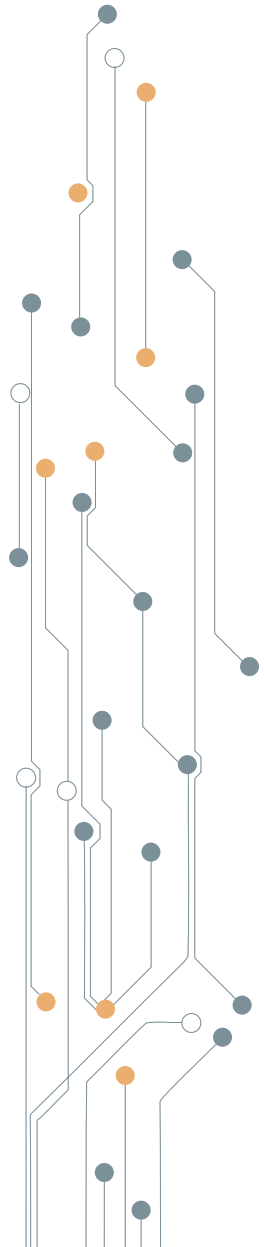
Otras características de la Programación Orientada a Objetos

Telefónica
EDUCACIÓN DIGITAL

Índice



Otras características de la Programación Orientada a Objetos

1. Otras características de la Programación Orientada a Objetos

Como hemos indicado, los lenguajes orientados a objetos ofrecen una serie de características de gran valor durante el desarrollo de las aplicaciones. Aunque tendrás la oportunidad de profundizar en el uso de dichas características durante el estudio del lenguaje de programación específico de tu itinerario, te presentamos a continuación los fundamentos de las más interesantes.

1.1 | Sobrecarga de métodos

Ya hemos comentado anteriormente el concepto de sobrecarga aplicado a constructores. Pues bien, este concepto es aplicable también a los métodos en general, es decir, una clase puede sobrecargar no solo constructores, sino también métodos.

La sobrecarga de métodos consiste en definir en una misma clase varios métodos con el mismo nombre, si bien deben diferenciarse en el número o tipo de parámetros.

Por ejemplo, en la clase Calculadora que creamos anteriormente podríamos tener los siguiente métodos suma definidos:

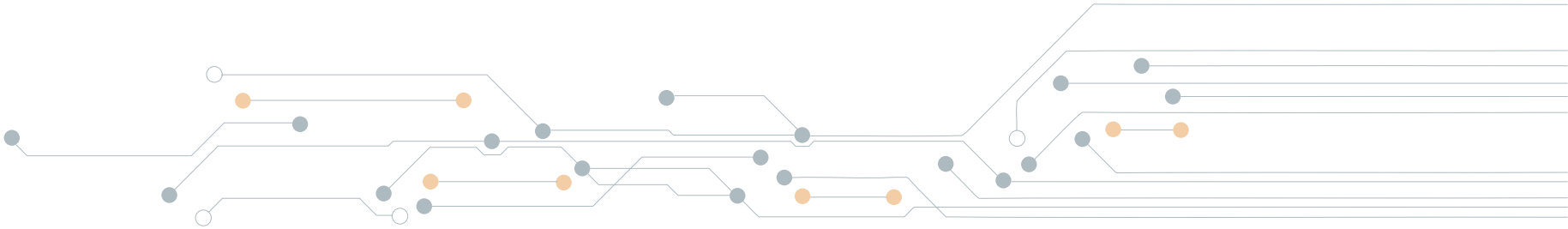
```
Function Sumar()
Function Sumar(a Integer)
Function Sumar (a Integer, b Integer)
```

Lo que **no** podríamos es tener por ejemplo los siguientes métodos definidos en la misma clase:

```
Function Restar(m Integer)
Function Restar(k Integer)
```

Pues aunque les hayamos dado distintos nombres, los tipos de los parámetros coinciden, luego están incumpliendo la regla de la sobrecarga.

La sobrecarga de métodos hace más cómodo el uso de una clase cuando esta dispone de varios métodos que realizan la misma tarea. Y es que si vamos a tener tres métodos que van a realizar la operación Sumar, pero con diferentes parámetros, es preferible que se llamen igual, dado que van a hacer la misma función, que darles nombres diferentes.



