

Interfaz	Descripción
<code>BinaryOperator&lt;T&gt;</code>	Contiene el método <code>apply</code> que recibe dos argumentos, realiza una operación sobre ellos (como un cálculo) y devuelve un valor de tipo <code>T</code> .
<code>Consumer&lt;T&gt;</code>	Contiene el método <code>accept</code> que recibe un argumento <code>T</code> y devuelve <code>void</code> . Realiza una tarea con su argumento <code>T</code> , como mostrar el objeto en pantalla, invocar a un método del objeto, etc.
<code>Function&lt;T,R&gt;</code>	Contiene el método <code>apply</code> que recibe un argumento <code>T</code> y devuelve el resultado de ese método.
<code>Predicate&lt;T&gt;</code>	Contiene el método <code>test</code> que recibe un argumento <code>T</code> y devuelve un <code>boolean</code> .
<code>Supplier&lt;T&gt;</code>	Contiene el método <code>get</code> que no recibe argumentos y produce un valor de tipo <code>T</code> . A menudo se usa para crear un objeto colección en donde se colocan los resultados de la operación de un flujo.
<code>UnaryOperator&lt;T&gt;</code>	Contiene el método <code>get</code> que no recibe argumentos y devuelve un valor de tipo <code>T</code> .

**Fig. 1** Las seis interfaces funcionales genéricas básicas en el paquete **java.util.function**.

Interfaz	Descripción	interfaces funcionales genéricas básicas en el paquete <code>java.util.function</code>
<b>Consumer&lt;T&gt;</b>	Contiene el método <code>accept</code> que recibe un argumento <code>T</code> y devuelve <code>void</code> . Realiza una tarea con su argumento <code>T</code> , como mostrar el objeto en pantalla, invocar a un método del objeto, etc.	

```

1  .../
25 package java.util.function;
26
27 import java.util.Objects;
28
29 Represents an operation that accepts a single int-valued argument and returns no result. This is the primitive
30 type specialization of Consumer for int. Unlike most other functional interfaces, IntConsumer is expected
31 to operate via side-effects.
32 This is a functional interface whose functional method is accept(int).
33 Since: 1.8
34 See Also: Consumer
41 @FunctionalInterface
42 public interface IntConsumer {
43
44     Performs this operation on the given argument.
45     Params: value – the input argument
46
47     void accept(int value);
48
49     Returns a composed IntConsumer that performs, in sequence, this operation followed by the after
50     operation. If performing either operation throws an exception, it is relayed to the caller of the
51     composed operation. If performing this operation throws an exception, the after operation will not be
52     performed.
53     Params: after – the operation to perform after this operation
54     Returns: a composed IntConsumer that performs in sequence this operation followed by the after
55     operation
56
57     @Contract(pure = true) @NotNull
58     default IntConsumer andThen( @NotNull IntConsumer after) {
59         Objects.requireNonNull(after);
60         return (int t) -> { accept(t); after.accept(t); };
61     }
62 }

```

**Fig. 2** La interfaz funcional **consumer**

```

1  import java.util.stream.IntStream;
2
3  public class UsingIntStream {
4      public static void main(String[] args) {
5          int fuente[] = { 3, 1, 4, 1, 5, 9 };
6
7          IntStream.of(fuente).forEach(i -> System.out.println(" " + i));
8      }
9  }

```

**Operación Terminal `forEach` de Stream:** Realiza un procesamiento sobre cada elemento en un flujo (por ejemplo, mostrar cada elemento en pantalla).

java.util.function  
@FunctionalInterface  
public interface IntConsumer

Represents an operation that accepts a single int-valued argument and returns no result. This is the primitive type specialization of Consumer for int. Unlike most other functional interfaces, IntConsumer is expected to operate via side-effects. This is a functional interface whose functional method is accept(int).

Since: 1.8  
See Also: Consumer  
< 1.8 >

**Fig. 3** Ejemplo de uso de la interfaz **consumer** mediante **lambda** en java

## Operaciones intermedias con flujos

<code>filter</code>	Produce un flujo que contiene sólo los elementos que satisfacen una condición.
<code>distinct</code>	Produce un flujo que contiene sólo los elementos únicos.
<code>limit</code>	Produce un flujo con el número especificado de elementos a partir del inicio del flujo original.
<code>map</code>	Produce un flujo en el que cada elemento del flujo original está asociado a un nuevo valor (posiblemente de un tipo distinto); por ejemplo, asociar valores numéricos a los cuadrados de los valores numéricos. El nuevo flujo tiene el mismo número de elementos que el flujo original.
<code>sorted</code>	Produce un flujo en el que los elementos están ordenados. El nuevo flujo tiene el mismo número de elementos que el flujo original.

**Fig. 3** Operaciones intermedias comunes con Stream.



Operaciones terminales con Stream	
forEach	Realiza un procesamiento sobre cada elemento en un flujo (por ejemplo, mostrar cada elemento en pantalla).
<b>Operaciones de reducción:</b> toman todos los valores en el flujo y devuelven un solo valor	
average	Calcula el <i>promedio</i> de los elementos en un flujo numérico.
count	Devuelve el <i>número de elementos</i> en el flujo.
max	Localiza el valor <i>más grande</i> en un flujo numérico.
min	Localiza el valor <i>más pequeño</i> en un flujo numérico.
reduce	Reduce los elementos de una colección a un <i>solo valor</i> mediante el uso de una función de acumulación asociativa (por ejemplo, una lambda que suma dos elementos).
<b>Operaciones de reducción mutables:</b> crean un contenedor (como una colección o un <i>StringBuilder</i> )	
collect	Crea una <i>nueva colección</i> de elementos que contienen los resultados de las operaciones anteriores del flujo.
toArray	Crea un <i>arreglo</i> que contiene los resultados de las operaciones anteriores del flujo.
<b>Operaciones de búsqueda</b>	
findFirst	Encuentra el <i>primer</i> elemento del flujo con base en las operaciones intermedias; termina inmediatamente el procesamiento de la canalización de flujo una vez que se encuentra dicho elemento.
findAny	Encuentra <i>cualquier</i> elemento de flujo con base en las operaciones intermedias anteriores; termina de inmediato el procesamiento de la canalización de flujo una vez que se encuentra dicho elemento.
anyMatch	Determina si <i>alguno</i> de los elementos del flujo coincide con una condición especificada; cuando un elemento coincide, ésta termina de inmediato el procesamiento de la canalización de flujo.
allMatch	Determina si <i>todos</i> los elementos en el flujo coinciden con una condición especificada.

**Fig. 4** Operaciones terminales comunes con Stream.