Tomasz Ligęza

Programowanie równoległe. Przetwarzanie równoległe i rozproszone.

Sprawozdanie z laboratorium 6.

Cel zajęć:

- opanowanie podstaw tworzenia wątków w języku Java,
- opanowanie podstawowych metod synchronizacji w języku Java,

W ramach zajęć zrealizowałem następujące kroki:

1. Utworzyłem katalog roboczy lab_6,
2. Uruchomiłem metodę main z klasy HistogramTest oraz sprawdziłem, czy kod wykonuje się prawidłowo,

Wariant 1:

3. Zaimplementowałem klasę Watek dziedziczącą po klasie Thread:

```
public class Watek extends Thread {
    Obraz obraz;
    int numOfChar = -1;
    public Watek(int numOfChar, Obraz obraz_1) {
        obraz = obraz_1;
        this.numOfChar = numOfChar;
    }
    @Override
    public void run() {
        obraz.calculateHistogramOf(numOfChar);
        obraz.printHistogramOf(numOfChar);
    }
}
```

4. Dodałem dodatkowy parametr konstruktora klasy Obraz tak, żeby móc przesyłać do niego ilość różnych znaków (również zastąpiłem wszystkie wystąpienia “magic number” równego 93 nową zmienną LICZBA_ZNAKÓW):

```
public Obraz(int n, int m, int numOfChars) {
    LICZBA_ZNAKOW = numOfChars;
    this.size_n = n;
    this.size_m = m;
    tab = new char[n][m];
    tab_symb = new char[LICZBA_ZNAKOW];
    [...]
}
```

5. W metodzie main tworzę tyle Wątków, ile jest różnych znaków w tablicy:

```
System.out.println("Set number of threads");
int num_threads = scanner.nextInt();

Watek[] NewThr = new Watek[num_threads];

for (int i = 0; i < num_threads; i++) {
   (NewThr[i] = new Watek(i, obraz_1)).start();
}

for (int i = 0; i < num_threads; i++) {
   try {
       NewThr[i].join();
   } catch (InterruptedException e) {
   }
}

System.out.println("Czy oba histogramy sie zgadzaja? "
                   + obraz_1.checkBothHistograms());
```

6. Dodałem metody do klasy Obraz realizujące obliczanie i wypisywanie histogramu dla konkretnego znaku:

```
public void calculateHistogramOf(int numOfChar) {
   for (int i = 0; i < size_n; i++) {
       for (int j = 0; j < size_m; j++) {
           if (tab[i][j] == tab_symb[numOfChar])
parallel_histogram[numOfChar]++;
       }
   }
}

public synchronized void printHistogramOf(int numOfChar) {
   System.out.print("Watek [" + numOfChar + "]: "
                    + tab_symb[numOfChar] + " "
                    + parallel_histogram[numOfChar] + " ");
   for (int i = 0; i < parallel_histogram[numOfChar]; i++) {
       System.out.print("=");
   }
   System.out.println();
}
```

Wariant 2:

7. Zaimplementowałem klasę WatekDec implementującą interfejs Runnable:

```
public class WatekDec implements Runnable {
    Obraz obraz;
    int start;
    int end;
    public WatekDec(int start, int end, Obraz obraz_1) {
        obraz = obraz_1;
        this.start = start;
        this.end = end;
    }
    @Override
    public void run() {
        obraz.calculateHistogramBetween(start, end);
        obraz.printHistogramBetween(start, end);
    }
}
```

8. Dodałem kod wykorzystujący klasę WatekDec w metodzie main:

```
Thread[] DecThr = new Thread[num_threads];

int charsForThread = (int) Math.ceil((double)NUMBER_OF_CHARS/num_threads);

for (int i = 0; i < num_threads; i++) {
    int start = charsForThread * i;
    int end = Math.min(((i + 1) * charsForThread), NUMBER_OF_CHARS);
    (DecThr[i] = new Thread(new WatekDec(start, end, obraz_1))).start();
}

for (int i = 0; i < num_threads; i++) {
    try {
        DecThr[i].join();
    } catch (InterruptedException e) {
    }
}

System.out.println("Czy oba histogramy sie zgadzaja? " +
                   obraz_1.checkBothHistogramsDec());
```

