


Interfaces `java.util.function`



Introducción

- Conjunto de interfaces funcionales incorporadas en Java SE 8 dentro del paquete `java.util.function`.
 - Utilizadas como argumentos en métodos que manipulan datos para el establecimiento de criterios de filtrado, operación sobre elementos, transformación, etc.
 - Implementadas habitualmente mediante expresiones lambda
- 

Interfaz Predicate<T>

- Dispone del método abstracto *test*, que a partir de un objeto realiza una comprobación y devuelve un boolean:

```
boolean test(T t)
```

- Utilizada para la definición de criterios de filtrado, por ejemplo, método *removeIf()* de Collection:

```
//elimina los elementos que cumplen con la  
//condición del filtro  
boolean removeIf(Predicate<? super T> filtro)
```



```
//eliminación de numeros pares de una colección:  
lista.removeIf(n->n%2==0);
```

- Variante BiPredicate<T,U> con dos parámetros:

```
boolean test(T t, U u)
```

Interfaz Function<T,R>

➤ Método abstracto *apply*, que a partir de un objeto realiza una operación y devuelve un resultado:

`R apply(T t)`

➤ Utilizado en operaciones de transformación de datos. Por ejemplo, método `map()` de `Stream`:

```
//Transforma cada elemento del Stream de tipo T  
//en otro de tipo R  
Stream <R> map(Function<? super T,? extends R> mapper)
```



```
//genera un nuevo Stream con la longitud  
//de las cadenas del Stream original  
st.map(cad->cad.length());
```

➤ Variante `BiFunction<T,U,R>` con dos parámetros:

`R apply(T t, U u)`

Interfaz Consumer<T>

- Dispone del método abstracto *accept*, que realiza algún tipo de procesamiento con el objeto recibido:

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

- Utilizada en operaciones de procesamiento de datos. Por ejemplo, método *forEach()* de listas y conjuntos:

```
//aplica las operaciones del método a cada  
//elemento de la lista  
void forEach(Consumer<? super T> action)
```



```
//imprime el contenido de la lista:  
lista.forEach(n->System.out.println(n));
```

- Variante BiConsumer<T,U> con dos parámetros:

```
void accept(T t, U u)
```

Interfaz Supplier<T>

- Dispone del método abstracto *get*, que no recibe ningún parámetro y devuelve como resultado un objeto:

`T get()`

- Utilizada para implementar operaciones de extracción de datos. Por ejemplo, método *generate()* de Stream:

```
//genera una secuencia infinita de elementos  
//proporcionados por llamadas a get()  
static Stream<T> generate(Supplier<T> s)
```



```
//devuelve un Stream de números aleatorios  
//entre 1 y 500:  
Stream.generate(()->(int)(Math.random()*500+1));
```

Revisión conceptos



Indica que interfaz funcional de `java.util.function` implementa cada una de las siguientes expresiones lambda:

- a. `()->Math.random()*5`
- b. `(n)->n.length()`
- c. `a->a>10`
- d. `(x,y)->System.out.println(x+":"+y)`

Respuesta

- a. Supplier
- b. Function
- c. Predicate
- d. BiConsumer

Interfaz UnaryOperator<T>

- Subinterfaz de Function donde el tipo de entrada coincide con el de devolución:

`T apply(T t)`

- Al igual que Function, se emplea en contextos de transformación de datos. Ejemplo, método `replaceAll()` de `Collection`:

```
//reemplaza cada elemento de la colección por otro  
//resultante de aplicar la función  
void replaceAll(UnaryOperator<? super T> oper)
```



```
//sustituye cada elemento de la colección  
//por su cuadrado  
lista.replaceAll(n->n*n);
```

- Variante `BinaryOperator<T>` equivalente a `BiFunction<T,T,T>`

Interfaces para tipos primitivos

- **IntPredicate** ➡ `boolean test(int t)`
- **IntFunction<R>** ➡ `R apply(int t)`
- **IntConsumer** ➡ `void accept(int t)`
- **IntSupplier** ➡ `int getAsInt()`
- **IntUnaryOperator** ➡ `int applyAsInt(int t)`
- **Versiones también para long y double**

Revisión conceptos



Dada la siguiente lista, escribir un bloque de código que muestre solamente los números pares ordenados de mayor a menor:

```
List<Integer> nums=List.of(10,4,21,3,17,8,20,11);
```

Respuesta

```
List<Integer> nums=List.of(10,4,21,3,17,8,20,11);  
List<Integer> datos=new ArrayList<>(nums);  
datos.removeIf(n->n%2!=0);  
datos.sort((a,b)->b-a);  
//o de esta otra forma  
//datos.sort(Comparator.reverseOrder());  
datos.forEach(n->System.out.println(n));
```