwiają ich bardziej efektywne wykorzystanie.

Celem niniejszego sprawozdania jest porównanie klasycznych technik wielowątkowości z metodami wysokopoziomowymi na przykładzie aplikacji symulującej działanie bankomatu. Analiza uwzględnia pomiar czasu wykonania, zużycia procesora oraz zmienności stanów wątków.

2 Opis programu

Zadaniem programu jest symulacja operacji bankowych na współdzielonym koncie przy użyciu 500 klientów, gdzie każdy z nich dokonuje modyfikacji salda konta. Program zaimplementowano w dwóch wersjach:

- **Wersja klasyczna:** Wykorzystuje interfejs `Runnable`. Każdy klient jest uruchamiany jako osobny wątek.
- **Wersja wysokopoziomowa:** Korzysta z puli wątków implementowanej za pomocą `ThreadPoolExecutor`. Klienci są zgłaszani jako zadania do wykonania przez pulę wątków.

3 Implementacja

3.1 Wersja klasyczna

Kod kluczowy dla wersji klasycznej został zaimplementowany w pliku `ATMSimulatorThread.java`. Każdy wątek został utworzony i uruchomiony osobno. Operacje modyfikacji salda zostały zsynchronizowane, aby zapobiec problemom współbieżności.

```
for (int i = 0; i < 500; i++) {
    new Client(account, new Random().nextInt(0, 1000)).run();
}
```

3.2 Wersja wysokopoziomowa

Kod kluczowy dla wersji wysokopoziomowej został zaimplementowany w pliku `ATMSimulatorThreadPool`. Zadania zostały dodane do puli wątków za pomocą `invokeAll`, co umożliwia równoczesne wykonanie operacji przez ograniczoną liczbę wątków.

```
ThreadPoolExecutor executor = (ThreadPoolExecutor) Executors.newFixedThreadPool
    (8);
List<Callable<Void>> tasks = new ArrayList<>();
for (int i = 0; i < 500; i++) {
    tasks.add(() -> {
        new Client(account, new Random().nextInt(0, 1000)).run();
        return null;
    });
}
executor.invokeAll(tasks);
executor.shutdown();
```

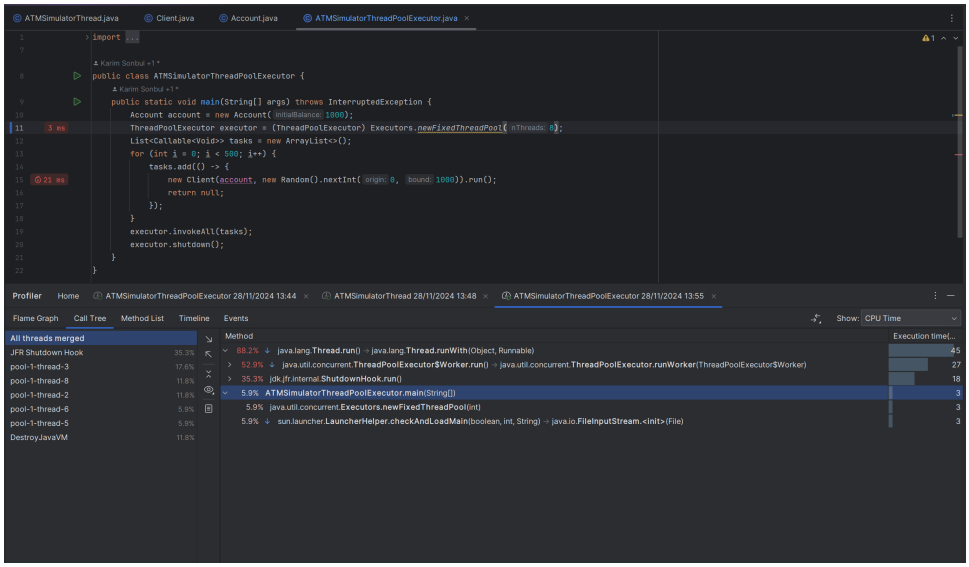
4 Testy i analiza

Do przeprowadzenia analizy wydajności obu wersji aplikacji wykorzystano profiler wbudowany w środowisko IntelliJ IDEA. Skupiono się na trzech głównych metrykach:

