**Java 7/ 8 /11 features**

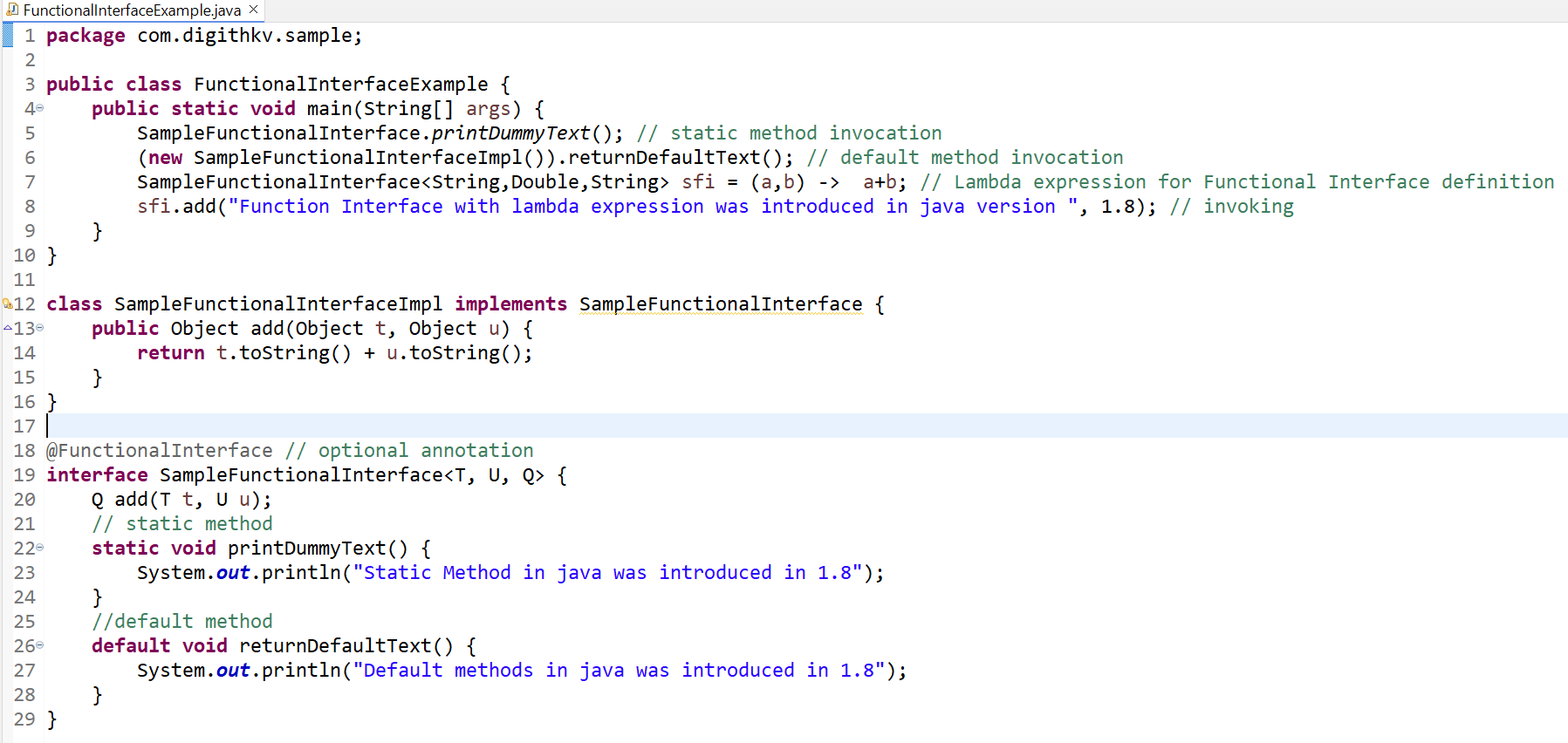
Java 8 features:

* Functional (SAM) Interface, static and default methods, Lambda expression. Interfaces that came with java 8 - Function, Predicate, Consumer, Supplier
* Streams & Parallel Streams
* CompletableFuture
* Local Date API
* Nashorn Engine

Functional Interface (SAM):

Interfaces with only **S**ingle **A**bstract **M**ethod - annotated with @FunctionalInterface (optional). static and default methods in interface were also introduced. Below given code contains:

* SAM Interface (SampleFunctionalInterface) with abstract method add(T t, U u) method.
* Lambda expressions are used to define functional interface (line 7)
* static methods -> avoid utility methods defined in classes (can be called directly without an instance)
* default methods -> avoid repetition in implementing class

eg:

before java 8 :

Runnable(with run()), Callable(with call()), Comparable(with compareTo(T o))

in java 8:

Function (with apply(T t)), Consumer(with accept(T t)), Supplier (with get()), Predicate(with test())

**Lambda Expression** -> to define functional interface and make code concise (part of functional programming

Streams & Parallel Stream:

To process collection of elements in a functional style of programming we have **interface** Stream. Intermediate operations over streams **are executed only when terminal operator is there**. Parallel Stream allows you to operate on multiple cores of your machine. Advantages of Streams include:

* No Storage
* Pipeline of Functions
* Laziness
* Can be infinite
* Can be parallelized

How to create a stream:

1. <Collections>.stream() / parallelStream()
2. Stream.of(T... values) // varags
3. Insteam.range(inclusive, exclusive) / Instream.rangeClosed(inclusive, inclusive)
4. Stream.iterate(initial, UnaryOperator) -> used in Fibonacci series in java 8

There are two kinds of operations in Streams:

1. Intermediate (operates and gives another stream)-> map(), flatMap(), filter(), sorted(), distinct(), mapToLong/Int/Double/Object(), flatMapToLong/Int/Double/Object(), sorted(Comparator comparator), limit(long number), skip(long number)
2. Terminal Operations -> collect(Collector collector), reduce(BinaryOperator accumulator), findAny(),findFirst(), toList(), toArray(), distinct(), count(), forEach(Consumer consumer)

Once stream is operated / closed if attempted another stream operation you will get “*stream has already been operated upon or closed*”