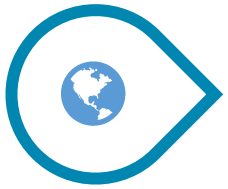
PROGRAMACIÓN ORIENTADA A OBJETOS

DESARROLLO DE APLICACIONES MÓVILES AVANZADAS

LUIS HUMBERTO RIVAS RODRÍGUEZ
INGENIERO EN SISTEMAS INFORMÁTICOS
Y MÁSTER EN DIRECCIÓN ESTRATÉGICA DE EMPRESAS.

Agenda

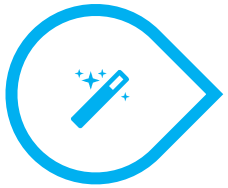


1 **Introducción**

Acerca de POO

Principios

Conceptos Claves 2



3 **POO en Kotlin**

Características de Kotlin para POO

Primero pasos

Prácticas en el Desarrollo Android 4

5 **Cierre**

Valoraciones Finales

RESULTADOS DE APRENDIZAJE

Competencias a desarrollar al finalizar la sesión.

INTERPRETA

Los componentes de software necesarios para construir una interfaz gráfica en Android.

COMPRENDE

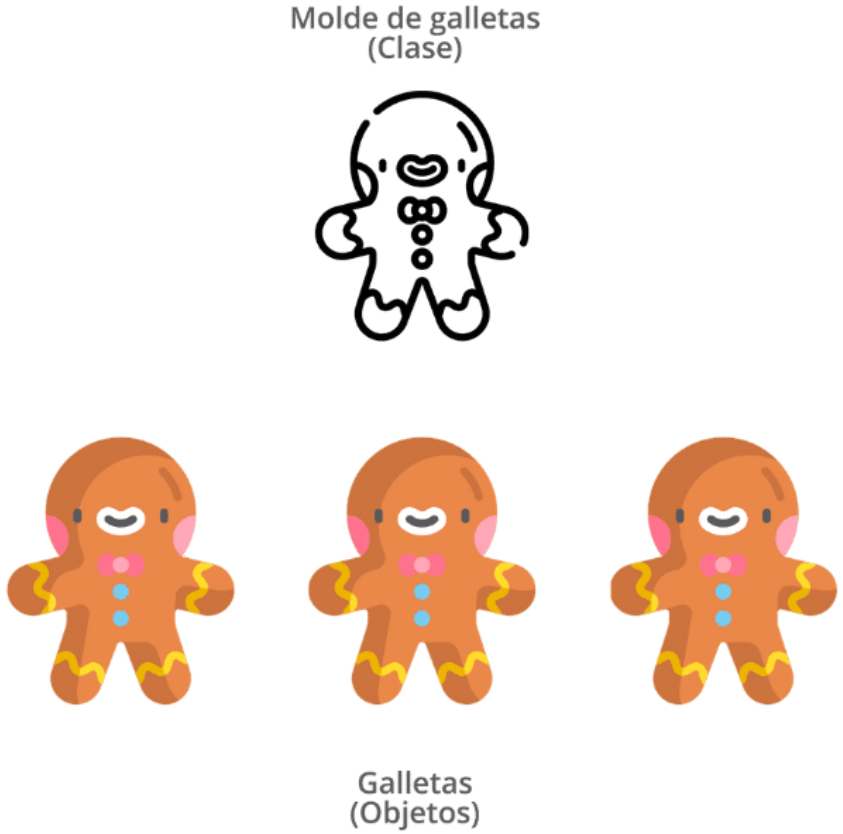
La relación que tienen los componentes y su importancia en el desarrollo de aplicaciones orientadas a objetos basadas en Android.

CONSTRUYE

Aplicaciones Android orientadas a objetos haciendo uso de los componentes avanzados que ofrece el entorno de desarrollo de Android Studio y el lenguaje Kotlin.

