ITEDES

Educación Digital

Módulo:

BackEnd

Segmento:

Programación Orientada a Objetos

Tema:

Herencia

Prof. Germán C. Basisty
german.basisty@itedes.com

Índice de Contenidos

Herencia

El concepto de herencia

La **herencia** es específica de la programación orientada a objetos, donde una clase nueva se crea a partir de una clase existente. La herencia (a la que habitualmente se denomina **clase derivada**) proviene del hecho de que la **clase derivada** (la nueva clase creada) contiene las atributos y métodos de la clase primaria (**superclase**). La principal ventaja de la **herencia** es la capacidad para definir atributos y métodos nuevos para la **clase derivada**, que luego se aplican a los atributos y métodos heredados.

Esta particularidad permite crear una **estructura jerárquica de clases** cada vez más especializada. La gran ventaja es que uno ya no debe comenzar desde cero cuando desea especializar una clase existente. Como resultado, se pueden adquirir bibliotecas de clases que ofrecen una base que puede especializarse a voluntad.

Reutilización del código

Por otra parte, otro de los mecanismos que cualquier lenguaje de programación debe proveer es la posibilidad de **reutilizar el código**. En la programación estructurada tenemos las funciones, así que ya hemos podido reutilizar código de alguna manera. Así pues, el equivalente a las funciones, los métodos, ya nos da un grado de reutilización, pero no llegan al nivel de potencia de las que encontraremos en la herencia.

No necesitamos decirte mucho más para entender las bondades de la **reutilización**: en inglés lo resume el término **DRY** , **Don't Repeat Yourself** (**no te repitas**) y es uno de los enunciados que debes tener más presente cuando programas. *No es mejor programador quien más líneas de código hace, sino quien mejor las reutiliza*.

Se debe evitar escribir dos veces el mismo código, evitar los copia/pega y pensar que la reutilización nos ayuda seriamente en el **mantenimiento del software**.

Jerarquía de clase

La relación padre-hijo entre clases puede representarse desde un punto de vista jerárquico, denominado vista de clases en árbol. La vista en árbol comienza con una clase general llamada **superclase** (a la que algunas veces se hace referencia como clase primaria, clase padre, clase principal, o clase madre; existen muchas metáforas genealógicas). Las **clases derivadas** (clase secundaria o subclase) se vuelven cada vez más especializadas a medida que van descendiendo en el árbol. Por lo tanto, se suele hacer referencia a la relación que vincula una clase secundaria con una clase primaria mediante la frase es una x o y.

Herencia Múltiple

Algunos lenguajes orientados a objetos, como **C++** permiten herencias múltiples, lo que significa que una clase puede heredar los atributos de otras dos o más **superclases**. Este método puede utilizarse para agrupar atributos y métodos desde varias clases dentro de una sola.

Ejemplos

Java

```java
public class Mamifero {
    private int patas;
    private String nombre;

    public Mamifero(String nombre, int patas) {
            this.nombre = nombre;
            this.patas = patas;
    }

    public void imprimirPatas() {
        System.out.println(nombre + " tiene " + patas
            + " patas");
    }
}
```

```java
public class Perro extends Mamifero {
    public Perro(String nombre) {
            super(nombre, 4);
    }

    public void ladrar() {
        System.out.println("Guau Guau!");
    }
}
```

```java
public class Gato extends Mamifero {
    public Gato(String nombre) {
        super(nombre, 4);
    }

    public void maullar() {
        System.out.println("Miau Miau!");
    }
}
```

```java
public class CreaPerro {
    public static void main(String [] args) {
        Perro bobi = new Perro("Bobi");
        bobi.imprimirPatas();
        bobi.ladrar();
    }
}
```

C#

```
namespace ejemplocs {
    class Persona
    {
        public string Nombre;
        public int Edad;
        public string NIF;

        void Cumpleaños()
        {
            Edad++;
        }
        public Persona(string nombre, int edad, string nif)
        {
            Nombre = nombre;
            Edad = edad;
            NIF = nif;
        }
    }
}
```

```
namespace ejemplocs {
    class Trabajador : Persona
    {
        public int Sueldo;

        Trabajador(string nombre, int edad, string nif, int sueldo)
            : base(nombre, edad, nif)
        {
            Sueldo = sueldo;
        }
    }
}
```

```
using System;

namespace ejemplocs {
    class Program {
        public static void Main()
        {
            Trabajador p = new Trabajador(
                "Juan",
                22,
                "77588260-Z",
                100000
            );

            Console.WriteLine("Nombre=" + p.Nombre);
            Console.WriteLine("Edad=" + p.Edad);
            Console.WriteLine("NIF=" + p.NIF);
            Console.WriteLine("Sueldo=" + p.Sueldo);
            Console.ReadKey();
        }
    }
}
```

Python

```python
class Vehiculo:
    def __init__(self, nombre, color):
        self.__nombre = nombre      # __name es privado
        self.__color = color

    def getColor(self):
        return self.__color

    def setColor(self, color):
        self.__color = color

    def getNombre(self):
        return self.__nombre

class Auto(Vehiculo):
    def __init__(self, nombre, color, modelo):
        super().__init__(nombre, color)
        self.__modelo = modelo

    def getDescripcion(self):
        return self.getNombre() + self.__modelo + \
            " de color " + self.getColor()

c = Auto("Ford Mustang", "rojo", "GT350")
print(c.getDescripcion())
print(c.getNombre())
```