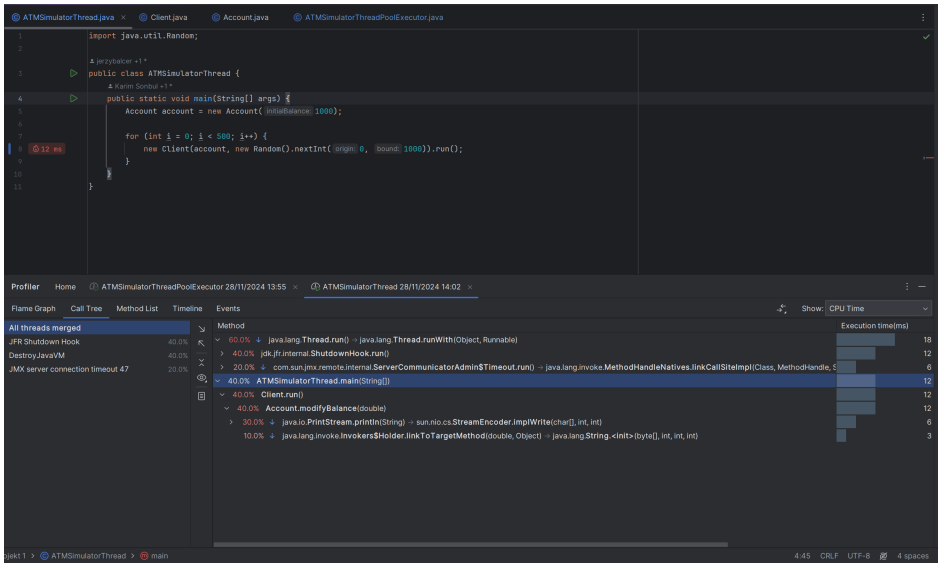
- **Czas wykonania (CPU time):** Porównano całkowity czas procesora dla obu implementacji.
- **Zużycie pamięci:** Analizowano ilość pamięci wyczerpanej podczas działania programu.
- **Działanie wątków w czasie:** Przeanalizowano aktywność wątków dla obu implementacji, przedstawioną na wykresie.

4.1 Czas wykonania (CPU time)

Z wyników testów wynika, że implementacja oparta na pulach wątków (`ThreadPoolExecutor`) jest znacznie bardziej wydajna. Czas procesora wynosił jedynie **3 ms**, podczas gdy dla klasycznej implementacji (`Runnable`) wynosiła aż **12 ms**.



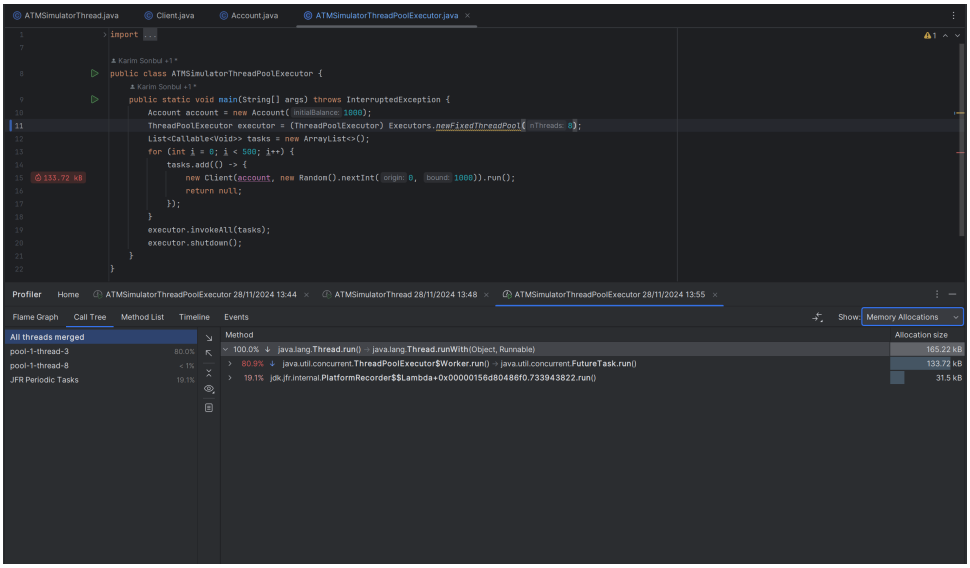
Rysunek 1: Czas wykonania (CPU time) dla implementacji z pulą wątków



