2º DAM

Programación Multimedia y Dispositivos Móviles

Programación Android

José A. Lara

Contenido

POO en Kotlin

Programación Android

Clases en Kotlin

Para definir una clase utilizamos la palabra reservada ***class***. Si queremos incluir parámetros, podemos hacerlo en la cabecera de la clase (se comporta como el constructor de la clase).

```
fun main(){
    val persona:Persona = Persona( nombre: "Pepe", apellidos: "Lopez")
    persona.saludar()
}

class Persona(private val nombre:String,private val apellidos:String) {
    fun saludar(){
        println ("Bienvenido $nombre $apellidos")
    }
}
```

```
"C:\Program Files\Android\Android S
Bienvenido Pepe Lopez

Process finished with exit code 0
```

android

Programación Android

Clases en Kotlin

Para crear un constructor vacío, se proporcionará un valor por defecto, pero para poderles asignar valores posteriormente, tenemos que quitar ***private*** (modificador de acceso) y hacer los parámetros de tipo ***var***.

```
fun main(){
    val persona:Persona = Persona()
    persona.nombre="Juan"
    persona.apellidos="Perez"
    persona.saludar()
}

class Persona(var nombre:String="",var apellidos:String="") {
    fun saludar(){
        println ("Bienvenido $nombre $apellidos")
    }
}
```

```
"C:\Program Files\Android\Android S
Bienvenido Juan Perez

Process finished with exit code 0
```

android

Programación Android

Clases en Kotlin

Podemos crear además otro constructor además del creado implícitamente en la cabecera de la clase.

```
class School(var name:String, var address:String, val active:Boolean=true, var numCode:String="") {

    constructor(): this( name: "", address: "")

    fun hello(){
        println("Hello")
    }
}
```

android

Programación Android

Sobrecarga de métodos

```
fun main(){
    showProduct( name: "Pan",  promo: "10%")
    showProduct( name: "Pera")
    showProduct( name: "Chocolate", promo: "5%", validity: "agotar existencias")
    showProduct( name: "Pasta",validity = "5 de junio")
}

fun showProduct(name:String, promo:String="sin promoción", validity:String ="agotar existencias"){
    println("$name = $promo hasta $validity")
}
```

```
"/Applications/Android Studio.app/Contents/jre/
Pan = 10% hasta agotar existencias
Pera = sin promoción hasta agotar existencias
Chocolate = 5% hasta agotar existencias
Pasta = sin promoción hasta 5 de junio
```

android

Programación Android

Sobrecarga de métodos

Podemos sobreescribir métodos con la palabra reservada ***override***.

```
class School(var name:String, var address:String, val active:Boolean=true, var numCode:String="") {
    constructor(): this( name: "", address: "")

    override fun toString(): String {
        if (active) {
            return "name: $name at $address"
        } else {
            return "Escuela $name inactiva"
        }
    }
}
```

```
fun main(){
    val school=School( name: "Harv", address: "Calle Principal,14")
    println(school)

    val schoolInactive=School( name: "Harv", address: "Calle Principal,14", active: false)
    println(schoolInactive)
}
```

```
"C:\Program Files\Android\Android S
name: Harv at Calle Principal,14
Escuela Harv inactiva

Process finished with exit code 0
```

android

Programación Android

this

```
class School(var name:String, var address:String, val active:Boolean=true, var numCode:String="") {
    var staff:MutableList<Person>
    init{
        staff = mutableListOf()
    }

    constructor(): this( name: "", address: "")

    constructor(name:String, address: String, staff:MutableList<Person>): this(name,address){
        this.staff=staff
    }

    override fun toString(): String {
        if (active) {
            return if (staff.size>0){
                "name: $name at $address, with ${staff.size} members"
            } else {
                "name: $name at $address"
            }
        } else {
            return "Escuela $name inactiva"
        }
    }
}
```

```
class Person(val firstName:String, val lastName:String) {
}
```