9. Dodałem metody do klasy Obraz realizujące obliczanie i wypisywanie histogramu wykorzystując dekompozycję blokową:

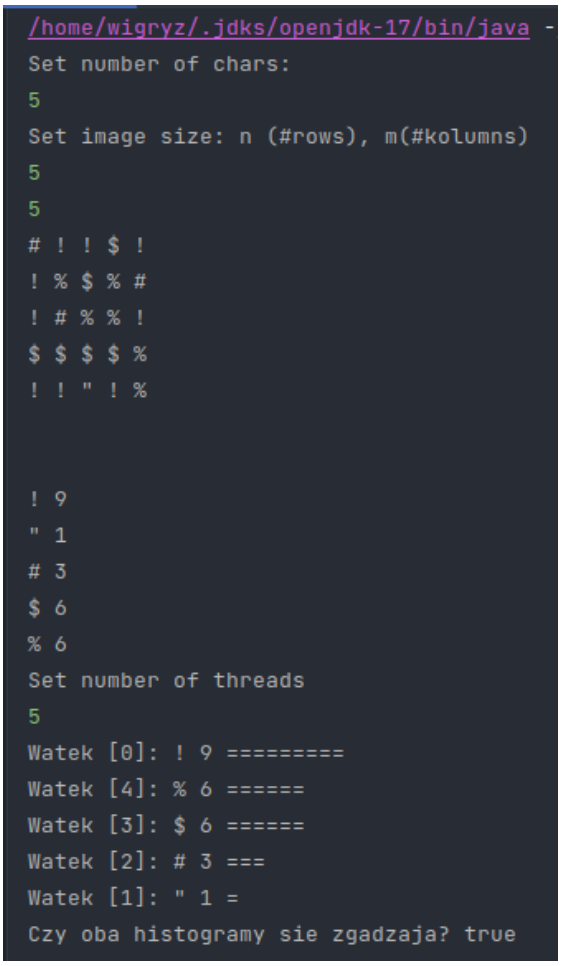
```
public void calculateHistogramBetween(int start, int end) {
    for (int i = 0; i < size_n; i++) {
        for (int j = 0; j < size_m; j++) {
            for (int k = start; k < end; k++) {
                if (tab[i][j] == tab_symb[k]) dec_histogram[k]++;
            }
        }
    }
}

public synchronized void printHistogramBetween(int start, int end) {
    for(int k=start ; k < end ; k++) {
        System.out.print("Watek ["+ Thread.currentThread().getName() + "]:"
                         + tab_symb[k] + " "
                         + dec_histogram[k] + " ");
        for (int i = 0; i < dec_histogram[k]; i++) {
            System.out.print("=");
        }
        System.out.println();
    }
}
```

Kod metod sprawdzających poprawność policzonych histogramów:

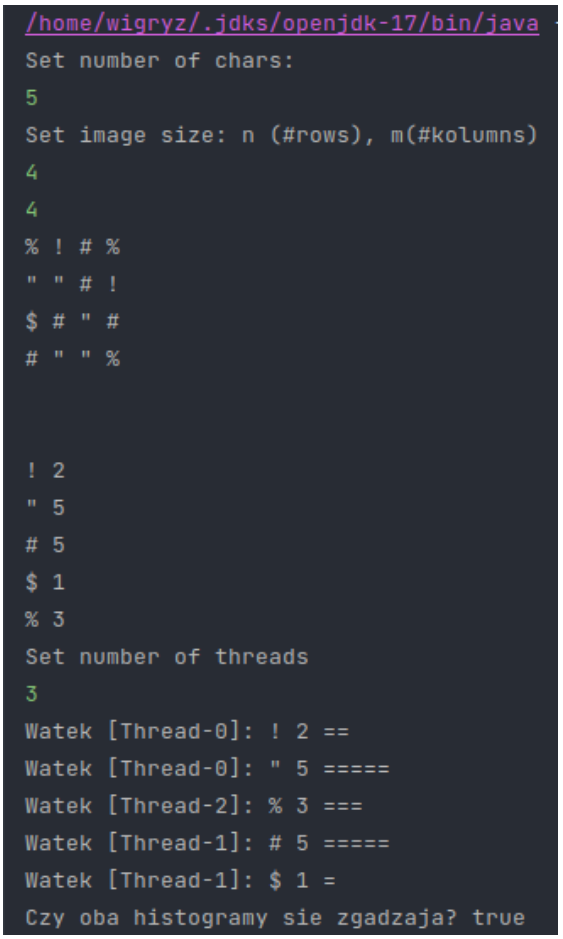
```
public boolean checkBothHistograms() {
    for (int i = 0; i < histogram.length; i++) {
        if (histogram[i] != parallel_histogram[i])
            return false;
    }
    return true;
}

public boolean checkBothHistogramsDec() {
    for (int i = 0; i < histogram.length; i++) {
        if (histogram[i] != dec_histogram[i])
            return false;
    }
    return true;
}
```

Zrzuty ekranu z wykonywania programu:

Wersja z równą ilością znakóœ i wątków:



Wersja z dekompozycją blokową:



Wnioski:

Składnia tworzenia wątków i pracy z nimi w Javie w porównaniu do C jest znacznie prostsza. Obsłużenie kodu mającego działać współbieżnie wyłącznie jednym słowem kluczowym synchronized używanym w definicji metody znacznie upraszcza pisanie prostych współbieżnych programów.

Jednym z nieoczywistych problemów współbieżnych jest wypisywanie tekstu w konsoli. Może wydawać się, że wypisując tekst będziemy go tylko odczytywali ze zmiennych, więc race condition nie zajdzie, aczkolwiek należy pamiętać, że korzystamy wtedy również z obiektu System.out. Dokumentacja nie wspomina, aby System.out był ”thread-safe”, więc skoro obiektu tego używa kilka wątków, to należałoby go zabezpieczyć.