Introducción a la Programación Orientada a Objeto con Kotlin

Guillermo Palma

Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información

Plan

Introducción

Objetivo

Se quiere introducir algunos de los principios básicos de la Programación Orientada a Objetos con Kotlin, para poder crear nuevos tipos de datos, específicamente para crear colas, pilas, árboles, tablas de hash, colas de prioridad, grafos, entre otras estructuras que se verán durante los cursos de algoritmos y estructuras de datos.

Programación Orientada a Objetos (POO)

Definición de la POO

De Wikipedia tenemos la siguiente definición [1]:

> *La programación orientada a objetos, es un paradigma de programación basado en el concepto de "objetos", los cuales son estructuras de datos que contienen data, en forma de campos conocidos como* atributos*; y código en forma de procedimientos, conocidos como* métodos*. Una característica de los objetos es que los procedimientos de un objeto, pueden acceder y modificar los campos de datos de los objetos con los cuales ellos están asociados.*

[1] https://en.wikipedia.org/wiki/Object-oriented_programming

Terminología de la POO

Clase: Mecanismo para definir un nuevo tipo de datos, que contiene un conjunto de datos (**atributos**) y procedimientos (**métodos**).

Instancia: Un objeto de cierta clase. Ej. un objeto p de la clase Persona es una instancia de la clase Persona.

Variable de clase: Variables que comparten todas las instancias de una clase.

Variable de instancia: Variables que contiene datos de los objetos (atributos) y están asociadas a la instancia actual de la clase.

Método: función o procedimiento definido en una clase.

Objeto : Una instancia única de una estructura de datos que está definida por su clase.

Ejemplo: Clase `Persona`

Clase: Se define el tipo de datos `Persona`.

Instancias: *Fabian* y *Diana* son instancias de la clase `Persona`.

Variables de clase: Número de instancias de tipo persona creadas, en este caso el valor es 2.

Variable de instancia o atributos :

- Nombre
- Cédula de identidad
- Fecha de Nacimiento
- Trabajos
- Sueldo

Métodos:

- Obtener nombre
- Cambiar nombre
- Obtener atributo X (C.I., Fecha Nacimiento, etc)
- Cambiar atributo Y (C.I., Fecha Nacimiento, etc)
- Calcular edad
- Dar aumento de sueldo

POO con Kotlin

Sobre la POO con Kotlin

- Kotlin soporta múltiples paradigmas de programación, incluyendo la programación orientada a objetos.
- Los principios de la programación orientada a objetos están implementados de manera clara y concisa en Kotlin.

Sobre la implementación de la clase `Persona`

- Se desea hacer la implementación de la clase `Persona` en Kotlin
- Se mostrará como crear un nuevo tipo de datos `Persona` a través de una clase `Persona`
- Se presentará el constructor de la clase que crea un nuevo objeto `Persona`
- Se crearán los métodos de la clase `Persona`
- Se implementará una variable global de clase que cuenta en número de personas creadas
- Se mostrará un cliente que hace uso de objetos de tipo `Persona`
- Se mostrará el código en el mismo orden en el que aparace en los archivos fuentes

Clases importadas para la clase `Persona`

```
1  import java.time.LocalDate
2  import java.time.format.DateTimeFormatter
3  import java.time.Period
4  import java.util.Arrays
```

Constructor de la clase `Persona`

```
class Persona(nombreParam: String,
              val ci: String,
              val fNac: LocalDate,
              val trabajos: Array<String>,
              sueldoParam: Double) {
```

Variable global de clase de la clase `Persona`

```
    companion object {
        var numeroDePersonas = 0
    }
```

Método para cambiar el nombre de la Persona

```
    var nombre = nombreParam
        set(value) {
            field = value
        }
```

Inicialización de la clase `Persona`

```
init {
  println("Una nueva persona llamada ${nombre} fue creada")
  ++numeroDePersonas
}
```

Método para cambiar el sueldo de la Persona

```
1     var sueldo = sueldoParam
2     set(value) {
3       if (value >= 0) {
4         field = value
5       }
6     }
```

Sueldo en divisas de la Persona

```
1     val sueldoEnDivisas: Double
2         get() = sueldo / 4.30
```

Método para dar un aumento a la Persona

```
1    fun darAumento(porcentaje: Int) {
2      sueldo += sueldo*porcentaje/100.0
3    }
```

Método para obtener la edad de una Persona

```
1    fun obtenerEdad() : Int {
2      val today = LocalDate.now()
3      val period = Period.between(fNac, today)
4      return period.getYears()
5    }
```

String que representa a una clase `Persona`

```
    override fun toString() : String {
      val fnStr = fNac.format(DateTimeFormatter.ofPattern("
    dd/MM/yyyy"))
      return """
      Nombre: ${nombre}
      CI: ${ci}
      Fecha de nacimiento: ${fnStr}
      Trabajos: ${Arrays.toString(trabajos)}
      Sueldo: ${sueldo}
      """
    }
}
```

Cliente de la clase `Persona`

```
    fun printPersonas(personas : Array<Persona>) {
      println("Imprimiendo Secuencia de personas:")
      for(p in personas) {
        println(p.toString())
      }
    }
```

Listado 1: Imprime un arreglo de objetos de tipo Persona.

Main del cliente de la clase Persona

```
fun main() {
  val kamuiFNac = LocalDate.of(1985, 12, 9)
  val kamuiTrabajos = arrayOf("Motorizado", "MotoTaxista",
                              "Delivery")
  val kamui = Persona("Kamui", "20569855", kamuiFNac,
    kamuiTrabajos, 2150.0)
  println(kamui.toString())
  kamui.darAumento(52) // Aumento de 52%
  println("Sueldo de ${kamui.nombre} despues de un aumento:
    %.2f".format(kamui.sueldo));
  kamui.sueldo = 6000.0
  println("Nuevo sueldo de ${kamui.nombre}: %.2f".format(
    kamui.sueldo))
  println("Sueldo de ${kamui.nombre} en divisas: %.2f".
    format(kamui.sueldoEnDivisas))
  kamui.nombre = "Pedro"
  println("Nuevo nombre de Kamui: ${kamui.nombre}")
  println(kamui.toString())
  println("Numero de personas: ${Persona.numeroDePersonas}")
```

Main del cliente de la clase Persona, continuación

```
  val camila = Persona("Camila", "28123774",
                       LocalDate.of(1982, 6, 30),
                       arrayOf("Desarrollador", "Analista"),
                       7000.0)
  println(camila.toString())
  println("Numero de personas: ${Persona.numeroDePersonas}")
  val familia = arrayOf(kamui, camila,
                       Persona("Juana", "26980361",
                               LocalDate.of(1989, 2, 15),
                               arrayOf("Abogada"),
                               5000.0))
  printPersonas(familia)
  println("Numero de personas: ${Persona.numeroDePersonas}")
}
```

Archivos de la clase `Persona` y su cliente

- Main.kt
- Persona.kt
- Makefile

Compilando y ejecutando `Persona` y su cliente

```
>make
```

Figura 1: Compilando la clase `Persona` y su cliente

```
>kotlin MainKt
```

Figura 2: Ejecutando el programa cliente de la clase `Persona`

Resumen

- Se dio la definición de la POO junto con algunos de sus principales elementos.
- Se realizó la implementación de una clase en Kotlin.
- Se aplicó la POO en Kotlin para crea un nuevo tipo de datos llamado *Persona*.
- Se mostró como instanciar un objeto de una clase en Kotlin.
- Se ilustró como acceder a los atributos de un objeto y como aplicar sus métodos en Kotlin.