Zaawansowane programowanie w Javie

Studia zaoczne – Wykład 4

dr hab. Andrzej Zbrzezny, profesor UJD

Katedra Matematyki i Informatyki
Uniwersytet Jana Długosza w Częstochowie

6 kwietnia 2024

Zalecana Literatura

Literatura podstawowa

- Cay Horstmann.
  Java. Przewodnik doświadczonego programisty. Wydanie III.
  Wydawnictwo Helion. Gliwice, październik 2023.
- Joshua Bloch.
  Java. Efektywne programowanie. Wydanie III.
  Wydawnictwo Helion. Gliwice, sierpień 2018.
- Cay Horstmann.
  Java. Podstawy. Wydanie XII.
  Wydawnictwo Helion. Gliwice, grudzień 2022.
- https://dev.java/

Strumienie

Wprowadzenie

- Strumienie prezentują dane w sposób, który umożliwia przeniesienie przetwarzania danych na wyższy poziom abstrakcji, niż ma to miejsce w przypadku kolekcji.
- W przypadku strumieni określamy, co powinno zostać wykonane, a nie – w jaki sposób to wykonać. Wykonanie operacji pozostawiamy implementacji.
- Dla przykładu przypuśćmy, że chcemy obliczyć średnią wartość wybranego parametru.
- Określamy źródło danych i parametr, a biblioteka obsługująca strumień optymalizuje obliczenia, korzystając przykładowo z wielu wątków do obliczania sum i zliczania, a następnie przetwarzania rezultatów.

Strumienie

Kluczowe zagadnienia

- Iteratory narzucają konkretną strategię przetwarzania i uniemożliwiają wydajne przetwarzanie równoległe.
- Można utworzyć strumienie z kolekcji, tablic, generatorów czy iteratorów.
- Do wybierania elementów używa się metody `filter`, a do przekształcania elementów metody `map`.
- Inne operacje przekształcające strumienie to: `limit`, `distinct` oraz `sorted`.

Strumienie

Kluczowe zagadnienia

- Aby pobrać wynik działania ze strumienia, używamy operatorów redukujących, takich jak `count`, `max`, `min`, `findFirst` lub `firstAny`. Niektóre z tych metod zwracają wartość typu `Optional`.
- Typ `Optional` stanowi bezpieczny, alternatywny sposób pracy z wartościami `null`. Aby bezpiecznie go używać, wykorzystuje się metody `ifPresent` i `orElse`.
- Można zebrać wyniki ze strumieni w kolekcjach, tablicach, ciągach znaków lub mapach.
- Metody `groupingBy` i `partitioningBy` klasy `Collectors` pozwalają podzielić zawartość strumienia na grupy i ustalić wyniki dla każdej z grup.

Strumienie

Kluczowe zagadnienia

- Istnieją specjalne strumienie dla typów prostych: `int`, `long` oraz `double`.
- Równoległe strumienie automatyzują równoległe wykonywanie operacji na strumieniach.
- Podczas pracy ze strumieniami równoległymi należy unikać efektów ubocznych i rozważyć zrezygnowanie z warunków narzucających uporządkowanie.
- Aby użyć biblioteki strumieniowej, trzeba znać pewną liczbę interfejsów funkcyjnych.

Od iteratorów do operacji strumieniowych

- Przetwarzając kolekcję, zazwyczaj przechodzimy przez jej kolejne elementy i wykonujemy na każdym z nich odpowiednie operacje.
- Przypuśćmy, że chcemy policzyć wszystkie długie wyrazy w książce. Najpierw umieścimy je na liście:

```
// Wczytujemy plik do ciągu znaków
String contents = new String(Files.readAllBytes(
    Paths.get("alice.txt")), StandardCharsets.UTF_8);
// Dzielimy ciąg znaków na słowa
// Znaki inne niż litery są ogranicznikami
List<String> wordList =
    Arrays.asList(contents.split("\\PL+"));
```

- Wyrażenie regularne `"\\PL+"` określa wszystkie łańcuchy znaków, których elementami nie są litery.

