Streams

Functional interface ?

Any interface with a SAM(Single Abstract Method) is a functional interface, and its implementation may be treated as lambda expressions.

Optional

Optional is a container object used to contain not-null objects. Optional object is used to represent null with absent value. This class has various utility methods to facilitate code to handle values as ‘available’ or ‘not available’ instead of checking null values. It is introduced in Java 8 and is similar to what Optional is in Guava.

Predicate

```
public interface Predicate<T> {

    boolean test(T t);

}
```

```
public class CheckForNull implements
Predicate {

    @Override

    public boolean test(Object o) {

        return o != null;

    }

}
```

Function

```
public interface Function<T,R> {

    public <R> apply(T parameter);

}
```

```
public class AddThree implements
Function<Long, Long> {

    @Override
    public Long apply(Long aLong) {
        return aLong + 3;
    }
}
```

UnaryOperator

```
UnaryOperator<Person> unaryOperator =

        (person) -> { person.name = "New Name"; return person; };
```

BinaryOperator

```
BinaryOperator<MyValue> binaryOperator =

        (value1, value2) -> { value1.add(value2); return value1; };
```

Supplier

```
Supplier<Integer> supplier = () -> new Integer((int) (Math.random() * 1000D));
```

Consumer

```
Consumer<Integer> consumer = (value) -> System.out.println(value);
```

Method reference

```
list.forEach(s -> System.out.println(s));
list.forEach(System.out::println);
```

Что такое lambda ?

Lambda Expressions were added in Java 8. A lambda expression is **a short block of code which takes in parameters and returns a value**. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

Что такое Stream ?

A stream is **a sequence of objects that supports various methods which can be pipelined to produce the desired result**. The features of Java stream are – A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.

Получение объекта Stream

java.util.stream.Stream.

- Пустой стрим: `Stream.empty() // Stream<String>`
- Стрим из List: `list.stream() // Stream<String>`
- Стрим из Map: `map.entrySet().stream() // Stream<Map.Entry<String, String>>`
- Стрим из массива: `Arrays.stream(array) // Stream<String>`
- Стрим из указанных элементов: `Stream.of("a", "b", "c") // Stream<String>`

Stream

- Обработка не начнётся до тех пор, пока не будет вызван терминальный оператор. list.stream().filter(x -> x > 100); не возьмёт ни единого элемента из списка.
- Стрим после обработки нельзя переиспользовать.

Stream API

- Источники
- Промежуточные операторы
- Терминальные операторы

Источники

- empty()
- of(T value)
- of(T... values)
- generate(Supplier s)
- concat(Stream a, Stream b)
- …

Промежуточные операторы

- filter(Predicate predicate)
- map(Function mapper)
- flatMap(Function<T, Stream<R>> mapper)
- limit(long maxSize)
- skip(long n)
- sorted()
- sorted(Comparator comparator)
- distinct()
- peek(Consumer action)
- …

Терминальные операторы

- void forEach(Consumer action)
- long count()
- R collect(Collector collector)
- List<T> toList()
- Optional reduce(BinaryOperator accumulator)
- Optional findAny()
- anyMatch(Predicate predicate)
- Optional findFirst()
- noneMatch(Predicate predicate)
- sum()
- …

toList(), toSet()

- .toList()
- toSet()
-

toMap()

```
1.  Map<Integer, Integer> map1 = Stream.of(1, 2, 3, 4, 5)
2.      .collect(Collectors.toMap(
3.          Function.identity(),
4.          Function.identity()
5.      ));
6.  // {1=1, 2=2, 3=3, 4=4, 5=5}
7.
8.  Map<Integer, String> map2 = Stream.of(1, 2, 3)
9.      .collect(Collectors.toMap(
10.         Function.identity(),
11.         i -> String.format("%d * 2 = %d", i, i * 2)
12.     ));
13. // {1="1 * 2 = 2", 2="2 * 2 = 4", 3="3 * 2 = 6"}
14.
15. Map<Character, String> map3 = Stream.of(50, 54, 55)
16.     .collect(Collectors.toMap(
17.         i -> (char) i.intValue(),
18.         i -> String.format("<%d>", i)
19.     ));
```

Stream exercises

- Obtain a list of products belongs to category “Books” with price > 100
- Obtain a list of product with category = “Toys” and then apply 10% discount
- Get the cheapest products of “Books” category
- Calculate total lump sum of all orders placed i
- Obtain a data map with order id and order’s product count