Lab 01

Współbieżność w Javie

Adrian Madej 02.10.2023

1. Treści zadań

1. Napisać program, który uruchamia 2 wątki, z których jeden zwiększa wartość zmiennej całkowitej o 1, drugi wątek zmniejsza wartość o 1. Zakładając że na początku wartość zmiennej Counter była 0, chcielibyśmy wiedzieć jaka będzie wartość tej zmiennej po wykonaniu 10000 operacji zwiększania i zmniejszania przez obydwa wątki.

2. Na podstawie 100 wykonań programu z p.1, stworzyć histogram końcowych wartości zmiennej Counter.

3. Spróbować wprowadzić mechanizm do programu z p.1, który zagwarantowałby przewidywalną końcową wartość zmiennej Counter. Nie używać żadnych systemowych mechanizmów, tylko swój autorski.

2. Rozwiązanie zadań

1. Zaimplementowanie programu

W wykonywanym programie dwa wątki korzystają jednocześnie z dzielonego zasobu, co może doprowadzić do niespodziewanych wyników z powodu braku synchronizacji.

Klasa Counter pochodzi ze szkieletu programu i pozostaje w stanie niezmienionym.

```
 public class Counter {
    private int _val;
    public Counter(int n) {
        _val = n;
    }
    public void inc() {
        _val++;
    }
    public void dec() {
        _val--;
    }
    public int value() {
        return _val;
    }
}
```

Implementujemy klasę zmniejszającą wartość zmiennej.

```
 public class DThread extends Thread{
    private Counter cnt;

    DThread(Counter cnt){
        this.cnt = cnt;
    }

    public void run(){
        for (int i = 0; i < 10000; i++) {
            this.cnt.dec();
        }
    }
}
```

Implementujemy klasę zwiększającą wartość zmiennej.

```
public class IThread extends Thread{

    private Counter cnt;

    IThread(Counter cnt){
        this.cnt = cnt;
    }

    public void run(){
        for(int i = 0; i < 10000; i++){
            this.cnt.inc();
        }
    }
}
```

Tworzymy nowe wątki.

```
public class Race {
    public static void main(String[] args) {
        Counter cnt = new Counter(0);

        IThread increaseThread = new IThread(cnt);
        DThread decreaseThread = new DThread(cnt);

        increaseThread.start();
        decreaseThread.start();

        try {
            increaseThread.join();
            decreaseThread.join();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        System.out.println("stan=" + cnt.value());
    }
}
```

2. Tworzenie histogramu

Do utworzenia histogramu posłuży nam hashmapa. Klucze będą reprezentowały końcowe wyniki zmiennej Counter, natomiast wartości posłużą nam do określenia liczby ich wystąpień.

Tworzymy nową klasę Histogram.

```
import java.util.HashMap;

public class Histogram {

    private final HashMap<Integer, Integer> histogram = new
HashMap<>();

    public void addToHistogram(Counter cnt) {
        int variable = cnt.value();
        if (histogram.containsKey(variable)) {
            histogram.put(variable, histogram.get(variable) +
1);
        } else {
            histogram.put(variable, 1);
        }
    }

    public void display() {
        histogram.forEach((key, value) ->
                System.out.println("wartość=" + key + ",
wystąpienia: " + value)
        );
    }
}
```

3. Synchronizacja wątków

Jeśli chcemy mieć gwarancję co do wartości końcowej zmiennej Counter, musimy dokonać synchronizacji wątków. W tym celu dodamy do klasy Counter flagę oznaczoną jako volatile co poinformuje kompilator, że zmienna może zmieniać się asynchronicznie. Pętle while w metodach inc() oraz dec() będą „czekały” aż poprzedni wątek przestanie pracować na zmiennej. Wartość flagi false oznacza, że zmienna będzie zwiększana, natomiast wartość true, że będzie zmniejszana.

Ostatecznie klasa Counter wygląda następująco.

```
public class Counter {
    private int _val;
    private volatile boolean flag = false;
    public Counter(int n) {
        _val = n;
    }
    public void inc() {
        while(flag) {}
        _val++;
        flag = true;
    }
    public void dec() {
        while (!flag){}
        _val--;
        flag = false;
    }
    public int value() {
        return _val;
    }
}
```

Powyższe rozwiązanie gwarantuje nam przewidzenie końcowego wyniku, gdyż tylko jeden wątek w danym czasie ma możliwość do zmiany wartości zmiennej. Zastosowanie flagi jest całkowicie niezależne od mechanizmów systemowych oraz proste w implementacji. Takie rozwiązanie nie jest jednak optymalne gdyż program czeka w nieskończonej pętli while na zmianę flagi, co prowadzi do wzmożonej pracy procesora.

