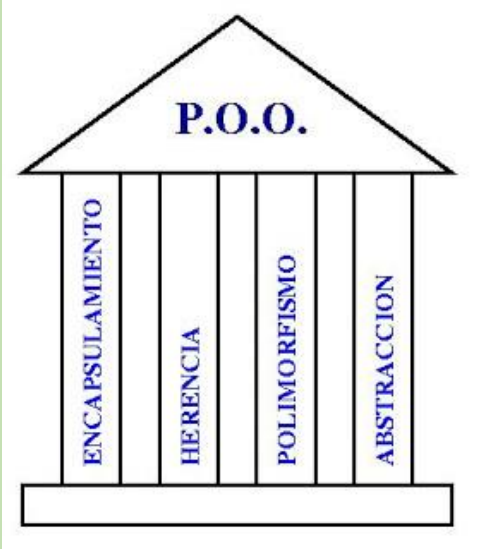
Capítulo 6

Introducción a la Programación Orientada a Objetos

Continuar

Introducción

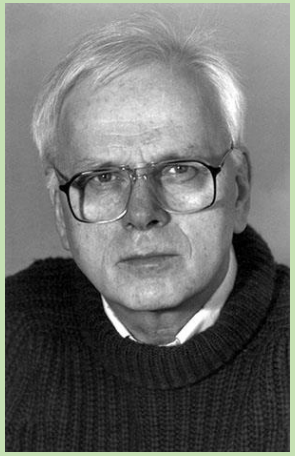
Un paradigma de programación es una forma específica de dar solución a un problema. Es decir, cuando se tiene un problema pueden existir varias alternativas de solución y cada una de ellas representa un paradigma diferente. Existen diversos paradigmas de programación como el imperativo, el declarativo, el funcional, el orientado a aspectos, el orientado a componentes, entre otros; pero, el paradigma de Programación Orientada a Objetos (POO) es uno de los más utilizados, debido a los beneficios que trae consigo; uno de ellos es que el diseño y la construcción de programas son muy similares a la forma en que los humanos entienden el entorno.

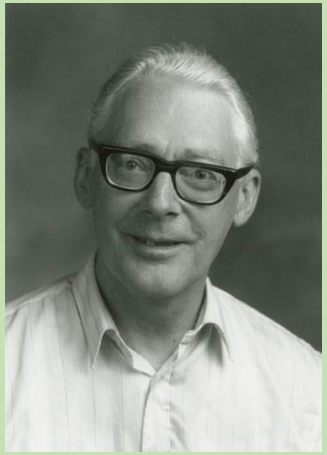


Orígenes de la Programación Orientada a Objetos

La Programación Orientada a Objetos surgió en el año de 1967 en el Centro Noruego de Computación, cuando Kristen Nygaard (Premio Turing 2001) y Ole Johan Dahl (Premio Turing 2001) diseñaron el lenguaje de programación Simula 67. En 1980, se dio a conocer Smalltalk, un lenguaje diseñado por Alan Kay (Premio Turing 2003), en el Centro de Investigación en Palo Alto de Xerox. Este lenguaje fue creado con fines educativos, para que a través de ese “mundo virtual” fuera posible interactuar con “objetos virtuales” los cuales se comunicaban entre sí por medio de mensajes. De esta manera, se definieron por primera vez los conceptos del paradigma de la POO: clase, objeto, atributo, método, etc.

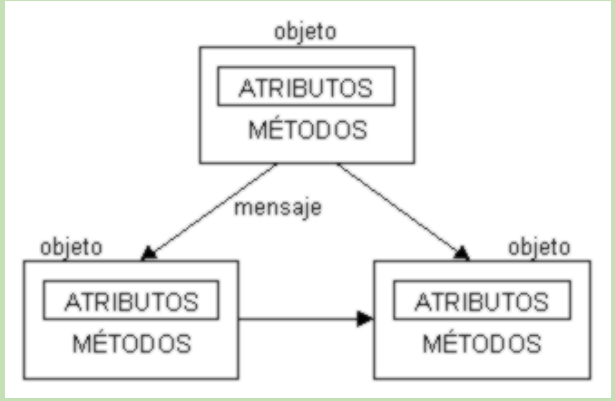
Finalmente, en la década de los noventas nació Java, otro lenguaje con alto soporte para el paradigma de la Programación Orientada a Objetos.





Beneficios de la Programación Orientada a Objetos

- Modelación comprensible: La forma de modelar una solución mediante el paradigma de la POO es muy similar a la visión que tienen los seres humanos respecto a la realidad.
- Reutilización: La posibilidad de no comenzar nuevas aplicaciones desde cero, pues el diseño que da la POO al código permite usar nuevamente código creado con anterioridad.
- Flexibilidad: El paradigma de la POO crea código flexible y facilita la adaptación de código ya escrito a nuevas necesidades sin mucho esfuerzo. Esto repercute en en un mantenimiento más sencillo y menos costoso de las aplicaciones.
- Escalabilidad: Las aplicaciones pueden ir creciendo a la par de las nuevas necesidades debido a la reutilización y características como la herencia.



Clases

Una clase es un tipo de dato que contiene en una misma estructura atributos (variables) y métodos (funciones). La sintaxis básica para definir una clase en Java es la siguiente:

especificador_acceso **class** nombre_clase {

//atributos

//métodos

}

El primer elemento, el especificador de acceso, indica la visibilidad de la clase (quiénes pueden verla), existen dos tipos:

- Public
- Default

El segundo elemento es escribir la palabra reservada **class** y después el nombre de la clase para entre llaves colocar los atributos y los métodos de dicha clase.

