# Práctica 0.
# Funciones lambda.

# 1. Enunciado.

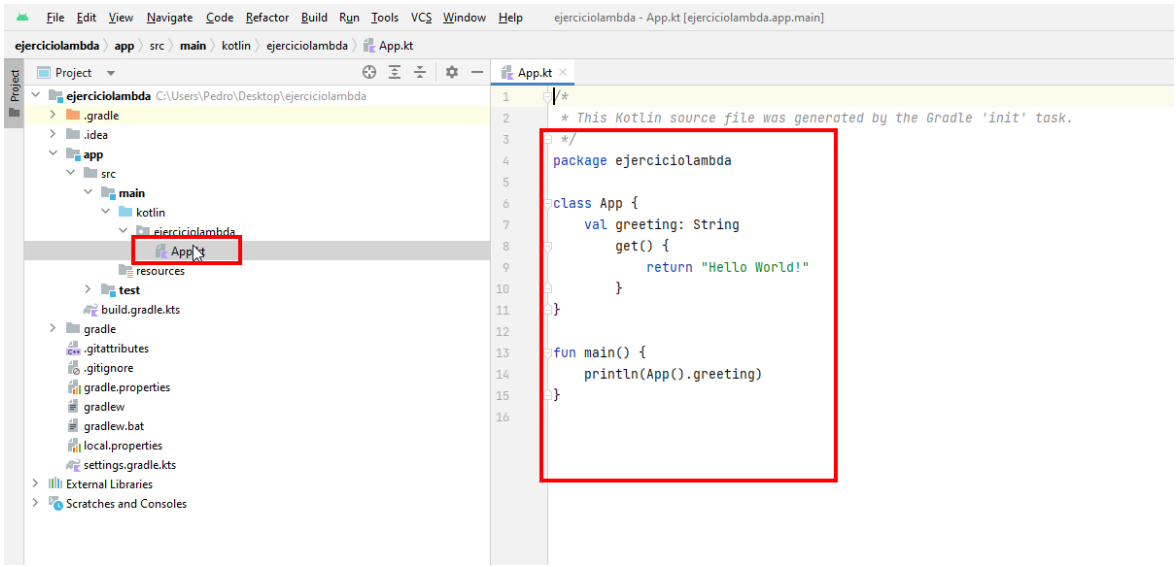
Se desea gestionar una pequeña página web que contiene categorías y productos, se tiene una estructura para almacenar todos los productos y otra estructura para almacenar las categorías, de forma que se pueda realizar búsquedas por productos y búsquedas por categorías.

1. Pensar en qué tipo de estructura de datos usar para tener categorías y dentro productos.
2. Crear la clase categoría (nombre, estado(enabled, disabled), img (en este caso una cadena) y estructura para soportar los productos) y la clase producto (nombre, precio, imagen, característica y estado), ya que tienen elementos en común usar herencia y clases abstractas.
3. Definir la estructura necesaria para gestionar las categorías.
4. Insertar la categoría: Portátiles y asignarle 2 productos:
   - Nombre: Lisa, precio 100€, características: Uno de los primeros ordenadores de Apple.
   - Nombre: IBMPC, precio 50€, características: El primer equipo para el gran público.
5. Insertar la categoría: Impresoras y asignar 2 productos:
   - Nombre: Brother Colo1234, precio 10€, características: Impresora a color.
   - Nombre: HP Laser, precio 20€, características: Impresora láser.
6. Obtener la lista de categorías de la tienda.
7. Listar los productos de portátiles.
8. Volver a listar las categorías.
9. Obtener el precio medio de los productos.
10. Obtener el precio medio de los productos por categorías.
11. Crear un método que permita buscar por nombre.
12. Crear un método que permita buscar por precio.
13. Crear un método que permita buscar por nombre y precio.
14. Las búsquedas han de estar ordenadas por precio.
15. Borrar las impresoras con un precio mayor que un 15€

# 2. Empezando.

Se puede crear el proyecto usando Gradle y seleccionando Kotlin como lenguaje, y a continuación abrirlo con AndroidStudio.

