Streams

Java module 2

Intermediate Operations

filter()

filter(): Returns a stream consisting of elements that match the given predicate.

```
List<String> words = Arrays.asList("apple", "banana", "pear", "peach");
words.stream().filter(s -> s.startsWith("p")).forEach(System.out::println);
/* output:
* pear
* peach
*/
```

map()

map(): Returns a stream consisting of the results of applying the given function to the elements of this stream.

```
List<String> words = Arrays.asList("apple", "banana", "pear", "peach");
words.stream().map(s -> s.toUpperCase()).forEach(System.out::println);
/* output:
* APPLE
* BANANA
* PEAR
* PEACH
*/
// with method reference
words.stream().map(String::toUpperCase).forEach(System.out::println);
```

flatMap()

flatMap(): Returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided function.

```
String[][] list2d = {
        {"1", "2", "3", "4", "5"},
        {"6", "7", "8", "9", "10"},
        {"11", "12", "13", "14", "15"}
};

String[] flattened = Arrays.stream(list2d)
        .flatMap(Arrays::stream)
```

```
        .toArray(String[]::new);

System.out.println(Arrays.toString(flattened));
/* output:
* [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
*/

int[][] list2d = {
        {1, 2, 3, 4, 5},
        {6, 7, 8, 9, 10},
        {11, 12, 13, 14, 15}
};

int[] list1d = Arrays.stream(list2d)
        .flatMapToInt(Arrays::stream) // Flatten the 2D array into an
IntStream
        .toArray(); // Collect the stream into a 1D array

System.out.println(Arrays.toString(list1d)); // Convert the array to a
String
/* output:
* [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
*/
```

distinct()

distinct(): Returns a stream with distinct elements (according to Object.equals(Object))

sorted()

sorted(): Returns a stream sorted according to the natural order of the elements.

```
List<String> list = Arrays.asList("9", "A", "Z", "1", "B", "Y", "4", "a",
"c");
List<String> sortedList =
list.stream().sorted().collect(Collectors.toList());
sortedList.forEach(System.out::println);
```

peek()

peek(): Returns a stream consisting of the elements of this stream, additionally performing the provided action on each element as elements are consumed from the resulting stream.

limit()

limit(long): Truncates the stream to be no longer than the given size.

skip()

skip(long): Skips the first N elements of the stream.

Terminal Operations

forEach()

forEach(): Performs an action for each element of this stream.

forEachOrdered()

forEachOrdered(): Performs an action for each element of this stream, guaranteeing that each element is processed in the encounter order for streams that have a defined encounter order.

toArray()

toArray(): Returns an array containing the elements of this stream.

min(), max()

min(), max(): Returns the minimum or maximum element of this stream according to the provided comparator.

count()

count(): Returns the count of elements in this stream.

anyMatch()

anyMatch(): Returns true if any elements of this stream match the provided predicate.

allMatch()

allMatch(): Returns true if all elements of this stream match the provided predicate.

noneMatch()

noneMatch(): Returns true if no elements of this stream match the provided predicate.

findFirst()

findFirst(): Returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty.

findAny()

findAny(): Returns an Optional describing some element of the stream, or an empty Optional if the stream is empty.