Práctica 24: Continuando la práctica 23, agrega el método: public abstract String toString(); ¿ lo acepta ? ¿cómo es que lo acepta si únicamente se podía un único método abstracto? Ahora añade el método: public abstract int otroAbstracto(); ¿lo acepta? (toma captura de pantalla) ¿qué error muestra? Comenta la línea con: @FunctionalInterface ¿sigue quejándose? ¿por qué lo acepta ahora?

```
public class P24Main {

    @FunctionalInterface
    interface Operaciones<T> {

        String operacion(T a, T b);

        public abstract String toString();


        default public void miNombre() {
            System.out.println("Kevin");
        }

        default public void misApellidos() {
            System.out.println("Hernández García");
        }

    }
```

Vemos que permite el método abstracto toString(), esto es porque las interfaces funcionales solo pueden tener un método abstracto, sin embargo pueden tener otros métodos abstractos que vengan de sobreescribir otros métodos que hayan heredado, y toString() es un método heredado de la clase Object.

```
 *
 * @author Kevin Her
 */
public class P24Main

    @FunctionalInterface
    interface Operaciones<T> {

        String operacion(T a, T b);

        public abstract String toString();

        public abstract int otroAbstracto();


        default public void miNombre() {
            System.out.println("Kevin");
        }

        default public void misApellidos() {
            System.out.println("Hernández García");
        }

    }
```

```
Unexpected @FunctionalInterface annotation
Operaciones is not a functional interface
  multiple non-overriding abstract methods found in interface Operaciones
---
(Alt-Enter shows hints)
```

Si intentamos añadir el método abstracto otroAbstracto(), vemos que el IDE si nos mostrara un error, esto lo hace porque con @FunctionalInterface estamos indicando que la función es funcional, y esta solo puede tener un método abstracto que no esté sobrescribiendo otro que haya heredado.

```java
public class P24Main {

//    @FunctionalInterface
    interface Operaciones<T> {

        String operacion(T a, T b);

        public abstract String toString();

        public abstract int otroAbstracto();


        default public void miNombre() {
            System.out.println("Kevin");
        }

        default public void misApellidos() {
            System.out.println("Hernández García");
        }

    }

    public static void mostrarResultado(int x, int y, Operaciones op) {
        System.out.println(op.operacion(x, y));
    }

    public static void main(String[] args) {
        Operaciones<Integer> suma = (a, b) -> "x + y=" + (a + b);
        Operaciones<Integer> resta = (a, b) -> "x - y=" + (a - b);
        Operaciones<Integer> multiplica = (a, b) -> "x * y=" + (a * b);
        mostrarResultado(2, 3, suma);
        mostrarResultado(5, 1, resta);
        mostrarResultado(4, 7, multiplica);
        suma.miNombre();
        suma.misApellidos();
    }
```

Si comentamos @FunctionalInterface lo que ocurre es que le estamos diciendo al IDE que esta función no es funcional, y por lo tanto te permite crear tantos métodos abstractos como quieras, sin embargo, en el main estamos haciendo uso de lambdas, y estas requieren de un Interfaz funcional para poder funcionar correctamente.