PROGRAMACIÓN ORIENTADA A OBJETOS

FUNDAMENTOS



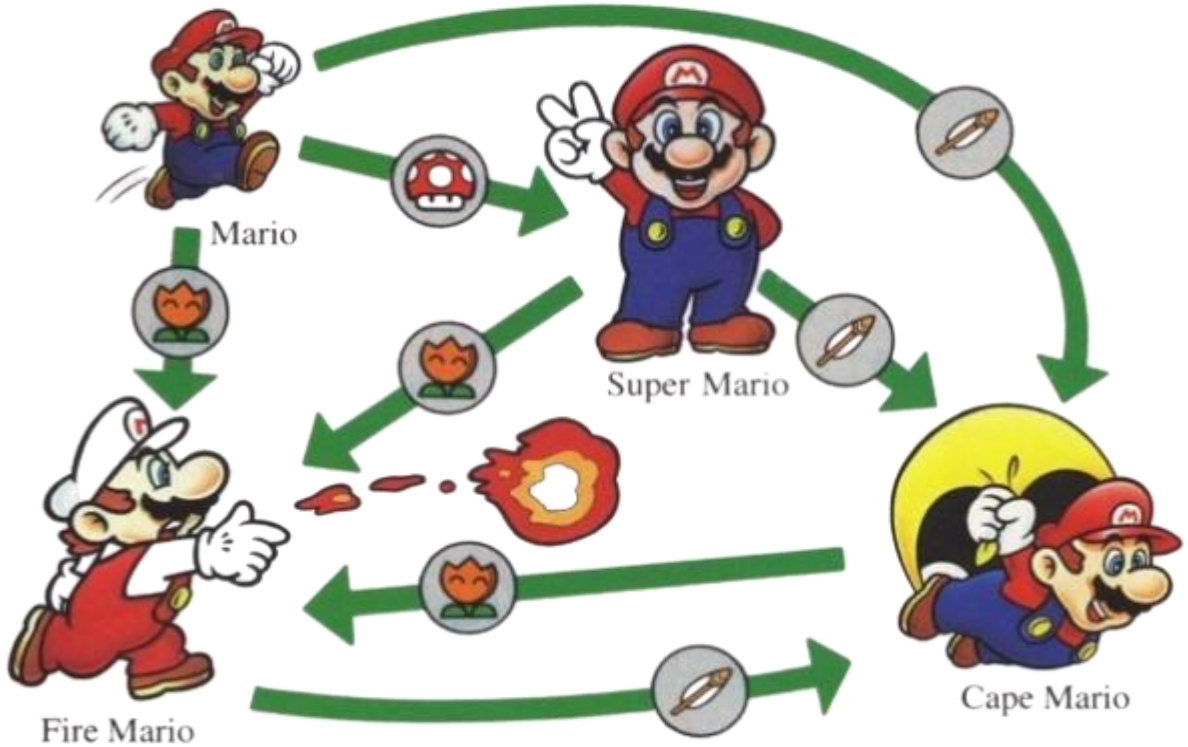
- **Propiedad:** características que tiene un objeto, dichas características son heredadas por las que tenga la clase.
- **Método:** es una serie de rutinas de código que buscan modificar el valor de una propiedad. (Setter).
- **Funciones:** son un conjunto de rutinas que a diferencia del método (que busca modificar una propiedad) este se encarga de averiguar el valor que tiene una propiedad. (Getter).
- **Constructor:** es el método inicial que se ejecuta sin ser invocado por el usuario, y es ideal cuando se desea programar ciertos atributos de la clase para que aparezcan con datos por defecto.

PROGRAMACIÓN ORIENTADA A OBJETOS

FUNDAMENTOS

¿CUÁNTAS CLASES HAY VISUALMENTE?

