Reseña:

Podemos verificar que kotlin es simple y menos código que java, eso podría ser que lo haga mas eficiente a la hora d programar.

Declaración de una clase

Para declarar una **clase en Kotlin** se utiliza la palabra class, seguida del nombre de la clase. Por ejemplo:

class Mascota{}

Veamos un ejemplo sencillo de una clase en Kotlin:

|  |  |
| --- | --- |
|  | /\*\* |
|  | \* Clase en Kotlin |
|  | \* |
|  | \* @author parzibyte |
|  | \* @link https://parzibyte.me/blog |
|  | \* |
|  | \* \*/ |
|  | class Mascota { |
|  | var nombre: String |
|  |  |
|  | init { |
|  | println("Init del objeto") |
|  | nombre = "" |
|  | } |
|  |  |
|  | constructor(nombre: String) { |
|  | this.nombre = nombre |
|  | // Aquí el constructor |
|  | println("Se construye el objeto") |
|  | } |
|  |  |
|  |  |
|  | fun dormir() { |
|  | println("ZZZZ $nombre duerme") |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/parzibyte/23388839aeedb7e7ca7549b49639f861/raw/9452e120e11ebe6d98d6cd1434e867fee51738f9/Clase.kt)[**Clase.kt**](https://gist.github.com/parzibyte/23388839aeedb7e7ca7549b49639f861#file-clase-kt) hosted with  by [**GitHub**](https://github.com/)

Las propiedades de la clase se declaran con var o con val, y siguen las mismas [reglas](https://parzibyte.me/blog/2019/07/02/val-var-kotlin/) que cualquier [variable](https://parzibyte.me/blog/2019/07/08/variables-constantes-kotlin/).

Podemos tener [funciones](https://parzibyte.me/blog/2019/07/18/funciones-kotlin/) dentro de la clase, y se puede acceder a las propiedades de la misma.

El init es para establecer el valor inicial de las variables. Es muy útil cuando declaramos con val y las mismas se deben inicializar obligatoriamente.

Dentro de una clase se puede tener **varios constructores** con distinta firma, es decir, que reciban distintos argumentos o una cantidad diferente. Lo vamos a ver más abajo.

Instanciar una clase

Para crear un objeto o una **instancia** de una clase, se hace así:

val instancia = Clase()

(también podrías usar var, aunque no es recomendable)

No necesitamos ninguna palabra reservada como new o algo así. Después de tener un objeto de la clase podemos llamar a los métodos de la misma.

Veamos el ejemplo de **creación de un objeto en Kotlin**:

|  |  |
| --- | --- |
|  | fun main(argumentos: Array<String>) { |
|  | val mascota = Mascota("Maggie") |
|  | mascota.dormir() |
|  | println("El nombre de la mascota: ${mascota.nombre}") |
|  | } |

[**view raw**](https://gist.github.com/parzibyte/8c780fb33959f6f9c271603d1bc9992e/raw/3fbc4da53746340fb1d5a47b079d2b32fa49f14c/main.kt)[**main.kt**](https://gist.github.com/parzibyte/8c780fb33959f6f9c271603d1bc9992e#file-main-kt) hosted with  by [**GitHub**](https://github.com/)

En la línea 2 hago la instancia de la clase, pasando el nombre de la mascota (pues es un argumento necesario)

En la línea 3 invoco a un método de la clase que internamente imprime un mensaje usando las [backticks](https://parzibyte.me/blog/2019/07/02/cadenas-plantillas-cadena-kotlin/).

Como la propiedad nombre es pública, puedo acceder a ella desde la instancia.

Puedes probar el código [aquí](https://pl.kotl.in/nul3t1Vr9).

Modificadores de acceso

Kotlin tiene 4 modificadores de acceso:

public: por defecto las funciones y variables son públicas. Es decir, especificarlo con public o no especificar su acceso es lo mismo.

private: solo se puede acceder desde la clase. No se puede acceder a instancia.propiedad desde fuera de la clase.

internal: cualquiera (dentro del mismo módulo) que pueda acceder a la clase puede acceder a la propiedad.

protected: es como private, pero las subclases de de esta clase pueden acceder a la propiedad.

Constructores

Pueden existir varios constructores dentro de una clase de Kotlin. Todos ellos son definidos con la palabra reservada constructor.

En Kotlin existe el constructor primario, el cual **puede ser definido dentro de la definición de la clase**.

