Análisis del capítulo 7A- creando clases jerárquicas

En primer lugar, antes de adentrarnos en el tema debemos de saber claramente la definición de un concepto, el cual se repetirá a lo largo de todo este capítulo. El termino herencia hace alusión al conjunto de bienes, derechos y obligaciones que adquiere una persona luego de heredar a alguien tras el fallecimiento de esta. En el caso de la programación orientada a objetos tenemos que la herencia es la capacidad de crear una clase como base con atributos y métodos ya definidos por el usuario y que de esta clase base podamos heredar a demás clases derivadas y que así estas adquieran todos sus métodos y atributos.

La finalidad del uso del recurso de la herencia es crear una clase base que guarde todos los atributos que luego van a necesitar las otras demás clases derivadas (del mismo tipo), un ejemplo práctico es si tenemos una clase llama *área* en la cual creamos un método para calcular el área de diferentes figuras geométricas. Dado que no a todas figuras geométricas se les calcula el área de la misma manera en el método base pondremos la entrada de datos como la cantidad de lados que tiene la figura más un atributo llamado *lado* Y dependiendo de esto con un switch usaremos un método u otro, pero todos los demás heredarían a este método base llamado *área.*

Además de manejar los atributos y métodos dentro de una clase base para luego ser heredada también podemos heredar otras cosas de los métodos creados en la clase base, como lo son las excepciones. Si esto lo aplicamos a el ejemplo anterior de un programa que nos calcule el área de diferentes figuras, pues en el método de base podemos implementar un tipo de manejo de excepción el cual no le permita al usuario ingresar datos de tipo string en la parte de la cantidad de lados de la figura y de ser así que un catch reciba este error para que programa no finalice y en lugar de esto le diga al usuario que digite un número en lugar de una letra u otro carácter.

En el entorno de desarrollo de C# por defecto se puede heredar a cualquier clase. Dado que esta posibilidad existe se debe de tener cuidado a la hora de utilizar esta herramienta en códigos amplios, ya que algunos atributos o métodos podrían solapar a los que fueron heredados y viceversa. Pero, para estos casos existe una alternativa que es sellar la clase, al ser sellada la clase esta no podría ser heredada por ninguna de las demás. Evitando de esta manera posibles complicaciones en el código debido a ambigüedades de clases que tengan atributos parecidos y solapamiento entre constructores.

Por otro lado, tenemos a la clase abstracta, la cual, si nos permite que la heredemos a otras clases, pero no nos permite

crear instancias de está. Por lo cual es utilizada como esqueleto o clase base, para en ella definir los parámetros o atributos que manejaran las que la sucedan. Un buen ejemplo de esto seria nuestra clase área la cual se usa para calcular el área de diferentes figuras. Bueno pues dentro de esta en un constructor podemos definir los parámetros o atributos que utilizaran sus predecesoras, dado que esta la utilizaremos como base fundamental o esqueleto de todas las demás.

A la hora definir las jerarquías de las clases debemos de tener en cuenta algo llamado modificadores de acceso, los cuales pueden ser públicos y privados y de estos depende si una clase que heredó a otra puede acceder o no a los parámetros de la que la antecede. Entonces recapitulando si se desea poder acceder en cualquier momento a los atributos, métodos y constructores de una clase esta debe de ser publica y si por el contrario se desea que no pueda ser así pues esta debe de ser privada.

Cuando una clase derivada hereda un método de la clase de base. Esta hereda la manera en la que esta implementa los métodos. Como programador se puede querer que la clase de derivada posea un método que se llame igual, posea la misma cantidad de parámetros que la clase base. Estos parámetros posean el mismo tipo de dato que el que se hereda de la clase de base. Aunque es posible mediante una sobrecarga de métodos (algo que veremos más adelantes), lo mas factible es lo que se conoce como sobrescribir el método base, dándole atributos que el anterior no poseía y sobre todo haciendo que el programa pueda diferenciar entre uno y el otro.

Los métodos heredados por la clase ya derivada pueden sobrecargar, a esto se le llama sobrecarga de métodos y sucede cuando el nombre, y el tipo de datos de dos métodos que están dentro de una misma clase son iguales. Para que esto pueda acontecer se necesita que la cantidad de parámetros de los métodos que se desean sobrecargar sean diferentes los unos con los otros.

Si un método tiene el mismo nombre que otro dentro de la misma clase y además usa el mismo tipo de dato dentro de sus parámetros, pero, además de esto, no esta utilizando la palabra reservada override oculta de manera correcta el método dentro de la clase. Aunque en la mayoría de los casos estos se considera una muy mala práctica de programación, ya que esto puede terminar llevando a un solapamiento entre los métodos de una misma clase. Aunque el programa se ejecutara correctamente visual estudio nos presenta una advertencia subrayando con color verde el nombre del método que se este sobrescribiendo de manera implícita dejándonos saber que uno de los dos (pueden ser n cantidad de métodos dentro de cada clase) no se está ejecutando correctamente.

Referencias

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