PROGRAMACIÓN FUNCIONAL EN JAVA v.1.0

CARACTERÍSTICAS PRINCIPALES DE LA P. FUNCIONAL

- **Transparencia referencial**: la salida de una función debe depender sólo de sus argumentos y debe ser determinista.
- **Inmutabilidad de los datos**: los datos deben ser inmutables para evitar posibles efectos colaterales.
- **Composición de funciones**: la salida de una función se puede tomar como entrada para la siguiente.
- **Funciones de primer orden**: funciones que permiten tener otras funciones como parámetros, a modo de callbacks.

INTERFAZ FUNCIONAL / CLASES ANÓNIMAS / LAMBDAS

Una IF es una interfaz (todos los métodos abstractos) con un único método y este es abstracto.
Creamos IF (IFMensaje) con un método abstracto "imprimir" que no hace nada (es abstracto). Tenemos tres alternativas:

```
// CASO A:  SIN CLASE ANÓNIMA, CON IMPLEMENTS
//  Definimos la IF con implements (MensajeImplementado)
//  Creamos una instancia de esa subclase
//  Llamamos a "imprimir": imprime UNA vez el texto
```

[MensajeImplementado.java]

```
public class MensajeImplementado implements IFMensaje {
   //@Override
   public String imprime(String t) {
      return t;  } }
```

[Main.java]

```
MensajeImplementado m1 = new MensajeImplementado();
```

```
// CASO B:  CON CLASE ANÓNIMA
// Creamos una instancia de esa subclase
// Al crear la instancia definimos el método con @Override
// Llamamos al método "imprimir": imprime DOS veces el texto
```

[Main.java]

```
 IFMensaje m2 = new IFMensaje() {
        //@Override
        public String imprime(String t) {
           return t + t;         }     };
```

```
// CASO C:  CON FUNCIÓN LAMBDA
// Creamos una instancia de esa subclase
// Al crear la instancia definimos el método con UNA LAMBDA
// Llamamos a "imprimir": imprime TRES veces el texto
```

[Main.java]

```
 IFMensaje m3 = (t) -> (t + t + t);
```

COMPARADORES CON LAMBDAS (JAVA) de menor a mayor

La IF **Comparator** pide implementar un método **compare**, que recibe dos datos del tipo a tratar (T), y devuelve un entero indicando si el primero es menor, mayor, o son iguales.

Solo si son enteros:

```
personas.sort((p1,p2)→ p1.getEdad(()-p2.getEdad());   // o bien
personas.sort((p1,p2)→ Integer.compare(p1.getEdad((), p2.getEdad());
```

*Si son reales (**debe devolver un entero**):*

```
ps.sort((p1,p2)→ Double/Float.compare(p1.getPrecio((),p2.getPrecio());
```

*Si son cadenas (**debe devolver un entero**):*

```
personas.sort((p1,p2)→ p1.getNombre(), p2.getNombre()); // o bien
personas.sort((p1,p2)→ compareTo(p1.getNombre(), p2.getNombre());
```

STREAMS INTERMEDIOS (DEVUELVEN STREAMS)

```
//  FILTRADO
Stream<Libro> librosFiltradosOrdenadosPorPrecioAscendente
 = libros.stream()
.filter(p -> p.getPrecio() > 1)
.sorted((p1, p2) -> Double.compare(p2.getPrecio(), p1.getPrecio()));

librosFiltradosOrdenadosPorPrecioAscendente
.forEach(p -> System.out.println(p.getTitulo()));

// MAPEADO Y FILTRADO
Stream<String> tituloDeStreamFiltradosOrdenadosMapeados
= libros.stream()
.filter(p -> p.getPrecio() > 1)
.sorted((p1, p2) -> Double.compare(p2.getPrecio(), p1.getPrecio()))
.map(p -> p.getTitulo());

tituloDeStreamFiltradosOrdenadosMapeados
.forEach(p -> System.out.println(p));

Stream<Double> precioDeStreamFiltradosOrdenadosMapeados
= libros.stream()
.filter(p -> p.getPrecio() > 1)
.sorted((p1, p2) -> Double.compare(p2.getPrecio(), p1.getPrecio()))
.map(p -> p.getPrecio());

precioDeStreamFiltradosOrdenadosMapeados
.forEach(p -> System.out.println(p));
```

STREAMS FINALES (DEVUELVEN OTRA COSA)

```
// STREAMS FINALES
libros.stream()
.filter(p -> p.getPrecio() > 1)
.sorted((p1, p2) -> Double.compare(p2.getPrecio(), p1.getPrecio()))
.map(p -> p.getTitulo())
.forEach(p -> System.out.println(p));
// COLLECT
List<Double> preciosAltosOrdenados
= libros.stream()
.filter(p -> p.getPrecio() > 1)
.sorted((p1, p2) -> Double.compare(p2.getPrecio(), p1.getPrecio()))
.map(p -> p.getPrecio())
.collect(Collectors.toList());
// COLLECT JOINING
String librosPreciosAltosOrdenadosSeparadosComas
= libros.stream()
.filter(p -> p.getPrecio() > 1)
.sorted((p1, p2) -> Double.compare(p2.getPrecio(), p1.getPrecio()))
.map(p -> p.getTitulo())
.collect(Collectors.joining(",", "Texto antes => ", " <= Texto después"));
// MapToInt con Average
double mediaPaginas
= libros.stream()
.mapToInt(p -> p.getPaginas()).average().getAsDouble();
```

Ejemplos completos:

https://github.com/sergiobadal/PRG_EJEMPLOS/tree/master/funcional

Autor:
Sergio Badal