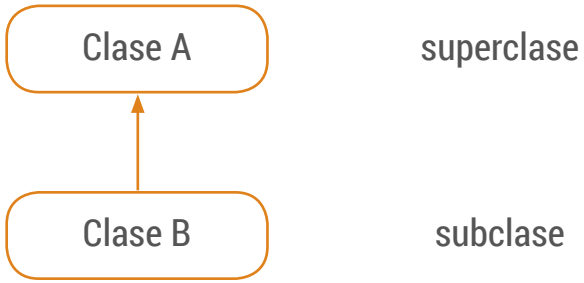
1.2 | Herencia

Después del concepto de clase y objeto, la herencia es la característica más importante de los lenguajes orientados a objetos.

La herencia permite crear clases que adquieran (hereden) todos los métodos y propiedades de otras clases ya existentes. Esta característica la aplicaremos cuando queramos crear una clases, que debe contener la misma funcionalidad que otra clases ya existente, pero que desea incorporar características y métodos adicionales.

En vez de codificar de nuevo los métodos y propiedades en la nueva clase, la herencia nos permite reutilizar el código de la clase ya existente.

La herencia se representa de forma gráfica de la siguiente manera:



A la clase padre o clase heredada se le conoce con el nombre de **superclase** o clase base, mientras que la que hereda es conocida como **subclase** o clase derivada.

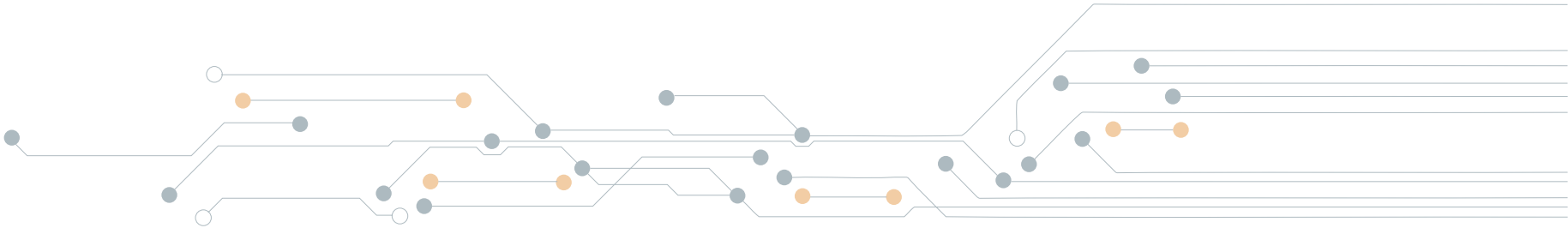
1.3 | Herencia en Pseudocódigo

De cara a crear una clase que herede otra clase, cada lenguaje de programación orientado a objetos dispone de su propia notación. En el caso de pseudocódigo, vamos a emplear la palabra extends (es la que utilizan lenguajes como Java) para indicar que una clase va a heredar otra clase:

```
Class NuevaClase extends ClasePadre
      //código de la nueva clase
End Class
```

Como ejemplo de aplicación, vamos a crear una clase que herede a la clase Cuenta del ejercicio anterior, a la que añadiremos una nueva propiedad llamada "clave" que represente una clave asociada a la cuenta. Por otro lado, crearemos un nuevo método llamado "resetear" que pondrá el saldo de la cuenta a 0 cuando sea llamado:

```
Class CuentaClave extends Cuenta
      Property clave Integer
      CuentaClave(c Integer)
            clave=c
      End
      Sub Resetear()
            saldo=0  //acceso a propiedad
heredada
      End Sub
End Class
```



Como vemos, la nueva clase solamente define los métodos y propiedades nuevos que incorpora, todo lo definido en la clase Cuenta también pertenece a CuentaClave, de ahí que podamos acceder directamente a la propiedad saldo.

Si creásemos un objeto de CuentaClave podríamos acceder directamente a todos sus métodos:

```
Cc CuentaClave
Cc=New CuentaClave(100) //le pasa la clave de
la cuenta al constructor
//llamada a métodos
Cc.Ingresar(20)
Cc.Extrear(10)
Cc.Resetear()
```

Sin embargo, la clase CuentaClave no está completa. Al crear el objeto no se establece ningún valor para el saldo, por lo que originariamente estará a 0. Lo habitual es que en el constructor de una clase se inicialice tanto las propiedades propias como las heredadas, aprovechando el constructor de la superclase para esto último.