Od iteratorów do operacji strumieniowych

- Teraz możemy rozpocząć iterację:

```
int count = 0;
for (String word : wordList) {
    if (word.length() > 12) ++count;
}
```

Od iteratorów do operacji strumieniowych

- Co jest w poprzednim przykładzie nie tak? Tak naprawdę nic – poza tym, że trudno jest zrównoleglić kod.
- Gdy korzystamy ze strumieni, przykład z poprzedniego slajdu wygląda tak:

```
long count = wordList.stream()
    .filter(w -> w.length() > 12)
    .count();
```

- Metoda `stream` zwraca strumień dla listy słów.
- Metoda `filter` zwraca inny strumień, który zawiera tylko słowa o długości większej niż dwanaście.
- Metoda `count` redukuje ten strumień do wyniku.

Strumienie

Od iteratorów do operacji strumieniowych

- Strumienie wydają się być bardzo podobne do kolekcji, gdyż pozwalają przekształcać i pobierać dane. Istnieją jednak znaczące różnice:
  1. Strumień nie przechowuje swoich elementów. Mogą one być przechowywane w wykorzystywanej przez strumień kolekcji lub generowane na żądanie.
  2. Operacje strumienia nie modyfikują danych źródłowych. Np. metoda `filter` nie usuwa elementów ze strumienia, ale zwraca nowy strumień, w którym pewne elementy nie są umieszczane.
  3. Operacje wykonywane przez strumień są leniwe, jeżeli tylko jest to możliwe. Oznacza to, że ich wykonanie jest opóźniane do chwili, gdy potrzebny jest wynik działania. Na przykład jeżeli zażądamy pierwszych pięciu długich słów, a nie wszystkich, metoda `filter` zatrzyma filtrowanie po odnalezieniu piątego słowa.
  4. Dzięki temu można korzystać nawet ze strumieni o nieskończonej długości!

Strumienie

Od iteratorów do operacji strumieniowych

- Strumień może być w łatwy sposób zrównoleglony:

```
long count = wordList.parallelStream()
    .filter(w -> w.length() > 12)
    .count();
```

- Prosta zamiana `stream` na `parallelStream` pozwala bibliotece obsługującej strumienie zrównoleglić filtrowanie i zliczanie.
- Strumienie kierują się zasadą „co, nie jak”. W przykładzie ze strumieniem opisujemy, co chcemy wykonać: wybrać długie wyrazy i je policzyć.
- Nie określamy, w jakiej kolejności ani w jakim wątku ma to być wykonane – inaczej niż w pętli zaprezentowanej na slajdzie nr 7, gdzie opisano, jakie dokładnie obliczenia mają zostać wykonane, co uniemożliwiało wprowadzenie jakichkolwiek optymalizacji.

Od iteratorów do operacji strumieniowych

- Taki przepływ pracy jest typowy przy pracy ze strumieniami. Przygotowujemy zestaw operacji w trzech etapach:
  1. Tworzymy strumień.
  2. Określamy pośrednie operacje przekształcające początkowy strumień do innej postaci; może to wymagać wykonania kilku kroków.
  3. Wykonujemy końcową operację generującą wynik. Ta operacja wymusza wykonanie leniwych operacji niezbędnych do jej zakończenia. Po tym kroku strumień nie może być dalej wykorzystywany.
- W naszym przykładzie strumień był tworzony za pomocą metody `stream` lub `parallelStream` z interfejsu `Collection`.
- Metoda `filter` przekształciła ten strumień, a na końcu została wykonana operacja `count`.

Strumienie

Od iteratorów do operacji strumieniowych

- Operacje strumieniowe nie są wykonywane na elementach w kolejności w które są wywoływane w strumieniach.
- Gdy metoda `count` prosi o pierwszy element, wtedy metoda `filter` zaczyna żądać elementów, dopóki nie znajdzie takiego, który ma długość > 12.

Strumienie

Przykład (Od iteratorów do operacji strumieniowych)

- Program `r08/r08_01/CountLongWords.java`
  - Kompilacja:
    ```
    javac r08/r08_01/CountLongWords.java
    ```
  - Wykonanie:
    ```
    java r08.r08_01.CountLongWords
    ```

Zadanie

Przeanalizuj powyższy program i upewnij się, że rozumiesz wszystkie zastosowane w nim konstrukcje.

