

Mercado Salinas Fhernando

- *Máster Universitario en Sistemas Embebidos*
 - Mondragon Unibertsitatea, Basque Country, Spain
- *Ingeniería en Sistemas Computacionales*
 - Tecnológico de Estudios Superiores de Jocotitlán

Programación Lógica y Funcional

v 0.0, Advanced Level

¿Qué es la Programación Funcional?

La programación funcional
es un **paradigma de programación**
basado en el uso
de **funciones matemáticas**

¿Qué es una Expresión Lambda?

Una expresión lambda es esencialmente una función sin nombre, que tiene un cuerpo de función y puede o no recibir parámetros.

En Java, una expresión lambda siempre va a representar **el método abstracto de una interfaz funcional**.

La expresión lambda va a separar su lista de parámetros de su cuerpo de función por medio del operador flecha ->

Una expresión lambda luce así.

```
1 | (argumentos) ->  
2 | {  
3 |     //cuerpo de la expresión lambda  
4 | }
```

¿Qué es una Interfaz Funcional?

Antes de hablar de las interfaces funcionales, comentaré brevemente las novedades en las interfaces en general que la versión 8 de Java ha implementado.

La primera y mas impresionante característica es que ahora las interfaces permiten tener una implementación por defecto para sus métodos. Si un método de una interfaz tiene alguna implementación, es necesario declarar ese método como default. La idea es que todas las clases que implementen esta interfaz tengan cierta funcionalidad "por defecto" en los métodos de sus interfaces.

¿Qué es una Interfaz Funcional?

Una interfaz puede tener cualquier cantidad de métodos implementados siempre y cuando sean declarados como default.

```
1  public interface Iterable<T> {  
2      Iterator<T> iterator();  
3  
4      default void forEach(Consumer<? super T> action) {  
5          Objects.requireNonNull(action);  
6          for (T t : this) {  
7              action.accept(t);  
8          }  
9      }  
10 }
```


¿Qué es una Interfaz Funcional?

La otra característica es que las interfaces pueden tener métodos estáticos implementados.

```
1  import java.util.List;
2
3  public interface Producto {
4
5      public int getPrecio();
6
7      public static int importeTotal(List<Producto> listaProduct
8
9          return listaProductos.stream().mapToInt((p)->p.getPrec
10     }
11 }
```

En cuanto a los métodos abstractos, no hay cambios; las interfaces pueden tener cualquier cantidad de métodos abstractos "en sus filas".

¿Qué es una Interfaz Funcional?

Entonces, ¿Qué es una interfaz funcional?

Las interfaces funcionales se utilizan principalmente para permitir **el paso de funciones a métodos**. Esto es, que podemos pasar una implementación de una función como argumento de un método.

En el fondo, seguiremos pasando un objeto a dicho método, pero la idea de la interfaz funcional (y de la programación funcional en sí) es trabajar con funciones mas que con objetos. La interfaz funcional nos va a permitir *simular* que pasamos una función implementada como argumento de un método, aunque no es su único uso.

¿Qué es una Interfaz Funcional?

Para que una interfaz sea funcional debe cumplir con un solo requisito: ***Solo debe tener un método abstracto.***

La interfaz funcional puede tener varios métodos estáticos y default si quiere, pero solo un método abstracto.

Más adelante veremos que esta característica le permite acoplarse perfectamente con las expresiones lambda.

Es posible para nosotros crear nuestras propias interfaces funcionales, sin embargo, la versión 8 de Java ha incluido una gran cantidad de estas interfaces dentro del paquete

`java.util.function`

Sintaxis de una Expresión Lambda

```
3  
4 parametros -> cuerpo  
5
```

```
6 public interface Ficticia {  
7  
8     public void aceptar();  
9  
10 }  
11  
12 Ficticia f = new Ficticia(){  
13     @Override  
14     public void aceptar(){  
15         //cuerpo del método implementado aceptar  
16     }  
17 }  
18
```

Método Abstracto

Antes

Sintaxis de una Expresión Lambda

```
3
4 parametros -> cuerpo
5
6 public interface Ficticia {
7
8     public void aceptar();
9
10 }
11
12 Ficticia f = new Ficticia(){ Antes
13     @Override
14     public void aceptar(){
15         System.out.println("Hola mundo");
16     }
17 }
18
19 Ficticia f = () -> { Ahora System.out.println("Hola mundo"); }
```