Programación Android

this

```
val school=School( name: "Harv", address: "Calle Principal,14")
println(school)

val schoolInactive=School( name: "Harv", address: "Calle Principal,14", active: false)
println(schoolInactive)

val highSchool=School( name: "Stan", address: "Constitución, 232",
    mutableListOf(Person("Jose","Lara")))
println(highSchool)
```

```
"C:\Program Files\Android\Android Studio\jre\bin
name: Harv at Calle Principal,14
Escuela Harv inactiva
name: Stan at Constitución, 232, with 1 members

Process finished with exit code 0
```

android

Programación Android

Métodos y propiedades

```
class Person(val firstName:String, val lastName:String) {
    var tax:Float=10.0f

    var salary:Float=0f

    fun getFullName():String="$firstName $lastName"

    fun showWork():String{
        return "Capturando datos..."
    }
}
```

```
val person=Person("Pepe","Grillo")
println(person.getFullName())
println(person.showWork())
person.salary=1000f
println(person.salary)
```

```
"C:\Program Files\Android\Android
Pepe Grillo
Capturando datos...
1000.0

Process finished with exit code 0
```

android

Programación Android

Set y Get

```
class Person(val firstName:String, val lastName:String) {
    var tax:Float=10.0f
    get() = 1-(field*0.01f)

    var salary:Float=0f
    set(value) {
        field = value
    }
    get() = field * tax

    fun getFullName():String="$firstName $lastName"

    fun showWork():String{
        return "Capturando datos..."
    }
}
```

```
fun main(){
    val person=Person("Pepe","Grillo")
    println(person.getFullName())
    println(person.showWork())
    person.salary=1000f
    println(person.salary)
}
```

```
"C:\Program Files\Android\Android
Pepe Grillo
Capturando datos...
900.0

Process finished with exit code 0
```

android

Programación Android

POO en Kotlin

```
class Telefono {
    fun llamar(){
        println("Llamando...")
    }
}
```

```
fun main(){
    val tfno:Telefono=Telefono()
    tfno.llamar()
}
```

```
class Telefono(val numero:String) {
    fun llamar(){
        println("Llamando...")
    }

    fun verNumero(){
        println(numero)
    }
}
```

```
fun main(){
    val tfno:Telefono=Telefono( numero: "952876321")
    tfno.llamar()
    tfno.verNumero()
}
```

```
"/Applications/Android Studio.app/
Llamando...
952876321

Process finished with exit code 0
```

android

Programación Android

Herencia

```
open class Telefono(protected val numero:String) {
    fun llamar(){
        println("Llamando...")
    }

    open fun verNumero(){
        println("El número es el $numero")
    }
}
```

```
class Smartphone(numero:String,val esPrivado:Boolean):Telefono(numero) {
    override fun verNumero(){
        if (esPrivado)
            println("Desconocido")
        else
            super.verNumero()
    }
}
```

```
fun main(){
    val tfno:Telefono=Telefono( numero: "952876321")
    tfno.llamar()
    tfno.verNumero()
    val smart:Smartphone= Smartphone( numero: "678456231", esPrivado: true)
    smart.llamar()
    smart.verNumero()
}
```

```
"/Applications/Android Studio.app/
Llamando...
El número es el 952876321
Llamando...
Desconocido

Process finished with exit code 0
```

android

Programación Android

Herencia - Encapsulamiento

Si no se especifica nada, significa que es público

```
open class Person(val firstName:String, private val lastName:String) {
    var tax:Float=10.0f
    get() = 1-(field*0.01f)

    var salary:Float=0f
    set(value) {
        field = value
    }
    get() = field * tax

    fun getFullName():String="$firstName $lastName"

    open fun showWork():String{
        return "Capturando datos..."
    }
}
```

Programación Android

Funciones de alcance - with

Su significado es: “con este objeto haz algo…”

```
fun main(){
    val tfno:Telefono=Telefono( numero: "952876321")
    val smart:Smartphone=Smartphone( numero: "678456231", esPrivado: true)

    with(smart){  this: Smartphone
        println("Privado? $esPrivado")
        llamar()
    }
}
```

```
"/Applications/Android Studio.app/
Privado? true
Llamando...

Process finished with exit code 0
```

android

Programación Android

Funciones de alcance - with

```
class smartphone(numero:String, var marca:String): telefono(numero) {
    lateinit var modelo: String
    fun infoSmartphone(){
        with( receiver: this){  this: smartphone
            if (asignarValor()) {
                println(toString())
            }
        }
    }
    private fun asignarValor(): Boolean{
        if (!this::modelo.isInitialized){
            modelo="Desconocido"
        }
        return modelo.isNotEmpty()
    }

    override fun toString(): String {
        return "N° $numero - Marca $marca - Modelo $modelo"
    }
}
```

```
open class telefono(val numero:String) {
    fun llamar(){
        println("Llamando...")
    }
    fun verNumero(){
        println(numero)
    }
}
```