Strumienie

Tworzenie strumieni

- Widzieliśmy już, że można zamienić każdą kolekcję w strumień za pomocą metody `stream` interfejsu `Collection`.
- W przypadku tablicy można do tego wykorzystać statyczną metodę `Stream.of`.

```
Stream<String> words =
    Stream.of(contents.split("\\PL+"));
// split zwraca tablicę typu String[]
```

- Metoda `of` ma argument o zmiennej liczbie elementów (varargs), dzięki czemu można utworzyć strumień z dowolnej liczby argumentów:

```
Stream<String> song =
    Stream.of("gently", "down", "the", "stream");
```

Strumienie

Tworzenie strumieni

- Aby utworzyć strumień z fragmentu tablicy, używamy `Arrays.stream(tablica, od, do)`.
- Aby utworzyć pusty strumień, wykorzystujemy statyczną metodę `Stream.empty`:

```
// Typ generyczny Stream<String> jest wywnioskowany;
// dlatego jest to to samo co Stream.<String>empty()
Stream<String> silence = Stream.empty();
```

- Interfejs `Stream` ma dwie statyczne metody do tworzenia nieskończonych strumieni: `generate` oraz `iterate`.
- Metoda `generate` pobiera bezargumentową funkcję (lub, dokładniej, obiekt implementujący interfejs `Supplier<T>`).
- Gdy jest potrzebna wartość ze strumienia, funkcja ta jest wywoływana, aby utworzyć kolejną wartość.

Strumienie

Tworzenie strumieni

- Można otrzymać strumień jednakowych wartości za pomocą wyrażenia:

```
Stream<String> echos =
    Stream.generate(() -> "Echo");
```

lub strumień liczb losowych, wywołując:

```
Stream<Double> randoms =
    Stream.generate(Math::random);
```

Tworzenie strumieni

- Aby utworzyć nieskończone ciągi takie jak $0, 1, 2, 3, \ldots$, można skorzystać z metody `iterate`.
- Przyjmuje ona wartość początkową oraz funkcję `f` (technicznie `UnaryOperator<T>`) i w pętli wywołuje tę funkcję, podając jako argument poprzedni wynik.
- Przykładowo:

```
Stream<BigInteger> integers = Stream.iterate(
    BigInteger.ZERO, n -> n.add(BigInteger.ONE));
```

- Pierwszym elementem ciągu jest wartość początkowa (ang. seed) równa `BigInteger.ZERO`.
- Kolejnymi elementami są `f(seed)`, czyli 1 (wartość typu `BigInteger`), `f(f(seed))`, czyli 2 itd.

Strumienie

Tworzenie strumieni

- Aby utworzyć skończony strumień, należy dodać predykat określający, kiedy iteracja powinna się zakończyć:
- Przykładowo:

```
BigInteger limit = new BigInteger("10000000");
Stream<BigInteger> integers = Stream.iterate(
    BigInteger.ZERO,
    n -> n.compareTo(limit) < 0,
    n -> n.add(BigInteger.ONE)
);
```

- Gdy tylko predykat odrzuci wygenerowaną iteracyjnie wartość, strumień się kończy.

Strumienie

Tworzenie strumieni

- Wiele metod z Java API zwraca strumienie.
- Przykładowo, klasa `Pattern` zawiera metodę `splitAsStream`, dzielącą zmienną typu `CharSequence` z wykorzystaniem wyrażenia regularnego.
- Można wykorzystać poniższe wyrażenie, by podzielić ciąg znaków na słowa:

```
Stream<String> words = Pattern.
    compile("\\PL+").splitAsStream(contents);
```

- Metoda `Scanner.tokens` dostarcza strumień tokenów skanera. Innym sposobem na uzyskanie strumienia słów z ciągu znaków jest:

```
Stream<String> words =
    new Scanner(contents).tokens();
```

Strumienie

Tworzenie strumieni

- Statyczna metoda `Files.lines` zwraca strumień zawierający wszystkie wiersze pliku.
- Nadinterfejsem interfejsu `Stream` jest interfejs `AutoCloseable`.
- Gdy w strumieniu zostanie wywołana metoda `close`, plik bazowy również zostanie zamknięty.
- Aby mieć pewność, że tak się stanie, najlepiej jest użyć instrukcji try-with-resources wprowadzonej w `Java 7`:

```
try (Stream<String> lines = Files.lines(path)) {
    Do something with lines
}
```

- Strumień i związany z nim plik zostaną zamknięte, gdy blok `try` zakończy działanie normalnie lub poprzez wyjątek.

