



Adrian Parrilla Egido

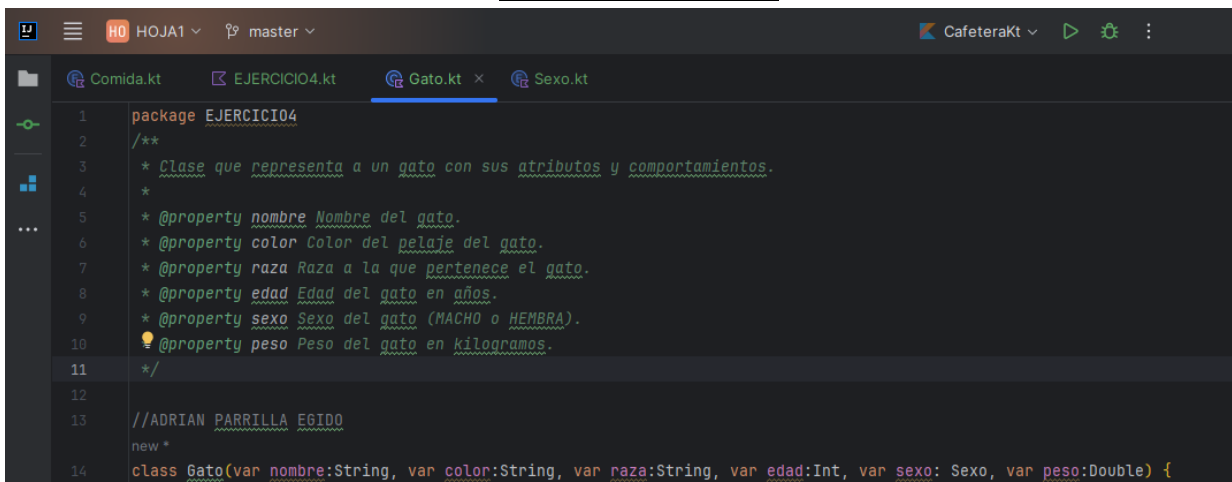
Elabora un manual de uso de KDoc generado desde IntelliJ con todas sus funcionalidades.

Debes usar un ejemplo real de aplicación en Kotlin que contenga varias clases (Puedes usar algún ejercicio de clase previamente programado).

El manual debe incluir capturas de pantalla del procedimiento seguido para alcanzar el objetivo que aquí se plantea.

**No olvides acreditar la autoría del trabajo haciendo que en las capturas de pantalla aparezca tu nombre.**

## CLASE GATO



```
1 package EJERCICIO4
2 /**
3  * Clase que representa a un gato con sus atributos y comportamientos.
4  *
5  * @property nombre Nombre del gato.
6  * @property color Color del pelaje del gato.
7  * @property raza Raza a la que pertenece el gato.
8  * @property edad Edad del gato en años.
9  * @property sexo Sexo del gato (MACHO o HEMBRA).
10  * @property peso Peso del gato en kilogramos.
11  */
12
13 //ADRIAN PARRILLA EGIDO
14 new *
15 class Gato(var nombre:String, var color:String, var raza:String, var edad:Int, var sexo: Sexo, var peso:Double) {
```

- Para cada propiedad (nombre, color, raza, edad, sexo, peso), utilizo @property seguido del nombre de la propiedad y una breve descripción.



Adrian Parrilla Egido

```
HOJA1 master
Comida.kt EJERCICIO4.kt Gato.kt Sexo.kt

15 /**
16  * Método que hace que el gato maúlle.
17  */
18 new *
19 fun maullar(){
20     println("${nombre} Miauuu")
21 }
22 //ADRIAN PARRILLA EGIDO
23 /**
24  * Método que hace que el gato ronronee.
25  */
26 new *
27 fun ronronear(){
28     println("${nombre} Mrrr")
29 }
30 /**
31  * Método que simula el acto de comer del gato.
32  *
33  * @param comidaGato Tipo de comida que el gato intenta comer.
34  */
35 new *
36 fun comer(comidaGato: Comida){
37     if (comidaGato == Comida.PESCADO){
38         println("${nombre} te da las gracias por ${Comida.PESCADO}")
39     }else{
40         println("${nombre} rechaza la comida")
41     }
42 }
43 //ADRIAN PARRILLA EGIDO
44 /**
45  * Método que simula una pelea entre dos gatos.
46  *
47  * @param gatoPeelador Gato con el que se va a pelear.
48  */
49 new *
50 fun pelear(gatoPeelador: Gato){
51     if ((sexo == Sexo.MACHO) && (gatoPeelador.sexo == Sexo.MACHO)) {
52         println("${nombre} y ${gatoPeelador.nombre} estan peleando")
53     }else{
54         println("No se pueden pelear ")
55     }
56 }
```

