1. Introducción a la Programación Orientada a Objetos

La programación orientada a objetos es un paradigma de la programación que consiste en trasladar o representar las características y comportamientos que tienen los objetos en la vida real al código de la programación.

El objetivo es la reutilización de código con el fin de hacer más fácil la programación

La programación orientada a objetos nos ayudará también a estructurar mejor nuestros programas.

PHP a partir de su versión 5 tiene implementaciones orientadas a objetos, lo que lo hace tener código más reutilizable y mantenible.

1.1 Conceptos sobre Programación, Clase, Objeto e Instancia

Clase

Modelo donde se redactan las características comunes de un grupo de objetos.

Una clase, es simplemente una abstracción que hacemos de nuestra experiencia sensible. El ser humano tiende a agrupar seres o cosas -objetos- con características similares en grupos -clases-. Así, aun cuando existen por ejemplo multitud de vasos diferentes, podemos reconocer un vaso en cuanto lo vemos, incluso aun cuando ese modelo concreto de vaso no lo hayamos visto nunca. El concepto de vaso es una abstracción de nuestra experiencia sensible.

Objeto

Un objeto es una entidad que existe en tiempo de ejecución y que tiene comportamiento.

Componente o código de software que contiene en sí mismo tanto sus características (campos) como sus comportamientos (métodos).

Instancia de una clase. Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos), los mismos que consecuentemente reaccionan a eventos. Se corresponden con los objetos reales del mundo que nos rodea, o con objetos internos del sistema (del programa).

Por ejemplo un carro, un carro tiene atributos y características que lo definen, con lo cual se conforma una clase.

Propiedades / Estado / Atributos

Son las variables en programación estructurada, en la POO son datos específicos que determinan las características que definen a la clase.

Es decir, con base al ejemplo, son las características que tienen los objetos tales como el color, tamaño, peso, etc.

Métodos / comportamiento / funciones

Es la implementación de un algoritmo que representa una operación o función que un objeto realiza. El conjunto de los métodos de un objeto determina el comportamiento del objeto.

Es decir, con base al ejemplo son las acciones o comportamientos que pueden hacer los objetos, tales como arrancar, frenar, acelerar, etc.

Instancia

Ejemplar perteneciente a una clase.

Es un objeto con características comunes pero que se diferencian entre ellos ya que toma valores individuales.

Clase: Carros	
Atributos / Propiedades	Métodos / Funciones
Color **Peso** **Alto** **Largo**	Arrancar Frenar Girar Acelerar
Objeto: Carro	
Instancia: Ford Focus	

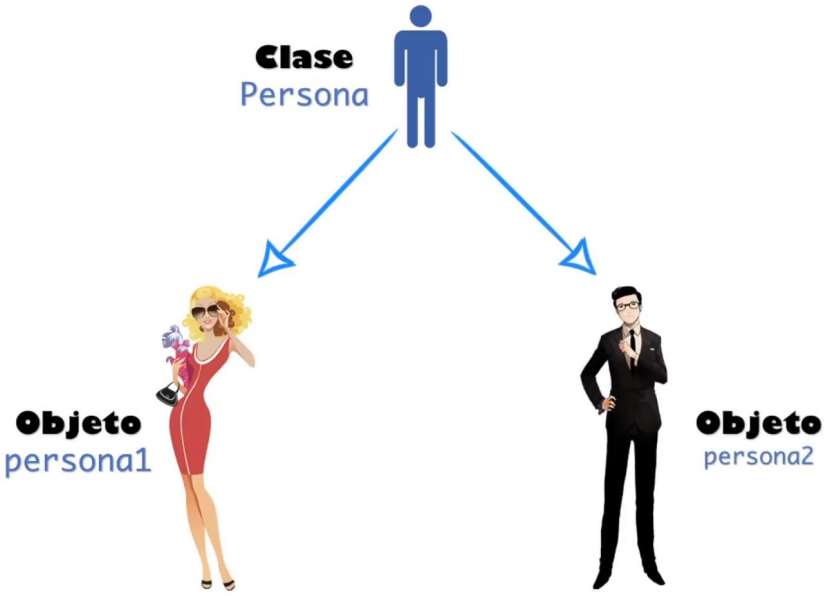
1.2 Características de la Programación Orientada a Objetos