Sintaxis de una Expresión Lambda

```
7
8     public void aceptar();
9
10 }
11
12 Ficticia f = new Ficticia(){
13     @Override
14     public void aceptar(){
15         System.out.println("Hola mundo");
16     }
17 }
18
19 Ficticia f = () -> { System.out.println("Hola mundo"); };
```

Punto y Coma

Sintaxis de una Expresión Lambda

```
2
3
4 parametros -> cuerpo
5
6 public interface Ficticia {
7
8     public void aceptar(int valor);
9
10 }
11
12 Ficticia f = new Ficticia(){
13     @Override
14     public void aceptar(int valor){
15         System.out.println("Hola mundo");
16     }
17 }
18
19 Ficticia f = (valor) -> System.out.println("Hola mundo" + valor)
```

Inferir el tipo de dato

Sintaxis de una Expresión Lambda

```
3
4 parametros -> cuerpo
5
6 public interface Ficticia {
7
8     public void aceptar(int valor);
9
10 }
11
12 Ficticia f = new Ficticia(){
13     @Override
14     public void aceptar(int valor){
15         System.out.println("Hola mundo");
16     }
17 }
18
19 Ficticia f = valor -> System.out.println("Hola mundo "+ valor)
```

De esta manera; sí recibe un solo parámetro
y el cuerpo de la expresión lambda
es solo una línea de código

Sintaxis de una Expresión Lambda

```
1
2 public interface ActionListener{
3     public void actionPerformed(ActionEvent event);
4 }
5
6 JButton button = new JButton("Hola");
7
8 button.addActionListener(
9     (event) -> System.out.println("Hola");
10 );
11
12 button.addActionListener(new ActionListener(){
13
14     @Override
15     public void actionPerformed(ActionEvent event){
16         lksjdfklkjdf
17     }
18
19 });
```


Inferencia de tipo en la expresión lambda

```
1
2
3 public interface Function {
4     public void accept(int entero);
5 }
6
7 Funcion f = parametro -> System.out.println(parametro);
8
9 MiClase obj = new MiClase();
10
11 obj.metodo( new Function() {
12
13     ...@Override
14     ...public void accept(int entero){
15
16         ...kjaskjldfjkl
17     ...}
18 });
19
20
21 //Clase MiClase
22 public class MiClase{
23
24     public void metodo(Function parametro){
25
26         ///Hace algo con el parámetro
27
28     }
29 }
```

