Javier Carrillo – Grupo #1 – 10138957

Pilares de POO

La programación orientada a objetos es una orientación implementada por los nuevos y actuales lenguajes como Java y C# en comparación a otros más antiguos que se basa en enfocarse en los datos para darle estructura y sentido al programa, con esto vienen las clases con sus atributos y métodos, que sirven como plantillas para crear objetos a partir de estas.

Conociendo cómo funcionan las clases no basta para comprender en su totalidad las formas eficaces de estructurar un programa orientado a objetos y reducir de paso su complejidad. Este es el papel que desarrollan los denominados 4 pilares de la POO, bases teóricas que, de una manera u otra implementan los lenguajes por lo que es pertinente la comprensión y conocimiento de estos. Estos pilares son los siguientes:

Encapsulación

Es uno de los conceptos más básicos de los cuatro pilares que, por lo general, es la base que da concepción a la estructuración y forma de los objetos. Aunque pasa desapercibido muchas veces es importante reconocer su destacada función en la estructuración de nuestro programa.

La encapsulación no es más que el hecho de que las propiedades de un objeto residen y son propias de él. Esto quiere decir que si se quiere o desea interactuar con alguno de estos atributos o métodos se debe solicitar su uso o acceder a ellos mediante la clase que los posea, esto permite que los datos sean pertenecientes estrictamente a un renglón de parámetros estructurados y únicos, lo que permite la creación de tantos objetos con determinadas funciones y datos característicos como se requieran a partir de la clase base.

Además de la “apropiación” de datos únicos que nos permite comprender la encapsulación, ayuda a optimizar el uso de datos, esto quiere decir que, si necesitamos cierta información de un objeto, simplemente utilizaremos lo que se necesita de toda esa selección de datos.

Abstracción

Este elemento es una situación muy parecida a la anterior. Este concepto implementa la exclusión del programa hacia cualquier índice de complejidad que posea un objeto, es decir, que los procesos y métodos que posea un objeto no son pertinentes a quien solicite la información, simplemente le entregará la información.

Estos procesos complejos siempre son obviados y esto es importante, ya que sobrecargar de información cualquier instancia que quiera solicitar cierto dato de un conjunto de elementos también generará un uso innecesario de la información.

Ejemplificando para poder ambientar el concepto que hemos desarrollado de la abstracción, podemos plantear el escenario de un restaurante al que asisten dos personas, un supervisor de calidad y un comensal. El comensal pedirá la comida que desee y se le será entregada por la clase que sería el restaurante, pero él no necesita saber cómo se preparó el plato o de donde provengan los ingredientes que se está comiendo, sin embargo, el supervisor necesita toda esa información para garantizar la salud en el local.

Al final la abstracción desempeña un papel bastante necesario a la hora de seccionar la información

Herencia

La herencia es el tercer pilar de la programación orientada a objetos (OOP), se caracteriza por tomar datos y rasgos ya establecidos en el programa. Es decir que una clase puede heredar las características de una clase anterior dentro del mismo programa. Teniendo en cuenta que la clase heredad toma los rasgos de la primera podemos añadir nuevas características y modificar ciertos parámetros de esta misma. Ahora bien, a la primera clase se le llama clase base y la clase que hereda las características clase derivada.

Una implementación de esto es si tenemos una clase llamada Persona donde pones datos básicos. Luego agregamos nuevas clases como Medico, Policía o Ingeniero que van a heredar las características de Persona, pero para clases nuevas creadas se le van a agregas rasgos más específicos.

Un ejemplo, los objetos de la clase Medico le agregamos un método nuevo Operar() que tiene la función de atender a los pacientes en una sala de Hospital dentro de un juego. De igual manera, le podemos agregar al departamento al que pertenece con Departamento para identificar a que departamento pertenece, así mismo le podemos poner un nombre para diferenciarlo Nombdoctor. Teniendo esto en cuenta, podemos hacer esto con cada objeto derivado de Persona

Polimorfismo

El polimorfismo es el cuarto y último pilar de programación orientada a objetos, establece que diversos objetos de diferentes clases y que tienen una base común, pero se pueden utilizar de manera indistintas, sin tener la necesidad de saber de qué clase son. Es decir que nos permite utilizar a los objetos de manera genérica, aunque de manera interna se comporta de forma diferente y especifica.