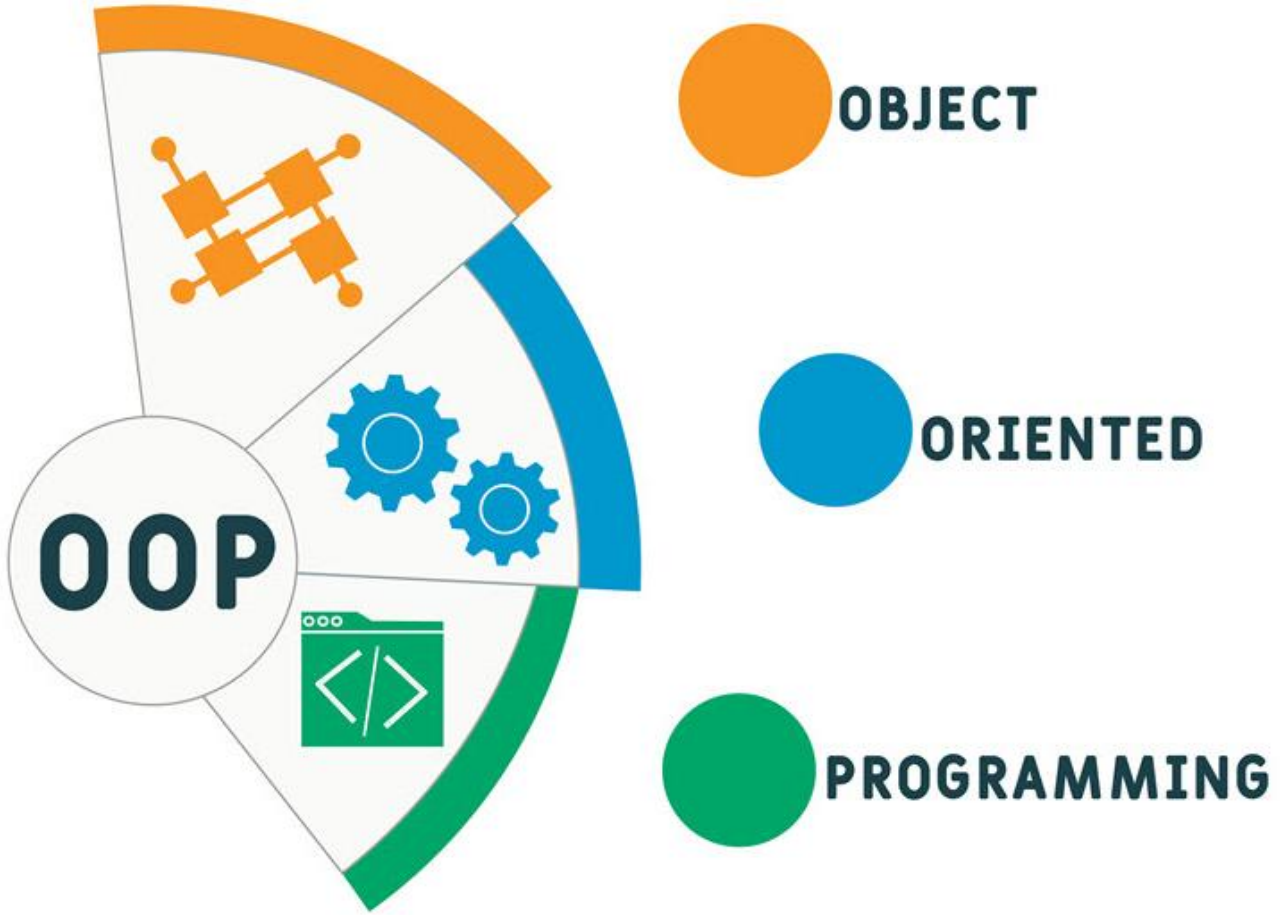
¿ES ORIENTADO A ESTADOS?



