Preparador Informática

www.preparadorinformatica.com

TEMA 27. INFORMÁTICA

PROGRAMACIÓN ORIENTADA A OBJETOS. OBJETOS. CLASES. HERENCIA. POLIMORFISMO. LENGUAJES

TEMA 29. SAI

PROGRAMACIÓN ORIENTADA A OBJETOS. OBJETOS. CLASES. HERENCIA. POLIMORFISMO.

TEMA 27 INF: PROGRAMACIÓN ORIENTADA A OBJETOS. OBJETOS. CLASES. HERENCIA. POLIMORFISMO. LENGUAJES

TEMA 29 SAI: PROGRAMACIÓN ORIENTADA A OBJETOS. OBJETOS. CLASES. HERENCIA. POLIMORFISMO.

1. INTRODUCCION

A lo largo de la historia de la informática, han ido apareciendo diferentes paradigmas de programación. En primer lugar, apareció la programación secuencial, que consistía en secuencias de sentencias que se ejecutaban una tras otra. El lenguaje ensamblador o el lenguaje COBOL son lenguajes secuenciales. Entonces no existía el concepto de función, que apareció más adelante en los lenguajes procedimentales, como el BASIC o C. La evolución no terminó aquí, y continuó hasta el paradigma más extendido en la actualidad: la programación orientada a objetos. Java, C++ o C# son ejemplos de lenguajes basados en este paradigma. Cada nuevo paradigma ha extendido el anterior, de manera que podemos encontrar características de un lenguaje secuencial en uno procedimental, y las de uno procedimental en uno orientado a objetos.

En el presente tema se estudian los conceptos de la programación orientada a objetos desde un punto de vista global, sin particularizar para ningún lenguaje de programación específico.

2. PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos permite una representación más directa del modelo del mundo real en el código. El resultado es que se reduce considerablemente la transformación de los requisitos del sistema (definido en términos de usuario) a la especificación del sistema (definido en términos de computador). Es un paradigma de programación que usa objetos en sus interacciones, para diseñar aplicaciones y programas informáticos. Está basada en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

Fue a principios de la década de los 90 cuando su uso se popularizó y en la actualidad probablemente sea uno de los paradigmas de programación más utilizados existiendo gran variedad de lenguajes de programación que soportan la orientación a objetos.

2.1. VENTAJAS

- **Comprensión.** Facilita la comprensión al representar los problemas en términos del mundo real. Es una representación más cercana a nuestra forma de pensar.
- **Modularidad**. Al estar las definiciones de objetos en módulos, hace que las aplicaciones estén mejor organizadas y sean más fáciles de entender.
- **Fácil mantenimiento**. Cualquier modificación en las acciones queda automáticamente reflejada en los datos.
- **Seguridad.** No se pueden modificar los datos de un objeto directamente, sino que se debe hacer mediante las acciones definidas para ese objeto.
- **Reutilización de código**. Facilita la reutilización mediante el uso de bibliotecas de clases.

2.2. CARACTERÍSTICAS

Existen diversas características ligadas a la orientación a objetos muy importantes y que trataremos posteriormente en el tema de manera más profunda. Estas características son:

- Abstracción
- Encapsulación.
- Herencia.
- Polimorfismo.

3. OBJETOS

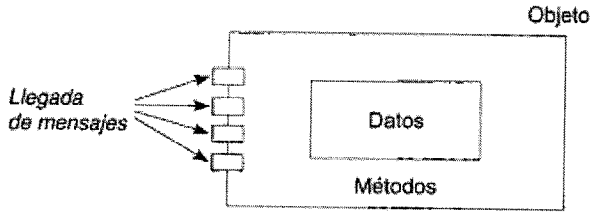
Un objeto se corresponde con una entidad del mundo real que se caracteriza por un estado (datos) y un comportamiento (operaciones).

a) **Estado**. El estado de un objeto viene determinado por los valores que toman sus datos. Los datos se denominan también ***atributos o propiedades*** y componen la estructura del objeto. Por ejemplo, un objeto Coche su estado podría estar definido por atributos como Marca, Modelo, Color, etc.

b) **Comportamiento**. El comportamiento de un objeto viene determinado por las operaciones (***métodos***) que se pueden realizar sobre el objeto. Por ejemplo,

un objeto Coche su comportamiento podría estar definido por acciones como arrancar, parar, acelerar, etc.