8 lines, 96 characters selected

Inferencia de tipo en la expresión lambda

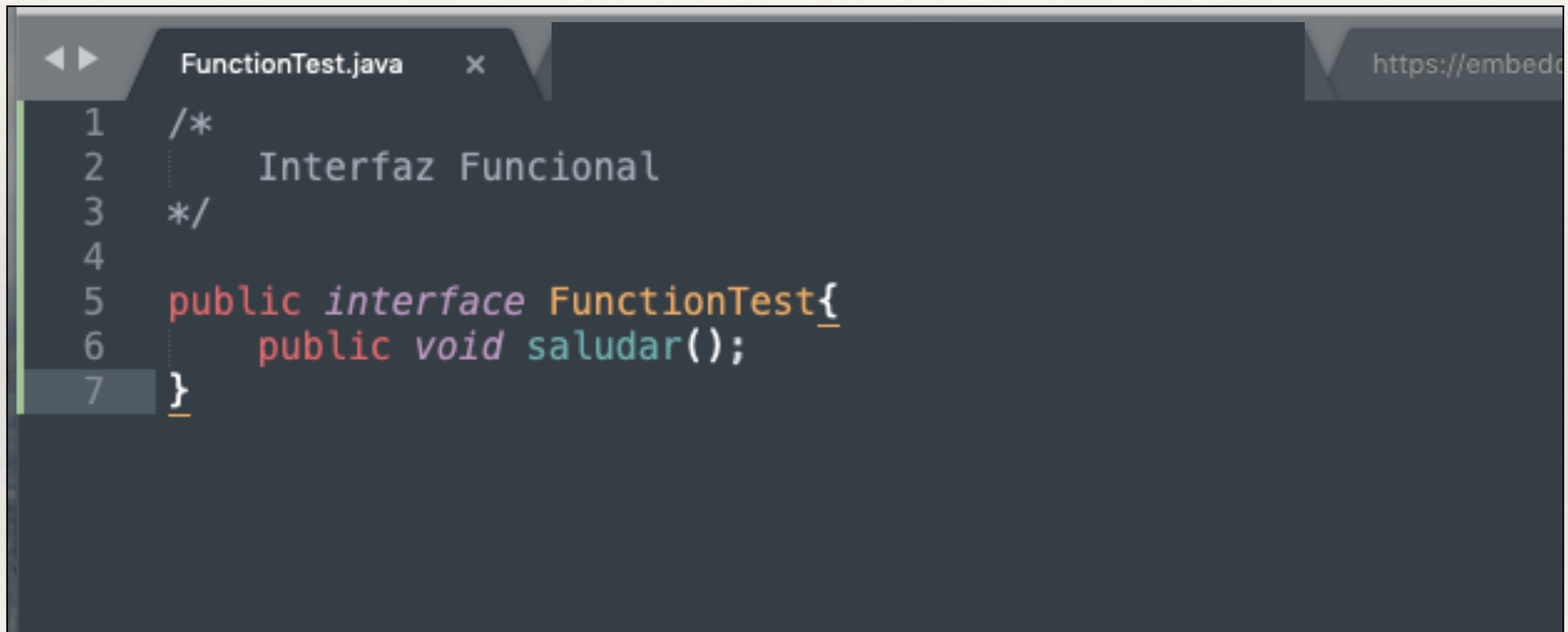
```
1
2
3 public interface Function {
4     public void accept(int entero);
5 }
6
7 Funcion f = parametro -> System.out.println(parametro);
8
9 MiClase obj = new MiClase();
10
11 obj.metodo( parametro -> System.out.println(parametro) );
12
13
14 //Clase MiClase
15 public class MiClase{
16
17     public void metodo(Function parametro){
18
19         ///Hace algo con el parámetro
20
21     }
22 }
23
```


Inferencia de tipo en la expresión lambda

```
1
2
3 public interface Function {
4     public void accept(int entero, String cadena);
5 }
6
7 Funcion f = parametro -> System.out.println(parametro);
8
9 MiClase obj = new MiClase();
10
11 obj.metodo( (parametro, parametro2) -> System.out.println(parametro) );
12
13
14 //Clase MiClase
15 public class MiClase{
16
17     public void metodo(Function parametro){
18
19         ///Hace algo con el parámetro
20
21     }
22 }
23
```

Ejemplos

Expresiones lambda sin parámetros



The screenshot shows a code editor window titled "FunctionTest.java". The code defines a functional interface named "FunctionTest" with a single method "saludar()". The interface is annotated with a Javadoc comment: "/* Interfaz Funcional */". The code is as follows:

```
1  /*  
2      Interfaz Funcional  
3  */  
4  
5  public interface FunctionTest{  
6      public void saludar();  
7  }
```


Expresiones lambda sin parámetros

```
FunctionTest.java x LambdaTest.java x
1 public class LambdaTest{
2     public static void main(String[] args) {
3
4         // Expresión lambda ==> representa un objeto de una interfaz funcional
5         FunctionTest ft = () -> System.out.println("Hola Mundo"); // Implementación del método abstracto "saludar()"
6                             // de la Interfaz Funcional.
7
8         ft.saludar();
9     }
10 }
```