android

Programación Android

Funciones de alcance - apply

Su significado es: “aplica las configuraciones…”

```
fun main(){
    var tfno:telefono=telefono( numero: "952876321")

    with(tfno){ this: telefono
        verNumero()
        llamar()
    }

    var smart:smartphone= smartphone( numero: "678012345", marca: "iPhone")
    smart.infoSmartphone()
    smart.apply { this: smartphone
        modelo="12 Pro"
        marca="Apple iPhone"
    }
    smart.infoSmartphone()
}
```

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...
952876321
Llamando...
N° 678012345 - Marca iPhone - Modelo Desconocido
N° 678012345 - Marca Apple iPhone - Modelo 12 Pro

Process finished with exit code 0
```

android

Programación Android

Funciones de alcance - run

Su significado es: “ejecuta el siguiente bloque…”

```
var smart:smartphone= smartphone( numero: "678012345", marca: "iPhone")
smart.infoSmartphone()
smart.apply { this: smartphone
    modelo="12 Pro"
    marca="Apple iPhone"
}
smart.infoSmartphone()
smart.run{ this: smartphone
    modelo="12 mini"
    infoSmartphone()
}
```

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...
952876321
Llamando...
N° 678012345 - Marca iPhone - Modelo Desconocido
N° 678012345 - Marca Apple iPhone - Modelo 12 Pro
N° 678012345 - Marca Apple iPhone - Modelo 12 mini

Process finished with exit code 0
```

android

Programación Android

Funciones de alcance - let

Permite ejecutar un bloque de código no nulo verificado con ?

```
smart.infoSmartphone()
smart.run{ this: smartphone
    modelo="12 mini"
    infoSmartphone()
}

var movil:smartphone? = null
movil?.let { it: smartphone
    movil.modelo="S21"
    movil.marca="Samsung"
    println("Si el objeto es diferente de null, permite imprimirlo $(it)")
}
```

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...
952876321
Llamando...
N° 678012345 - Marca iPhone - Modelo Desconocido
N° 678012345 - Marca Apple iPhone - Modelo 12 Pro
N° 678012345 - Marca Apple iPhone - Modelo 12 mini

Process finished with exit code 0
```

android

Programación Android

Funciones de alcance - let

Permite ejecutar un bloque de código no nulo verificado con ?

```
var movil:smartphone? = smartphone( numero: "601987543",  marca: "Samsung")
movil?.let {  it: smartphone
    movil.modelo="S21"
    movil.marca="Samsung"
    println("Si el objeto es diferente de null, permite imprimirlo ${it}")
}
```

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...
952876321
Llamando...
N° 678012345 - Marca iPhone - Modelo Desconocido
N° 678012345 - Marca Apple iPhone - Modelo 12 Pro
N° 678012345 - Marca Apple iPhone - Modelo 12 mini
Si el objeto es diferente de null, permite imprimirlo N° 601987543 - Marca Samsung - Modelo S21

Process finished with exit code 0
```

android

Programación Android

Funciones de alcance - also

Su significado es: “Y además, ejecuta el siguiente código…”

```
var miMovil:smartphone = smartphone( numero: "661199882",  marca: "Xiaomi")
miMovil.modelo="Redmi 9".also {  it: String
    println("Mi móvil es el ${it}")
}
```

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...
952876321
Llamando...
N° 678012345 - Marca iPhone - Modelo Desconocido
N° 678012345 - Marca Apple iPhone - Modelo 12 Pro
N° 678012345 - Marca Apple iPhone - Modelo 12 mini
Si el objeto es diferente de null, permite imprimirlo N° 601987543 - Marca Samsung - Modelo S21
Mi móvil es el Redmi 9

Process finished with exit code 0
```

android

Programación Android

Constantes y métodos estáticos (companion object)

De esta forma Kotlin hace que las constantes sean estáticas

```
companion object{
    const val ACTIVE = true
    const val INACTIVE = false
}
```

```
class School(var name:String, var address:String, val active:Boolean=ACTIVE, var numCode:String="") {
    var staff:MutableList<Person>
    init{
        staff = mutableListOf()
    }

    constructor(): this( name: "", address: "")

    constructor(name:String, address: String, staff:MutableList<Person>): this(name,address){
```

Programación Android

Clases anidadas

