Análisis del capítulo 7B- Sobrescribiendo métodos de la clase base

En primer lugar, antes de entrar de manera directa a al núcleo de este capítulo, analicemos las dos ideas principales sobre la que este se sustenta, la cuales son: métodos primordiales de una clase y como usar el calificador en una clase derivada.

Tal y como vimos en el capitulo anterior, una clase abstracta es aquella en la que no se permiten crear instancias de esta por eso se utiliza de clase de soporte o esqueleto de todas las demás clases a partir de esta. Declarando dentro de esta todos los atributos necesarios para las demás clases que la van a suceder o heredar. Pero, aparte de este método tenemos otra alternativa que es definir una interfaz la cual definirá en el método en el que se alberguen todos los atributos de dicha clase. Estas clases que implementan una interfaz son llamadas por contrato para la implementación de los atributos declarados en ellas (cosa que no se puede alterar). Esta manera de hacer el código sirve para que el usuario sepa en todo momento que métodos son los que están a su disposición, los valores de retorno y la manera en la que lo debe de llamar.

Una clase implementa una interfaz utilizando el punto y como seguido de el nombre de la interfaz. Cuando una clase implemente el uso de una interfaz debe de proporcionarse el código de implementación para todas y cada unas de las implementaciones de métodos definidos por interfaz. Dado a que el uso de una interfaz es muy similar al uso de una clase abstracta puede que se pregunten el porque de tomarse la molestia de hacerlo con interfaces en lugar de una clase abstracta (dado que de la segunda manera es más sencillo en términos de cantidad de código). Pues bien, una de las razones por las cuales esto se hace de esta manera es que frente a una clase abstracta las interfaces poseen una ventaja y es el hecho de que .Net Framework no admite la herencia de mas de una clase. Lo cual hace que, a la hora de trabajar en proyecto amplio y bien estructurado este resulte imposible de llevar, dado que en cada clase se tendrían que crear los mimos métodos con los mismos objetos. Sin embargo, como solución a esto se incluyo la capacidad de poseer varias interfaces.

Por otro lado, tenemos el polimorfismo el cual es la capacidad que tiene una clase derivada de la clase base de responder a la misma llamada de un método de la clase base, todo esto de manera única. La ventaja de esto es poder hacer

que el código se mas simple de maneja y por ende de depurar y corregir errores.

Como ejemplo de esto supongamos que se posee una aplicación para saber el valor de una cuenta bancaria. Todo estos como el mismo método *ObtenerInfoClient* el cual recorrería diferentes clases hasta encontrar a cuál es que se le esta haciendo referencia en la llamada de el usuario. De esta manera al agregar la información de una nueva cuenta pues no se necesitará alterar la estructura del código sino agregar otra interfaz.

Además de esto poseemos tres tipos de polimorfismos, los cuales son:

polimorfismo por herencia: Esto es cuando se hereda una clase y se convierte cual en esta. Este como es lógico es el más básicos de los tres que existen y tiene como peculiaridad heredar a su clase padre y sustituirla.

polimorfismos por abstracción: Este es básicamente el mismo ejemplo anterior, pero con una clase abstracta a la cual se sustituye por su clase hijo, heredando de esta manera sus atributos(ya que al ser abstracta no puede poseer instancias). Si, por ejemplo, implementamos una clase llamada perro y la definimos como abstracta. Y dado que cada perro duerme de manera diferente pues creamos otra instancia que sea dormir dentro de la misma.

polimorfismo por interfaz: Esta es la posibilidad que nos da el OOP de crear una interfaz y heredarla pasando a sustituirla. Este que, aunque este mencionado en ultima instancia se considera el polimorfismo más importante, debido a que este esta basado en los contratos, que se encargan decirle lo que puede o no hacer.

A la hora de implementarlo poseemos dos opciones, las cuales son:

El polimorfismo estático es aquél en el que los atributos a los que se aplica el polimorfismo deben ser explícitos y estar todos declarados antes de aplicarle el polimorfismo

El polimorfismo utilizando la palabra reservada *new.* Este lo utilizamos cuando queremos hacer entender que queremos sobrescribir el método de la clase padre en la clase heredada hijo. Esto lo que hace es que nos crea un método totalmente desvinculado y nuevo del anterior método base, pero, manteniendo los atributos e instancias creadas anteriormente. Algo para recalcar es que eso es única y exclusivamente para esa que se este heredando.

Referencias

Clark, D. (2013). Beginning C# Object-Oriented Programming (2nd Ed.). New York: Apress

Netmentor.es. 2021. Polimorfismo en programación orientada a objetos. [online] Available at: <https://www.netmentor.es/entrada/polimorfismo-poo> [Accessed 30 June 2021].