Los objetos se comunican unos con otros llamando a sus métodos. Estos métodos determinan cómo actúan los objetos cuando reciben un mensaje. Un ***mensaje*** es la acción que hace un objeto. Y un método especifica cómo se ejecuta un mensaje.



Los objetos son entidades dinámicas, que existen en el tiempo; por ello deben ser creados o instanciados, normalmente a través de otros objetos. Esta operación se hace a través de métodos especiales llamados constructores, que puede ser ejecutados implícitamente por el compilador o explícitamente por el programador mediante la invocación a los citados constructores.

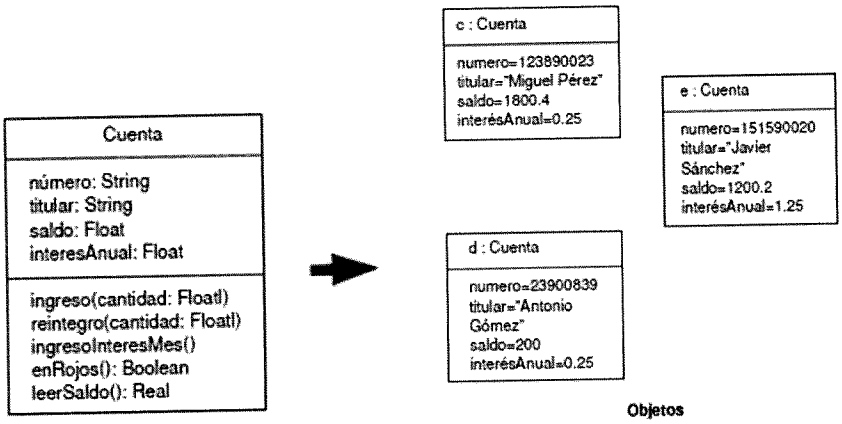
4. CLASES

Una clase es la caracterización abstracta de un conjunto de objetos y define los datos que se utilizan para representar un objeto y las operaciones que se pueden realizar sobre esos datos. Se pueden definir muchos objetos de la misma clase. Es decir, una clase es la declaración de un tipo objeto.

Las clases son similares a los tipos de datos y equivalen a modelos o plantillas que describen:

- Los **atributos o propiedades** comunes a todos los objetos de la clase.
- Los **métodos o funciones** que son las operaciones que pueden utilizarse para manejar esos objetos.

Cada vez que se construye un objeto a partir de una clase, estamos creando lo que se llama una instancia de esa clase. En general, instancia de una clase y objeto son términos intercambiables. Veamos un ejemplo:



Por ejemplo, la declaración de una clase en Java tiene la siguiente estructura general:

```
[modificadores] class <NombreClase> [herencia] [interfaces] { // Cabecera de la clase
   // Cuerpo de la clase
   Declaración de los atributos
   Declaración de los métodos
}
```

Un ejemplo de básico de clase en Java podría ser:

```
/**
 *
 * Ejemplo de clase Punto
 */

class Punto {
    // Atributos
int x,y;

    // Métodos
int obtenerX () { return x; }
int obtenerY()  {return y;}
void establecerX (int vx) { x= vx; };
void establecerY (int vy) { y= vy; };
}
```

5. ABSTRACCIÓN

La abstracción es la propiedad que permite representar las características esenciales de un objeto desde un punto de vista determinado y no tener en cuenta las restantes características, reduciendo así la complejidad. Es decir, permite no preocuparse por los detalles no esenciales.

Por ejemplo, diferentes modelos de abstracción del objeto Coche podrían ser:

- Un coche es la combinación de diferentes partes, tales como motor, carrocería, número de puertas, etc.
- Un coche puede clasificarse por el nombre del fabricante, por su categoría (turismo, deportivo, todoterreno...), por el carburante que utilizan (gasolina, gasoil, híbrido...).

Por ejemplo, la segunda abstracción vista anteriormente se utilizará siempre que la marca, la categoría o el carburante sean significativos. Por tanto, la abstracción a utilizar dependerá de la naturaleza del problema a resolver

6. ENCAPSULACIÓN DE DATOS

La encapsulación es la propiedad que permite asegurar que el contenido de la información de un objeto está oculta. La encapsulación (también se conoce como ocultación de la información), en esencia, es el proceso de ocultar todos los secretos de un objeto que contribuyen a sus características esenciales.

La encapsulación permite la división de un programa en módulos. Estos módulos se implementan mediante clases, de forma que una clase representa la encapsulación de una abstracción. En la práctica, esto significa que cada clase debe tener dos partes: una interfaz y una implementación. La interfaz de una clase captura sólo su vista externa y la implementación contiene la representación de la abstracción, así como los mecanismos que realizan el comportamiento deseado.