3. Wyniki

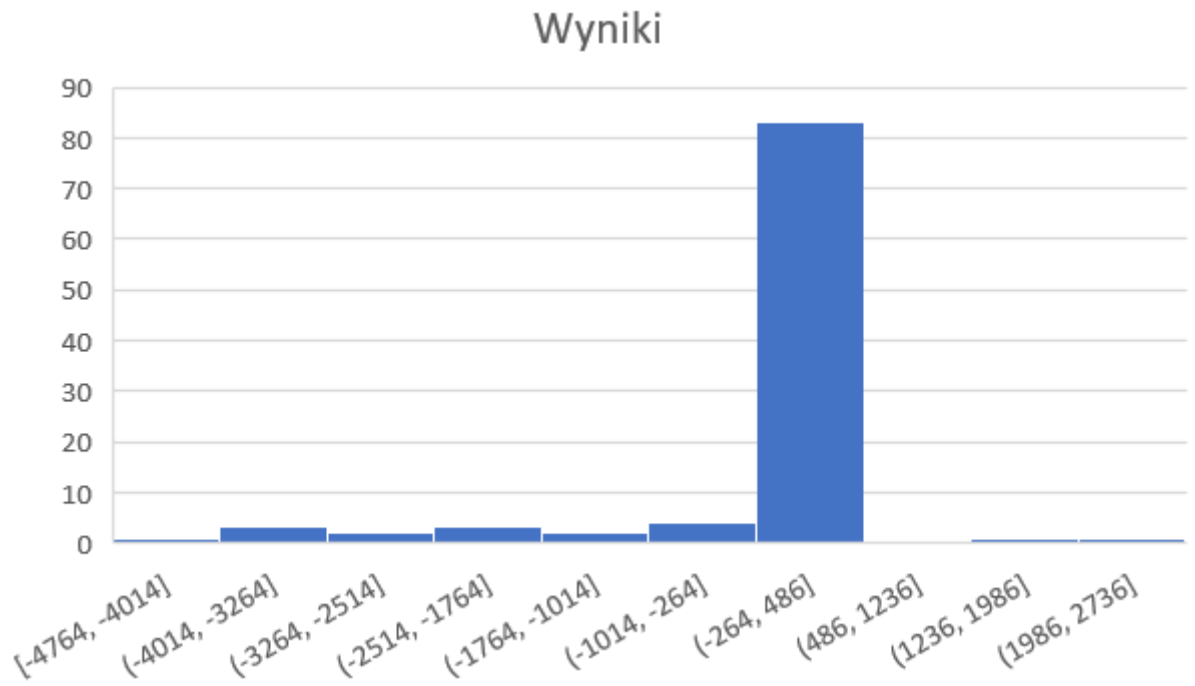
Dla zadań 1-2 zadania wyniki zebrałem w postaci poniżej tabeli. Jak widać rezultaty są różne i nie jesteśmy w stanie przewidzieć wartości końcowej z powodu braku synchronizacji wątków.

Wartość końcowa	Ilość wystąpień
0	82
-1025	1
-2594	1
-3938	1
-2531	1
-452	1
-3846	1
1413	1
-1990	1

-781	1
-1328	1
-2128	1
-2166	1
-122	1
-4764	1
2556	1
-382	1
-799	1
-3616	1

Tabela 1.

Z powyższych wyników utworzyłem histogram.



Histogram 1.

Po wprowadzonej synchronizacji w zadaniu 3 wartości końcowe mają się następująco.

Wartość końcowa	Ilość wystąpień
0	100

Tabela 2.

Zatem jak widać synchronizacja przebiegła poprawnie.

4. Wnioski

1. Programowanie wielowątkowe jest potencjalnie trudnym zadaniem ze względu na potrzebę synchronizacji i współdzielenia danych między wątkami. Zmienne warunkowe są jednym z narzędzi, które można użyć do zarządzania współbieżnością.
2. Wykorzystanie zmiennej warunkowej pozwala na kontrolę kolejności działania wątków i może pomóc w uniknięciu problemów związanych z wyścigami tzw. race conditions.
3. Histogram jest przydatnym narzędziem do analizy rozkładu wyników w programach wielowątkowych. Pomaga on zrozumieć, jakie wartości mogą pojawić się po wykonaniu wielu iteracji programu pomagając zidentyfikować ewentualne problemy z synchronizacją.
4. Implementacja własnych mechanizmów synchronizacji może być skomplikowana, podatna na błędy oraz niewydajna. W przypadku bardziej zaawansowanych problemów warto rozważyć użycie gotowych narzędzi i mechanizmów synchronizacji dostępnych w języku Java, takich jak synchronized, wait, notify lub klasy z pakietu java.util.concurrent.

5. Bibliografia

- Materiały do laboratorium
- Książka Inside the Java Virtual Machine
- Jacek Rumiński, Język Java
- Dokumentacja Javy