Uniwersytet Rzeszowski

Instytut Informatyki

Apliklacja sprawdzająca poprawność rozwiązania sudoku

Systemy Operacyjne 2

Prowadzący: mgr inż. Aleksander Wojtowicz

Autor: **Michał Pilecki**

Numer indeksu: **125151**

Data: **06.01.2025**

Rok akademicki: 2025/2026

Spis treści

1 Wprowadzenie

Celem projektu jest stworzenie aplikacji walidującej rozwiązanie Sudoku przy użyciu wielowątkowości w języku Java. Aplikacja umożliwia sprawdzenie, czy podane rozwiązanie Sudoku jest poprawne, zgodnie z regułami tej gry logicznej. Projekt pozwala na praktyczne zastosowanie koncepcji programowania obiektowego oraz technik współbieżności, zwiększając jednocześnie wydajność obliczeń dzięki równoległemu przetwarzaniu danych.

2 Funkcjonalności

Aplikacja realizuje następujące funkcje:

- Walidacja poprawności wierszy Sudoku,
- Walidacja poprawności kolumn Sudoku,
- Walidacja poprawności podsiatek 3x3,
- Wykorzystanie wielowątkowości do równoległego sprawdzania wierszy, kolumn i podsiatek,
- Wyświetlenie wyniku walidacji w konsoli.

3 Implementacja w Javie

3.1 Struktura kodu

Aplikacja została zaimplementowana w języku Java, korzystając z mechanizmu wielowątkowości w celu zwiększenia wydajności walidacji. Każdy wiersz, kolumna oraz podsiatka Sudoku są weryfikowane w osobnych wątkach. Dzięki temu obliczenia przebiegają równolegle, co znacznie przyspiesza cały proces.

3.2 Główne komponenty

Kod składa się z następujących elementów:

- **Klasa `SudokuValidator`**
  Klasa zawiera logikę walidacji Sudoku. Odpowiada za zarządzanie wątkami oraz synchronizację wyników walidacji.
- **Metody walidacyjne**:
  Metody sprawdzające poprawność wierszy, kolumn i podsiatek Sudoku:
  - `sprawdzWiersz(int wiersz)`: sprawdza, czy wszystkie liczby w danym wierszu są unikalne i mieszczą się w zakresie od 1 do 9.
  - `sprawdzKolumne(int kolumna)`: weryfikuje poprawność kolumny na podobnych zasadach co wiersze.
  - `sprawdzPodsiatke(int poczatekWiersz, int poczatekKolumna)`: sprawdza, czy podsiatka 3x3 zawiera wszystkie liczby od 1 do 9.
- **Siatka Sudoku**:
  Dwuwymiarowa tablica `int[][]` przechowująca rozwiązanie Sudoku do walidacji.

3.3 Kod źródłowy

Kod źródłowy klasy `SudokuValidator` przedstawia się następująco:

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.atomic.AtomicBoolean;

public class SudokuValidator {

    private static final int ROZMIAR = 9;
```

```
private static final int ROZMIAR_PODSIATKI = 3;
private static final int[][] siatkaSudoku = {
        {6, 2, 4, 5, 3, 9, 1, 8, 7},
        {5, 1, 9, 7, 2, 8, 6, 3, 4},
        {8, 3, 7, 6, 1, 4, 2, 9, 5},
        {1, 4, 3, 8, 6, 5, 7, 2, 9},
        {9, 5, 8, 2, 4, 7, 3, 6, 1},
        {7, 6, 2, 3, 9, 1, 4, 5, 8},
        {3, 7, 1, 9, 5, 6, 8, 4, 2},
        {4, 9, 6, 1, 8, 2, 5, 7, 3},
        {2, 8, 5, 4, 7, 3, 9, 1, 6}
};

private static final AtomicBoolean czyPoprawne = new AtomicBoolean(true);

public static void main(String[] args) {
    ExecutorService wykonawca = Executors.newFixedThreadPool(ROZMIAR + 2);

    // Walidacja wierszy
    for (int i = 0; i < ROZMIAR; i++) {
        int wiersz = i;
        wykonawca.submit(() -> sprawdzWiersz(wiersz));
    }

    // Walidacja kolumn
    for (int i = 0; i < ROZMIAR; i++) {
        int kolumna = i;
        wykonawca.submit(() -> sprawdzKolumne(kolumna));
    }