PROGRAMACIÓN ORIENTADA A OBJETOS

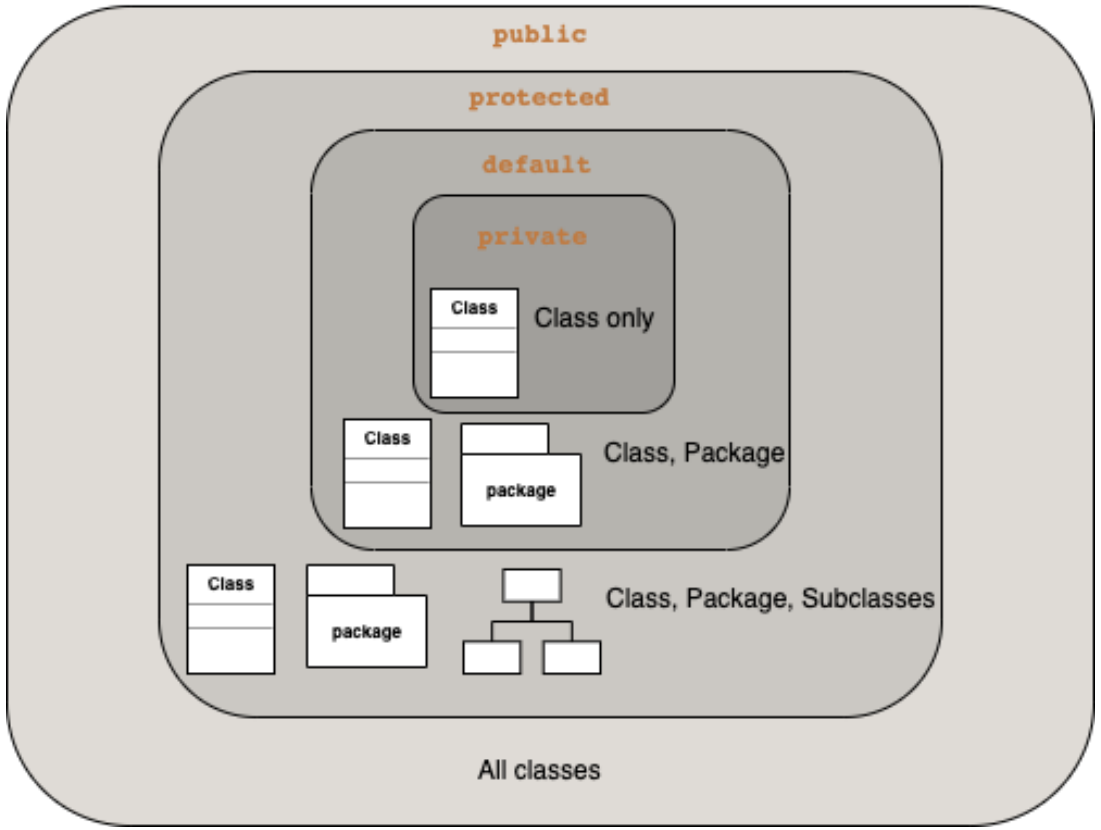
FUNDAMENTOS

- **Programación basada en estados** se enfoca en cómo los cambios de estado afectan a la UI, y cómo los datos fluyen a través de la aplicación para actualizar automáticamente la interfaz.
- **Programación orientada a objetos** se centra en modelar el mundo real con objetos que tienen propiedades y comportamientos.



PROGRAMACIÓN ORIENTADA A OBJETOS

FUNDAMENTOS



Los **modificadores de acceso** permiten dar un nivel de seguridad mayor a nuestras aplicaciones restringiendo el acceso a diferentes atributos, métodos, constructores asegurándonos que el usuario deba seguir una "ruta" especificada por nosotros para acceder a la información.

Encapsulamiento y Modificadores de Visibilidad: Kotlin permite controlar el acceso a las propiedades y métodos de una clase mediante modificadores de visibilidad:

- **public:** Accesible desde cualquier lugar (valor predeterminado).
- **private:** Accesible solo dentro de la clase.
- **protected:** Accesible dentro de la clase y sus subclases.
- **internal:** Accesible dentro del mismo módulo.
- **init:** Bloques de inicialización (init) se ejecutan cuando se crea una instancia de la clase, útiles para realizar configuraciones adicionales.

PRINCIPIOS

CONCEPTOS CLAVE

- **Clase:** Es un molde o plantilla que define propiedades (atributos) y comportamientos (métodos).
- **Objeto:** Es una instancia de una clase.
- **Encapsulación:** Proceso de restringir el acceso directo a ciertos componentes de un objeto y controlar su modificación.

```
class Persona(val nombre: String, var edad: Int) {
    fun saludar() {
        println("Hola, soy $nombre y tengo $edad años.")
    }
}
```

```
val persona1 = Persona("Luis", 25)
persona1.saludar()
```

```
class Persona(private val nombre: String, private var edad: Int) {
    fun getEdad() = edad // Getter para la edad
    fun setEdad(nuevaEdad: Int) { if (nuevaEdad > 0) edad = nuevaEdad } // Validación simple

    fun saludar() = println("Hola, soy $nombre y tengo $edad años.")
}

fun main() {
    val persona = Persona("Juan", 25)
    persona.saludar()

    persona.setEdad(30) // Modificar la edad
    println("Nueva Edad: ${persona.getEdad()} años")
}
```

PRINCIPIOS

CONCEPTOS CLAVE

- **Herencia:** La herencia promueve la reutilización de código al permitir que las subclases extiendan o sobrescriban funcionalidades específicas de la superclase. En Kotlin, se utiliza la palabra clave **open** para permitir que una clase sea heredada.
- **Polimorfismo:** Permite que métodos sobrescritos sean invocados dinámicamente.

```kotlin
// Clase base
open class Figura {
    open fun area() {
        println("Calculando el área...")
    }
}
// Clase derivada: Círculo
class Circulo : Figura() {
    override fun area() {
        println("Área del círculo: π * r^2")
    }
}
// Clase derivada: Cuadrado
class Cuadrado : Figura() {
    override fun area() {
        println("Área del cuadrado: lado * lado")
    }
}
fun main() {
    val figuras: List<Figura> = listOf(Circulo(), Cuadrado())

    figuras.forEach { it.area() }
}
```

```kotlin
// Clase base
open class Animal(val nombre: String) {
    fun dormir() {
        println("$nombre está durmiendo.")
    }
}

// Clase derivada
class Perro(nombre: String) : Animal(nombre) {
    fun ladrar() {
        println("$nombre está ladrando.")
    }
}

fun main() {
    val perro = Perro("Firulais")
    perro.dormir()
    perro.ladrar()
}
```

