Constructores Kotlin

Un constructor es una función miembro especial que se invoca automáticamente cuando se crea un objeto de la clase. Su objetivo principal es inicializar propiedades y otras variables. Una clase debe tener un constructor y, si no declaramos ningún constructor, el compilador genera un constructor predeterminado.

Kotlin tiene dos tipos de constructores:

- Constructor principal o primario
- Constructor secundario

Una clase en Kotlin puede tener como máximo un constructor principal y uno o más constructores secundarios. El constructor principal se utiliza cuando simplemente queremos hacer asignaciones de valores a las propiedades. El constructor secundario se usa cuando se requiere hacer las inicializaciones con algo de lógica adicional.

El constructor predeterminado

En el ejemplo que sigue, la clase Persona no define ningún constructor y por tanto se puede utilizar el constructor predeterminado para crear un objeto de esa clase. El constructor predeterminado se invoca con el nombre de la clase seguido de parentesis vacios.

In [5]:
```
class Persona{
    val nombre:String="yo"
    val edad:Int=22
}
val p=Persona()//usando constructor predeterminado

print("soy ${p.nombre} y tengo ${p.edad} años")
```

```
soy yo y tengo 22 años
```

El constructor principal

Hay unas cuantas variantes sintácticas a la hora de escribir un constructor principal. Nos centramos en lo más básico.

El constructor principal se escribe simplemente indicando despues del nombre de la clase la palabra clave *constructor*, y a continuación entre paréntesis, un conjunto de parámetros separados por comas.

Lo más habitual y simple es que los parámetros sean variables. La definición de las variables puede hacerse con *val/var* o sin *val/var*. Si queremos que los parámetros definan al mismo tiempo propiedades es necesario hacerlo con *var/val* Es decir, en este caso el uso de *var/val* convierte a un parámetro variable en propiedad de la clase. Los parámetros también pueden ser funciones pero no ejemplificamos por el momento este caso para simplificar.

In [8]:
```
class Persona constructor(var nombre:String, var edad:Int){
   //nombre y edad son propiedades definidas en el constructor princ
   var email="chuchy@gmail.com" //propiedad no definida en construct
   fun printMe() {
      println(nombre+ " " + edad)
   }
}
val p1 = Persona("yo",15)
println(p1.nombre)//nombre es una propiedad y utilizamos el punto pa
println(p1.email)//email también es una propiedad aunque se declaro
p1.printMe()
val p2 = Persona("tu",35)
p2.printMe()
```

```
yo
chuchy@gmail.com
yo 15
tu 35
```

La palabra clave *constructor* se puede omitir si no hay anotaciones o modificadores de acceso especificados como público, privado o protegido.

In [3]:
```
class Persona (var nombre:String, var edad:Int){
   fun printMe() {
      println(nombre+ " " + edad)
   }
}
val p1 = Persona("yo",15)
p1.printMe()
val p2 = Persona("tu",35)
p2.printMe()
```

```
yo 15
tu 35
```

Se pueden inicializar los parámetros del constructor con valores predeterminados.

In [17]:
```
class Persona (var nombre:String, var edad:Int=90){
   fun printMe() {
      println(nombre+ " " + edad)
   }
}
val p1 = Persona("yo")
p1.printMe()
val p2 = Persona("tu",35)
p2.printMe()
```

```
yo 90
tu 35
```

parametros que no son propiedades

Sí no indicamos en los paréntesis del constructor los parámetros con val/var entonces es que preferimos definir dentro de la clase las propiedades y entre los paréntesis simplemente se indican variables que no tienen el rango de propiedades de la clase.

In [5]:
```
class Persona (elnombre:String, laedad:Int){
   var nombre=elnombre
   var edad=laedad
   fun printMe() {
      println(nombre+ " " + edad)
   }
}
val p1 = Persona("yo",15)
p1.printMe()
val p2 = Persona("tu",35)
p2.printMe()
```

```
yo 15
tu 35
```

Bloques init

El constructor principal simplemente tiene capacidad para hacer asignaciones de valores a variables. Podemos añadir lógica a la inicialización con un bloque *init*. Puede haber más de un bloque de inicialización durante la inicialización de una instancia, los bloques de inicialización se ejecutan en el mismo orden en que aparecen en el cuerpo de la clase.

