

**kurs języka Java**  
**strumienie i lambdy**

Instytut Informatyki  
Uniwersytetu Wrocławskiego

Paweł Rzechonek

---

Strumienie to jeden z najbardziej wydajnych i elastycznych mechanizmów w programowaniu – pozwalają one na manipulowanie danymi w sposób deklaratywny i funkcyjny z użyciem lambd. Strumienie zawierają bogaty zbiór operacji do przetwarzania danych, dzięki czemu umożliwiają pisanie czystych, czytelnych i łatwych do utrzymania aplikacji. Konsepcja strumieni jest znana w wielu językach programowania, ale jej pełen potencjał staje się widoczny szczególnie w Javie, która oferuje naprawdę potężne narzędzia do pracy z danymi pozyskiwanymi ze strumieni. W tym zadaniu należy wykonać szereg czynności obliczeniowych na danych udostępnianych przez strumień.

Część 1.

Przygotuj plik tekstowy z danymi (liczby całkowite) zgodnie z następującym formatem:

- w pliku zapisane są liczby całkowite z zakresu od 1 do  $10^9$  w postaci dziesiętnej bez zer wiodących;
- w jednej linii może być zapisana co najwyżej jedna liczba (jeśli linia nie zawiera liczby to nazywamy ją linią pustą);
- na końcu każdej linii może się znajdować komentarz, rozpoczynający się od sekwencji dwóch ukośników " // " (linia pusta także może zawierać komentarz);
- białe znaki (spacje i tabulacje) na początku i na końcu linii należy zignorować.

Plik powinien zawierać co najmniej 20 liczb, linię pustą i komentarz. Przeczytaj plik z danymi wiersz po wierszu, sprawdź za pomocą wyrażeń regularnych czy format pliku jest zgodny ze specyfikacją (zgłoś wyjątek jeśli nie jest) a wczytane liczby umieść tablicowej implementacji listy `ArrayList<Integer>`. Zastosuj konstrukcję *try-with-resources* przy czytaniu danych z pliku:

```
try (BufferedReader br = new BufferedReader(new FileReader(path))) {  
    for (String ln = br.readLine(); ln != null; ln = br.readLine()) {  
        ...  
    }  
}  
catch (Exception ex) { ... }
```

Następnie za pomocą operacji agregujących na strumieniu `Stream<Integer>` i wyrażeń lambda wykonaj następujące polecenia:

1. wypisz liczby z kolekcji uporządkowane od największej do najmniejszej wartości;
2. wypisz te liczby z kolekcji, które są liczbami pierwszymi;
3. wypisz sumę wszystkich liczb z kolekcji, które są  $< n$  (np. dla  $n = 1000$ );
4. wypisz ile spośród wszystkich liczb w kolekcji jest podzielnych przez  $n$  (np. dla  $n = 7$ ).

## Część 2.

Przygotuj plik tekstowy z danymi (trójkąt opisany długościami trzech boków) zgodnie z następującym formatem:

- w pliku zapisane są trójki liczb rzeczywistych w postaci dziesiętnej z opcjonalną częścią ułamkową po kropce;
- w jednej linii może być zapisana co najwyżej jedna trójkątka liczb (jeśli linia nie zawiera liczby to nazywamy ją linią pustą);
- na końcu każdej linii może się znajdować komentarz, rozpoczynający się od sekwencji dwóch ukośników " / " (linia pusta także może zawierać komentarz);
- białe znaki (spacje i tabulacje) na początku i na końcu linii należy zignorować.

Plik powinien zawierać co najmniej 10 trójkątów liczb (10 trójkątów), linię pustą i komentarz. Przeczytaj plik z danymi wiersz po wierszu, sprawdź za pomocą wyrażeń regularnych czy format pliku jest zgodny ze specyfikacją (zgłoś wyjątek jeśli nie jest) a wczytane liczby umieść listowej implementacji listy `LinkedList<Trojkat>`, gdzie `Trojkat` to zdefiniowana klasa reprezentująca trójkąt w postaci długości trzech boków (w konstruktorze sprawdź, czy podane długości boków spełniają warunek trójkąta). Zastosuj konstrukcję `try-with-resources` przy czytaniu danych z pliku.

Następnie za pomocą operacji agregujących na strumieniu `Stream<Trojkat>` i wyrażeń lambda wykonaj następujące polecenia:

1. wypisz trójkąty z kolekcji uporządkowane od najmniejszego do największego obwodu;
2. wypisz te trójkąty z kolekcji, które są trójkątami prostokątnymi;
3. wypisz ile spośród wszystkich trójkątów w kolekcji jest równobocznych;
4. wypisz dwa trójkąty z kolekcji, których pola są odpowiednio najmniejsze i największe.

## Część 3.

Zdefiniuj rekurencyjne lambdy realizujące następujące zadania:

1. zdefiniuj statyczną lambdę obliczającą długość ciągu Collatza zaczynającego się od wartości naturalnej  $n$  (dla  $n = 1$  długość ciągu wynosi 1); lambda ma być implementacją interfejsu funkcyjnego `UnaryOperator<Integer>`; podstawowe informacje o problemie Collatza możesz znaleźć na Wikipedii:  
[https://pl.wikipedia.org/wiki/Problem\\_Collatza](https://pl.wikipedia.org/wiki/Problem_Collatza)
2. zdefiniuj instancijną lambdę obliczającą największy wspólny dzielnik liczb naturalnych  $a$  i  $b$  zgodnie z algorytmem Euklidesa; lambda ma być implementacją interfejsu funkcyjnego `BinaryOperator<Integer>`; podstawowe informacje o algorytmie Euklidesa możesz znaleźć na Wikipedii:  
[https://pl.wikipedia.org/wiki/Algorytm\\_Euklidesa](https://pl.wikipedia.org/wiki/Algorytm_Euklidesa)