Sprawozdanie: Implementacja histogramu znaków ASCII w Javie

Wykonał:

Mikita Shmialiou

Opis zadania:

Celem zadania było obliczenie histogramu częstotliwości występowania znaków ASCII w dwuwymiarowej tablicy znaków.

Program zaimplementowano w dwóch wersjach: **sekwencyjnej** (pojedynczy wątek) oraz **równoległej** (wiele wątków). W wersji równoległej wyróżniono dwa warianty: (a) *wariant 1* – każdy wątek liczy wystąpienia dokładnie jednego znaku ASCII (klasa rozszerzająca Thread), (b) *wariant 2* – każdy wątek odpowiada za blok kilku kolejnych znaków ASCII (klasa implementująca Runnable). Program najpierw generuje tablicę o zadanych wymiarach wypełnioną losowymi znakami ASCII, następnie oblicza histogram sekwencyjnie i wypisuje wynik, a potem uruchamia obliczenia równoległe (tworzy odpowiednie wątki, scala wyniki) i porównuje je z wersją sekwencyjną.

Zrealizowane kroki:

- Generacja danych – utworzenie tablicy n×m z losowymi znakami ASCII i wyświetlenie jej.
- Obliczenie histogramu sekwencyjnie (jednowątkowo): zliczenie wystąpień każdego znaku.
- *Wariant 1 (Thread per char)* – uruchomienie 94 wątków, każdy liczy wystąpienia przypisanego znaku.
- *Wariant 2 (Runnable, blokowy)* – podział 94 znaków na zadaną liczbą wątków liczbę bloków; każdy wątek liczy znaki ze swojego bloku.
- Porównanie otrzymanych histogramów i wyświetlenie informacji o zgodności wyników.

Kod:

```
122   // ===== Wariant 1: Thread – po 1 znaku =====
123   class Watek extends Thread {
124       private int startIndex;
125       private int endIndex;
126       private Obraz obraz;
127
128       public Watek(int startIndex, int endIndex, Obraz obraz) {
129           this.startIndex = startIndex;
130           this.endIndex = endIndex;
131           this.obraz = obraz;
132       }
133
134       public void run() {
135           for (int k = startIndex; k < endIndex; k++) {
136               int count = obraz.calculate_for_char(k);
137               obraz.print_for_char(k, count, this.getName());
138           }
139       }
140   }
141
142   // ===== Wariant 2: Runnable – blok znaków =====
143   class HistogramRunnable implements Runnable {
144       private int startIndex;
145       private int endIndex;
146       private Obraz obraz;
147
148       public HistogramRunnable(int startIndex, int endIndex, Obraz obraz) {
149           this.startIndex = startIndex;
150           this.endIndex = endIndex;
151           this.obraz = obraz;
152       }
153
154       @Override
155       public void run() {
156           obraz.calculate_for_range(startIndex, endIndex);
157       }
158   }
159
```

Wnioski

- **Poprawność:** Obie wersje programu powinny dawać identyczne wyniki histogramu (co weryfikujemy porównując tablice wyników). Testy pokazały, że połączenie

wyników wątków (zarówno w wariancie 1 jak i 2) zgadza się z obliczeniem sekwencyjnym. Program informuje o *zgodności histogramów* po zakończeniu obliczeń.

- **Wydajność:** Wersja równoległa może być szybsza przy dużych danych i gdy liczba wątków jest dostosowana do liczby rdzeni procesora. Należy jednak pamiętać, że tworzenie i zarządzanie wątkami generuje narzut czasowy. Uruchomienie bardzo wielu wątków (np. 94 wątki dla każdego znaku) może być mniej efektywne od użycia mniejszej liczby wątków, z powodu przełączania kontekstu i kosztów administracyjnych.
- **Zalecenia konstrukcyjne:** W Javie zwykle korzystniej jest implementować Runnable niż rozszerzać klasę Thread, ponieważ pozwala to oddzielić logikę zadania od mechanizmu wykonania. Dzięki temu kod jest bardziej elastyczny (np. można wykorzystać pule wątków). W zadaniu pokazano obie metody (klasę Thread i Runnable) zgodnie z wymaganiami ćwiczenia.
- **Zrzuty ekranu:**

```
➜  lab2 java Histogram_test
Set image size: n (#rows), m(#columns)
10 10
~ w d v } M \ / B W
n ) ^ s h L ` > ) }
r ( J } I G b S A )
H a X 6 : [ ? q 3 -
t _ ^ # [ M t \ k y
2 | 1 [ p W j ) \ Z
u h D ^ i e Q G h 6
" G p C p C r ` \ w
: q H i # u j U 6 $
% = < S 1 { F ) S b


Sequential histogram:
! 0
" 1
# 2
$ 1
% 1
& 0
' 0
( 1
) 5
* 0
+ 0
```

```
Wątek-1: J
Wątek-1: K =
Wątek-1: L =
Wątek-1: M ==
Wątek-1: N ==
Wątek-1: O ==
Wątek-1: P
Wątek-3: {
Wątek-3: | ==
Wątek-3: } =
Wątek-3: ~ =

[Version 2] Parallel histogram results:
Histograms match!
```