Adrian Parrilla Egido

- Para cada función (maullar, ronronear, comer, pelear), uso `/** ... */` para agregar comentarios KDoc sobre la funcionalidad de cada método. Documento el propósito de la función y describo los parámetros y el tipo de retorno, si es relevante.

```

49         println("$nombre y $gatoPeleeador.nombre} estan peleando")
50     }else{
51         println("No se pueden pelear ")
52     }
53 }
54 /**
55  * Sobrescritura del método toString para representar el objeto como una cadena de texto.
56  *
57  * @return Representación en cadena del objeto Gato.
58  */
59 //ADRIAN PARRILLA EGIDO
60 new *
61 override fun toString(): String {
62     return "$nombre de color $color con raza $raza, con $edad años, de sexo$sexo pesa $peso kg"
63 }
64 }
    
```

- Utilizo `/** ... */` para explicar el propósito de la función `toString()` y especifico la cadena de retorno.
- Añado comentarios para explicar la lógica de las funciones, como en la función `comer` donde describo el comportamiento según el tipo de comida.

## CLASE SEXO

```

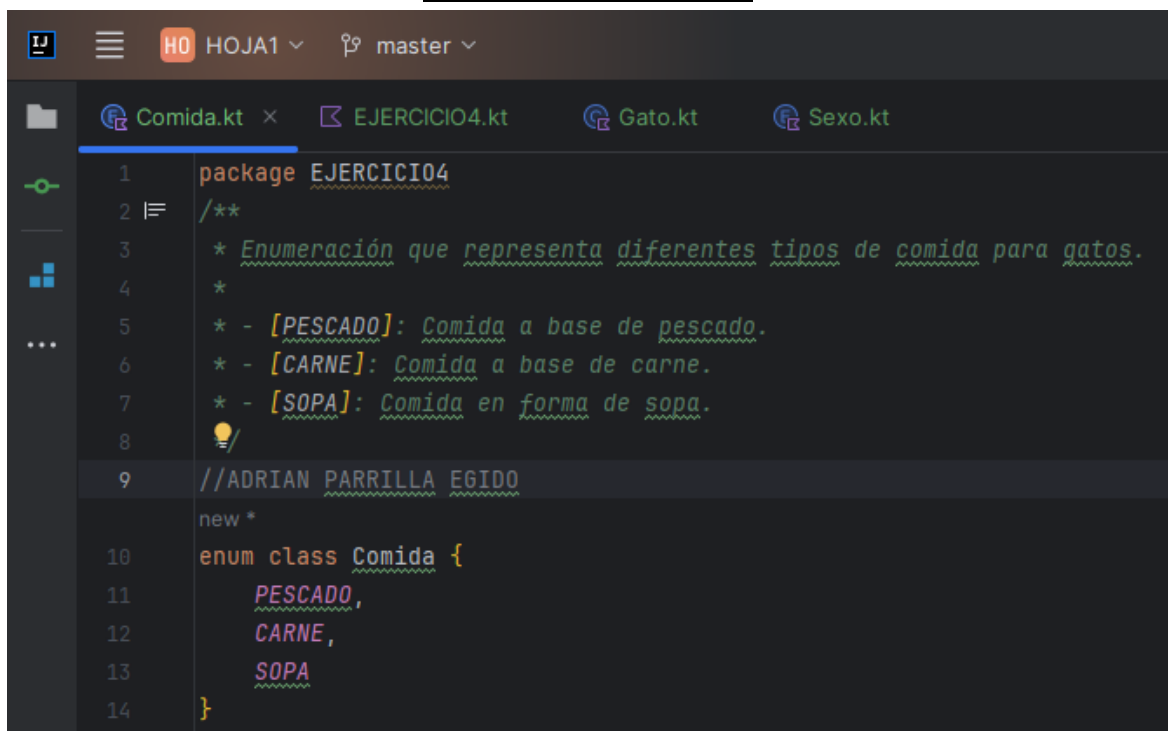
1 package EJERCICIO4
2 /**
3  * Enumeración que representa los posibles sexos de un gato.
4  *
5  * - [MACHO]: Sexo masculino.
6  * - [HEMBRA]: Sexo femenino.
7  * - [BINARIO]: Otra opción para casos específicos.
8  */
9 //ADRIAN PARRILLA EGIDO
10 new *
11 enum class Sexo {
12     MACHO,
13     HEMBRA,
14     BINARIO
15 }
    
```



Adrian Parrilla Egido

- Utilizo `/** ... */` antes de la declaración de la enumeración para agregar comentarios KDoc. Proporciono una descripción general de la enumeración y de sus posibles valores.
- Proporciono descripciones detalladas para cada uno de los valores de la enumeración (MACHO, HEMBRA, y BINARIO). Cada descripción indica el significado asociado con el valor respectivo.
- Utilizo una lista (`- [MACHO]`, `- [HEMBRA]`, `- [BINARIO]`) para resaltar y organizar los valores de la enumeración.