In [6]:
```
class Persona (var nombre:String, var edad:Int){
   init{
       if(edad<0) edad=0
   }
   init{
       println("Segundo init")
   }
   fun printMe() {
      println(nombre+ " " + edad)
   }
}
val p1 = Persona("yo",-15)
p1.printMe()
val p2 = Persona("tu",35)
p2.printMe()
```

```
Segundo init
yo 0
Segundo init
tu 35
```

un uso típico de *this*

Indicamos que progresivamente iran saliendo casos que precisan el uso de *this*. Un caso muy habitual es que coincidan los nombres de los parámetros con los nombres de las propiedades. En este caso, el nombre del parámetro oculta al de la propiedad por lo que para referirnos a la propiedad debemos de utilizar la palabra reserva *this*.

En el siguiente ejemplo decidimos definir las propiedades dentro de la clase, no através de los parámetros. Observa que los parámetros del constructor no llevan var/val. Además, también a proposito, decidimos que coincidan los nombres de los parámetros y las propiedades. Inevitablemente, necesitamos usar this para referirnos a las propiedades.

In [4]:
```
class Persona (nombre:String,edad:Int){
   var nombre:String
   var edad:Int
   init{
       this.nombre=nombre
       this.edad=edad
       if(this.edad<0) this.edad=0
   }

   fun printMe() {
      println(nombre+ " " + edad)
   }
}
val p1 = Persona("yo",-15)
p1.printMe()
val p2 = Persona("tu",35)
p2.printMe()
```

```
yo 0
tu 35
```

Puedes comprobar modificando el ejemplo de arriba que los parámetros(variables sin indicar var/val) son realmente variables val, de forma que si queremos cambiar su valor dentro de la clase no es posible.

Otra posiblilidad sintáctica consiste en inicializar las propiedades con los parámetros justo en el momento de declarlos, en este caso no se puede usar this, al ir el nombre de la propiedad detrás de var no hay ambigüedad en el contexto de que es propiedad y que es parámetro.

In [1]:
```
class Persona (nombre:String,edad:Int){
   var nombre:String=nombre//no hay ambigüedad, no hace falta this (
   var edad:Int=edad//no hay ambigüedad, no hace falta this
   init{
      if(this.edad<0) this.edad=0
   }

   fun printMe() {
      println(nombre+ " " + edad)
   }
}
val p1 = Persona("yo",-15)
p1.printMe()
val p2 = Persona("tu",35)
p2.printMe()
```

```
yo 0
tu 35
```

Constructores secundarios

Recuerda que el constructor primario se declaraba en la cabecera de la clase con la palabra reservada *constructor*(opcional en ciertas situaciones). Un constructor secundario también se declara con la palabra reservada *constructor* pero se hace dentro del cuerpo de la clase.

En el siguiente ejemplo no hay constructor primario pero hay un constructor secundario

In [4]:
```
class Persona{
   var nombre:String
   var edad:Int

   constructor(nombre:String, edad:Int){
       this.nombre=nombre
       this.edad=edad
   }



   fun printMe() {
      println(nombre+ " " + edad)
   }
}
val p1 = Persona("yo",15)
p1.printMe()
val p2 = Persona("tu",35)
p2.printMe()
```

```
yo 15
tu 35
```

In [ ]:

Puede haber varios constructores secundarios

es un efecto similar a la sobrecarga de funciones

In [5]:
```
class Persona{
   var nombre="sin nombre"
   var edad:Int

   constructor(edad:Int){
       this.edad=edad
   }
   constructor(nombre:String, edad:Int){
       this.nombre=nombre
       this.edad=edad
   }



   fun printMe() {
      println(nombre+ " " + edad)
   }
}
val p1 = Persona("yo",15)
p1.printMe()
val p2 = Persona(35)
p2.printMe()
```

```
yo 15
sin nombre 35
```

puedem coexistir simultáneamente constructor primario con secundarios

En este caso es obligatorio usar la expresión *this* en la cabecera del secundario para delegarle los parámetros que requiera.

In [2]:
```
class Persona(var nombre:String){
   var edad:Int
   init{
       edad=99
   }

   constructor(nombre:String, edad:Int):this(nombre) {

       this.edad=edad
   }



   fun printMe() {
      println(nombre+ " " + edad)
   }
}
val p1 = Persona("yo",15)
p1.printMe()
val p2 = Persona("tu")
p2.printMe()
```

```
yo 15
tu 99
```