Expresiones lambda sin parámetros

```
_Ejemplo_001_ — fm5@Air-Fhm5-5 — ..Ejemplo_001_ — -zsh — 110x25
└─[fm5 Air-Fhm5-5] - [~/Documents/_reposGit_github/_Programacion_Logica_y_Funcional/_Ejemplo_001_] - [Fri Feb 21, 14:52]
└─[$] <> javac LambdaTest.java
└─[fm5 Air-Fhm5-5] - [~/Documents/_reposGit_github/_Programacion_Logica_y_Funcional/_Ejemplo_001_] - [Fri Feb 21, 14:52]
└─[$] <> ls -lrat
total 32
drwxr-xr-x  4 fm5  staff   128 Feb 21 14:44 ..
-rw-r--r--@ 1 fm5  staff    56 Feb 21 14:46 FunctionTest.java
-rw-r--r--@ 1 fm5  staff   326 Feb 21 14:51 LambdaTest.java
-rw-r--r--  1 fm5  staff  1010 Feb 21 14:52 LambdaTest.class
drwxr-xr-x  6 fm5  staff   192 Feb 21 14:52 .
-rw-r--r--  1 fm5  staff   129 Feb 21 14:52 FunctionTest.class
└─[fm5 Air-Fhm5-5] - [~/Documents/_reposGit_github/_Programacion_Logica_y_Funcional/_Ejemplo_001_] - [Fri Feb 21, 14:52]
└─[$] <> java LambdaTest
Hola Mundo
└─[fm5 Air-Fhm5-5] - [~/Documents/_reposGit_github/_Programacion_Logica_y_Funcional/_Ejemplo_001_] - [Fri Feb 21, 14:52]
└─[$] <> |
```


Expresiones lambda sin parámetros



```
1 public class LambdaTest{
2     public static void main(String[] args) {
3
4         // Expresión lambda ==> representa un objeto de una interfaz funcional
5         FunctionTest ft = () -> System.out.println("Hola Mundo!!!"); // Implementación del método abstracto "saludar()"
6                                     // de la Interfaz Funcional.
7
8         //ft.saludar();
9
10        LambdaTest objeto = new LambdaTest();
11        objeto.miMetodo(ft);
12    }
13
14
15    public void miMetodo(FunctionTest parametro){
16        parametro.saludar();
17    }
18 }
```