Un ejemplo de funcionamiento del polimorfismo, si tuviéramos un montón de personajes en un mismo lugar. Hay varios médicos, policías y muchas más clases o tipos de personas. En determinado momento todos van a necesitar hablar y como cada uno lo hace de una manera diferente porque son personajes diferentes sería bastante tedioso tener que localizar a una de las personas en específico y luego ponerlo hablar con otro y así sucesivamente. Ahora bien, la idea es tratarlo a todos como si fueran personas, independientemente del tipo en específico de persona que sea para así simplificar la acción

Ya a la hora de derivar toda la clase Persona todos pueden hablar y a la hora de llamar el método Hablar () cada uno de ellos lo ejecutara según este programado para cada clase de Persona con las particularidades que debe seguir en cada caso.

Diferencias de la programación orientada a objetos con la programación imperativa clásica:

Antes de entrar a detalle en las diferencias de estos tipos de programación es preciso entrar en detalle sobre cuales son las definiciones de cada uno en general, para que a la hora de entrar en detalle con las diferencias sea mas claro ver esta con el contexto que nos proveen las definiciones.

La programación orientada a objetos es un tipo de programación en el cual se utiliza el concepto de objeto y clases para la organización de los datos y variables. Mientras que la programación imperativa clásica es el concepto de programación que consiste en una serie de instrucciones que el ordenador lleva acabo en una secuencia especifica. Teniendo en cuenta estos breves conceptos que nos permiten a la realización de las diferencias entre estos dos tipos de programación:

La programación orientada a objeto es el avance de la programación imperativa, ya que este se basa en los fundamentos de la programación imperativa. Sin embargo, mejora conceptos implementando mayor organización y capacidad de modificación.

La programación orientada a objetos permite la creación de variables abstractas y virtuales, que pueden cambiar su función a lo largo del programa.

La programación orientada a objetos nos permite que a la hora de elaborar un nuevo código se pueda reutilizar parte de un código antiguo para una funcionalidad en específico lo que permite que la inversión que se hace a la hora de realizar un código sea recuperada por los inversores de manera sencilla.

La programación orientada a objetos permitió la incorporación de partes de código que son integradas por parte de un tercero lo cual permitió el ahorro de costos y la reducción de trabajo.

A diferencia de la programación clásica, la orientada a objeto posee mas flexibilidad en la confección del código al este estar compuestos de clases y tener la opción de poseer métodos abstractos los cuales dan mucho juego debido a que su funcionalidad puede variar dependiendo de lo que requiera el código en un momento en específico.

El mantenimiento de un código de programación orientada a objetos es más fácil de mantener y darle soporte debido a que estos códigos son más fáciles de leer y están estructurados de una manera donde es más fácil aislar los problemas a la hora de resolver estos o tener que hacer algún cambio en el código.

Los programas que son creados en base a la programación orientada a objetos normalmente son más intuitivos y fácil de leer. Esto nos permite el crear una interfaz gráfica para que el cliente usuario pueda interactuar con este de una manera más intuitiva.

A diferencia de la programación tradicional con la programación orientada a objetos podemos dividirnos los trabajos por tareas a la hora de realizarlos ya que al estos no ser tan lineales es posible que los programadores puedan enfocarse en partes especificas del programa en la que estos se sientan más cómodos y trabajen de forma más eficiente.

Con el análisis de los distintos ejemplos mostrados queda claro que las principales diferencias entre la programación orientada a objetos y la programación imperativa clásica, radica en la forma en como los códigos son confeccionados y la flexibilidad que nos da la implementación de objetos y clases a la hora de realizar un código. Estos son debido a que la implementación de clases estructura el código y permite que este tenga parámetros fáciles de modificar que rigen el comportamiento de esté, mientras que con la programación clásica el programa está compuesto por una serie de instrucciones lineales que son poco flexibles y difíciles de modificar. La programación orientada a objetos surgió como una necesidad para remplazar el paradigma de la programación imperativa clásica, ya que este vino como una solución a los distintos problemas que traía la programación antigua, la cual quedó relegada a un puesto secundario donde los proyectos que son realizados son usando la programación orientada a objetos.

Bibliografía

Clank, D. C. (2013). Beginning C Object Oriented Programming (Segunda edición). APRESS.