## CLASE COMIDA



```
1 package EJERCICIO4
2 /**
3  * Enumeración que representa diferentes tipos de comida para gatos.
4  *
5  * - [PESCADO]: Comida a base de pescado.
6  * - [CARNE]: Comida a base de carne.
7  * - [SOPA]: Comida en forma de sopa.
8  */
9 //ADRIAN PARRILLA EGIDO
10 new *
11 enum class Comida {
12     PESCADO,
13     CARNE,
14     SOPA
15 }
```

- Utilizo `/** ... */` antes de la declaración de la enumeración para comenzar los comentarios KDoc. Proporciono una descripción general de la enumeración y su propósito.
- Proporciono descripciones detalladas para cada uno de los valores de la enumeración (PESCADO, CARNE, y SOPA). Cada descripción indica el tipo de comida asociado con el valor respectivo.
- Utilizo una lista (`- [PESCADO]`, `- [CARNE]`, `- [SOPA]`) para resaltar y organizar los valores de la enumeración.



# ENTORNOS DE DESARROLLO

## UT 4 – JAVADOC-KDOC

### Ejercicios

Dpto. INFORMÁTICA



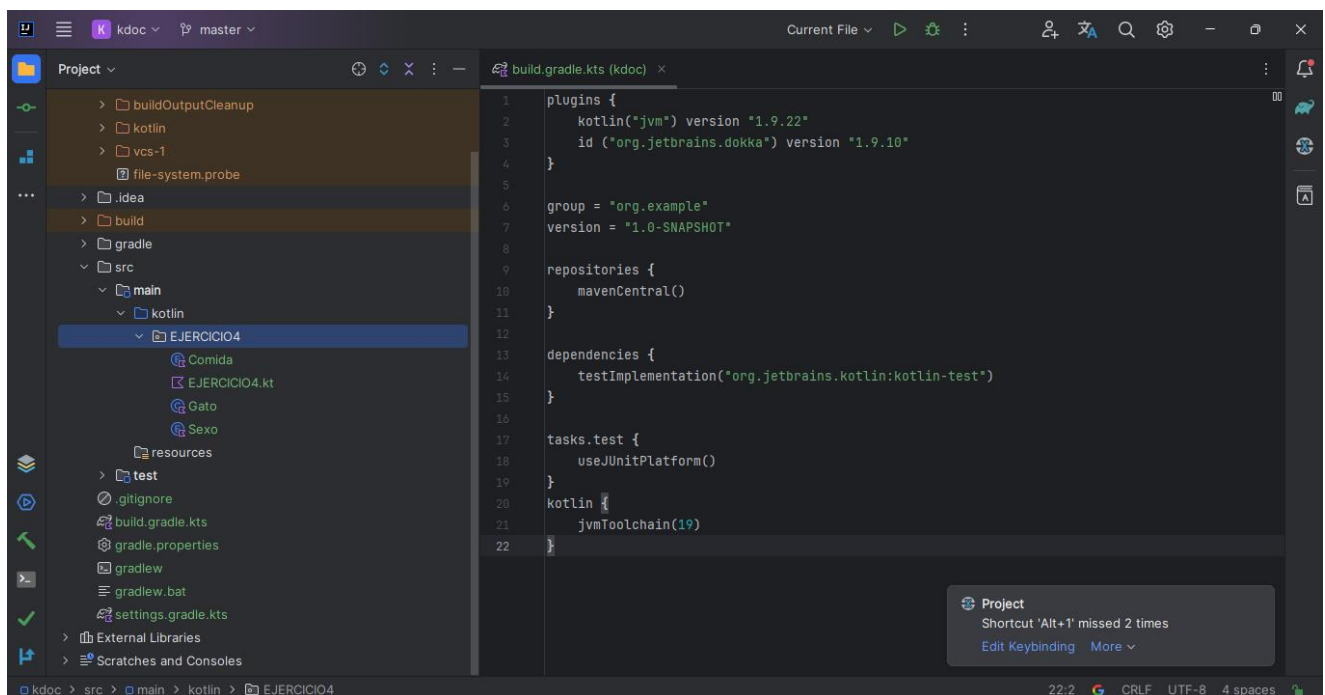
Curso: 2023-24

Adrian Parrilla Egido

Añade comentarios a tu código fuente con Kdoc y genera automáticamente la documentación para cada clase y cada método de tu proyecto.