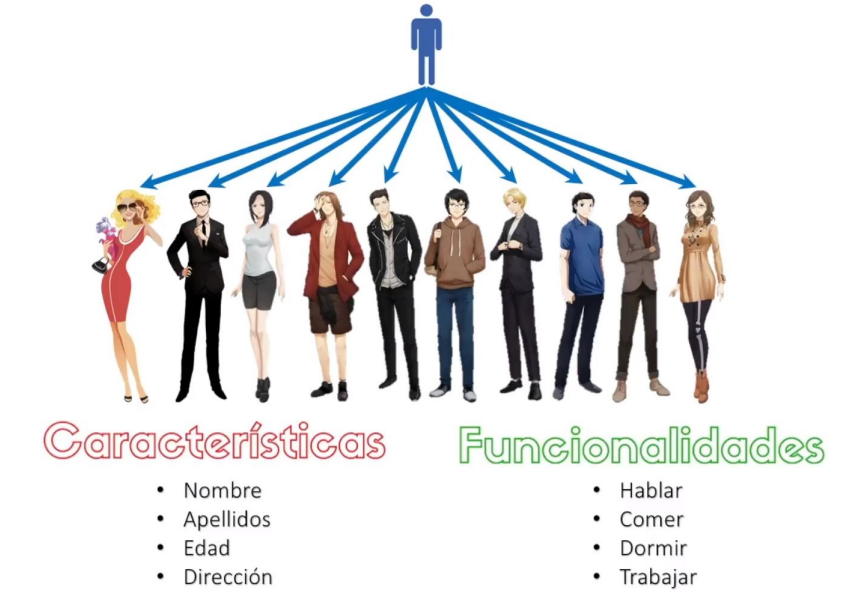
- Se basa en la percepción de objetos en la vida real.
- Es una técnica o paradigma de programación.
- Conjunto de reglas y técnicas que nos permiten incrementar enormemente nuestro proceso de producción de software.
- La reutilización es la principal característica de la POO, la cual se logra mediante:
- Abstracción { Herencia, encapsulamiento, polimorfismo}

1.2.1 Abstracción

Esta propiedad permite captar y distinguir a un objeto de los demás, observando sus características y comportamientos y representarlos en clases por medio de atributos (características) y métodos (comportamientos) de dicha clase , pensando en qué es y no en cómo se codifica en un lenguaje.

Capacidad del ser humano para entender una situación excluyendo detalles y sólo viéndola a alto nivel. El hombre ha comprendido el mundo con la abstracción. Con la abstracción se destaca lo importante y se ignora lo irrelevante, o sea, hay ocultamiento de información. Hay abstracción de datos al declarar una variable tipo *integer*, ya que internamente el compilador lo implementa en 2 *bytes*, lo cual es transparente al programador, o al declarar una variable *date*, el compilador controla los días de los meses, acepta sólo operaciones válidas entre las fechas, permitiendo al programador abstraerse de esos detalles. Estos tipos de datos abstractos coleccionan valores y operaciones, los cuales se usan transparentemente sin importar su implementación: otro lo implementa y yo lo uso.





1.2:2 Herencia

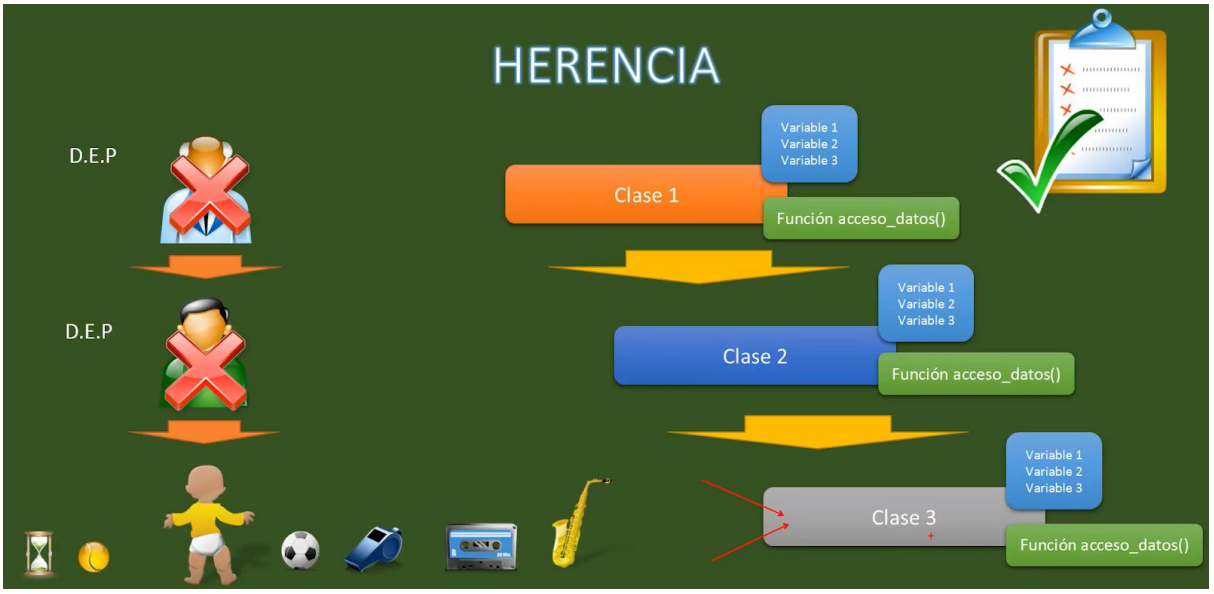
Propiedad que permite a los objetos ser construidos a partir de otros; es recibir de un módulo superior sus características, tales como atributos o funciones (campos y métodos o comportamientos), para usarlos en el módulo actual.

