Day 2 functional programming

Monday, August 28, 2017 11:11 AM

Functional programming - The process of building software composed purely of functions.

Lambda - anonymous function.
Functional interface - Single Abstract Method (SAM) interface with only one abstract method declaration.

Steams - Classes to support functional-style operations on streams of elements, such as map-reduce transformations on collections.

- No storage. A stream is not a data structure that stores elements; instead, it conveys elements from a source such as a data structure, an array, a generator function, or an I/O channel, through a pipeline of computational operations.
- Functional in nature. An operation on a stream produces a result, but does not modify its source. For example, filtering a Stream obtained from a collection produces a new Stream without the filtered elements, rather than removing elements from the source collection.
- Laziness-seeking. Many stream operations, such as filtering, mapping, or duplicate removal, can be implemented lazily, exposing opportunities for optimization. For example, "find the first String with three consecutive vowels" need not examine all the input strings. Stream operations are divided into intermediate (Stream-producing) operations and terminal (value- or side-effect-producing) operations. Intermediate operations are always lazy.
- Possibly unbounded. While collections have a finite size, streams need not. Short-circuiting operations such as limit(n) or findFirst() can allow computations on infinite streams to complete in finite time.
- Consumable. The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be generated to revisit the same elements of the source.

From <https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html>

- Sequential - operates in one thread, similar to a normal for loop
- Parallel - operates in multiple threads, to increase efficiency.