Consumers are functional interfaces that take in a parameter and do not produce anything.

Below are some of the functional interfaces which can be categorized as Consumers.

Consumer <t></t>	Represents an operation that accepts a single (reference type) input argument and returns no result	<pre>void accept(T t)</pre>
DoubleConsumer	Accepts a single double-value argument and returns no result	<pre>void accept(double value)</pre>
IntConsumer	Accepts a single int-value argument and returns no result	<pre>void accept(int value)</pre>
LongConsumer	Accepts a single long-value argument and returns no result	<pre>void accept(long value)</pre>
BiConsumer <t, u=""></t,>	Represents an operation that accepts two (reference type) input arguments and returns no result	void accept(T t, U u)
ObjDoubleConsumer <t></t>	Accepts an object-value and a double-value argument, and returns no result	<pre>void accept(T t, double     value)</pre>
<pre>ObjIntConsumer<t></t></pre>	Accepts an object-value and an int-value argument, and returns no result	<pre>void accept(T t, int value)</pre>
	Accepts an object-value and a	

## Consumer<T>

ObjLongConsumer<T>

This interface takes a parameter of type T and does not return anything.

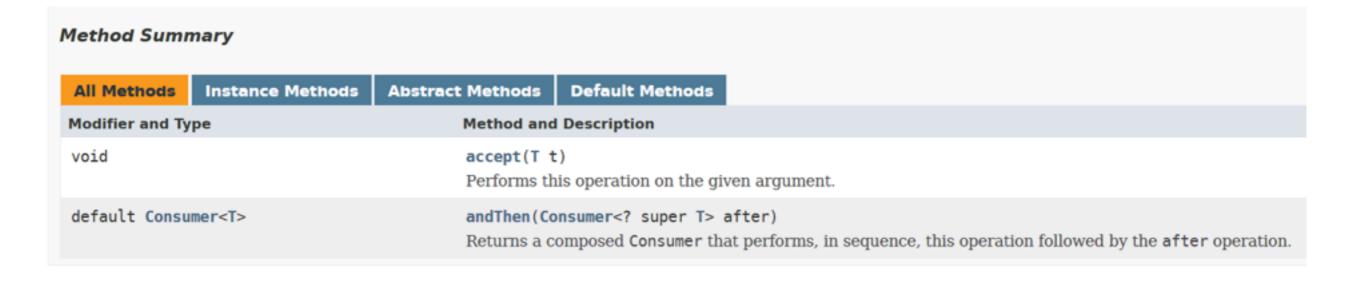
A consumer can be used in all contexts where an object needs to be consumed, i.e. taken as input, and some operation is performed on the object without returning any result.

long-value argument, and re-

turns no result

void accept(T t, long value)

Below is the list of methods in the Consumer interface. Consumer<T> has an abstract method accept() and a default method called andThen(), which is used for chaining.



import java.util.function.Consumer;

syntax of this method.

In the below example, we will crate a **Consumer** which prints a value.

```
public class ConsumerDemo {
    public static void main(String[] args) {
            Consumer<String> stringConsumer = s -> System.out.println(s);
            stringConsumer.accept("Hello World.");
            Consumer<Integer> intConsumer = i -> System.out.println("Integer value = " + i);
10
            intConsumer.accept(5);
11
12
13 }
                                                                                                         Run
                                                                                                 Reset
```

Consumer<T> andThen(Consumer<? super T> after)

The andThen() method, which is a default method in the Consumer interface is used for chaining. Here is the

```
The andThen() method returns a composed Consumer that performs this operation followed by the after
operation. In the below example, we will create two consumers, and we will chain them together using the
```

andThen() method. import java.util.function.Consumer; C

```
public class ConsumerDemo {
          public static void main(String[] args) {
   5
             Consumer<String> consumer1 = (arg) -> System.out.println(arg + "My name is Jane.");
             Consumer<String> consumer2 = (arg) -> System.out.println(arg + "I am from Canada.");
  10
              consumer1.andThen(consumer2).accept("Hello. ");
  11
  12
  14 }
  Run
                                                                                      Save
                                                                                              Reset
BiConsumer<T,U>
```

This interface takes two parameters and returns nothing.

• T - the type of the first argument to the operation

• U - the type of the second argument to the operation.

This interface has the same methods as present in the Consumer<T> interface.

```
import java.util.function.BiConsumer;
    public class BiConsumerDemo {
        public static void main(String[] args) {
         BiConsumer<String, String> greet = (s1, s2) -> System.out.println(s1 + s2);
         greet.accept("Hello", "World");
10
11
Run
                                                                                                  Reset
```