POO EN KOTLIN

Características de Kotlin para POO

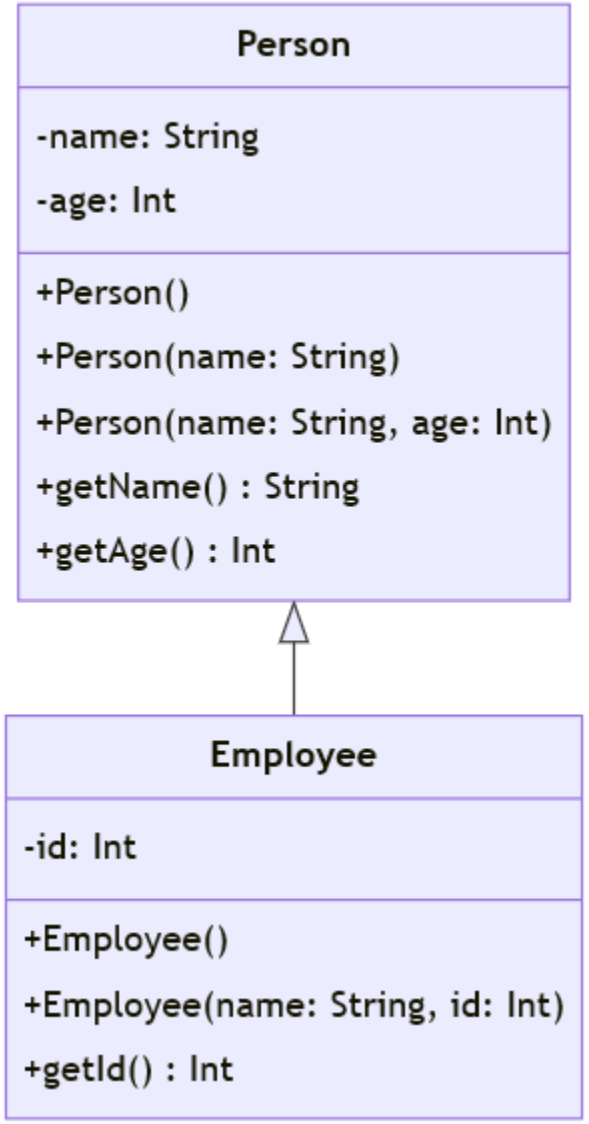
- Las interfaces se definen con **interface** y las clases pueden implementarlas con las interfaces pueden contener métodos con implementación por defecto.

  Además, están las propiedades y Getters/Setters, las propiedades se definen dentro de las clases con **var** o **val**. Kotlin genera automáticamente los métodos getter y setter, también cuenta con **Clases Abstractas**, las cuales se pueden crear utilizando la palabra clave **abstract**, que no pueden ser instanciadas pero pueden ser heredadas; y los objetos Singleton que se implementa con la palabra clave **object**.

