Pontificia Universidad católica Madre y Maestra

Facultad de Ciencias de la Ingeniería

Programación 1

Tarea 1

Presentado por:

Eduardo Miguel Vila Cestero | 2018-5755
Dayan Paniagua Robles | 2019-5025
Freddy Antonio Cruz Valerio | 2019-5033

Grupo 2

Fecha de entrega:

26/05/2021

Programación orientada a objetos.

En la programación orientada a objetos existen una diversidad de conceptos tales como clases y objetos, pero, en el desarrollo de software con POO, podemos encontrar un conjunto de ideas fundamentales que a su vez son los cimientos del desarrollo de este. A estos conceptos se les conocen como los pilares de la programación orientada a objetos. Estos pilares son la abstracción, el encapsulamiento, la herencia y el polimorfismo.

Cuando nos referimos a una abstracción, se quiere omitir ciertos detalles, los cuales se ven como no necesarios, mientras que se busca mostrar solamente los datos que son relevantes. Si observamos desde el punto de vista del desarrollo de software, se puede observar que se puede realizar una abstracción de un ente que exista en el mundo real mediante una clase, esto quiere decir que, si utilizamos de ejemplo un vehículo, podemos guardar datos de este que estén relacionados con el año y la marca de susodicho. A la vez también se puede pensar que existen un sinnúmero de datos que se podrían utilizar, pero es posible que para el software estos datos no tengan relevancia y por ende sean omitidos. Esto quiere decir que para una clase que abstrae los datos de cualquier entidad, solamente es necesario tomar en cuenta lo que es de interés para la misma y de esta manera descartar lo demás que no sea de interés.

Con relación al encapsulamiento, este pilar nos permite controlar quienes pueden o no utilizar o ver los módulos que tiene un sistema. Con relación a las clases, el encapsulamiento permite definir quienes tendrán acceso a los miembros de dicha clase.
Para esto podemos utilizar modificadores de acceso, los cuales permiten definir cuáles agentes externos podrán controlar las diversas partes de un sistema, tales como las clases, las interfaces o los miembros de las clases. Siguiendo con el ejemplo del vehículo utilizado anteriormente, podemos usar una variable que llamaremos aceleración, la cual se quiere agregar a la clase vehículo, esta indicara que tan rápido este pasa de 0 a 100. Para esto se quiere que solo dentro de la clase se pueda modificar y ver dicha variable, esto se puede lograr utilizando una propiedad. Cuando se haga una instancia de la clase, no se podrá adentrar al valor de dicha propiedad y tampoco se podrá modificar desde afuera, lo que se podría hacer es utilizar la función para aumentar o disminuir la aceleración del vehículo dependiendo de cómo se hace la propiedad. Esto quiere decir que con el encapsulamiento se nos

permite tener el control de cómo se pueda alterar la data de un objeto dentro del sistema.

La herencia nos permite una manera de poder compartir código lo cual es crucial dentro de cualquier proyecto, ya que esto permite que podamos ahorrar trabajo y tiempo al querer por ejemplo realizar algún cambio a un sistema, permitiendo que un algoritmo pueda procesar diversas clases de entidades.
La herencia se podría decir que permite una relación entre dos clases, siendo estas la clase derivada y la clase base, donde la clase derivada consigue la forma de utilizar algunas propiedades o funcionalidades de la clase base. Esto quiere decir que la clase derivada "hereda" varias características que tiene la clase base. Como ejemplo utilizaremos la clase vehículo anteriormente mencionada, existen diversos tipos de vehículos como un carro, un camión o hasta un avión, los cuales todos comparten varios aspectos como la velocidad y la aceleración. Si tenemos varias clases, tales como vehículo, carro y avión, las clases carro y avión podrían heredar de la clase vehículo, lo cual significa que estas clases pueden heredar la función aceleración utilizada anteriormente y también de la misma forma los campos y propiedades de esta. Pero que estos puedan heredar algunas funciones de la clase base, esto no significa que los mismos no puedan definir sus propias funciones que no estén relacionadas con la clase base.

También existen las clases abstractas, que vendrían siendo una clase que no puede ser instanciada, estas serían de utilidad en casos en los cuales no se quiere que un usuario o varios usuarios instancie la clase base, sino que estas solo puedan instanciar las clases derivadas. También están las interfaces que facilitan realizar otro tipo de herencia, una clase base permite la implementación por defectos de algunos métodos, con las interfaces se pueden implementar un conjunto de miembros que las clases que lo implementan deben implementar y al igual que las clases abstractas, las interfaces no pueden ser instanciadas.

Por último, tenemos el polimorfismo, la idea del polimorfismo es que se pueda tener una función que reciba un parámetro, tal como una clase base y poder pasarle al mismo objeto que sean instancias de clases derivadas de la misma clase base, al igual que si el método recibe como dicho parámetro una interfaz. Así se puede traspasar al método cualquier clase que pueda implementar dicha interfaz.
De modo que podemos ver que el polimorfismo es cuando se recibe un parámetro que tiene varios tipos.

Una de las ventajas que ofrece este pilar es que se permite generalizar algoritmos para que estos puedan funcionar con distintos tipos.

