**Stream API**

To zestaw metod do strumieniowego przetwarzania danych.

Po streamie możemy przejechać się tylko raz czyli inaczej niż w kolekcjach.

Przykład

**public static void** main(String[] args) {  
 Supplier<List<Student>> supplier= AppSupplier::*createData*;  
 Predicate<Student> over30=student -> student.getAge()>=30;  
 Function<Student,String> getStudentName= student -> student.getName();  
 Consumer<String> print=s -> System.***out***.println(s);  
 List<Student> students=supplier.get();  
**students.stream().filter(over30).map(getStudentName).forEach(print);**}

**Z czego składa się strumień**

Pobranie danych, Operacje **pośrednie**, Operacje **terminalne**

# Generowanie wartości dla strumieni

1 sposób (predefiniowane wartości)

Stream.*of*(**"A"**,**"B"**,**"C"**).forEach(print);

2 sposób (kolekcje)

List<Student> studentList=*createData*();  
studentList.stream().filter(over30).map(getStudentName).forEach(print);

3 sposób (kolekcje)

Stream.*generate*(listSupplier).limit(4).forEach(printConsumer);

Tylko, ze trzeba napisać jeszcze consumera dla listy studentów

Consumer<List<Student>> printConsumer=s-> System.***out***.println(s);

Jeśli nie będzie 4, to będzie generowane w nieskończoność

4 sposób (Supplier)

Stream.*generate*(()->Math.*random*()).limit(10).forEach(System.***out***::println);

5 sposób (iterate, wyświetlenie 20 liczb parzystych, począwszy od 0)

Stream.*iterate*(0,i->i+2).limit(20).forEach(System.***out***::println);

6 sposób (liczby podzielne przez 2)

IntStream.*rangeClosed*(5,100).filter(i->i%2==0).forEach(System.***out***::println);

# Filter

Filter to **operacja pośrednia** na strumieniach. Filter przyjmuje jako argument Predicate.

Niech będzie dane

**public static** Stream<Student> createDataStream(){  
 Student student1=**new** Student(**"Paweł"**,38);  
 Student student2=**new** Student(**"Jacek"**,34);  
 Student student3=**new** Student(**"Kasia"**,38);  
 Student student4=**new** Student(**"Tomasz"**,39);  
**return** Stream.*of*(student1,student2,student3,student4);  
}

**public static void** main(String[] args) {  
 Predicate<Student> over30= student -> student.getAge()>30;  
 Consumer<String> println= System.***out***::println;  
 Function<Student,String> getStudentName=Student::getName;  
  
*createDataStream*().filter(over30).map(getStudentName).forEach(println);  
}

Ostatnia instrukcja wyświetla imiona studentów powyżej 30 lat.

# Map

Map to **operacja pośrednia** na strumieniach. Map przyjmuje jako argument Function.

**Map i Filter** można stosować wiele razy i na przemian.

# ForEach

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

# FindFirst, AnyMatch,AllMatch,NoneMatch

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

Wyświetlimy imię pierwszego studenta, którego wiek przekracza 30 lat

*createDataStream*().filter(over30).map(getStudentName).**findFirst().**ifPresent(System.***out***::println);

**AnyMatch** – czy dowolny obiekt w strumieniu spełnia Predicate

Czy jest jakiś student o imieniu Kasia?

System.***out***.println(*createDataStream*().map(getStudentName).anyMatch(name->name.equals(**"Kasia"**)));

**AllMatch** – czy wszystkie obiekty spełniają określony warunek

Czy wszystkie imiona są palindromami

System.***out***.println(*createDataStream*().map(Student::getName).allMatch(s->**new** StringBuilder(s).reverse().equals(s)));

Czy wszystkie wyrazy znajdujące się w pliku są palindromami?

System.***out***.println(Files.*readAllLines*(Paths.*get*(**"src/main/java/strumienie/palindrom.txt"**)).stream().map(String::toLowerCase).allMatch(s->**new** StringBuilder(s).reverse().equals(s)));

Jeszcze jest **noneMatch()**

# Reduce

Metoda **terminalna**. Redukuje strumień do jednej, konkretnej wartości.Można użyć do wyszukiwania min,max czy do łączenia stringów.

**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, (aDouble, aDouble2) -> aDouble+aDouble2));

Lub

System.***out***.println(Stream.*generate*(Math::*random*).limit(10).reduce(0.0, Double::*sum*));

**Jak znaleźć najstarszego studenta?**

*createDataStream*().map(Student::getAge).max(Comparator.*naturalOrder*()).ifPresent(i-> System.***out***.println(i));

lub

*createDataStream*().map(Student::getAge).reduce(Integer::*max*).ifPresent(System.***out***::println);

# Collect

**Metoda terminalna**, jest to specjalny typ reduce, który pozwala nam np. na zebranie wszystkich elementów w listę.

Jak uzyskać listę wieku studentów?

System.***out***.println(*createDataStream*().map(Student::getAge).collect(Collectors.*toList*()));

Counting – zwraca liczbę elementów.

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

String wiek=*createDataStream*().map(Student::getAge).map(s->s.toString()).collect(Collectors.*joining*(**","**));

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

Map<Integer, List<Student>> list=*createDataStream*().collect(Collectors.*groupingBy*(Student::getAge));

list.forEach(**new** BiConsumer<Integer, List<Student>>() {  
@Override  
**public void** accept(Integer integer, List<Student> students) {  
 System.***out***.println(**"Wiek "**+integer);  
 students.stream().map(Student::getName).forEach(println);  
 }  
});

lub jako lambda

list.forEach((integer, students) -> {  
 System.***out***.println(**"Wiek "**+integer);  
 students.stream().map(Student::getName).forEach(println);  
});

# 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)

# Strumienie typów prymitywnych

**Zaleta?**

Szybciej pracuje się na strumieniach prymitywnych

IntStream intStream=*createDataStream*().map(Student::getAge).mapToInt(value -> value.intValue());

Może być jeszcze tylko Long i Double

**Wada?**

Od teraz wszędzie w parametrach trzeba podawać Integerowe odmiany interfejsów funkcyjnych czyli np. IntPredicate zamiast Predicate.