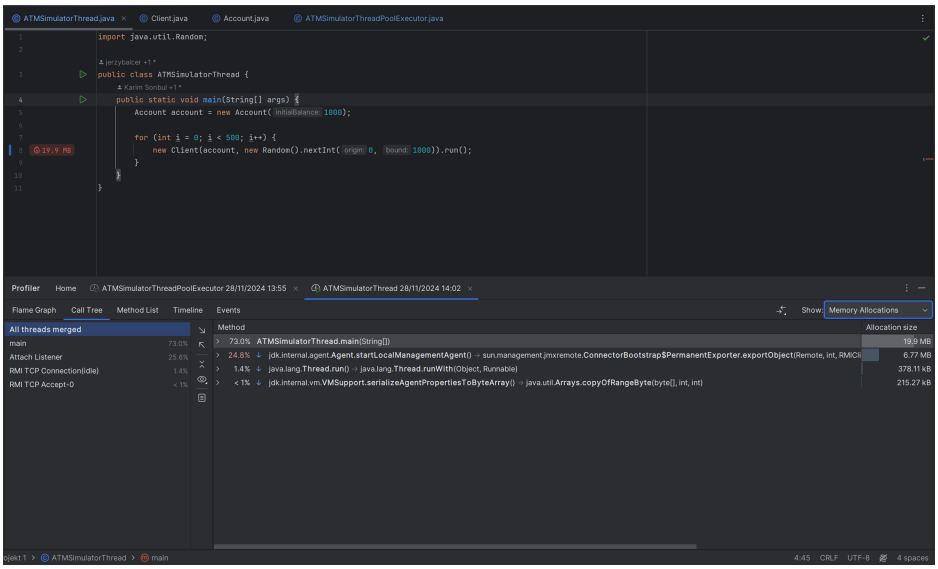
Rysunek 2: Czas wykonania (CPU time) dla implementacji klasycznej

4.2 Zużycie pamięci

Wyniki testów pokazały znaczącą różnicę w wykorzystaniu pamięci. Implementacja oparta na pulach wątków zużyła zaledwie **165,22 KB**, podczas gdy klasyczna implementacja wymagała aż **19,9 MB** pamięci.



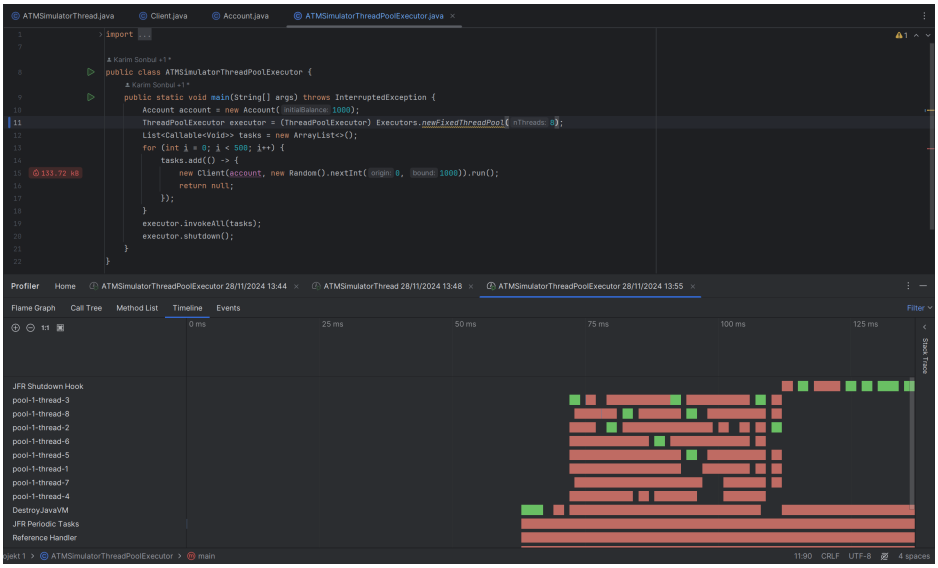
Rysunek 3: Zużycie pamięci dla implementacji z pulą wątków