```
class Teacher(firstName:String, lastName:String, var Students:Short) : Person(firstName, lastName){

    var classroom: Classroom = Classroom( key: "N/A")

    override fun showWork(): String {
        return "Agendando"
    }

    class Classroom(var key:String){
        override fun toString(): String {
            return "Clasroom: $key"
        }
    }
}
```

```
fun main(){
    val teacher=Teacher( firstName: "Pepito", lastName: "Grillo", Students: 20)
    println(teacher.classroom)
    teacher.classroom.key="4°A"
    println(teacher.classroom)
}
```

```
"C:\Program Files\Android\Android
Clasroom: N/A
Clasroom: 4°A

Process finished with exit code 0
```

android

Programación Android

Inner Class

Al ser una clase ***inner*** podrá acceder a los atributos de la clase padre.

```
class Teacher(firstName:String, lastName:String, var students:Short) : Person(firstName, lastName){

    var classroom: Classroom = Classroom( key: "N/A")

    override fun showWork(): String {
        return "Agendando"
    }

    inner class Classroom(var key:String){
        override fun toString(): String {
            return "Clasroom: $key"
        }

        fun getInfo():String="Classroom ${key} with teacher $firstName and $students students"
    }

}
```

```
fun main(){
    val teacher=Teacher( firstName: "Pepito", lastName: "Grillo", students: 20)
    println(teacher.classroom)
    teacher.classroom.key="4°A"
    println(teacher.classroom)
    println(teacher.classroom.getInfo())
}
```

```
"C:\Program Files\Android\Android Studio\jre\bin\j
Clasroom: N/A
Clasroom: 4°A
Classroom 4°A with teacher Pepito and 20 students

Process finished with exit code 0
```

android

Programación Android

Interface

```
class Teacher(firstName:String, lastName:String, var students:Short) : Person(firstName, lastName),
    Boss{

    var classroom: Classroom = Classroom( key: "N/A")

    override fun showWork(): String {
        return "Agendando"
    }

    override fun namePerson(): String = getFullName()

    override fun netSalary(): Float = salary

    inner class Classroom(var key:String){
        override fun toString(): String {
            return "Clasroom: $key"
        }

        fun getInfo():String="Classroom ${key} with teacher $firstName and $students students"
    }
}
```

```
interface Boss {
    fun namePerson():String
    fun netSalary():Float
}
```

android

Programación Android

Interface

```
fun main(){
    val teacher=Teacher( firstName: "Pepito", lastName: "Grillo", students: 20)
    println(teacher.classroom)
    teacher.classroom.key="4°A"
    println(teacher.classroom)
    println(teacher.classroom.getInfo())
    teacher.salary=1000f
    val boss: Boss=teacher
    println(boss.namePerson())
    println(boss.netSalary())
}
```

```
"C:\Program Files\Android\Android Studio\jre\bin\ja
Clasroom: N/A
Clasroom: 4°A
Classroom 4°A with teacher Pepito and 20 students
Pepito Grillo
900.0

Process finished with exit code 0
```

android

Programación Android

Data class

Son clases que se utilizan para almacenar algún dato en concreto

```
fun main(){
    val usuario = Usuario( nombre: "Pepe",  edad: 25)
    println(usuario.toString())
}

data class Usuario(val nombre:String, val edad:Int)
```

```
"C:\Program Files\Android\Android S
Usuario(nombre=Pepe, edad=25)

Process finished with exit code 0
```

android

Programación Android

Data class

Para generar otro objeto Usuario al que le cambiamos uno de los valores podemos utilizar la función ***copy***.

```
fun main(){
    val usuario = Usuario( nombre: "Pepe",  edad: 25)
    val usuario2 = usuario.copy(edad=20)
    println(usuario2.toString())
}

data class Usuario(val nombre:String, val edad:Int)
```

```
"C:\Program Files\Android\Android St
Usuario(nombre=Pepe, edad=20)

Process finished with exit code 0
```

android

Programación Android

Data class

Para saber si son iguales dos objetos, utilizaremos el operador ==

```
fun main(){
    val usuario = Usuario( nombre: "Pepe",  edad: 25)
    val usuario2 = usuario.copy(edad=20)
    println("Son iguales: ${usuario==usuario2}")
}

data class Usuario(val nombre:String, val edad:Int)
```

```
"C:\Program Files\Android\Android
Son iguales: false

Process finished with exit code 0
```

android

Programación Android

Equals y Hashcode

