EJERCICIO U8_B4_E1:

escribe un "hola mundo " utilizando el operador ::

Lo hacemos de tres formas: 1. con clase anónima 2. con expresión lambda 3. con referencia a métodos

```
import java.util.function.Consumer;

class App {

    public static void main(String[] args) {
        Consumer<String> c1 = new Consumer<String>() {
            @Override
            public void accept(String t) {
                System.out.println(t);
            }
        };
        c1.accept("hola mundo 1");

        Consumer<String> c2= s -> System.out.println(s);
        c2.accept("hola mundo 2");

        Consumer<String> c3=System.out::println;
        c3.accept("Hola mundo 3");

    }
}
```

EJERCICIO U8_B4_E2:

```
import java.util.function.Function;

public class App{
    public static void main(String[] args){
        imprimir(new Function<String, String>(){
            @Override
            public String apply(String s){
                return s.toLowerCase();
            }
        }, "imprimir1:STRING TO LOWERCASE");

        imprimir(s -> s.toLowerCase(), "imprimir2: STRING TO LOWERCASE");

        imprimir(String::toLowerCase, "imprimir3: STRING TO LOWERCASE");
    }

    public static void imprimir(Function<String, String> function, String s){
        System.out.println(function.apply(s));
    }
}
```

Observa que en este pequeño ejemplo la programación funcional lo complica todo, para empezar el método imprimir es totalmente superfluo, pero en otros contextos verás cómo estos conceptos y sintaxis se convierten en "magia"

EJERCICIO U8_B4_E3:

```
import java.util.function.BiFunction;
import java.util.function.Function;

class Punto {
  private int x;
  private int y;

  public Punto(int x, int y) {
    this.x = x;
    this.y = y;
  }
  Punto(Punto p) {
    this.x = p.x;
    this.y = p.y;
  }

  public int getX() {
    return x;
  }

  public int getY() {
    return y;
  }
}

public class App {
  public static void main(String[] args) {
    //BiFunction<Integer, Integer, Punto> constructor = (x, y) -> new Punto(x, y);
    BiFunction<Integer, Integer, Punto> constructor = Punto::new;


    // Crear un objeto Punto con las coordenadas (3, 4)
    Punto punto = constructor.apply(3, 4);
    System.out.println("Coordenadas: (" + punto.getX() + ", " + punto.getY() + ")");

    //crear un punto copia del anterior
    //Function <Punto, Punto> constructorCopia = p -> new Punto(p);
    Function<Punto, Punto> constructorCopia = Punto::new;

    Punto copia = constructorCopia.apply(punto);
    System.out.println("Coordenadas copia: (" + copia.getX() + ", " + copia.getY() + ")");


  }
}
```