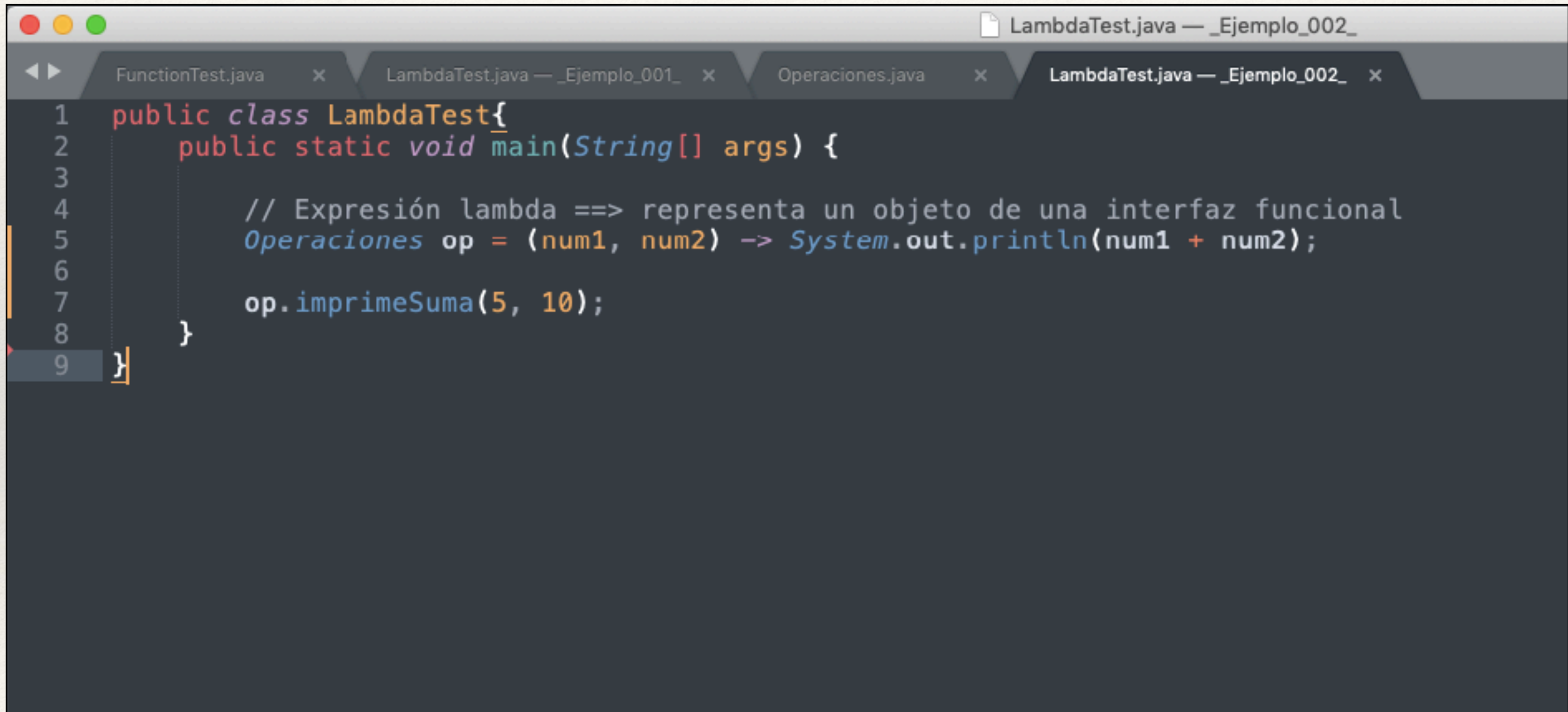
Otra manera, mismo ejemplo

Expresiones lambda con parámetros



```
1  /*
2     Interfaz Funcional
3  */
4
5  public interface Operaciones{
6      public void imprimeSuma(int num1, int num2);
7  }
```


Expresiones lambda con parámetros

A screenshot of a Java IDE window titled "LambdaTest.java — _Ejemplo_002_". The window contains a code editor with the following Java code:

```
1 public class LambdaTest{
2     public static void main(String[] args) {
3
4         // Expresión lambda ==> representa un objeto de una interfaz funcional
5         Operaciones op = (num1, num2) -> System.out.println(num1 + num2);
6
7         op.imprimeSuma(5, 10);
8     }
9 }
```

The code is color-coded: keywords like "public", "class", "static", "void", "main", and "System.out.println" are in red; identifiers like "LambdaTest", "Operaciones", "op", "num1", "num2", "5", and "10" are in blue; and literals like "5" and "10" are in orange. The IDE has a tab bar at the top with four tabs: "FunctionTest.java", "LambdaTest.java — _Ejemplo_001_", "Operaciones.java", and "LambdaTest.java — _Ejemplo_002_". The current tab is "LambdaTest.java — _Ejemplo_002_".

Expresiones lambda con parámetros

```
_Ejemplo_002_ — fm5@Air-Fhm5-5 — ..Ejemplo_002_ — -zsh — 110x25
[fm5 Air-Fhm5-5] - [~/Documents/_reposGit_gitHub/_Programacion_Logica_y_Funcional/_Ejemplo_002_] - [Fri Feb 21, 15:37]
[~[$] <> javac LambdaTest.java
[fm5 Air-Fhm5-5] - [~/Documents/_reposGit_gitHub/_Programacion_Logica_y_Funcional/_Ejemplo_002_] - [Fri Feb 21, 15:37]
[~[$] <> java LambdaTest
15
[fm5 Air-Fhm5-5] - [~/Documents/_reposGit_gitHub/_Programacion_Logica_y_Funcional/_Ejemplo_002_] - [Fri Feb 21, 15:37]
[~[$] <> |
```