**Observar la clase principal en el proyecto en Kotlin:**




**Crear la clase producto:**

```
package ejerciciolambda
enum class Estado{
    ACTIVO,
    INACTIVO
}
class Producto(
    var nombre: String,
    var precio: Float,
    var imagen: String,
    var caracteristicas: String,
    var estado: Estado = Estado.ACTIVO
)
```

**Seleccionar una estructura de datos, por ejemplo, un ArrayList para gestionar un conjunto de productos e insertar varios**:

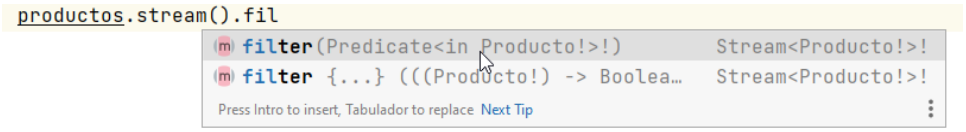
```
fun main() {
    var productos=ArrayList<Producto>()
    productos.add(Producto("Lavadora grande",345f ,"lg.jpeg","lavadora grande",Estado.ACTIVO))
    productos.add(Producto("Televisor 55\"",545f ,"tv55.jpeg","televisor grande",Estado.ACTIVO))
    productos.add(Producto("Portatil 14",654f ,"ptt.jpeg","portatil de 8gb pequeño",Estado.ACTIVO))

}
```

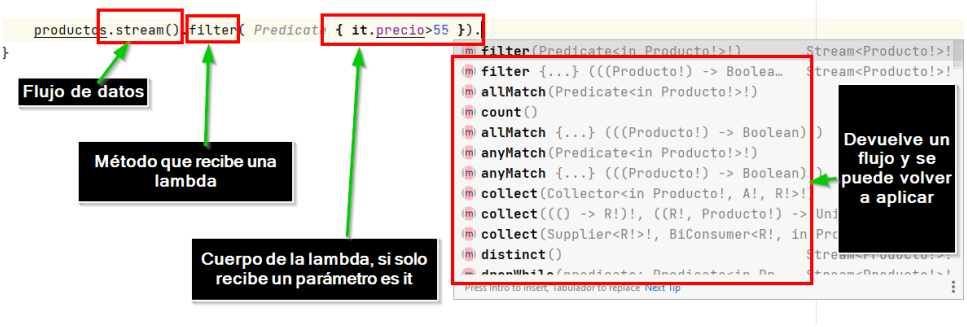
**A partir de este punto, y usando los métodos de ArrayList (todas las colecciones lo implementan, ya sea propio de la colección o con los flujos o streams) y pasándoles**

**funciones lambda a estos, ir cumpliendo con los puntos del ejercicio. Por ejemplo, si se desea obtener los productos cuyo precio sea mayor que una cantidad, se usa el filter del arraylist.**

**Por ejemplo, filter:**




**Recibe un predicado y devuelve a su vez un flujo de datos con los elementos ya filtrados, el predicado es una función lambda o anónima con un parámetro (en este caso el producto) y que devuelve cierto o falso**.




**Siendo a su vez posible volver a aplicar métodos, como en foreach que recibe una lambda con un parámetro y no devuelve nada:**

```
productos.stream().filter(        {        it.precio>555        }).forEach{
    println(it.nombre)
}
```

**Otra forma, de definir funciones lambda, más clara, en la que se observan los parámetros:**

```
productos.stream().filter{        item->        item.precio>555}.forEach{
    item2->                                        println(item2.nombre)
}
```

**Una operación interesante es la de aplanar, que consiste en crear una estructura a partir de otras, de forma que se pueda procesar como una sola la segunda. En la aplicación se tiene un mapa de categorías, y a su vez las categorías tienen productos:**

```
categorias.values.flatMap        {        it.productos        }.forEach{
    println                                                   (it.nombre)
}
```

**En el siguiente enlace se tiene las operaciones más comunes con funciones lambda sobre colecciones: [https://kotlinlang.org/docs/collection-aggregate.html](https://kotlinlang.org/docs/collection-aggregate.html)**