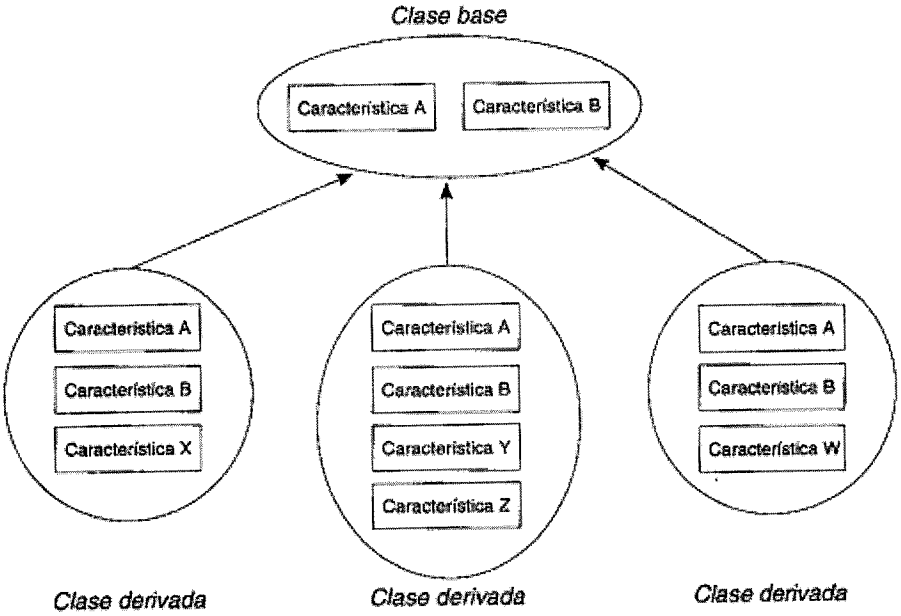
7. HERENCIA

La herencia es la propiedad que permite a los objetos ser construidos a partir de otros objetos. La idea fundamental es permitir crear nuevas clases aprovechando las características (atributos y métodos) de otras clases ya creadas evitando así tener que volver a definir esas características (reutilización).

A una clase que hereda de otra se le llama subclase (clase derivada, clase hija) y aquella de la que se hereda es conocida como superclase (clase base, clase padre).

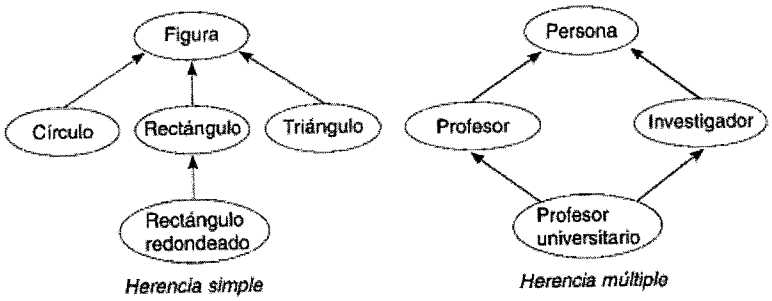
Las clases derivadas heredan el código y datos de su clase base, añadiendo su propio código y datos, incluso puede cambiar aquellos elementos de la clase base que necesita que sean diferentes. Por ejemplo, la clase animal se puede

dividir en las subclases anfibios, mamíferos, aves, etc., y la clase vehículo en coches, camiones, etc.

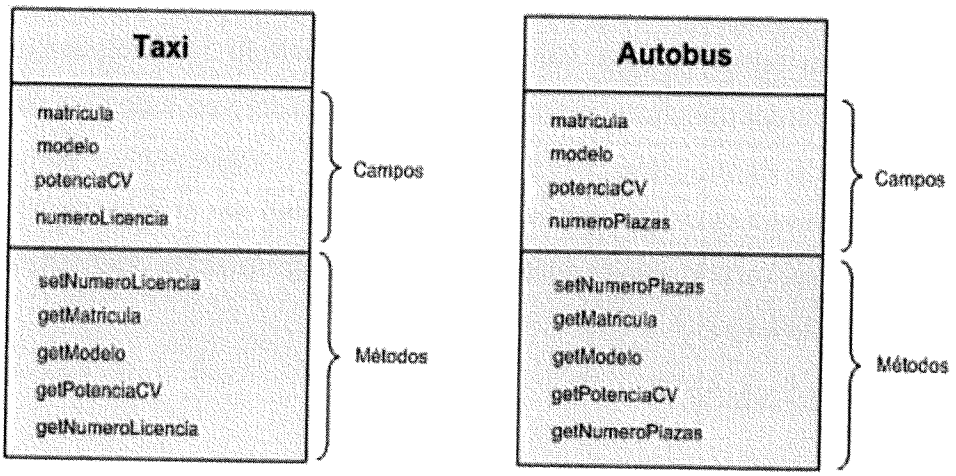


Existen dos tipos de herencia:

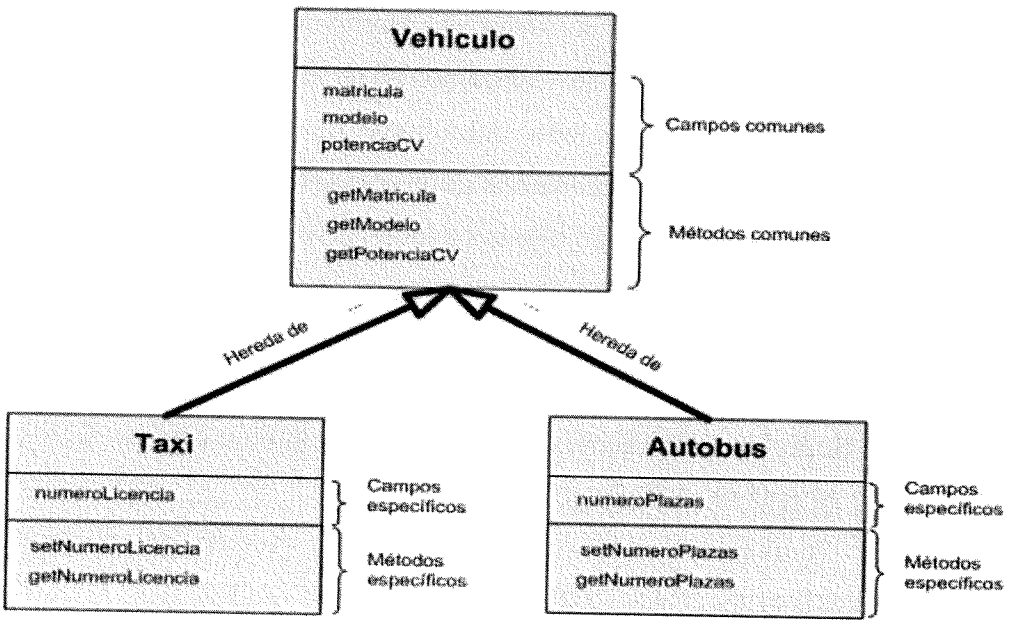
a) **Herencia simple**: una subclase puede heredar datos y métodos de una única clase. Java admite solo herencia simple.
b) **Herencia múltiple**: una subclase puede heredar datos y métodos de más de una clase. C++ admite herencia simple y múltiple.



Veamos otro ejemplo de herencia de manera más explícita, imaginando las clases Taxi y Autobús:



Aquí podemos ver que distintos objetos comparten campos y métodos que hacen aproximadamente lo mismo. Conceptualmente podemos imaginar una abstracción que engloba a Taxis y Autobuses: ambos podríamos englobarlos bajo la denominación de "Vehículos". Un Taxi sería un tipo de Vehículo y un Autobús otro tipo de Vehículo. De esta manera lo que haríamos sería definir una clase denominada Vehículo, de forma que la clase Taxi tuviera todas las propiedades de la clase Vehículo, más algunas propiedades y métodos específicos. Lo mismo ocurriría con la clase Autobús y otras que pudieran "heredar" de Vehículo quedando de la siguiente manera:



La herencia tiene como ventajas principales la reutilización y compartición de código (evita repetir código ahorrando mucho tiempo), la consistencia de la interfaz (el comportamiento que heredan será el mismo en todos los casos) y la ocultación de información. Pero también tiene inconvenientes como son la velocidad de ejecución (los métodos heredas suelen ser más lentos) y la curva de aprendizaje (al principio el aprendizaje de la programación orientada a objetos suele ser más lento).

8. POLIMORFISMOS

El polimorfismo es la propiedad que permite que un operador o un método actúen de modo diferente en función del objeto sobre el que se aplican. Cuando un operador existente en el lenguaje tal como +, = o * se le asigna la posibilidad de operar sobre un nuevo tipo de datos, se dice que está sobrecargado. La sobrecarga es una clase de polimorfismo.