```
override fun equals(other: Any?): Boolean {
    if (this === other) return true
    if (javaClass != other?.javaClass) return false
    other as Teacher
    if (getFullName() != other.getFullName()) return false
    return true
}

override fun hashCode(): Int {
    return getFullName().hashCode()
}
```

```
fun main(){
    val teacher=Teacher( firstName: "Pepito", lastName: "Grillo", students: 20)
    println(teacher.classroom)
    teacher.classroom.key="4ºA"
    println(teacher.classroom)
    println(teacher.classroom.getInfo())
    teacher.salary=1000f
    val boss: Boss=teacher
    println(boss.namePerson())
    println(boss.netSalary())
    val teacher2=Teacher( firstName: "Juanita", lastName: "Campanilla", students: 20)
    println(teacher.equals(teacher2))
    println(teacher.equals(teacher))
}
```

android

Programación Android

Equals y Hashcode

```
"C:\Program Files\Android\Android Studio\jre\bin\ja
Clasroom: N/A
Clasroom: 4ºA
Classroom 4ºA with teacher Pepito and 20 students
Pepito Grillo
900.0
false
true

Process finished with exit code 0
```

android

Programación Android

Operador ===

```kotlin
fun main(){
    val text1 = "text"
    val text2 = String("text".toCharArray())
    println(text1 == text2) // true
    println(text1 === text2) // false
}
```

