StudyNotes for Functional Programming Stream Interface.

1. Stream Creation:

* Empty Stream: Stream<String> emptyStream = Stream.empty();
* Stream.builder(), Stream.generated(), Stream.iterate() to add to emptyStream.
* **Collection<String> collection = Arrays.*asList*("a", "b", "c");**

**collection.stream() Collection can apply stream operation directly.**

* String[] strArray = { “a”, “b”, “c” }; Arrays.stream(strArray); or Arrays.stream(strArray, 1, 3); Array need to be converted with Arrays.stream(someArray);
* Stream of Primitives: IntStream, LongStream, DoubleStream
* **Java 8 streams can't be reused**
* **List and ArrayList:**

List<Course> courses = Arrays.*asList*(

**new** Course("Spring", "Framework", 98, 20000),

**new** Course("Spring Boot", "Framework", 95, 18000),

**new** Course("Microservices", "Microservices", 94, 25000),

**new** Course("FullStack", "FullStack", 91, 14000),

**new** Course("Azure", "Cloud", 99, 21000),

**new** Course("Docker", "Cloud", 95, 20000),

**new** Course("Kubernetes", "Cloud", 96, 25000)

);

* **intermediate operations which reduce the size of the stream should be placed before operations which are applying to each element**
* **don't leave an instantiated stream unconsumed**

1. Stream Pipeline:

* Intermediate operations: filter, map, skip, limit, sorted
* Terminal operations: forEach, collect, count, sum

1. Stream Reduction: two main methods: reduce() and collect()

* Reduce( identity, accumulator, combiner) identity: optional, initial value, accumulator: binaryOperator, required, combiner only for parallel mode to reduce the results from different threads.
* Collect(Collectors.toList()): accept an argument of the type Collector, which specifies the mechanism of reduction. There are some created, predefined Collectors for most common operations. (a) use together with map() to generate different lists, like name list, price list, etc. also can use the Collectors to collect specific things.

Map<Integer, List<Product>> collectorMapOfLists = productList.stream().collect(Collectors.groupingBy(Product::getPrice));

Multiple fields groupBy: <https://www.javacodegeeks.com/2021/05/java-8-streams-group-by-multiple-fields-with-collectors-groupingby.html>

TODO: https://docs.oracle.com/javase/tutorial/collections/interfaces/map.html

1. Sorted():

Type of Comparators:

* Collections.naturalOrder: default or Collections.reverseOrder();
* Comparator.comparingInt() or Long or Double;
* Comparator for multiple fields:

courses.stream()

.sorted(Comparator.*comparing*(Course::getCategory).thenComparing((o1, o2) -> {

**if** ( o1.getName().length() == o2.getName().length())

**return** 0;

**else** **if** ( o1.getName().length() > o2.getName().length())

**return** 1;

**else** **return** -1;

}))

.collect(Collectors.*toList*()));

Another Example:

Comparator<Employee> compareByName =

Comparator.comparing(Employee::getFirstName)

.thenComparing(Employee::getLastName);

* fds

1. Jfdsd
2. Jdfs
3. Fdsfd
4. Fdsfd

Collections and Algorithms:

1. Map: HashMap, TreeMap, LinkedListMap

* Collection views of the map: keyset, values, entrySet;
* Operations: put(key, value), get(key).
* putAll(anotherMap); containsAll(); removeAll(); retainAll()
* // ToDo: map attributes-value pairs, two sets: requiredAttr, permissibleAttr. permissibleAttr > requiredAttr

1. List: ArrayList

* Positional Access: get, set and remove; indexOf, lastIndexOf
* Find an element from the list: contains(), indexOf(), need to override the hashCode() and equals() code. For-Loop; stream filter/findAny.
* Remove: Iterator<String> iter = names.iterator();

While ( iter.hasNext() ) {

String name = iter.next();

If ( name.startWith(“A”)

Iter.remove();

}

* Remove: for collections, use removeIf(lamba)
* Sublist(fromIndex, toIndex)
* List Algorithms: binarySearch, sort and reverse use stream.

1. Dfsda
2. fds