Por ejemplo, la clase que estaba así:

|  |  |
| --- | --- |
|  | /\*\* |
|  | \* Clase en Kotlin |
|  | \* |
|  | \* @author parzibyte |
|  | \* @link https://parzibyte.me/blog |
|  | \* |
|  | \* \*/ |
|  | class Mascota { |
|  | var nombre: String |
|  |  |
|  | init { |
|  | println("Init del objeto") |
|  | nombre = "" |
|  | } |
|  |  |
|  | constructor(nombre: String) { |
|  | this.nombre = nombre |
|  | // Aquí el constructor |
|  | println("Se construye el objeto") |
|  | } |
|  |  |
|  |  |
|  | fun dormir() { |
|  | println("ZZZZ $nombre duerme") |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/parzibyte/23388839aeedb7e7ca7549b49639f861/raw/9452e120e11ebe6d98d6cd1434e867fee51738f9/Clase.kt)[**Clase.kt**](https://gist.github.com/parzibyte/23388839aeedb7e7ca7549b49639f861#file-clase-kt) hosted with  by [**GitHub**](https://github.com/)

Puede definirse así:

|  |  |
| --- | --- |
|  | /\*\* |
|  | \* Clase en Kotlin |
|  | \* |
|  | \* @author parzibyte |
|  | \* @link https://parzibyte.me/blog |
|  | \* |
|  | \* \*/ |
|  | class Mascota(var nombre: String) { |
|  |  |
|  | init { |
|  | println("Init del objeto") |
|  | nombre = "" |
|  | } |
|  |  |
|  |  |
|  | fun dormir() { |
|  | println("ZZZZ $nombre duerme") |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/parzibyte/218f1c87af82e649b86c9319412b39c4/raw/057f12b824a209951a0799b55500ac7f58afba09/Mascota.kt)[**Mascota.kt**](https://gist.github.com/parzibyte/218f1c87af82e649b86c9319412b39c4#file-mascota-kt) hosted with  by [**GitHub**](https://github.com/)

Mira la línea 8, dentro de la misma se declara la propiedad y el constructor.

Recuerda: el init no es obligatorio, pero en él puedes iniciar los elementos de la clase o simplemente hacer lo que se hace al construir tu objeto.

Varios constructores

Además del constructor primario, pueden existir más constructores. Los mismos deben llamar al constructor primario.

Veamos este ejemplo en donde se recibe, además del nombre, la edad de la mascota.

|  |  |
| --- | --- |
|  | /\*\* |
|  | \* Clase en Kotlin |
|  | \* |
|  | \* @author parzibyte |
|  | \* @link https://parzibyte.me/blog |
|  | \* |
|  | \* \*/ |
|  | class Mascota(var nombre: String, var edad: Int) { |
|  |  |
|  | constructor(nombre: String) : this(nombre, 0) { |
|  | // Opcionalmente aquí el cuerpo u otras acciones |
|  | } |
|  |  |
|  |  |
|  | fun dormir() { |
|  | println("ZZZZ $nombre duerme") |
|  | } |
|  |  |
|  | fun saludar() { |
|  | println("¡Hola! me llamo $nombre y tengo $edad años. Woof") |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/parzibyte/fd21299e6fe7f62fc1de053d3719a869/raw/313b9d2bcdbbdd82d1f8b780985d32153c64535f/Mascota.kt)[**Mascota.kt**](https://gist.github.com/parzibyte/fd21299e6fe7f62fc1de053d3719a869#file-mascota-kt) hosted with  by [**GitHub**](https://github.com/)

El constructor primario necesita el nombre y la edad (**línea 8**)

Pero en la **línea 10** definimos otro constructor, un **constructor secundario en Kotlin**. El mismo solo recibe el nombre, y llama a this (es decir, al constructor primario) estableciendo la edad en 0, que es el **valor por defecto**.

Opcionalmente, el segundo constructor puede tener un cuerpo en donde hacemos más acciones además de llamar al constructor primario.

La forma de invocación podría ser la siguiente:

|  |  |
| --- | --- |
|  | fun main(argumentos: Array<String>) { |
|  | val mascota = Mascota("Maggie") |
|  | mascota.dormir() |
|  | mascota.saludar() |
|  | val otraMascota = Mascota("Guayaba", 2) |
|  | otraMascota.saludar() |
|  | /\* |
|  | \* Salida: |
|  | ZZZZ Maggie duerme |
|  | ¡Hola! me llamo Maggie y tengo 0 años. Woof |
|  | ¡Hola! me llamo Guayaba y tengo 2 años. Woof |
|  | \*/ |
|  | } |

[**view raw**](https://gist.github.com/parzibyte/b8f4f8bc08cf497fac78167e23af927e/raw/132fa8f869d7a56c5394cf596fc2069e73803509/invocaci%C3%B3n.kt)[**invocación.kt**](https://gist.github.com/parzibyte/b8f4f8bc08cf497fac78167e23af927e#file-invocacion-kt) hosted with  by [**GitHub**](https://github.com/)

Tenemos dos instancias. Una de ellas es creada sin especificar la edad (por lo que ahora es 0 según nuestro segundo constructor)

La otra, se define con la edad.

Puedes probar el código [aquí](https://pl.kotl.in/1y4whr5tZ).

Getters y setters

Los gettes y setters en Kotlin son implícitos y heredan el mismo acceso que las propiedades.

Veamos cómo definir los getters y setters, y cómo declararlos de forma explícita.

**Nota:** ya lo dije pero lo digo de nuevo. Los getters y setters en Kotlin heredan el mismo acceso que las propiedades. Así que si defines una propiedad como privada, el getter y setter también serán privados y esto no puede ser cambiado.

