Interfaz	Descripción	
BinaryOperator <t></t>	Contiene el método app1y que recibe dos argumentos, realiza una operación sobre ellos (como un cálculo) v devuelve un valor de tipo T.	
Consumer <t></t>	Contiene el método accept que recibe un argumento T y devuelve void. Realiza una tarea con su argumento T, como mostrar el objeto en pantalla, invocar a un método del objeto, etc.	
Function <t,r></t,r>	Contiene el método app1y que recibe un argumento T y devuelve el resultado de ese método.	
Predicate <t></t>	Contiene el método test que recibe un argumento T y devuelve un boolean.	
Supplier <t></t>	Contiene el método get que no recibe argumentos y produce un valor de tipo T. A menudo se usa para crear un obieto colección en donde se colocan los resultados de la operación de un flujo.	
UnaryOperator <t></t>	Contiene el método get que no recibe argumentos y devuelve un valor de tipo T.	

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

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

```
/.../
         package java.util.function;
         import java.util.Objects;
           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).
           See Also: Consumer
         @FunctionalInterface
        public interface IntConsumer {
                Performs this operation on the given argument.
                Params: value - the input argument
               void accept(int value);
                Returns a composed IntConsumer that performs, in sequence, this operation followed by the after
                 operation. If performing either operation throws an exception, it is relayed to the caller of the
                composed operation. If performing this operation throws an exception, the after operation will not be
                performed.
                 Params: after – the operation to perform after this operation
                 Returns: a composed IntConsumer that performs in sequence this operation followed by the after
                       operation
               @Contract(pure = true) @NotNull
63 @
              default IntConsumer andThen(@NotNull IntConsumer after)
                    Objects.requireNonNull(after);
                    return (int t) -> { accept(t); after.accept(t); };
```

Fig. 2 La interfaz funcional consumer

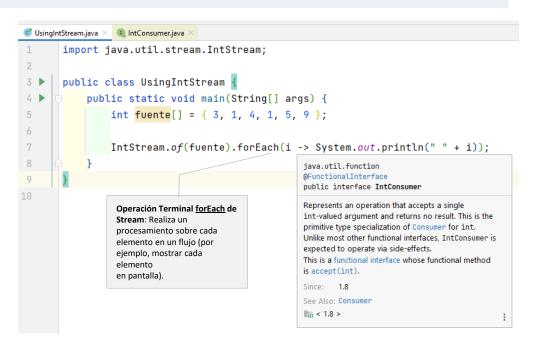


Fig. 3 Ejemplo de uso de la interfaz consumer mediante lambda en java

Operaciones intermedias con flujos

filter	Produce un flujo que contiene sólo los elementos que satisfacen una condición.
distinct	Produce un flujo que contiene sólo los elementos únicos.
limit	Produce un flujo con el número especificado de elementos a partir del inicio del flujo original.
map	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.
sorted	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).

Operaciones de reducción: toman todos los valores en el flujo y devuelven un solo valor

average Calcula el *promedio* de los elementos en un flujo numérico.

count Devuelve el número de elementos en el flujo.

max Localiza el valor *más grande* en un flujo numérico.

min Localiza el valor *más pequeño* en un flujo numérico.

reduce Reduce los elementos de una colección a un *solo valor* mediante el uso de una función de

acumulación asociativa (por ejemplo, una lambda que suma dos elementos).

Operaciones de reducción mutables: crean un contenedor (como una colección o un StringBuilder)

collect Crea una *nueva colección* de elementos que contienen los resultados de las operaciones anteriores

del flujo.

toArray Crea un *arreglo* que contiene los resultados de las operaciones anteriores del flujo.

Operaciones de búsqueda

findFirst Encuentra el *primer* 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 *cualquier* 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 *alguno* 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 *todos* los elementos en el flujo coinciden con una condición especificada.

Fig. 4 Operaciones terminales comunes con Stream.