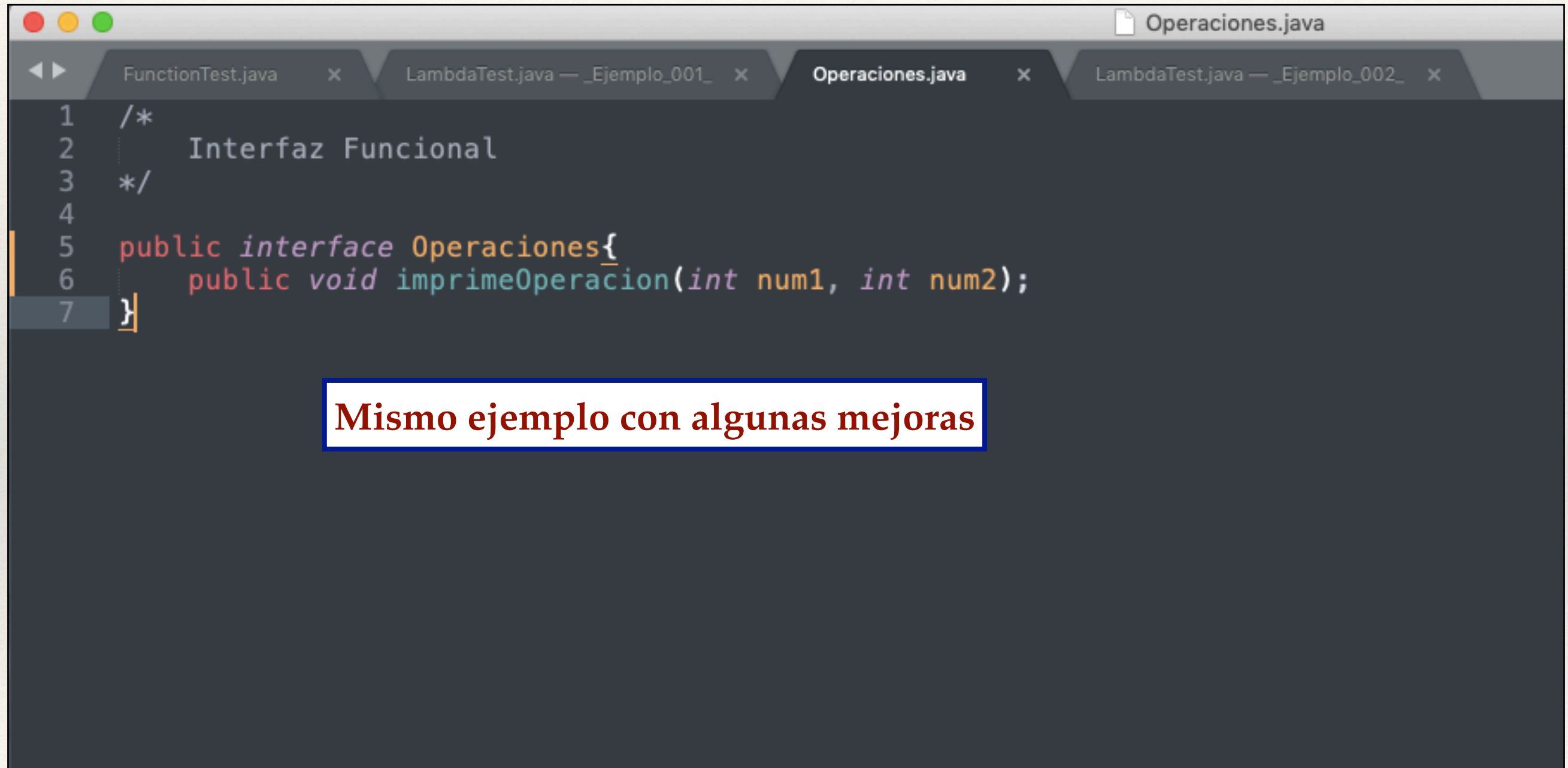

Expresiones lambda con parámetros



```
1 public class LambdaTest{
2     public static void main(String[] args) {
3
4         // Expresión lambda ==> representa un objeto de una interfaz funcional
5         Operaciones op = (num1, num2) -> System.out.println(num1 + num2);
6
7         //op.imprimeSuma(5, 10);
8
9         LambdaTest objeto = new LambdaTest();
10        objeto.miMetodo(op, 10, 10);
11    }
12
13    public void miMetodo(Operaciones op, int num1, int num2){
14        op.imprimeSuma(num1, num2);
15    }
16 }
17
```