```
PruebaKt
"C:\Program Files\Android\Android S
true
false

Process finished with exit code 0
```

android

Ejercicios Kotlin

Programación Android

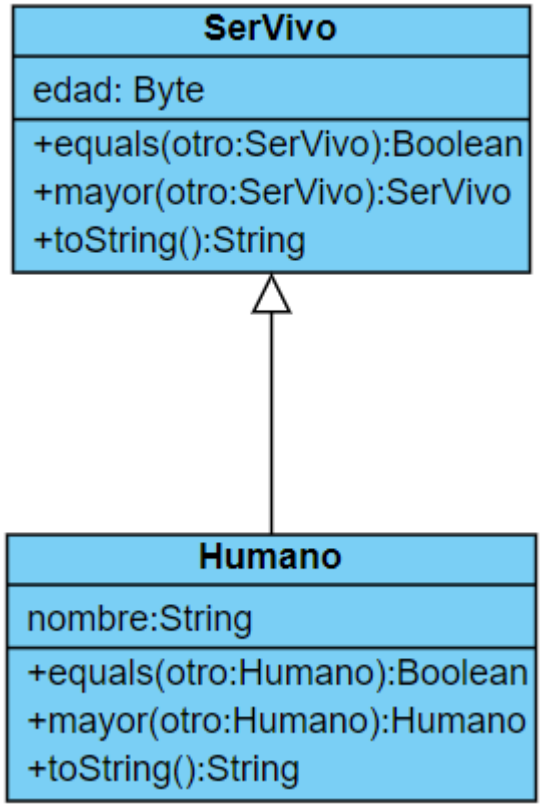
Ejercicio 1

a) Implementar las clases SerVivo y Humano en Kotlin.

El método **mayor** DEBE DEVOLVER un objeto de la clase SerVivo.
En el caso de la versión del método **mayor** que está en la clase SerVivo va a devolver al SerVivo (de entre los dos que se comparan) de MAYOR EDAD.
En el caso de la versión del método que se encuentra en la clase Humano DEBERÁ DEVOLVER al humano (de entre los dos que se comparan) cuya edad sea mayor. En caso de tener la misma edad, devolverá al humano cuyo nombre sea más largo.

El método **toString()** DEBE devolver un String que contenga TODOS los datos del objeto.
Para implementar el método **equals** DEBES SABER QUE:

- Dos seres vivos se consideran iguales si tienen la misma edad
- Dos humanos se consideran iguales si tienen la misma edad y el mismo nombre.



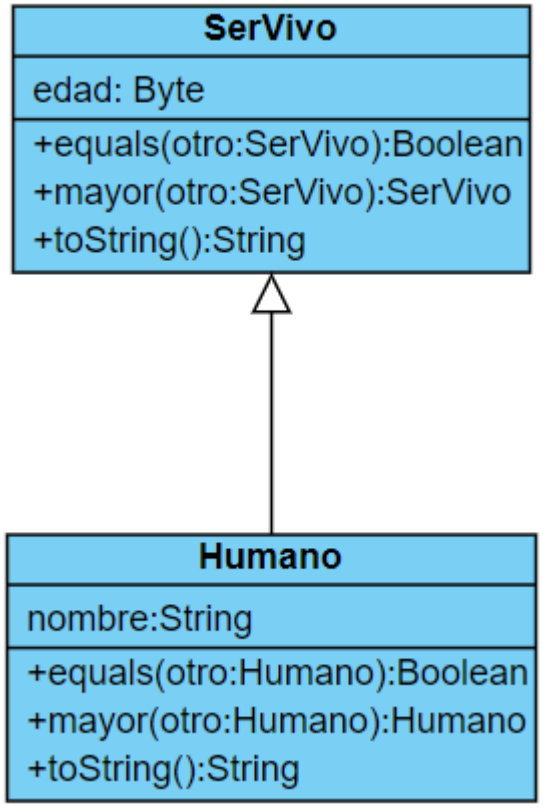
android

Programación Android

Ejercicio 1

b) En un archivo Main.kt, crear una función main() que cumpla con las siguientes instrucciones:

- Declarar DOS objetos de la clase SerVivo:
  - Objeto X tiene 3 años
  - Objeto Y tiene 5 años.
- Usando el método mayor() DEBERÁ imprimir el mayor SerVivo de los creados anteriormente.
- Crear DOS OBJETOS DE LA CLASE Humano y ASIGNARLOS A LAS VARIABLES DECLARADAS PREVIAMENTE. Los objetos son:
  - Objeto X: nombre Homero, 34 años
  - Objeto Y: nombre Bart, 9 años
- Usando el resultado devuelto por el método mayor() DEBERÁ imprimir el mayor Humano de los creados anteriormente.



android

Programación Android

Ejercicio 2

a) Crear una clase **Persona**, con los siguientes atributos:
   - nombre
   - apellido
   - teléfono (controlar que el teléfono solo acepte 9 dígitos)

b) Crear una clase **Cuenta**, con los siguientes atributos:
   - numeroCuenta
   - saldo
   - propietario. El propietario es un objeto tipo Persona.

La clase Cuenta debe tener un constructor con parámetros y otro sin parámetros, métodos getter y setter y debes sobreescribir **toString()** para estas clases. Tenemos que controlar que el **saldo de la cuenta** no debe ser menor que 0.

Ejercicio 2

Crear un método llamado **transaccion** que tenga como parámetros **cantidad** y **tipoTransaccion**. El parámetro **tipoTransaccion** puede ser “reintegro” o “ingreso”. Si es un reintegro, la cantidad se resta del saldo, y si es ingreso la cantidad se incrementa al saldo. El método transacción debe imprimir el tipo de transacción y el nuevo saldo.

En un archivo Main.kt, crear una función main(), crear dos cuentas pertenecientes a dos personas distintas (que también debemos crear) y hacer un ingreso y un reintegro en cada cuenta. Imprimir los valores de las personas, propietarios y transacciones.

Programación Android

Ejercicio 3

Crear un programa Kotlin que permita la gestión de una empresa agroalimentaria que trabaja con tres tipos de productos: productos **frescos**, productos **refrigerados** y productos **congelados**.

Todos los productos llevan esta información común: **fecha de caducidad (String)** y **número de lote**. A su vez, cada tipo de producto lleva alguna **información específica**. Los productos **frescos** deben llevar la **fecha de envasado(String)** y el **país de origen**. Los productos **refrigerados** deben llevar el **código del organismo de supervisión alimentaria**. Los productos **congelados** deben llevar la **temperatura de congelación recomendada**.

Crear el código de las clases Kotlin implementando una relación de herencia desde la superclase **Producto** hasta las subclases **ProductoFresco**, **ProductoRefrigerado** y **ProductoCongelado**.

Cada clase debe disponer de **constructor**(es) y permitir establecer (**set**) y recuperar (**get**) el valor de sus atributos y tener un **método** que permita **mostrar la información** del objeto.
Crear un archivo Kotlin Main con el método **main** donde se cree un objeto de cada tipo y se muestren los datos de cada uno de los objetos creados.