USO DE INTERFACES COMO PREDICATE, CONSUMER, FUNCTION Y SUPPLIER

1.

PREDICATE

PREDICATE

- Método ***boolean test(T t)***.
- Comprueba si se cumple o no una determinada condición.
- Muy utilizado con expresiones lambda para filtrar

```
listaPersonas
   .stream()
   .filter((p) -> p.getEdad() >= 351)
   .forEach(System.out::println);
```

PREDICATE

- Métodos útiles para constuir predicados complejos (*and*, *or*, *negate*…)

```
Predicate<Persona> edad = (p) -> p.getEdad() >= 35l;
Predicate<Persona> nombre = (p) ->
                          p.getApellidos().contains("e");
Predicate<Persona> complejo = edad.or(nombre);
```

2.

CONSUMER

CONSUMER

- Método ***void accept(T t)***.
- Sirve para consumir objetos.
- El ejemplo más claro es imprimir
- Muy utilizado con expresiones lambda para imprimir.

```
listaPersonas
   .stream()
   .filter((p) -> p.getEdad() >= 35l)
   .forEach(System.out::println);
```

CONSUMER

- Adicionalmente, tiene el método andThen, que permite componer *consumidores*, y encadenar así una secuencia de operaciones.

```
Consumer<Integer> consumer = i -> System.out.println(i);
Consumer<Integer> consumerWithAndThen =
                  consumer.andThen(i -> System.out.println("
                                       (hemos imprimido el " + i + ")"));
```

3.

FUNCTION

FUNCTION

- Método ***R apply(T t)***.
- Sirve para aplicar una transformación sobre un objeto.
- El ejemplo más claro es el mapeo de un objeto en otro.
- Muy utilizado con expresiones lambda para mapear.

```
lista
   .stream()
   .map((p) -> p.getNombre())
   .forEach(System.out::println);
```

FUNCTION

Adicionalmente, tiene otros métodos:

- *andThen*, que permite componer *funciones*.
- *compose*, que compone dos funciones, a la inversa de la anterior.
- *identity*, una función que siempre devuelve el argumento que recibe

4.

SUPPLIER

SUPPLIER

- Método ***T get()***.
- No recibe ningún parámetro
- Sirve para obtener objetos.
- Su uso está menos extendido que las anteriores.
- Su sintaxis como expresión lambda sería

```
() -> something
() -> { return something; }
```

SUPPLIER

Tiene interfaces especializados para tipos básicos

- *IntSupplier*
- *LongSupplier*
- *DoubleSupplier*
- *BooleanSupplier*