    // Walidacja podsiatek
    for (int wiersz = 0; wiersz < ROZMIAR; wiersz += ROZMIAR_PODSIATKI) {
        for (int kolumna = 0; kolumna < ROZMIAR; kolumna += ROZMIAR_PODSIATKI) {
            int poczatekWiersz = wiersz;
            int poczatekKolumna = kolumna;
            wykonawca.submit(() -> sprawdzPodsiatke(poczatekWiersz, poczatekKolumna));
        }
    }

    wykonawca.shutdown();
    while (!wykonawca.isTerminated()) {
        // Czekaj na zakończenie wszystkich wątków
    }

    if (czyPoprawne.get()) {
        System.out.println("Rozwiązanie Sudoku jest poprawne.");
    } else {
```

```
            System.out.println("Rozwiązanie Sudoku jest niepoprawne.");
        }
    }

    private static void sprawdzWiersz(int wiersz) {
        boolean[] widziane = new boolean[ROZMIAR];
        for (int kolumna = 0; kolumna < ROZMIAR; kolumna++) {
            int liczba = siatkaSudoku[wiersz][kolumna];
            if (liczba < 1 || liczba > ROZMIAR || widziane[liczba - 1]) {
                czyPoprawne.set(false);
                return;
            }
            widziane[liczba - 1] = true;
        }
    }

    private static void sprawdzKolumne(int kolumna) {
        boolean[] widziane = new boolean[ROZMIAR];
        for (int wiersz = 0; wiersz < ROZMIAR; wiersz++) {
            int liczba = siatkaSudoku[wiersz][kolumna];
            if (liczba < 1 || liczba > ROZMIAR || widziane[liczba - 1]) {
                czyPoprawne.set(false);
                return;
            }
            widziane[liczba - 1] = true;
        }
    }

    private static void sprawdzPodsiatke(int poczatekWiersz, int poczatekKolumna) {
        boolean[] widziane = new boolean[ROZMIAR];
        for (int wiersz = 0; wiersz < ROZMIAR_PODSIATKI; wiersz++) {
            for (int kolumna = 0; kolumna < ROZMIAR_PODSIATKI; kolumna++) {
                int liczba = siatkaSudoku[poczatekWiersz + wiersz][poczatekKolumna + kolumna];
                if (liczba < 1 || liczba > ROZMIAR || widziane[liczba - 1]) {
                    czyPoprawne.set(false);
                    return;
                }
                widziane[liczba - 1] = true;
            }
        }
    }
}
```

3.4 Opis działania

Aplikacja działa w następujący sposób:

1. Utworzony zostaje zestaw wątków za pomocą `ExecutorService`.
2. Dla każdego wiersza, kolumny i podsiatki uruchamiany jest osobny wątek walidacyjny.
3. Każdy wątek wykonuje odpowiednią metodę walidacyjną.
4. Wyniki walidacji są zapisywane w zmiennej `AtomicBoolean`, która przechowuje status poprawności całej planszy.
5. Po zakończeniu działania wszystkich wątków aplikacja wyświetla wynik walidacji w konsoli.

4 Instrukcja obsługi

4.1 Kroki do uruchomienia aplikacji

1. Upewnij się, że masz zainstalowane środowisko Java Development Kit (JDK) w wersji 8 lub nowszej.
2. Skopiuj plik `SudokuValidator.java` do wybranego katalogu na swoim komputerze.
3. Otwórz terminal lub wiersz poleceń i przejdź do katalogu, w którym znajduje się plik `SudokuValidator.java`.
4. Skompiluj plik za pomocą polecenia:

```
javac SudokuValidator.java
```

5. Uruchom aplikację za pomocą polecenia:

```
java SudokuValidator
```

6. Odczytaj wynik walidacji wyświetlony w konsoli.

4.2 Wymagania systemowe

- System operacyjny: Windows, Linux lub macOS,
- Zainstalowane JDK w wersji 8 lub nowszej,
- Dostęp do terminala lub wiersza poleceń.

5 Testowanie

5.1 Przygotowanie danych testowych

Do przetestowania aplikacji przygotowano zestawy plansz Sudoku:

- **Poprawna plansza**: Wszystkie wiersze, kolumny i podsiatki zawierają liczby od 1 do 9 bez powtórzeń.
- **Niepoprawna plansza**: Plansze z błędami, np. powtórzenie liczby w jednym wierszu, kolumnie lub podsiatce.
- **Pusta plansza**: Plansza wypełniona wartościami spoza zakresu 1–9.

5.2 Scenariusze testowe

- Walidacja poprawnej planszy (oczekiwany wynik: `Rozwiązanie Sudoku jest poprawne`).
- Walidacja planszy z błędem w jednym wierszu (oczekiwany wynik: `Rozwiązanie Sudoku jest niepoprawne`).
- Walidacja planszy z błędem w jednej kolumnie.
- Walidacja planszy z błędem w jednej podsiatce.
- Walidacja pustej planszy.

6 Podsumowanie

W ramach niniejszego projektu zrealizowano zaawansowaną aplikację do walidacji rozwiązań Sudoku w języku Java. Implementacja ta została zaprojektowana z wykorzystaniem technik wielowątkowości, co umożliwiło równoległe przetwarzanie i weryfikację wierszy, kolumn oraz podsiatek planszy Sudoku. Zastosowane podejście znacznie zwiększyło wydajność procesu walidacji poprzez optymalizację wykorzystania zasobów systemowych.

Aplikacja składa się z kilku kluczowych komponentów, w tym klasy SudokuValidator, która integruje logikę walidacji oraz zarządzanie wątkami. Wykorzystanie klasy ExecutorService zapewniło efektywne tworzenie i zarządzanie pulą wątków, co umożliwiło jednoczesne sprawdzanie poszczególnych segmentów planszy. Mechanizmy synchronizacji, takie jak AtomicBoolean, zagwarantowały spójność wyników walidacji, eliminując ryzyko błędów konkurencji.

Testy funkcjonalne przeprowadzone na różnych zestawach danych wykazały wysoką skuteczność aplikacji w identyfikowaniu zarówno poprawnych, jak i niepoprawnych rozwiązań Sudoku. Implementacja udowodniła swoją niezawodność, zachowując stabilność nawet przy intensywnym obciążeniu. Eksperymenty z różnymi układami Sudoku potwierdziły, że aplikacja prawidłowo identyfikuje wszelkie naruszenia reguł gry.

Projekt ten ilustruje praktyczne zastosowanie zaawansowanych koncepcji programowania obiektowego i współbieżności. Pozwolił na głębsze zrozumienie i wykorzystanie wzorców projektowych, takich jak model producent-konsument, oraz technik zarządzania wątkami. Uzyskane rezultaty podkreślają znaczenie współbieżności w nowoczesnym tworzeniu wydajnych aplikacji, które są w stanie sprostać wymaganiom współczesnych systemów komputerowych.

7 Bibliografia

- Dokumentacja języka Java: https://docs.oracle.com/javase/
- Materiały z wykładów z przedmiotu Programowanie Obiektowe.
- Strona poświęcona Sudoku: https://sudoku.com/
- Michał, "Programowanie obiektowe: przewodnik", Aviary, 2023: https://aviary.pl/programowanie-obiektowe/
- Mateusz Sowa, "4 filary programowania obiektowego", devmentor.pl, 2023: https://devmentor.pl/b/4-filary-programowania-obiektowego
- "Programowanie obiektowe: poznaj zastosowania, popularne języki", 3KM.pl: https://3km.pl/it-i-komputery/programowanie-obiektowe-kompleksowy-przewodnik-dla-poczatku
- "Java - Bardziej zaawansowane", JavaStart: https://javastart.pl/baza-wiedzy/java-zaawansowane
- "Baza wiedzy - JavaStart", JavaStart: https://javastart.pl/baza-wiedzy
- "Darmowy kurs Java - JavaStart", JavaStart: https://javastart.pl/baza-wiedzy/darmowy-kurs-java
- Guru99, "Wielowątkowość w Java", Guru99: https://www.guru99.com/pl/multithreading-java.html
- "Java 53: wątki – przetwarzanie wielowątkowe", Developer on the Go: https://developeronthego.pl/java-watki-przetwarzanie-wielowatkowe/
- "Współbieżność w Javie – synchronizacja i wielowątkowość", J-Labs: https://www.j-labs.pl/blog-technologiczny/wspolbieznosc-w-javie-synchronizacja-i-wielowatkowosc
- "Podstawy programowania wielowątkowego", Progromcy: https://bing.com/search?q=programowanie+wielow%c4%85tkowe+w+Javie