Otra manera, mismo ejemplo

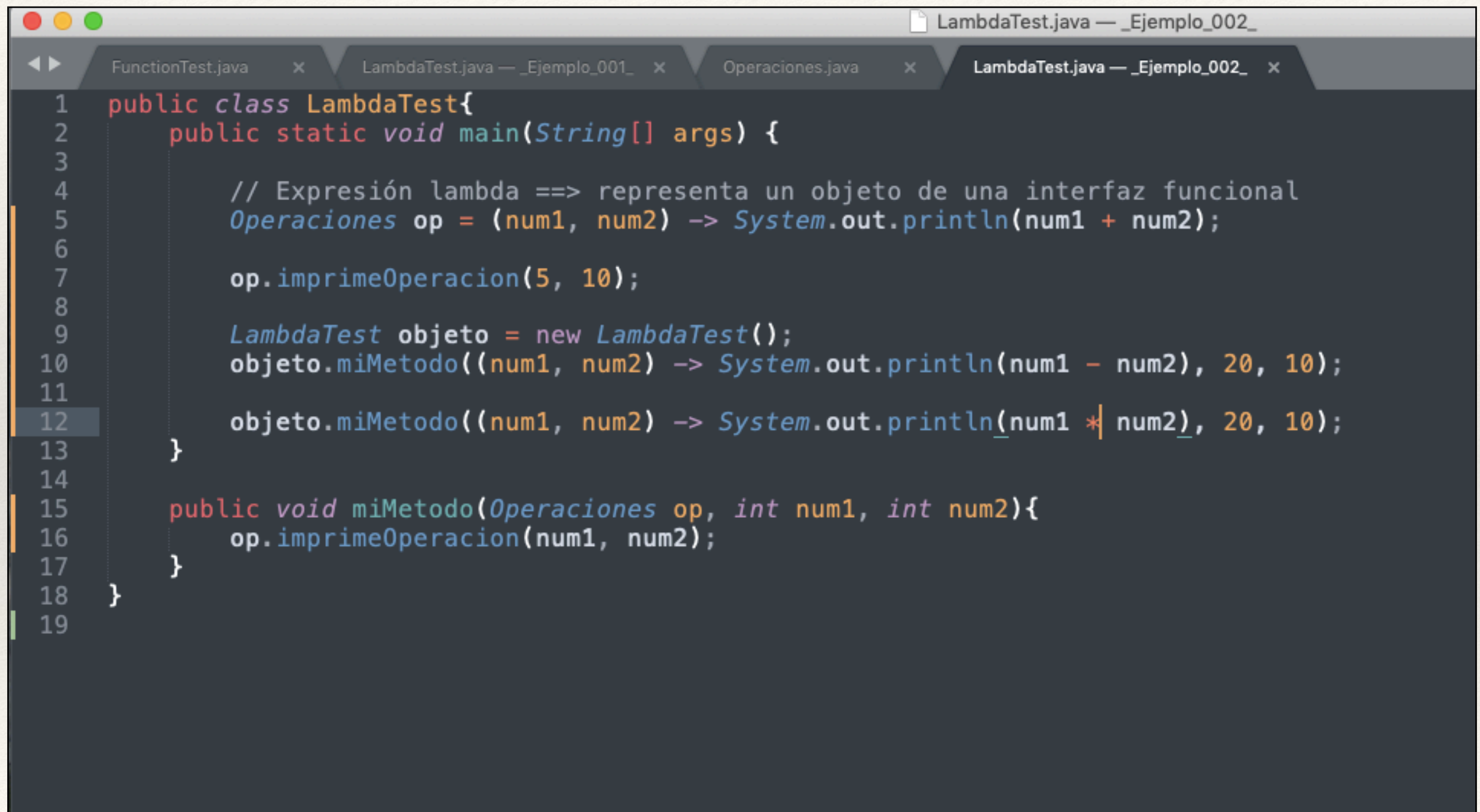
Expresiones lambda con parámetros



```
1  /*
2      Interfaz Funcional
3  */
4
5  public interface Operaciones{
6      public void imprimeOperacion(int num1, int num2);
7  }
```

Mismo ejemplo con algunas mejoras

Expresiones lambda con parámetros



The screenshot shows a Java IDE window titled "LambdaTest.java — _Ejemplo_002_". The code defines a class `LambdaTest` with a `main` method and a `miMetodo` method. The `main` method demonstrates the use of lambda expressions to create objects of the `Operaciones` interface and pass them to `miMetodo`.

```
1 public class LambdaTest{
2     public static void main(String[] args) {
3
4         // Expresión lambda ==> representa un objeto de una interfaz funcional
5         Operaciones op = (num1, num2) -> System.out.println(num1 + num2);
6
7         op.imprimeOperacion(5, 10);
8
9         LambdaTest objeto = new LambdaTest();
10        objeto.miMetodo((num1, num2) -> System.out.println(num1 - num2), 20, 10);
11
12        objeto.miMetodo((num1, num2) -> System.out.println(num1 * num2), 20, 10);
13    }
14
15    public void miMetodo(Operaciones op, int num1, int num2){
16        op.imprimeOperacion(num1, num2);
17    }
18 }
19
```