Consumer Functional Interface

1)Predicate< T > - takes an input perform conditional check and always return boolean value.

Input Type T.

2)Function< T , R > - takes an input ->preform some operation ->produce some result [the result which is need not to l

Input Type T , Return Type R

3)Consumer < T > - takes an input ->perform some operation -> it won't return any thing.

Input Type T

Syntax:

```
Interface Consumer< T >
{
public void accept( T t );
}
```

Ex: Consumer Functional Interface

```
package com.consumer;

import java.util.function.Consumer;
import java.util.function.Function;
import java.util.function.Predicate;

class Employee {
String name;
double salary;

public Employee(String name, double salary) {
super();
this.name = name;
this.salary = salary;
}

@Override
public String toString() {
return "Employee [name=" + name + ", salary=" + salary + "]";
}

}

public class ConsumerFI {
public static void main(String[] args) {
Function<Employee, String> f = e -> {
double salary = e.salary;
String position = "";
if (salary > 90000)
position = "Manager";
else if (salary > 80000)
position = "Team Lead";
else if (salary > 70000)
position = "Sr Software";
else if (salary > 50000)
position = "Jr Software";
else if (salary > 40000)
position = "Support";
else if (salary > 30000)
position = "others";

return position;

};

Predicate<Employee> p = e -> e.salary > 40000;

Consumer<Employee> c = e -> {
System.out.println("Employee name :" + e.name);
System.out.println("Employee salary :" + e.salary);
System.out.println("Employee position :" + f.apply(e));
System.out.println();
};

Employee[] e = { new Employee("anand", 80000), new Employee("kumar", 70000), new Employee("anji", 40000),
new Employee("bharath", 72000), new Employee("laddu", 35000), new Employee("sagar", 45000),
new Employee("lokesh", 23000), new Employee("nagarjuna", 29000), new Employee("ravi", 20000),
new Employee("mukesh", 15000), };

for (Employee e1 : e) {
   if (p.test(e1)) {     // if the condition is true then only accept the consumer for
             c.accept(e1);              the Employee name,salary and position.
                 }
                     }
}

}


output :

Employee name :anand
Employee salary :80000.0
Employee position :Sr Software

Employee name :kumar
Employee salary :70000.0
Employee position :Jr Software
```

Employee name :bharath
Employee salary :72000.0
Employee position :Sr Software

Employee name :sagar
Employee salary :45000.0
Employee position :Support

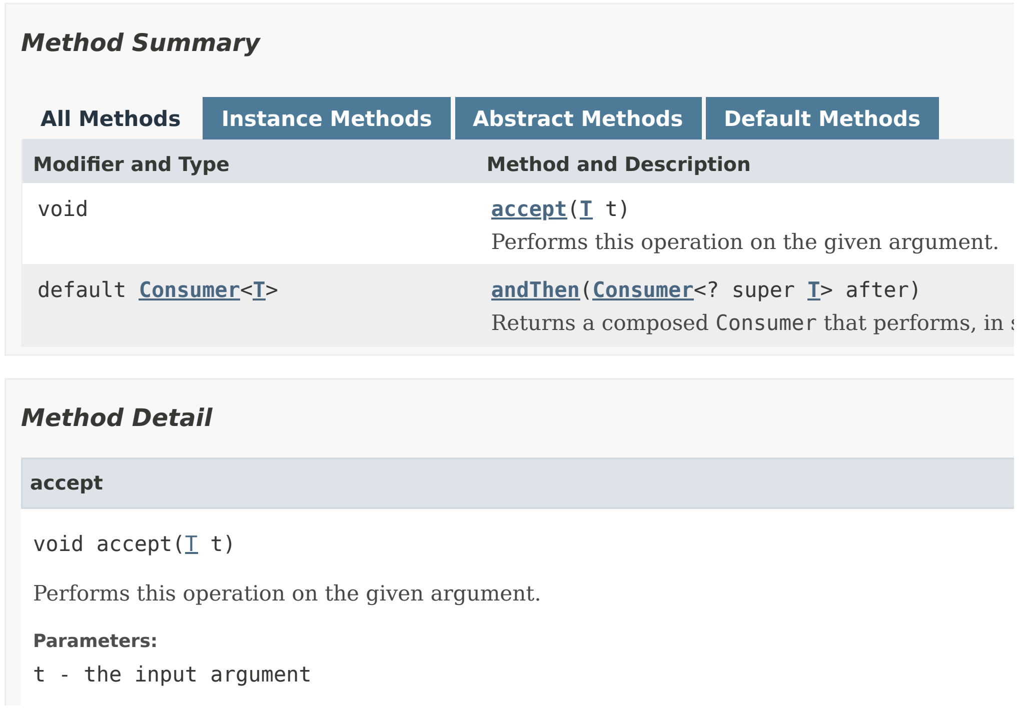
Consumer Chaining

Consumer < T > - takes an input ->perform some operation -> it won't return any thing.

Input Type T

Syntax:

```
Interface Consumer <T>
{
public void accept(T t);
}
```



Method Summary

All Methods | Instance Methods | Abstract Methods | Default Methods

Modifier and Type	Method and Description
`void`	`accept(T t)` Performs this operation on the given argument.
`default Consumer<T>`	`andThen(Consumer<? super T> after)` Returns a composed `Consumer` that performs, in

Method Detail

accept

```
void accept(T t)
```

Performs this operation on the given argument.

Parameters:

`t - the input argument`

andThen

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

Returns a composed `Consumer` that performs, in sequence, this operation followed by the `af`
relayed to the caller of the composed operation. If performing this operation throws an exce

Parameters:

`after - the operation to perform after this operation`

Returns:

`a composed Consumer that performs in sequence this operation followed by the a`

Throws:

`NullPointerException - if after is null`

Ex:

```
package com.consumer;

import java.util.function.Consumer;

class Movie {
String name;

public Movie(String name) {
super();
this.name = name;
}

}

public class ConsumerExample {
public static void main(String[] args) {
Consumer<Movie> c = m -> System.out.println("movie name is :" + m.name);
Consumer<Movie> c1 = m -> System.out.println("movie realesed :" + m.name);
Consumer<Movie> c2 = m -> System.out.println("movie flop " + m.name);
Consumer<Movie> c3 = m -> System.out.println("Audience not intersted to see the movie : " + m.name);
Consumer<Movie> cc = c1.andThen(c2).andThen(c3);

Movie m = new Movie(" 1- Nenokkadine");
cc.accept(m);
}
}
```

output:

```
movie realesed : 1- Nenokkadine
movie flop  1- Nenokkadine
Audience not intersted to see the movie :  1- Nenokkadine
```