Métodos

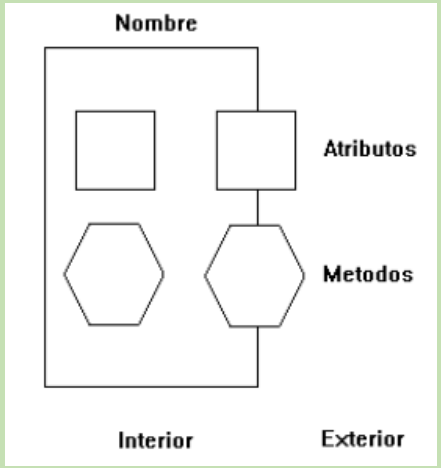
Los métodos son funciones de una clase, su sintaxis básica es la siguiente:

especificador _acceso [**static**] tipo_retorno nombre_método (parámetros) {

//sentencias

}

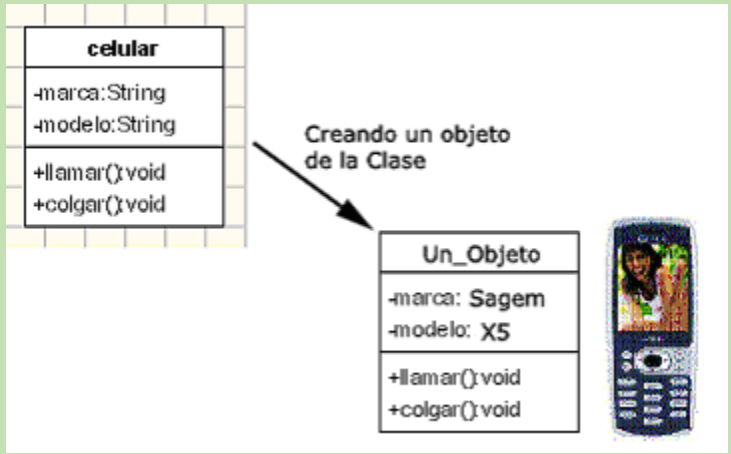
La palabra **static** puede o no ir, es opcional y se emplea para definir un método que puede llamarse desde otra clase sin necesidad de hacerlo a través de un objeto.



Objetos

Un objeto es una instancia de una clase, de la misma manera que una variable es una instancia de un tipo de dato (considerar el atributo int edad, la variable edad es una instancia del tipo de dato int); por lo que, se puede ver una clase como el tipo de dato de un objeto. Para acceder a los atributos y/o métodos de una clase es necesario crear un objeto, la sintaxis para esto es:

nombre_clase nombre_objeto = **new** nombre_clase ( );



Sobrecarga de métodos

Es una técnica que permite definir diferentes versiones de un método, todos con el mismo nombre pero diferentes parámetros (número y/o tipo). De esta manera, cuando se invoque al método sobrecargado, el compilador de Java determinará cuál elegir dependiendo del tipo y/o número de parámetros que requiere cada método.

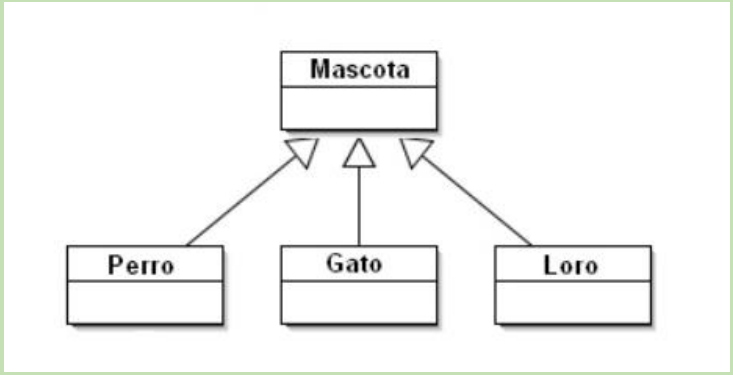
```
class Escritor
{
    public void Escribe(){
        Console.WriteLine("No tengo parámetros");
    }
    public void Escribe(int numero) {
        Console.WriteLine("Número: " + numero.ToString());
    }
    public void Escribe(int numero, string texto) {
        Console.WriteLine("Número: " + numero.ToString() +
            ", Texto: " + texto);
    }
    public void Escribe(int numero, string texto, DateTime fecha){
        Console.WriteLine("Número: " + numero.ToString() +
            ", Texto: " + texto + ", Fecha: " + fecha.ToString());
    }
}
```

Herencia

Esta técnica permite que una clase hija (clase que hereda de otra) adquiera de la clase padre las siguientes propiedades:

1. Automáticamente obtiene todos los atributos (variables) miembro de la clase padre.
2. Obtiene todos los métodos miembro de la clase padre; por lo que, la clase hija podrá funcionar de la misma manera que la padre.
3. La clase hija puede definir nuevos atributos (variables) y métodos (funciones).

Para utilizar la herencia en Java se emplea la palabra reservada **extends**, la cual indica que la nueva clase creada (hija) será una extensión (heredará) de la clase que se escribe justo después de la palabra **extends**.



Otros conceptos del paradigma de la Programación Orientada a Objetos

- Abstracción: Es la concentración sobre las características o hechos más importantes de los objetos, dejando de lado los detalles.
- Polimorfismo: Es la propiedad que tienen los objetos de responder de manera diferente a un mismo mensaje. Si se piensa en el mensaje (acción) *acomodar*, los objetos responderán de manera diferente dependiendo de objeto que se trate.
- Encapsulamiento: Concepto que se refiere a integrar los elementos necesarios dentro de una entidad.
- Ocultamiento: Propiedad que permite aislar o proteger del exterior a los atributos de un objeto, de tal manera que no puedan ser modificados de manera arbitraria.
- Modularidad: Es la propiedad que exige dividir una aplicación grande en partes más pequeñas, independientes entre sí.