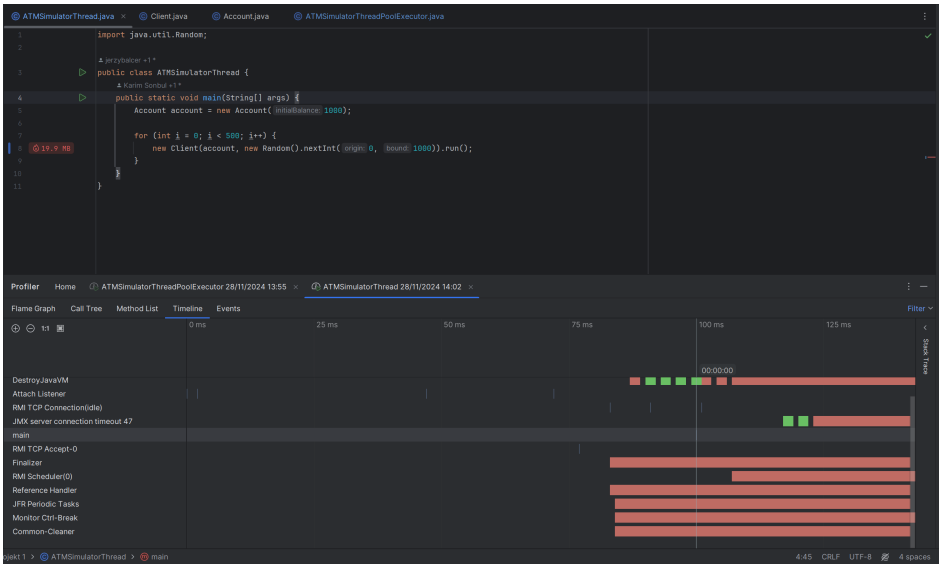
Rysunek 4: Zużycie pamięci dla implementacji klasycznej

4.3 Działanie wątków w czasie

Na poniższych wykresach przedstawiono aktywność wątków w czasie. W przypadku implementacji z pulą wątków widać wyraźnie efektywne wykorzystanie **8 wątków** w pulach, co przekłada się na równomierne obciążenie i lepsze zarządzanie zasobami. Natomiast w implementacji klasycznej aktywność wątków nie jest zbyt widoczna, co wskazuje na gorsze wykorzystanie możliwości współbieżności.



Rysunek 5: Działanie wątków w czasie dla implementacji z pulą wątków



Rysunek 6: Działanie wątków w czasie dla implementacji klasycznej

5 Wnioski

Analiza wykazała, że:

- Podejście klasyczne, choć prostsze, generuje znaczące obciążenie systemu w przypadku dużej liczby wątków.
- Wykorzystanie puli wątków redukuje to obciążenie, oferując bardziej skalowalne rozwiązanie.
- Synchronizacja metod w klasie `Account` była kluczowa w obu implementacjach, aby zapewnić integralność danych.

Zastosowanie `ThreadPoolExecutor` okazało się bardziej efektywne dla dużej liczby operacji, co potwierdza zalety podejścia wysokopoziomowego w zarządzaniu wieloma wątkami.