Ejemplos adicionales:

**1. Objetos mutables en Scala**

Scala tiene una clase ArrayBuffer que permite crear colecciones mutables.

object Main {  
 def main(args: Array[String]): Unit = {  
 import scala.collection.mutable.ArrayBuffer  
  
 val buffer = ArrayBuffer[Int](1, 2, 3)  
 buffer += 4 // Añade un elemento  
 *println*(buffer) // Output: ArrayBuffer(1, 2, 3, 4)  
  
 buffer(0) = 10 // Modifica el primer elemento  
 *println*(buffer) // Output: ArrayBuffer(10, 2, 3, 4)  
 }  
}

**2. Get y Set en Scala**

En Scala, puedes definir métodos get y set utilizando propiedades.

class Persona(private var *nombre*: String) {  
 def getNombre: String = *nombre* def setNombre(nuevoNombre: String): Unit = {  
 *nombre* = nuevoNombre  
 }  
}  
  
object Main {  
 def main(args: Array[String]): Unit = {  
 val persona = new Persona("Juan")  
 *println*(persona.getNombre) // Output: Juan  
 persona.setNombre("Pedro")  
 *println*(persona.getNombre) // Output: Pedro  
 }  
}

**3. Objetos públicos y privados en Scala**

Puedes definir miembros de clases como públicos o privados.

class Caja(private var *contenido*: String) {  
 def getContenido: String = *contenido* // Mé*todo público* private def setContenido(nuevoContenido: String): Unit = {  
 *contenido* = nuevoContenido // Mé*todo privado* }  
}  
  
object Main {  
 def main(args: Array[String]): Unit = {  
   
 val caja = new Caja("Juguetes")  
 *println*(caja.getContenido) // Output: Juguetes  
 // caja.setContenido("Comida") // Esto dará un error porque setContenido es privado  
 }  
}

**4. Override utilizando una funcionalidad del padre**

Aquí tienes un ejemplo de cómo puedes sobreescribir un método y utilizar la funcionalidad de la clase padre:

class Animal {  
 def hacerSonido(): String = "Sonido de animal"  
}  
  
class Perro extends Animal {  
 override def hacerSonido(): String = {  
 super.hacerSonido() + " y guau!"  
 }  
}  
  
object Main{  
 def main(args: Array[String]): Unit = {  
 val perro = new Perro  
 *println*(perro.hacerSonido()) // Output: Sonido de animal y guau!  
 }  
}

**5. Método Super en Scala**

class Vehiculo {  
 def info(): String = "Soy un vehículo"  
}  
  
class Coche extends Vehiculo {  
 override def info(): String = {  
 super.info() + ", y soy un coche"  
 }  
}  
object Main{  
 def main(args: Array[String]): Unit = {  
 val coche = new Coche  
 *println*(coche.info()) // Output: Soy un vehículo, y soy un coche  
 }  
}

**6. Ejemplo de constructor personalizado en Scala**

class Vehiculo {  
 def info(): String = "Soy un vehículo"  
}  
  
class Coche extends Vehiculo {  
 override def info(): String = {  
 super.info() + ", y soy un coche"  
 }  
}  
object Main{  
 def main(args: Array[String]): Unit = {  
 val coche = new Coche  
 *println*(coche.info()) // Output: Soy un vehículo, y soy un coche  
 }  
}

**7. Override utilizando una funcionalidad del padre**

class Animal {  
 def hacerSonido(): String = "Sonido de animal"  
}  
  
class Perro extends Animal {  
 override def hacerSonido(): String = {  
 super.hacerSonido() + " y guau!"  
 }  
}  
object Main{  
 def main(args: Array[String]): Unit = {  
 val perro = new Perro  
 *println*(perro.hacerSonido()) // Output: Sonido de animal y guau!  
 }  
}

Links de Interés:

* <https://docs.scala-lang.org/es/tour/singleton-objects.html>
* <https://docs.scala-lang.org/es/tour/classes.html>