En un sentido más general, el polimorfismo supone que un mismo mensaje puede producir acciones totalmente diferentes cuando se reciben por objetos diferentes.

El polimorfismo adquiere su máxima expresión en la herencia de clases, es decir, cuando se obtiene una clase a partir de una clase ya existente.

Por ejemplo, cuando se describe la clase base Mamífero se puede observar que la operación comer es una operación fundamental en su vida, de modo que cada tipo de mamífero (vaca, humano, león) debe poder realizar la operación comer, aunque cada uno de las clases derivadas que representan los distintos tipos de mamíferos la realizará de un modo diferente (polimorfismo).

9. LENGUAJES

Los podemos clasificar en dos categorías:

- Puros: Soportan únicamente el paradigma orientado a objetos. Ejemplos: Simula, SmallTalk, Eiffel y Java.
- Mixtos: También soportan otros paradigmas además del orientado a objetos. Ejemplos: C++, C#, Python y PHP.

Una panorámica de la evolución de los principales lenguajes de programación orientados a objetos hasta llegar a los utilizados actualmente es la siguiente:

- **Simula (1962)**. Simula fue el primer lenguaje que introdujo el concepto de objetos y de clase a la vez. En 1967 surgió Simula 67 que incorporaba un mayor número de tipos de datos, además del apoyo a objetos.

- **SmallTalk (1972)**. Basado en Simula 67, la primera versión fue Smalltalk 72, a la que siguió Smalltalk 76, versión totalmente orientada a objetos.

- **C++ (1985).** Lenguaje que deriva del C, al que añade una serie de mecanismos que le convierten en un lenguaje orientado a objetos. En este lenguaje es donde aparece el concepto de clase tal y como lo conocemos actualmente.

- **Eiffel (1986)**. Tiene una sintaxis similar a C. Soporta todas las propiedades fundamentales de los objetos, utilizado sobre todo en ambientes universitarios y de investigación.

- **Python (1989):** Es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma. Se trata de un lenguaje ideal para generar scripts y es muy personalizable gracias a la multitud de módulos existentes

- **Java (1995)**. Es un lenguaje orientado a objetos diseñado desde cero, que recibe muchas influencias de C++. La revolución de Internet ha influido mucho en el auge de Java.

- **PHP (1995):** Es un lenguaje de programación mixto e interpretado, diseñado originalmente para la creación de páginas web dinámicas. Se usa principalmente para la interpretación del lado del servidor.

- **C# (2000)**. Fue creado por Microsoft. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET.

10. RANKING DE LENGUAJES DE PROGRAMACIÓN MÁS UTILIZADOS

Según la consultora Tiobe Software y de acuerdo al índice TIOBE el ranking de los lenguajes de programación más utilizados por los desarrolladores son:

Noviembre 2019	Lenguaje de programación	Ratings
1	Java	16.246%
2	C	16.037%
3	Python	9.842%
4	C++	5.605%
5	C#	4.316%
6	Visual Basic .NET	4.229%
7	JavaScript	1.929%
8	PHP	1.720%
9	SQL	1.690%
10	Swift	1.653%

Como se puede ver la mayoría de los lenguajes de programación que están entre los diez más utilizados son lenguajes de programación orientados a objetos, lo que da una muestra más de la importancia que tiene este paradigma en el desarrollo de software en la actualidad.

11. CONCLUSIÓN

Como hemos visto en el presente tema, la programación orientada a objetos es el paradigma dominante actualmente en el desarrollo de software, ya que aporta multitud de ventajas, pero una de sus ventajas más importantes es que permite la posibilidad de representar un problema en términos del mundo real, es decir, mediante la representación de objetos. De este modo, resulta más lógico y sencillo, ver la relación entre los datos y sus instrucciones.

12. BIBLIOGRAFÍA

- Joyanes Aguilar, L. **Programación orientada a objetos**. Editorial McGraw-Hill
- **Cox, Brad J. Programación orientada a objetos: un enfoque evolutivo. Editorial Addison-Wesley**
- Joyanes Aguilar, L. **Fundamentos de programación. Algoritmos, estructuras de datos y objetos.**. Editorial McGraw-Hill
- http://atc.ugr.es/APrieto videoclases Departamento de Arquitectura y Tecnología de Computadores. Universidad de Granada.
- www.java.com
- https://docs.microsoft.com/es-es/dotnet/csharp/
- https://www.tiobe.com/tiobe-index/