Java 8 Features

Java8 Features

- Lambda Expressions
- Streams
- Default methods
- Optional
- LocalDate and LocalTime
- Functional Programming

Lambda Expressions

- Concise representation of anonymous function which can be passed around
- It enables developers to pass code in concise way

Syntax: (lambda parameters) -> (lambda body)

- It has
  - list of params
  - body
  - return type.(optional)

Why Lambda Expressions

- Easy way to pass a concise behavior.
- It allows developer to pass methos/behavior to method.
- This enables functional style programming , lambdas are introduced.

Functional Programming

- Functional Programming used to
  - Define anonymous functions
  - Assign function to a variable
  - Pass function as a parameter
  - Return function as a return value

- FP enables developer to write more readable , maintainable , clean & concise code.
- FP enable parallel processing
- FP uses declarative style of programming (just focus on what's to be done)

Lambda Expressions

- **Lambda examples**
  - (int a, int b) -> { return a + b; }
  - (a, b) -> a * b
  - (Employee e1, Employee e2) ->   e1.getEmpNo().compareTo(e2.getEmpNo())
  - () -> System.out.println(“Lambda")
  - (str) -> System.out.println(str)
  - () -> 100
  - () -> new Employee()

Where to use Lambda

- lambda expressions targets to Functional Interface (SAM) reference.
- Functional Interface is interface with only one abstract method (@FunctionalInterface)
- The signature of abstract method is called as **function descriptor**

  Ex.Function,Runnable,Comparable,Comparator, Iterable, Consumer,Predicate,Supplier, etc

Demo: Using Lambda for Collections.sort(), forEach() , removeIf()

Stream

- A sequence of elements from a source that supports data processing operations.
- Stream does not store elements. It simply conveys elements from a source such as a data structure, an array, or an I/O channel, through a pipeline of computational operations.
- Data processing operations - Supports common operations from functional programming languages.

Ex. filter, map, match, sort etc

Stream

- Stream is functional in nature. Operations performed on a stream does not modify it's 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.
- Stream is lazy and evaluates code only when required.
- 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.

Steps to use Stream

- 1. Convert to stream
- 2. Perform operation
- 3. Collect from Stream to collection