Stream API

Stream Api to zestaw metod do strumieniowego przetwarzania danych. Po streamie możemy przejechać się tylko raz czyli inaczej niż w kolekcjach.

Lazy initialization

Domyślnym sposobem przetwarzania jest **lazy initialization**

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 

final Integer[] integer = {2};

Stream<Integer> integerStream = numbers.stream()
        .map(number -> number * integer[0]);

integer[0] = 10;

integerStream.forEach(System.out::println);
```

Liczby nie będą przemnożone przez 2 ale przez 10! Dzieje się tak dlatego, że w zmiennej integerStream jest zapisana metoda, a nie wynik. W ostatnim wierszu jest wykonywana dopiero metoda, która odwołuje się do 0 elementu tablicy, a tam jest już 10, a nie 2!

Generowanie wartości dla strumieni

https://github.com/idzikpro/JavaBasics/blob/master/src/main/java/pl/idzikpro/stream/StreamValueGenerator.java

1 sposób (predefiniowane wartości)

```
Stream.of("A", "B", "c").forEach(s -> System.out.println(s));
```

2 sposób (kolekcje)

```
List<Student> students = Arrays.asList(
        new Student("Kasia", 22),
        new Student("Jacek", 32),
        new Student("Tomasz", 34)
);
students.stream().filter(s -> s.getAge() > 30).forEach(s -> System.out.println(s));
```

3 sposób (supplier)

```
List<String> stringList = Arrays.asList("Kasia", "Jacek", "Monika");
Random random = new Random();

Stream.generate(() ->
        new Student(
                stringList.get(random.nextInt(3)),
                random.nextInt(40) + 20)
).limit(10).forEach(s -> System.out.println(s));
```

Należy pamiętać, że metoda **generate** działa cały czas i trzeba ją zatrzymać. Najlepiej do tego celu nadaje się metoda **limit**.

4 sposób (iterate, wyświetlenie 10 liczb parzystych, począwszy od 0)

```
Stream.iterate(0,i->i+2).limit(10).forEach(System.out::println);
```

5 sposób (IntStream i znowu liczby podzielne przez 2 z zakresu od 0 do 18)

```
IntStream.rangeClosed(0,18).filter(i->i%2==0).forEach(System.out::println);
```

Podstawowe operacje na strumieniach

https://github.com/idzikpro/JavaBasics/blob/master/src/main/java/pl/idzikpro/stream/StreamOperations.java

Filtr – operacja pośrednia

Filter to **operacja pośrednia** na strumieniach. **Filter** przyjmuje jako argument **Predicate**. Służy do wyfiltrowania obiektów posiadających pewne cechy.

```
students.stream().filter(s -> s.getAge() > 30).forEach(s -> System.out.println(s));
```

Wyświetlamy więc imiona studentów powyżej 30 lat.

Map – operacja pośrednia

Map to **operacja pośrednia** na strumieniach. **Map** przyjmuje jako argument **Function.** Służy do mapowania obiektu na inne obiekty lub typy.

Map i Filter można stosować wiele razy i na przemian.

ForEach – operacja terminalna

ForEach to **metoda terminalna**, która kończy używanie strumienia.

FindFirst, AnyMatch,AllMatch,NoneMatch – operacje terminalne

Wszystkie trzy to **metody terminalne**, które po wykonaniu swoich zadań zamykają strumień.

FindFirst – znajduje pierwsze wystąpienie

Przykład. Wyświetlimy pierwszego studenta, który ma na imię Jacek.

```
students.stream()
        .filter(s -> "Jacek".equals(s.getName()))
        .findFirst()
        .ifPresent(s-> System.out.println(s));
```

findFirst zwraca Optional więc należy postąpić jak powyżej.

AnyMatch – czy dowolny obiekt w strumieniu spełnia Predicate

Przykład. Czy jest jakiś student o imieniu Jacek?

```
students.stream()
        .map(s->s.getName())
        .filter(s->"Jacek".equals(s))
        .findAny()
        .ifPresent(System.out::println);
```

AllMatch – czy wszystkie obiekty spełniają określony warunek

Przykład. Czy wszystkie imiona są palindromami?

```
System.out.println(students.stream()
        .map(s->s.getName())
        .allMatch(s->s.equals(new StringBuilder(s).reverse())));
```

Reduce – operacja terminalna

Metoda redukuje strumień do jednej wartości. Można jej więc użyć np.: do wyszukiwania min,max czy do łączenia stringów.

Przykład. Sumowanie dziesięciu liczb losowych (użycie jako drugiego parametru new BinaryOperator i przekształcić na lambdę)

```
System.out.println(Stream.generate(Math::random)
        .limit(10)
        .reduce(0.0, (a, b) -> a + b));
```

Przykład. Jak znaleźć wiek najstarszego studenta?

```
students.stream()
        .map(s -> s.getAge())
        .max(Comparator.naturalOrder())
        .ifPresent(age-> System.out.println(age));
```

lub

```
students.stream()
        .map(s -> s.getAge())
        .reduce(Integer::max)
        .ifPresent(age-> System.out.println(age));
```

Collect – operacja terminalna

Jest to specjalny typ reduce, który pozwala nam np. na zebranie wszystkich elementów do listy.

Przykład. Jak uzyskać listę wieku studentów?

```
List<Integer> ageList = students.stream()
        .map(s -> s.getAge())
        .collect(Collectors.toList());
System.out.println(ageList);
```

Ponadto **Collectors.counting()** – zwraca liczbę elementów.

Przykład. Jak połączyć wszystkie elementy ze strumienia do jednego stringa z separatorem „ , ” ?

```
String strStudent=students.stream()
        .map(s->s.getName())
        .collect(Collectors.joining(";"));
System.out.println(strStudent);
```

Przykład. Jak utworzyć mapę, klucz = age, wartość = liczba studentów w danym wieku

```
Map<Integer,List<Student>> studentMap=students.stream()
        .collect(Collectors.groupingBy(Student::getAge));
System.out.println(studentMap);
```

Limit, skip, distinct,sorted, count

Limit – ograniczenie do liczby elementów

Skip – pomija określoną liczbę elementów

Distnict – bierzemy pod uwagę tylko różne obiekty (hashcode i equals)

Sorted() – sortowanie wg naturalnego porządku lub new Comparator w nawiasie

Count – oblicza ilość elementów w strumieniu (reduktor)

summaryStatistics – podaje statystykę elementów w strumieniu, np. min, max itp. Działa tylko po zmapowaniu do typów liczbowych.

Strumienie typów prymitywnych

Zaleta?

Szybciej pracuje się na strumieniach prymitywnych

```
IntStream.rangeClosed(0, 18).filter(i -> i % 2 == 0).forEach(System.out::println);
```

Poza **IntStream** może być jeszcze tylko **Long** i **Double.**

Wada?

Od teraz wszędzie w parametrach trzeba podawać **integerowe** odmiany interfejsów funkcyjnych.