Heredar es compartir métodos y atributos.

Esta es la cualidad más importante de un sistema OOP, la que nos dará mayor potencia y productividad, permitiéndonos ahorrar horas y horas de codificación y de depuración de errores

Cuando una clase es heredada de otra se le conoce como subclase, y la clase de la que hereda es superclase. Así mismo una subclase puede ser super clase de otra.

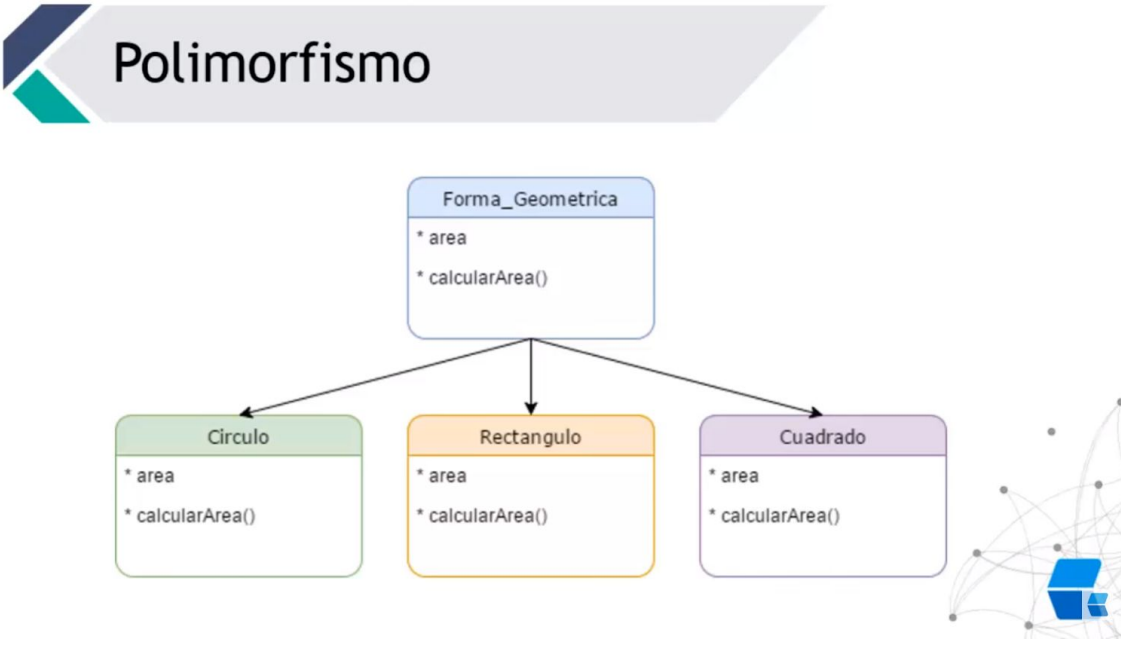
Meter imagen de ejemplo de herencia familiar



1.2.2 POLIMORFISMO

Esta propiedad indica que un elemento puede tomar distintas formas. Podemos definirlo como el uso de varios tipos en un mismo componente o función.