Strumienie

Przykład (Tworzenie strumieni)

- Program `r08/r08_02/CreatingStreams.java`
  - Kompilacja:
    `javac r08/r08_02/CreatingStreams.java`
  - Wykonanie:
    `java r08.r08_02.CreatingStreams`

Zadanie

Przeanalizuj powyższy program i upewnij się, że rozumiesz wszystkie zastosowane w nim konstrukcje.

Metody `filter`, `map` oraz `flatMap`

- Procedury przekształcające strumień tworzą strumień, którego elementy pochodzą z innego strumienia.
- Widzieliśmy już transformację `filter`, która zwraca nowy strumień z elementami spełniającymi podany warunek.
- Poniżej przekształcamy strumień ciągów znaków w inny, zawierający jedynie długie wyrazy:

```
List<String> wordList = ...;
Stream<String> words = wordList.stream();
Stream<String> longWords =
    words.filter(w -> w.length() > 12);
```

- Argumentem metody `filter` jest `Predicate<T>` – czyli funkcja przekształcająca `T` na **`boolean`**.

Metody `filter`, `map` oraz `flatMap`

- Często zdarza się, że konieczne jest jakieś przekształcenie danych ze strumienia.
- W takiej sytuacji wykorzystujemy metodę `map` i przekazujemy do niej funkcję, która wykona przekształcenie.
- Na przykład można zamienić wszystkie litery w słowach na małe w taki sposób:

```
Stream<String> lowercaseWords =
    words.map(String::toLowerCase);
```

- Użyliśmy tutaj metody `map` z referencją do metody. Często zamiast tego używa się wyrażenia lambda:

```
Stream<Character> firstChars = words.map(s -> s.charAt(0));
```

- Utworzony w ten sposób strumień zawiera pierwsze litery słów.

Metody `filter`, `map` oraz `flatMap`

- Gdy korzystamy z `map`, funkcja jest wykonywana na każdym elemencie, a wynikiem działania jest nowy strumień zawierający efekty jej działania. Załóżmy jednak, że mamy funkcję, która zwraca nie jedną wartość, ale strumień z wartościami, tak jak poniższa:

```
public static Stream<String> codePoints(String s) {
    List<String> result = new ArrayList<>();
    int i = 0;
    while (i < s.length()) {
        int j = s.offsetByCodePoints(i, 1);
        result.add(s.substring(i, j)); i = j;
    }
    return result.stream();
}
```

- Przykładowo, wywołanie `codePoints("boat")` zwraca strumień `['b', 'o', 'a', 't']`.

Metody `filter`, `map` oraz `flatMap`

- Załóżmy, że użyjemy metody `codePoints` do przetworzenia strumienia ciągów znaków:

```
Stream<Stream<Character>> result =
    words.map(w -> codePoints(w));
```

- Otrzymamy strumień strumieni:

```
[... ['y', 'o', 'u', 'r'], ['b', 'o', 'a', 't'], ...]
```

- Aby spłaszczyć strukturę do strumienia zawierającego litery

```
[... 'y', 'o', 'u', 'r', 'b', 'o', 'a', 't', ...],
```

należy użyć metody `flatMap` zamiast `map`:

```
// Przetwarza każde ze słów na litery
// i spłaszcza wyniki
Stream<Character> letters =
    words.flatMap(w -> codePoints(w))
```

Strumienie

Metody `filter`, `map` oraz `flatMap`

- Metoda `flatMap` znajduje się w klasach innych niż strumienie.
- Jest to ogólna koncepcja w inżynierii oprogramowania.
- Załóżmy, że mamy uogólniony typ `G` (taki jak `Stream`) i funkcję `f` przekształcającą pewien typ `T` na `G<U>` oraz funkcję `g` przekształcającą `U` na `G<V>`.
- Następnie łączymy je, czyli najpierw wywołujemy `f`, a potem `g`, korzystając z `flatMap`.
- Jest to kluczowy element teorii monad.
- Jednakże nie musimy się martwić – możemy korzystać z `flatMap`, nie wiedząc nic na temat monad.