Para poder comprender mejor algunas de las diferencias entre la programación orientada a objetos y la programación imperativa clásica es ideal el conocer un poco la programación imperativa clásica, ya que conocemos de cierto modo la POO.
La programación imperativa clásica se encuentra entre muchos de los paradigmas de programación y este se considera el paradigma clásico ya que este es uno de los primeros lenguajes de programación y esto significa que también dio origen a los primeros programas informáticos. Los cuales estaban basados completamente en este paradigma, el cual trae una secuencia de órdenes regularizada o también determinadas instrucciones. Esto también se puede decir que fue la base de algunos de los lenguajes aún vigentes como el lenguaje C y el pascal y la mayoría de los lenguajes ensambladores. En esta se puede decir además que el centro de atención es el poder trabajar lo más cercano al sistema como sea posible y por esto el código que resulta viene siendo sencillo de entender y abarcable.

Ahora, con relación a las diferencias entre la programación orientada a objetos y con la programación imperativa clásica. Es primordial entender las diferencias entre la programación orientada a objetos y la programación imperativa clásica.

El comprender los conceptos, las características y los lenguajes que se utilizan para cada uno. Al comprender y entender estos, se podrá ver de manera más clara cuáles son las diferencias que más se destacan a la hora de compararlos.
Primero lo primero, para ambos no existe una definición estándar, sino que para ambos se utilizaran unas definiciones resumidas de lo que más se asemeja a las mismas.
La programación orientada a objetos se podría decir que es un estilo en el cual los datos son tratados como objetos con atributos y métodos que a su vez es posible aplicar a los mismos y además pueden ser heredados por distintos objetos. Un buen ejemplo de esto podría ser Java, ya que este emplea este concepto, aunque este también emplea conceptos de otro método de programación.

La programación imperativa clásica, a diferencia de la programación orientada a objetos, vendría siendo un tipo de programación donde dentro de los procedimientos se hacen las declaraciones, las cuales podrían ser llamadas de ser necesario.

La programación orientada a objetos se concentra en clases y objetos, de forma que, al querer representar las variables como objetos, esto permite que se le pueda pasar una función. Dentro de una clase en particular puede pertenecer un objeto y este se puede tratar de forma independiente. Además, la programación orientada a objetos se puede basar en clases, de las cuales los objetos se basan predefinida mente. También existe la programación orientada a objetos basada en prototipos, en los cuales no son necesarias las clases y solo se necesitan utilizar los objetos.

Para la programación imperativa clásica, no son necesarios los objetos, sino que estos tienen procedimientos que podrían ser rutinas, subrutinas o estructura de datos.

Con relación a los términos utilizados en cada uno, estos en algunas ocasiones pueden llegar a tener los mismos significados, pero, en fin, en la programación orientada a objetos las funciones son llamadas métodos, mientras que, en la programación imperativa clásica, estas son llamados procedimientos. Con relación a la estructura de datos también se pueden observar algunas diferencias tales como los registros en la programación imperativa clásica y los objetos en la programación orientada a objetos. También la programación orientada a objetos utiliza una llamada de mensaje para que los objetos realicen acciones, mientras que la programación imperativa clásica, para llamar una función utiliza una llamada de procedimiento.

Una de las diferencias más características de ambos, viene siendo gracias a la herencia ya que mientras que en la programación imperativa clásica no admite herencia, ya que la misma solo se puede utilizar en objetos y esta no utiliza objetos.
La programación orientada a objetos recibe un gran impulso gracias a la herencia, ya que, gracias a esta, existe una mayor facilidad de reutilizar o extender un código existente. Los objetos que son creados tienen la capacidad de heredar algunas propiedades de objetos creados con anterioridad a los mismos, lo cual hace que la programación al crear nuevas variables estas pueden heredar múltiples características de varios objetos y facilitar su uso.

Otra diferencia importante es que, en la programación orientada a objetos, una función escrita de manera que los elementos de un tipo de datos, sea posible que estos funcionen en elementos de un distinto tipo de datos que estén relacionados, se puede lograr mediante el polimorfismo de subtipo. Aquí entra un principio llamado sustituibilidad, donde los objetos de un tipo, si existe

una relación, pueden ser cambiados por otros de otro tipo. Esto quiere decir que de cierto modo la programación orientada a objetos puede ser un poco más versátil o flexible que la programación imperativa clásica, la cual no puede lograr esto, ya que en la misma no se pueden declarar los subtipos y supertipos.

A diferencia de la programación imperativa clase la programación orientada a objetos también es capaz de que los datos sean vinculantes, de igual forma que los métodos manejan los datos. En este se forma una cápsula envolviendo los datos y métodos, de manera que los protege de toda interferencia externa. En la programación orientada a objetos la encapsulación funciona de manera eficiente y efectiva, por ende, el código se puede utilizar para detener la utilización de datos fuera de dicha cápsula.

Bibliografía

Programación orientada a objetos en PHP. (s. f.). Diego Lázaro. Recuperado 24 de mayo de 2021, de https://diego.com.es/programacion-orientada-a-objetos-enphp#:%7E:text=La%20programaci%C3%B3n%20orientada%20a%20objetos%20(Object%20Oriented%20Programming%20OOP)%20es,%2C%20abstracci%C3%B3n%2C%20polimorfismo%20o%20encapsulamiento

Weisfeld, M. (2019). The Object-Oriented Thought Process (5th ed.). Addison-Wesley Professional. https://doi.org/10.1145/2532780.2532797

Zapata-Ros, M., & Pérez-Paredes, P. (2019). El pensamiento computacional, análisis de una competencia clave: II Edición (2.ª ed.). Miguel Zapata Ros. https://doi.org/10.6018/red/46/6