Szymon Budziak 125582
PW9

Wątki w Javie można tworzyć np. poprzez utworzenie własnej klasy, która dziedziczy po klasie Thread, jest to sposób wykorzystywany przeze mnie w poniższych zadaniach. W tym przypadku konieczne jest nadpisane metody public void run() - zostaje ona wykonana jako ciało wątku.

Można również w swojej klasie zaimplementować interfejs Runnable, i w nim nadpisać metodę public void run().

W celu umożliwienia dostępu wielu wątkom do wspólnego zasobu i zabezpieczeniu się przed niechcianymi konsekwencjami możemy wewnątrz ciała wątku utworzyć blok synchronized(object), gdzie object jest obiektem do którego dostęp będzie miał tylko jeden wątek na raz.

Jeśli chcemy aby wątek rozpoczął wykonywanie wykonujemy na instancji Thread metodę start()
Jeśli chcemy zaczekać aż wątek umrze możemy użyć na jego instancji metodę join()

1. Napisz program sumujący dwa losowe wektory o ustalonej długości n. Zaprojektuj właściwe zrównoważenie obciążenia wątków dla przypadku gdy n nie jest podzielne przez liczbę wątków.

Klasa Vector przedstawia wspomniany w zadaniu wektor, ma pole size i content. Metodę initiateVecotr odpowiedzialną za wstawianie losowych liczb do wektora, oraz resetVector służącą wektorowy wynikowemu do wypełnienia swojego contentu nullami.

Klasa Thr dziedziczy po Thread, do swojego konstruktora przyjmuje dodawane wektory, wektor wynikowy, oraz indeksy początkowy i końcowy pomiędzy którymi ma operować. W nadpisywanej metodzie run() operując między wspomnianymi indeksami ustawia w wektorze wynikowym wynik dodawania odpowiadających wartości z wektorów wynikowych

```
@Override
public void run() {
    for (int i = start; i <= end; i++) {
        vector3.getContent().set(i, vector1.getContent().get(i) + vector2.getContent().get(i));
    }
}
```

W klasie Main tworzymy tyle porcji ile mamy mieć wątków, podstawowa porcja wynosi rozmiar/ilość wątków, jest to liczba całkowita, zakrąglona w dół. Następnie w celu jak najrówniejszego ustawienia wielkości porcji liczymy resztę z dzielenia rozmiar/ilość wątków, jeśli jest to liczba różna od 0, zwiększamy kolejne porcje o 1, dopóki nie skończy nam się reszta.

Następnie po stworzeniu wektorów uruchamiamy wątki każdemu przydzielając odpowiedni zakres na jakim ma operować.

Przykładowy wydruk na końcu sprawozdania*.

2. Napisać program współbieżny wyliczający histogram dla obrazu o wymiarze n na m, przy użyciu wątków Javy. Dla uproszczenia niech obraz będzie dwuwymiarową tablicą zmiennych typu char.

Klasa Thr dziedziczy po Thread, w swoim konstruktorze ma ArrayList characters, będącą rozłożoną tablicą dwuwymiarową której histogram mamy wykonać, do tego ArrayList usedChars, która przechowuje znaki już znalezione, swój numer id, oraz rozmiar tablicy.
Metoda countChar przyjmuje znak którego ma szukać a następnie iteruje po characters i zlicza wystąpienia danego znaku.
Metoda findFreeCharacter() używa zsynchronizowanej usedChars,
iteruje po characters i sprawdza czy dany znak znajduje się w usedChars, jeśli nie to zwraca go jako następny znak do liczenia.

Metoda run() na początku szuka wolnego znaku, jeśli go znalazła wchodzi do pętli while, oblicza wystąpnienia danego znaku i szuka następnego, do momentu aż po przejściu całego characters nie znajdzie żadnego wolnego znaku.

```
@Override
public void run() {
    Character freeChar = findFreeCharacter();
    while (freeChar != null) {
        countChar(freeChar);
        freeChar = findFreeCharacter();
    }
}
```

W klasie Main ustawiamy wymiary tablicy znaków, liczbę wątków, znaki którymi będzemy losowo wypełniać tę tablicę, konwertujemy tablicę dwuwymiarową na ArrayList w celu ułatwienia streamowania. Tworzymy wątki.

Przykładowy wydruk dla parametrów:
n = 1000
m = 555
liczba wątków = 4
wstawiane znaki = '!', '@', '#', '$', '%', '^', '&', '*'

```
Watek 1: @ 69282x
Watek 2: % 69580x
Watek 3: $ 69566x
Watek 0: & 69363x
Watek 1: * 69278x
Watek 2: ^ 69234x
Watek 0: ! 69687x
Watek 3: # 69010x
```

3. Napisz program znajdujący liczby pierwsze używając tzw. sita Eratostenesa

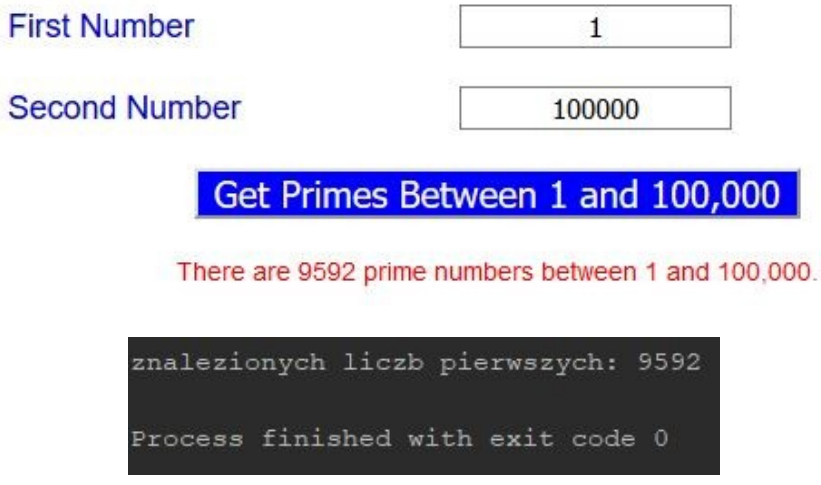
W celu rozwiązania zadania wykorzystałem algorytm z wikipedii

```
algorithm Sieve of Eratosthenes is
    input: an integer n > 1.
    output: all prime numbers from 2 through n.

    let A be an array of Boolean values, indexed by integers 2 to n,
    initially all set to true.

    for i = 2, 3, 4, ..., not exceeding √n do
        if A[i] is true
            for j = i², i²+i, i²+2i, i²+3i, ..., not exceeding n do
                A[j] := false

    return all i such that A[i] is true.
```

w celu podzielenia zakresu na części wykonujemy sqrt(size)/liczba wątków, następnie do każdego z wątków przekazujemy liczbę od od której ma zacząć, na której kończyć, ostatnią liczbę zakresu(rozmiar), oraz tablicę Booleanów.

W klasie Thr która dziedziczy po Thread w konstruktorze znajdują się ww. argumenty. Każdy z wątków wykonuje pokazany wyżej algorytm dla przekazanego mu zakresu, operując na tablicy Booleanów.

Poprawność wyników:




```
znalezionych liczb pierwszych: 9592

Process finished with exit code 0
```

*Wydruk dla zadania 1
Rozmiar:10
Wątki: 3