# Programación orientada a Objetos en Scala

S4N Campus

2 de marzo de 2021

## 1. Preliminares

Este taller tiene como objetivo poner en practica los conceptos vistos en clase sobre el tema de Clases en Scala.

## 2. Literales objetos

**Ejercicio 1.** Defina un objeto llamado `comp` con un método llamado `cuadrado` que acepte un valor de tipo flotante (`Float`) y otro método `cubo` que acepte un valor de tipo doble y utilizando el método `cuadrado` compute el cubo del valor entrado. □

**Ejercicio 2.** Pegue y copie el anterior objeto y lo renombra `comp2`, pero se debe cambiar todos los tipos para que utilicen valores de tipo `Long`. □

**Ejercicio 3.** Escriba el siguiente código y utilizando el `REPL` pruebe la salida del siguiente programa, mostrando el orden de ejecución de las instrucciones.

```
object prueba {
  def x = {
    println("x")
    1
  }
  val y = {
    println("y")
    x + 2
  }
  def z = {
```

```
11      println("z")
        x
13      x + "c"
    }
15  }
```

Una vez cargado el objeto en el `REPL` ejecute la siguiente instrucción.

```
scala> prueba.x + prueba.y + prueba.z
```

□

# 3. Objetos y clases

**Ejercicio 4.** Dada la siguiente tabla definir la clase `Gato` y crear un objeto de cada gato de la siguiente tabla.

| **Nombre** | **Color** | `Comida` |
|---|---|---|
| IO | Fawn | Churrus |
| Make | Red | Leche |
| Docker | Blue | Cuido |

□

**Ejercicio 5.** Defina un objeto VentaDeChurrus con un método despachar. Este método debe aceptar un gato y retornar `true` (Verdadero) si la comida favorita del gato son los Churrus y falso de otra forma. □

**Ejercicio 6.** Implementar las siguiente clases que se observan en la figura 1
□

**Ejercicio 7.** Implementar la clase `Contador` que se observa en la figura 2. El constructor de la clase debe tomar un entero. Los métodos `incr` y `decr` deben retonar ambos un nuevo contador `Counter`.

Aquí un ejemplo de uso:

```
scala > new Counter(10).incr.decr.inc.inc.contador
```

□

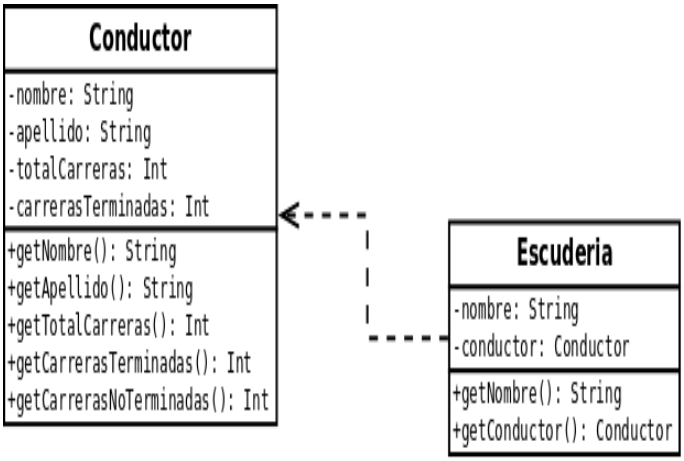


Figura 1: Relación de clases Escudería Conductor

**Ejercicio 8.** Aumentar la clase `Counter` del anterior ejercicio que permitas al usuario opcionalmente pasar un parámetro un `Int` cómo parámetro da `incr` y `decr`. Si el parámetro es definido este debe ser un valor por omisión de 1. □

**Ejercicio 9.** La siguiente código:

```
class Sumador(monto: Int) {
  def adicionar(valor: Int) = valor + monto
}
```

Aumente la clase `Contador` para adicionar un método que sea llame `ajuste`. Este método debe aceptar un `Sumador` y retornar un nuevo contador `Contador` con el resultado de aplicar el `Sumador` a el `Contador`.

# 4. Objetos de compañía

**Ejercicio 10.** El siguiente es el código de la clase `Persona`

```
class Persona(val nombre: String, val apellido: String) {
  def nombre = s"$nombre $apellido"
}
```

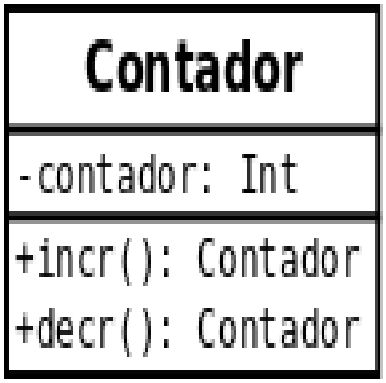


Figura 2: Clase contador

Implemente un objeto de compañia para la clase `Persona` que contenga una método `apply` que acepte todo el nombre como una sola cadena en vez de un nombre separado por espacios.

**Pista:** Se puede dividir una cadena dentro de un arreglo de la siguiente forma:

```
scala> val partes = "Juan Cardona".split(" ")
partes:res0: Array[String] = Array(Juan, Cardona)
scala> partes(0)
partes(0):res1: String = Juan
```

□

**Ejercicio 11.** La siguientes son las definiciones de dos clases: `Director` y `Película`.

```
class Director (
    val nombre: String,
    val apellido: String,
    val nacimiento: Int) {
  def nombre: String = s"$nombre $apellido"
  def copy(nombre: String  = this.nombre
           apellido: String = this.apellido,
           nacimiento: Int = nacimiento):Director =
      new Director(nombre, apellido, nacimiento)
}

class Pelicula (
    val nombre: String
```

```
    val presentacion: Int,
    val rangoIMDB: Double
    val director: Director) {

  def directorEdad = presentacion - director.nacimiento
  def esDirigidaPor(director:Director) =
      this.director == director
  def copy(
      nombre:String = this.nombre,
      presentacion = this.presentacion,
      rangoIMDB:Double = this.rangoIMDB,
      director:Director = this.director):Pelicula =
      new Pelicula(nombre,presentacion,rangoIMDB,director)
}
```

Escriba objetos de compañía para las clases `Director` y `Pelicula` como sigue:

Para `Director`:

- Un método `apply` que acepte los mismos parámetros del constructor de la clase y retorne un nuevo `Director`.
- Un método `esMayor` que acepte dos `Director`(es) y retorne el mayor de los dos.

Para `Pelicula`:

- Un método `apply` que acepte los mismos parámetros del constructor de la clase y retorne un nuevo `Pelicula`.
- Un método `mejorCalificada` que acepte dos `Pelicula`(s) y retorna la que tiene mayor `rangoIMDB` entre las dos.
- Un método `mayorDirectorEnElTiempo` que acepte dos `Pelicula`(s) y retorne el `Director` que fue mayor en el momento de presentar la película.

□