


SPRAWOZDANIE NR 1			
Nazwa ćwiczenia	Statystyka wyrazów		 POLITECHNIKA BYDGOSKA Wydział Telekomunikacji, Informatyki i Elektrotechniki
Przedmiot	Zaawansowane programowanie obiektowe		
Student grupa	Oskar Bartoszyński grupa 3		
Data ćwiczeń	10.03.2025	19.03.2025	Data oddania sprawozdania

Wnioski dotyczące kodu MainFrame

Wyrażenia lambda

Wyrażenia lambda zastosowane w kodzie znacząco upraszczają implementację interfejsów funkcyjnych. Na przykład:

```
Runnable producent = () -> {
    // kod producenta
};
```

Dzięki takiemu podejściu nie trzeba tworzyć osobnych klas implementujących interfejs Runnable. Wyrażenia lambda pozwalają na zwarte definiowanie zachowań i funkcjonalności, co czyni kod bardziej czytelnym i łatwiejszym w utrzymaniu. Są one szczególnie przydatne przy pracy z kolekcjami i strumieniami danych.

Strumienie danych

W metodzie getLinkedCountedWords() widać zastosowanie strumieni danych, które umożliwiają przetwarzanie sekwencji elementów w stylu funkcyjnym:

```
reader.lines()
    .flatMap(line -> Stream.of(line.split("\\s+")))
    .map(word -> word.replaceAll("[^a-zA-Z0-9ąęóśźńżĄĘÓŚĆŻŃŻ]", ""))
    .filter(word -> word.matches("[a-zA-Z0-9ąęóśźńżĄĘÓŚĆŻŃŻ]{3,}")
    .map(String::toLowerCase)
    .collect(Collectors.groupingBy(Function.identity(), Collectors.counting()))
```

Strumienie pozwalają na łańcuchowe wywoływanie operacji, co sprawia, że algorytm jest bardziej przejrzysty i zrozumiały. Operacje takie jak flatMap, map, filter i collect umożliwiają elegancko przetwarzanie danych bez konieczności używania pętli, co zmniejsza ryzyko błędów i czyni kod bardziej deklaratywnym.

Wzorzec Producent-Konsument

Kod implementuje klasyczny wzorzec Producent-Konsument, gdzie producenci dodają ścieżki do plików do współdzielonej kolejki, a konsumenci przetwarzają te pliki. Komunikacja między

producentami a konsumentami odbywa się poprzez BlockingQueue, która zapewnia bezpieczną i efektywną wymianę danych między wątkami.

Poison Pills

Interesującym aspektem implementacji jest wykorzystanie "poison pills" (trucizny) do sygnalizowania zakończenia pracy. Gdy użytkownik kliknie przycisk "Stop", producent wysyła `Optional.empty()` do kolejki dla każdego konsumenta:

```
for (int i = 0; i < liczbaKonsumentow; i++) {  
    kolejka.put(Optional.empty());  
}
```

Konsumenti, po otrzymaniu takiej wartości, wiedzą, że powinni zakończyć swoją pracę:

```
if (!optPath.isPresent()) {  
    // Odebrano poison pill, kończymy pracę  
    System.out.println("Konsument " + name + " otrzymał poison pill. Kończenie pracy.");  
    break;  
}
```

Jest to technika zakończenia pracy wątków bez konieczności używania brutalnych metod, takich jak `interrupt()`.

Zarządzanie wątkami

Kod wykorzystuje `ExecutorService` do zarządzania pulą wątków, co jest zgodne z najlepszymi praktykami programowania współbieżnego w Javie. Zamiast ręcznie tworzyć i zarządzać wątkami, używa się wysokopoziomowych abstrakcji, które zapewniają efektywne wykorzystanie zasobów.

Obsługa przerwania

Kod poprawnie obsługuje przerwanie wątków, co jest ważnym aspektem w programowaniu współbieżnym. W przypadku przerwania wątku, informacja jest logowana i odpowiednio obsługiwana:

```
catch (InterruptedException e) {  
    info = String.format("Oczekiwanie konsumenta %s przerwane!", name);  
    System.out.println(info);  
    Thread.currentThread().interrupt();  
}
```

Ponowne ustawienie flagi przerwania za pomocą `Thread.currentThread().interrupt()` jest dobrą praktyką, ponieważ pozwala na propagację przerwania w górę stosu wywołań.

Podsumowanie

Kod przedstawia zaawansowane wykorzystanie mechanizmów programowania współbieżnego w Javie. Zastosowanie wyrażeń lambda, strumieni danych, wzorca Producent-Konsument oraz technik takich jak poison pills.