Strumienie

Przykład (Metody `filter`, `map` oraz `flatMap`)

- Program `r08/r08_03/CodePointsDemo.java`
  - Kompilacja:
    `javac r08/r08_03/CodePointsDemo.java`
  - Wykonanie:
    `javac r08.r08_03.CodePointsDemo`
- Program `r08/r08_03/FilterMapDemo.java`
  - Kompilacja:
    `javac r08/r08_03/FilterMapDemo.java`
  - Wykonanie:
    `javac r08.r08_03.FilterMapDemo`

Zadanie

Przeanalizuj powyższe programy i upewnij się, że rozumiesz wszystkie zastosowane w nich konstrukcje.

Wycinanie podstrumieni i łączenie strumieni

- Wywołanie `stream.limit(n)` zwraca nowy strumień, który kończy się po `n` elementach (lub z końcem oryginalnego strumienia, jeżeli jest krótszy).
- Ta metoda jest szczególnie przydatna przy ograniczaniu długości nieskończonych strumieni do zadanej wielkości.
- Przykładowo:

```
Stream<Double> randoms =
    Stream.generate(Math::random).limit(100);
```

zwraca strumień zawierający 100 losowych liczb.

Wycinanie podstrumieni i łączenie strumieni

- Polecenie `stream.takeWhile(predykat)` pobiera wszystkie elementy ze strumienia, gdy predykat jest prawdziwy, a następnie zatrzymuje się.
- Przykładowo, załóżmy, że używamy metody `codePoints` ze slajdu 4 do podzielenia łańcucha `str` na znaki i chcemy pobrać wszystkie początkowe cyfry.
- Może to zrobić metoda `takeWhile` :

```
Stream<String> initialDigits =
    codePoints(str).takeWhile(
        s -> "0123456789".contains(s)
    );
```

Wycinanie podstrumieni i łączenie strumieni

- Metoda `dropWhile` działa odwrotnie, usuwając elementy, gdy warunek jest prawdziwy.
- Zwraca ona strumień wszystkich elementów, zaczynając od pierwszego, dla którego warunek był fałszywy.
- Przykładowo dla łańcucha znaków `str`:

```
Stream<String> withoutInitialWhiteSpace =
    codePoints(str).dropWhile(
        s -> s.trim().length() == 0
    );
```

Wycinanie podstrumieni i łączenie strumieni

- Wywołanie `stream.skip(n)` działa dokładnie odwrotnie. Odrzuca ono pierwszych `n` elementów.
- Jest to przydatne w przykładzie dotyczącym czytania książki, w którym z powodu sposobu działania metody `split` pierwszym elementem jest niepotrzebny pusty ciąg znaków.
- Można go odrzucić, wywołując `skip`:

```
Stream<String> words =
    Stream.of(contents.split("\\PL+")).skip(1);
```

Strumienie

Wycinanie podstrumieni i łączenie strumieni

- Można połączyć dwa strumienie za pomocą statycznej metody `concat` z klasy `Stream`:

```
Stream<Character> combined = Stream.concat(
    codePoints("Hello"), codePoints("World"));
// Dostarcza strumień
// ['H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd']
```

- Oczywiście pierwszy ze strumieni nie może być nieskończony – w takim przypadku drugi nigdy się nie pojawi.

Strumienie

Przykład (Wycinanie podstrumieni i łączenie strumieni)

- Program `r08/r08_04/ExtractingCombining.java`
  - Kompilacja:
    `javac r08/r08_04/ExtractingCombining.java`
  - Wykonanie:
    `java r08.r08_04.ExtractingCombining`

Zadanie

Przeanalizuj powyższy program i upewnij się, że rozumiesz wszystkie zastosowane w nim konstrukcje.