Intenta generar la documentación en HTML para poder compartirla con otros desarrolladores y clientes. Explica las herramientas empleadas y el procedimiento para instalarlas y para utilizarlas.

- Utilizo Gradle de la siguiente manera abriendo el terminal y escribo “Gradlew dokkaHtml”.





# ENTORNOS DE DESARROLLO

## UT 4 – JAVADOC-KDOC

### Ejercicios

Dpto. INFORMÁTICA



Curso: 2023-24

Adrian Parrilla Egido

The screenshot shows an IDE interface with a project structure on the left and a code editor on the right. The project structure includes folders like `buildOutputCleanup`, `kotlin`, `vcs-1`, `file-system.probe`, `.idea`, `build`, `gradle`, `src`, `main`, `kotlin`, and `EJERCICIO4`. The `EJERCICIO4` folder contains files `Comida`, `EJERCICIO4.kt`, `Gato`, and `Sexo`. The code editor shows the `build.gradle.kts` file with the following content:

```
11 }
12
13 dependencies {
14     testImplementation("org.jetbrains.kdoc")
15 }
16
17 tasks.test {
18     useJUnitPlatform()
19 }
20 kotlin {
21     jvmToolchain(19)
22 }
23 //Adrian Parrilla Egido
```

The terminal at the bottom shows the following output:

```
Running post-actions
BUILD SUCCESSFUL in 8s
1 actionable task: 1 executed
C:\Users\adria\Downloads\DAM\PROGRAMACION\kdoc>gradlew dokkaHtml
```



# ENTORNOS DE DESARROLLO

## UT 4 – JAVADOC-KDOC

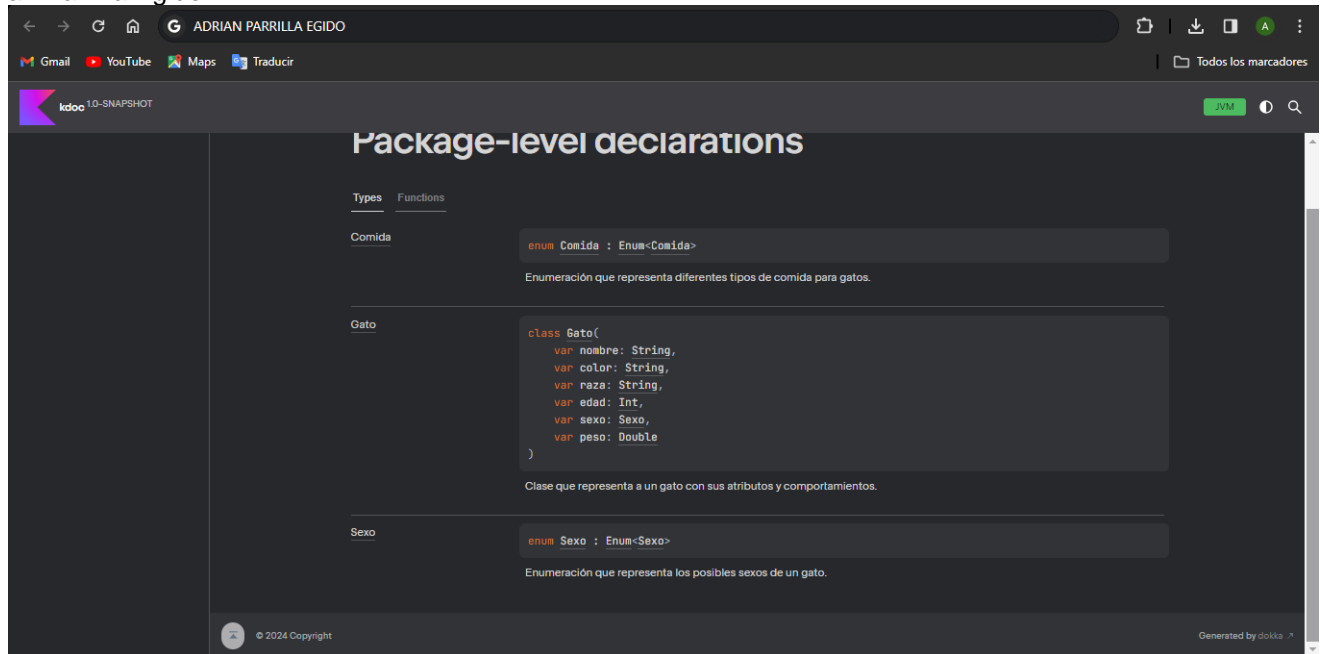
### Ejercicios

Dpto. INFORMÁTICA



Curso: 2023-24

Adrian Parrilla Egido



¿Sabrás crear esa misma documentación en formato pdf?

- Sería posible crearla pasando el html a pdf con alguna herramienta de internet como iLovePDF o html2PDF.