Si quieres proteger tus propiedades pero poner su modificador público, crea una función **que no lleve** el nombre de getTuPropiedad y setTuPropiedad, es decir, coloca un nombre distinto.

Get y set en Kotlin: implícito

Por defecto se definen estas funciones sin nosotros saberlo.

Es decir, si definimos la siguiente propiedad:

var nombre: String

Los getters y setters se crean y se usan así:

mascota.nombre = "Otro nombre"

var elNombre = mascota.nombre

No hay necesidad de llamar a funciones como getNombre y setNombre, simplemente hay que acceder a la propiedad o establecerla.

**Nota:** internamente Kotlin definirá y llamará a getNombre y setNombre.

Get y set en Kotlin explícito

Ahora veamos de nuevo nuestra clase:

|  |  |
| --- | --- |
|  | /\*\* |
|  | \* Clase en Kotlin |
|  | \* |
|  | \* @author parzibyte |
|  | \* @link https://parzibyte.me/blog |
|  | \* |
|  | \* \*/ |
|  | class Mascota { |
|  | var nombre: String |
|  | get() { |
|  | println("Llamada al getter de nombre") |
|  | return field |
|  | } |
|  | set(nuevoNombre) { |
|  | println("Llamada al setter de nombre") |
|  | field = nuevoNombre |
|  | } |
|  |  |
|  | // Para edad, dejamos el setter y getter implícito |
|  | var edad: Int |
|  |  |
|  |  |
|  | constructor(nombre: String, edad: Int) { |
|  | this.nombre = nombre |
|  | this.edad = edad |
|  | } |
|  |  |
|  | fun saludar() { |
|  | println("¡Hola! me llamo $nombre y tengo $edad años. Woof") |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/parzibyte/722da692a95fce3bc635d4da6eb29d01/raw/d85ed1c1bf68386a7ab2bcfea1c79fec5e79000b/Mascota.kt)[**Mascota.kt**](https://gist.github.com/parzibyte/722da692a95fce3bc635d4da6eb29d01#file-mascota-kt) hosted with  by [**GitHub**](https://github.com/)

Abajo de la declaración de la propiedad he establecido la **función get y set**. Lo único que hago en el cuerpo es imprimir que estamos usando las funciones.

*Como puedes ver, dentro de los getters y setters se utiliza field. Esta variable field es especial y se refiere / apunta al campo.*

En la **línea 12** no podemos regresar this.nombre ya que estaríamos llamando al get, es decir, sería una **recursión**. Por eso es que mejor se devuelve field, que apunta a la variable en sí.

Para el setter (**línea 14**) es lo mismo, se recibe un valor y se le asigna eso a field. No podemos hacer un this.nombre = nuevoNombre porque se llamaría al setter, creando una **recursividad**.

Podemos ejemplificar esto con el siguiente código:

|  |  |
| --- | --- |
|  | fun main(argumentos: Array<String>) { |
|  | val mascota = Mascota("Guayaba", 2) |
|  | val elNombre = mascota.nombre // se llama a getNombre internamente |
|  | println("El nombre de la mascota es $elNombre") |
|  | val laEdad = mascota.edad // Llamar a getEdad. Como no definimos el getter, no se imprime nada |
|  | println("La edad es: $laEdad") |
|  |  |
|  | // Cambiamos, llamamos a setNombre |
|  | mascota.nombre = "Panqué" |
|  | // Y ahora de nuevo a getNombre |
|  | println("El nuevo nombre es: ${mascota.nombre}") |
|  |  |
|  | /\* |
|  | \* Salida: |
|  | \* Primero el constructor: |
|  | Llamada al setter de nombre |
|  | \* Luego obtenemos el nombre y la edad |
|  | Llamada al getter de nombre |
|  | El nombre de la mascota es Guayaba |
|  | La edad es: 2 |
|  | \* Más tarde, llamada al setter e inmediatamente al getter |
|  | Llamada al setter de nombre |
|  | Llamada al getter de nombre |
|  | El nuevo nombre es: Panqué |
|  | \* \*/ |
|  | } |

[**view raw**](https://gist.github.com/parzibyte/d83be9fed58cbe8616297955f870518d/raw/af6c929b20944bee47f2b92343c6a3da655deb64/getters_setters.kt)[**getters\_setters.kt**](https://gist.github.com/parzibyte/d83be9fed58cbe8616297955f870518d#file-getters_setters-kt) hosted with  by [**GitHub**](https://github.com/)

Recuerda que el getter y setter de la *edad* no han sido modificados, en cambio los del nombre sí.

En el constructor (**línea 23** de la clase) se llama a this.edad = edad así que invocamos al setter, y como lo hemos modificado, se imprime dicha modificación.

Así es como podemos poner un cuerpo al setter y getter de las propiedades de una clase. Por defecto Kotlin crea uno por nosotros pero podemos “sobrescribirlos”

Prueba el código [aquí](https://pl.kotl.in/yQY_jjs3g).