Stream API

Programski jezici II

Stream API

- Stream-ovi imaju pogodnije metode nego liste
  - forEach, filter, map, reduce, min, sorted, distinct, limit, etc.
- Stream-ovi imaju korisne osobine koje listama nedostaju
  - Stream-ovi su „moćniji“, brži i memorijski efikasniji nego liste
  - Najvažnije osobine:
    - Lazy evaluacija
    - Automatska paralelizacija
    - Infinite (unbounded) stream-ovi
- Stream-ovi ne skladište podatke
  - Oni su programski wrapper-i oko postojećih izvora podataka

Stream API

- Interface Stream<T>

- boolean allMatch(Predicate<? super T> predicate)
- boolean anyMatch(Predicate<? super T> predicate)
- static <T> Stream.Builder<T> builder()
- <R,A> R collect(Collector<? super T,A,R> collector)
- <R> R collect(Supplier<R> supplier, BiConsumer<R,? super T> accumulator, BiConsumer<R,R> combiner)
- static <T> Stream<T> concat(Stream<? extends T> a, Stream<? extends T> b)
- long count()
- Stream<T> distinct()
- static <T> Stream<T> empty()
- Stream<T> filter(Predicate<? super T> predicate)
- Optional<T> findAny()
- Optional<T> ()
- <R> Stream<R> flatMap(Function<? super T,? extends Stream<? extends R>> mapper)
- DoubleStream flatMapToDouble(Function<? super T,? extends DoubleStream> mapper)
- IntStream flatMapToInt(Function<? super T,? extends IntStream> mapper)
- LongStream flatMapToLong(Function<? super T,? extends LongStream> mapper)
- void forEach(Consumer<? super T> action)
- void forEachOrdered(Consumer<? super T> action)

Stream API

static <T> Stream<T>	generate(Supplier<T> s)
static <T> Stream<T>	iterate(T seed, UnaryOperator<T> f)
Stream<T>	limit(long maxSize)
<R> Stream<R>	map(Function<? super T,? extends R> mapper)
DoubleStream	mapToDouble(ToDoubleFunction<? super T> mapper)
IntStream	mapToInt(ToIntFunction<? super T> mapper)
LongStream	mapToLong(ToLongFunction<? super T> mapper)
Optional<T>	max(Comparator<? super T> comparator)
Optional<T>	min(Comparator<? super T> comparator)
boolean	noneMatch(Predicate<? super T> predicate)
static <T> Stream<T>	of(T... values)
static <T> Stream<T>	of(T t)
Stream<T>	peek(Consumer<? super T> action)
Optional<T>	reduce(BinaryOperator<T> accumulator)
T	reduce(T identity, BinaryOperator<T> accumulator)
<U> U	reduce(U identity, BiFunction<U,? super T,U> accumulator, BinaryOperator<U> combiner)
Stream<T>	skip(long n)
Stream<T>	sorted()
Stream<T>	sorted(Comparator<? super T> comparator)
Object[]	toArray()
<A> A[]	toArray(IntFunction<A[]> generator)

Stream API

- Karakteristike stream-ova

- Stream-ovi ne skladište podatke
- Ne modifikuju strukturu na koju su povezani
- Dizajnirani za Lambde – sve Stream operacije koriste Lambde kao argumente
- Ne podržavaju indeksiran pristup
  - Moguće je zahtijevati prvi element, ali ne i drugi, treći, itd.
- Mogu dati izlaz u formi List-e ili niza
- Lazy
  - Većina Stream operacija se odlaže dok se ne utvrdi koliko podataka je potrebno
- Paralelizacija
  - Ako je Stream paralelan – operacija se mogu odvijati u paraleli
- Mogu biti *unbounded*

Stream API

- for-each

```
for(Employee e: empList) {
e.setSalary(e.getSalary() * 11/10);
}
```

- forEach

```
empList().stream().forEach(e ->
e.setSalary(e.getSalary() * 11/10));

empList().stream().parallel().forEach(e ->
e.setSalary(e.getSalary() * 11/10));
```

Stream API

- Map
  - Transformacija Stream-a propuštanjem elemenata kroz Function

```
Stream<String> stream = list.stream();
stream.map(e -> e + e).forEach(e ->
System.out.println(e));
```

Stream API

- Filter
  - Zadržavanje elemenata koji zadovoljavaju određeni Predicate

```
Stream<Integer> intStream = intList.stream();
intStream.filter(e -> e >
3).forEach(System.out::println);
```

Stream API

- findFirst
  - Vraća prvi element Stream-a ako postoji

```
Optional<Integer> optiInt = intStream.filter(e ->
e > 8).findFirst();
if(optiInt.isPresent()){
                System.out.println(optiInt.get());
}
```

Stream API

- orElse
  - Vraća prvi element ne Stream-a postoji možemo vratiti predefinisani

```
Integer optiInt = intStream.filter(e -> e >
8).findFirst().orElse(0);
System.out.println(optiInt);
```