A continuación te presentamos la versión completa de la clase:

```
Class CuentaClave extends Cuenta
      Property clave Integer
      CuentaClave(c Integer, s Decimal):base(s)
                  clave=c
      End
      Sub Resetear()
                  saldo=0  //acceso a propiedad heredada
      End Sub
End Class
```

Si observas la definición del constructor verás que a continuación del paréntesis de cierre aparece la expresión :base(s)

Lo que hacemos con esta expresión es una llamada al constructor de la clase base, al que le pasamos uno de los parámetros recibidos en el constructor de CuentaClave, concretamente el saldo. Es realmente una llamada a Cuenta(s Decimal)

Así pues, al crear un objeto CuentaClave, antes de ejecutarse el constructor de dicha clase, la expresión :base() hace que se produzca primero una llamada al constructor de la superclase, concretamente a aquel que coincida con los argumentos indicados en *base()*. Una vez que este constructor se ha ejecutado, el algoritmo continúa en el constructor de la subclase (CuentaClave).

1.4 | Sobrescritura de métodos

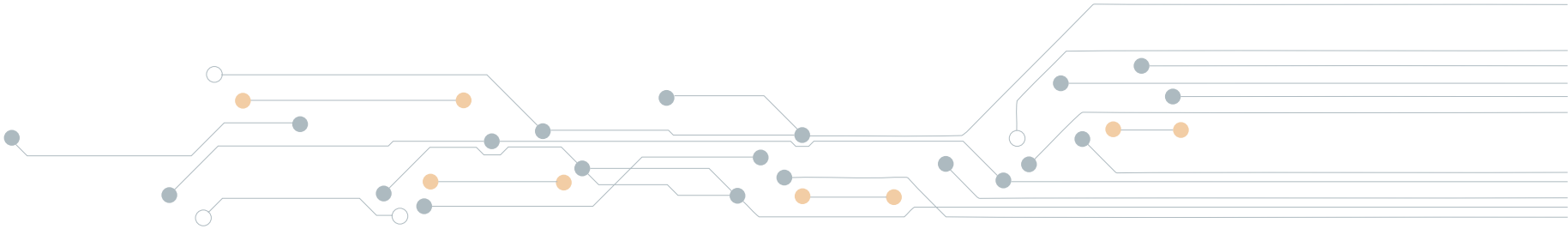
Hay ocasiones en las cuales alguno de los métodos que hereda una clase está implementado de manera que no se adapta a la funcionalidad que se le quiere dar a la nueva clase, o simplemente se desea realizar una mejora en el mismo.

En estos casos lo que se puede hacer es sobrescribir el método heredado en la nueva clase.

La sobrescritura de un método consiste en volver a definir en una subclase un método que ha sido heredado de la superclase. Durante la sobrescritura, se debe respetar el formato (nombre y parámetros) del método original.

Por ejemplo, supongamos que en la clase CuentaClave que hereda Cuenta quisiéramos tener una versión mejorada del método extraer(), de forma que solo se pudiera extraer dinero en caso de que hubiera saldo suficiente. Lo que tendríamos que hacer en este caso es volver a definir el método extraer(), es decir, sobrescribirlo, en la clase CuentaClave. El código actualizado de la clase quedaría:

```
Class CuentaClave extends Cuenta
	Property clave Integer
	CuentaClave(c Integer, s Decimal):base(s)
		clave=c
	End
Sub Extraer(cantidad Decimal)
	If(cantidad<saldo) Then
		saldo=saldo-cantidad
	End If
End Sub
	Sub Resetear()
		saldo=0  //acceso a propiedad heredada
	End Sub
End Class
```



Como vemos, a la hora de sobrescribir un método en pseudocódigo no utilizaremos ninguna palabra especial, simplemente se definirá el método con el mismo nombre y parámetros que el original

Si quisiéramos crear un objeto CuentaClave y llamar al método extraer:

```
Cc CuentaClave
Cc=New CuentaClave(111, 200)
Cc.Extraer(300)
Mostrar “Saldo: “, Cc.saldo
Tras ejecutar el código anterior se mostrará el mensaje
Saldo 200
```

Y es que, dado que la cantidad que se quiere extraer (300) es superior al saldo (200), al llamar a la versión sobrescrita del método extraer(), la operación de descuento de la cantidad al saldo no se producirá.

Telefónica

EDUCACIÓN DIGITAL