POO EN KOTLIN

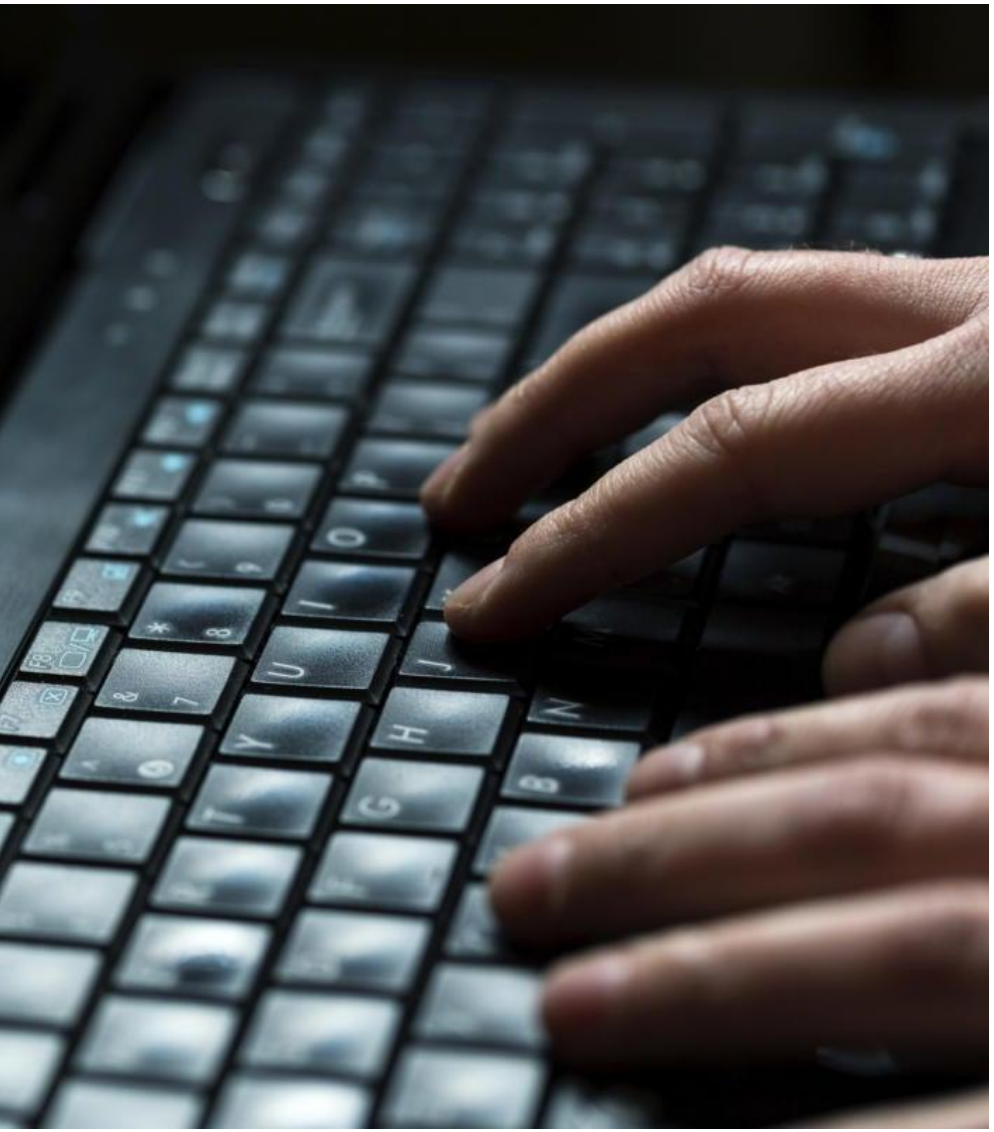
Características de Kotlin para POO

- **Constructores Primarios y Secundarios**
- **Constructor primario:** Definido directamente en la declaración de la clase.
- **Constructores secundarios:** Usan la palabra clave constructor y son útiles cuando se necesita mayor flexibilidad.
- **Data Classes:** Kotlin tiene clases especiales llamadas data classes que son ideales para contener datos, estas clases generan automáticamente los métodos toString(), equals(), hashCode() y copy().Delegación:Kotlin soporta la delegación a través del patrón by.



LIVE CODING

TRABAJO A REALIZAR



- Tomando como referencia los contenidos visto en la sesión, se elaborará una aplicación móvil que permita manejar el uso del lenguaje Kotlin y algunas acciones esenciales.

VALORACIONES **FINALES**

Comentarios sobre el tema.

1

ABSTRACCIÓN

Las clases definen el comportamiento y las propiedades esenciales para representar objetos del mundo real.

2

HERENCIA

Permite reutilizar código definiendo clases derivadas que extienden las funcionalidades de una clase base.

3

POLIMORFISMO

Diferentes clases pueden compartir una misma interfaz o clase base, facilitando el uso flexible y dinámico del código.

¿PREGUNTAS?

DESARROLLO DE APLICACIONES MÓVILES AVANZADAS

LUIS HUMBERTO RIVAS RODRÍGUEZ
INGENIERO EN SISTEMAS INFORMÁTICOS
Y MÁSTER EN DIRECCIÓN ESTRATÉGICA DE EMPRESAS.