Por ejemplo, una función que sume dos operandos, la cual maneja, o dos números o dos cadenas, para retornar un total de una suma o de una concatenación. También se denomina *subsumption* consiste en utilizar el método con un mismo nombre pero cambiarle el comportamiento que tuvo al declararse.



EJEMPLO EN CÓDIGO MÓDULO NO 3

1.2.3 Encapsulamiento

Ocultamiento de información, datos o funciones (atributos o métodos) especiales a los usuarios.

En el caso de la programación, el encapsulamiento es lo que permite que tanto la estructura (campos) como el comportamiento (métodos) se encuentren dentro del mismo cuerpo de código de la clase con la que se crean los objetos. Dentro de la clase se deben agrupar tanto la información o datos de los campos como las operaciones o métodos o funciones que operan sobre esta información.
Encapsulamiento: no importa el cómo sino el qué.

Modificadores de acceso

Son palabras clave que se utilizan para especificar el nivel de acceso que podemos tener hacia un elemento de una clase.

Permiten dar un nivel de seguridad mayor a nuestras aplicaciones restringiendo el acceso a atributos y métodos asegurándonos que el usuario debe seguir una ruta especificada por nosotros para acceder a la información.

Existen 3 modificadores de acceso public, protected y private

Public

Se puede acceder al elemento desde cualquier clase o instancia del proyecto sin importar el paquete o procedencia de este.

Protected

Nos permite acceso a elementos de la misma clase, además podemos acceder también desde clases del mismo paquete y desde las clases que hereden de ella.

Private

Básicamente cualquier elemento de una clase que sea privado, puede ser accedido únicamente por la misma clase nada más, es decir si un atributo es privado, solo puede ser accedido por los métodos o constructores de la misma clase, ninguna otra clase sin importas la relación que tengan, podrá tener acceso

Ventajas y desventajas

TAREA

VENTAJAS

- Todo el código se encuentra en un solo lugar
- Los componentes se pueden reutilizar.
- Facilidad de mantenimiento y modificación de los objetos existentes.
- Los objetos pueden tener varios atributos, por ejemplo, que lea un sensor y a la vez encienda.
- Es modular

- Se proporciona un buen marco que facilita la creación de rica interfaz gráfica de usuario aplicaciones (GUI).
- Se acopla bien a la utilización de bases de datos, debido a la correspondencia entre las estructuras.
- Son más fáciles de entender los códigos.
- La ejecución del programa es rápida y sencilla, todo se encuentra en una sola ventana.

DESVENTAJAS

- Los programas no pueden ser moldeados enteramente por la programación orientada a objetos.
- Para leer, modificar, o hacerles algo simplemente; en algunos programas debes realizar un paso extra para realizar estas acciones.
- Si se fuerza el lenguaje puede perder algunos objetos y características.
- Los objetos requieren una extensa documentación.
- Los objetos al ser abstracto pueden no coincidir la visión de un programador a otro.
- Limitaciones del programador: Es posible que el programador desconozca algunas características del paradigma y de hecho siga utilizando el paradigma estructurado.
- No hay una única forma de resolver los problemas. Esto puede llevar a que diferentes interpretaciones de la solución planteada emerjan.
- Se requiere una documentación amplia para determinar la solución planteada.

1.3 Lenguajes

TAREA

- C++
- Objective C
- Java
- Smalltalk
- Eiffel
- Lexico (en castellano)
- Ruby
- Python
- OCAML
- Object Pascal
- CLIPS
- Visual .net
- Java
- Actionscript
- COBOL
- Perl
- C#
- Visual Basic.NET
- PHP

Referencias

Fernández, L. A. (2018, 10 25). *msdn.microsoft.com/*. Retrieved from msdn.microsoft.com/: https://msdn.microsoft.com/es-es/library/bb972232.aspx

Kroenke, D., & Auer, D. (2009). *Database Concepts*. New Jersey: Prentice Hall.

Stair, R., & Reynolds, G. (2001). *Principles of Information Systems*. Boston: Course Technology.

https://msdn.microsoft.com/es-es/library/bb972232.aspx

https://styde.net/abstraccion-programacion-orientada-a-objetos/

https://drive.google.com/open?id=1O-S_gkVj--AsXqRoR_9gxhrYg_ctyvxg

https://kataix.umag.cl/~ruribe/Utilidades/Introduccion%20a%20la%20Programacion%20Orientada%20a%20Objetos.pdf