HERENCIA

Vimos en el concepto anterior que dos clases pueden estar relacionadas por la colaboración. Ahora veremos otro tipo de relaciones entre clases que es la Herencia.

La herencia significa que se pueden crear nuevas clases partiendo de clases existentes, que tendrá todos los atributos, propiedades y los métodos de su 'superclase' o 'clase padre' y además se le podrán añadir otros atributos, propiedades y métodos propios.

Clase padre:

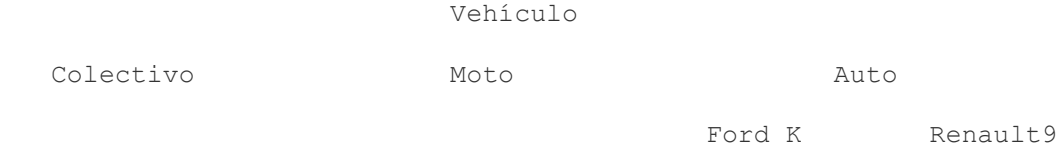
Clase de la que desciende o deriva una clase. Las clases hijas (descendientes) heredan (incorporan) automáticamente los atributos, propiedades y métodos de la la clase padre.

Subclase:

Clase descendiente de otra. Hereda automáticamente los atributos, propiedades y métodos de su superclase. Es una especialización de otra clase. Admiten la definición de nuevos atributos y métodos para aumentar la especialización de la clase.

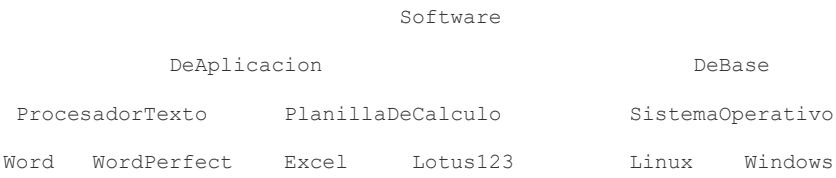
Veamos algunos ejemplos teóricos de herencia:

1) Imaginemos la clase Vehículo. Qué clases podrían derivar de ella?



Siempre hacia abajo en la jerarquía hay una especialización (las subclases añaden nuevos atributos, propiedades y métodos.

2) Imaginemos la clase Software. Qué clases podrían derivar de ella?



El primer tipo de relación que habíamos visto entre dos clases es la de colaboración. Recordemos que es cuando una clase contiene un objeto de otra clase como atributo.

Cuando la relación entre dos clases es del tipo "...tiene un..." o "...es parte de...", no debemos implementar herencia. Estamos frente a una relación de colaboración de clases no de herencia.

Si tenemos una ClaseA y otra ClaseB y notamos que entre ellas existe una relacion de tipo "... tiene un...", no debe implementarse herencia sino declarar en la clase ClaseA un atributo de la clase ClaseB.

Por ejemplo: tenemos una clase Auto, una clase Rueda y una clase Volante. Vemos que la relación entre ellas es: Auto "...tiene 4..." Rueda, Volante "...es parte de..." Auto; pero la clase Auto no debe derivar de Rueda ni Volante de Auto porque la relación no es de tipo-subtipo sino de colaboración. Debemos declarar en la clase Auto 4 atributos de tipo Rueda y 1 de tipo Volante.

Luego si vemos que dos clase responden a la pregunta ClaseA "..es un.." ClaseB es posible que haya una relación de herencia.

Por ejemplo:

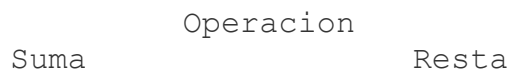
```
Auto "es un" Vehiculo
Circulo "es una" Figura
Mouse "es un" DispositivoEntrada
Suma "es una" Operacion
```

Problema 1: (Ejercicio 7)

Ahora plantearemos el primer problema utilizando herencia. Supongamos que necesitamos implementar dos clases que llamaremos Suma y Resta. Cada clase tiene como atributo valor1, valor2 y resultado. Las propiedades a definir son Valor1, Valor2 y Resultado, el método Operar (que en el caso de la clase "Suma" suma los dos Valores y en el caso de la clase "Resta" hace la diferencia entre Valor1 y Valor2.

Si analizamos ambas clases encontramos que muchas propiedades son idénticos. En estos casos es bueno definir una clase padre que agrupe dichas propiedades, atributos y responsabilidades comunes.

La relación de herencia que podemos disponer para este problema es:



Solamente el método operar es distinto para las clases Suma y Resta (esto hace que no lo podamos disponer en la clase Operacion), luego las propiedades Valor1, Valor2 son idénticos a las dos clases, esto hace que podamos disponerlos en la clase Operacion. Lo mismo las propiedades Valor1, Valor2 y Resultado se definirán en la clase padre Operacion.

Crear un proyecto y luego crear cuatro clases llamadas: Operacion, Suma, Resta y Prueba

Problema 2: (Ejercicio 8)

Confeccionar una clase Persona que tenga como atributos el nombre y la edad (definir las propiedades para poder acceder a dichos atributos). Definir como responsabilidad un método para imprimir.

Plantear una segunda clase Empleado que herede de la clase Persona. Añadir un atributo sueldo (y su propiedad) y el método para imprimir su sueldo.

Definir un objeto de la clase Persona y llamar a sus métodos y propiedades. También crear un objeto de la clase Empleado y llamar a sus métodos y propiedades.

CLASE PARCIAL (PARTIAL CLASS)

Hasta ahora hemos visto que una clase se la implementa en forma completa dentro de un archivo. El lenguaje C# permite la implementación de una clase en dos o más archivos. Para esto hay que agregarle el modificador *partial* cuando declaramos la clase.

Este concepto es ampliamente utilizado por el entorno del Visual Studio .Net en la generación de interfaces visuales.

Como veremos en conceptos futuros es necesario presentar "partial class" para su entendimiento.

Una clase parcial no es más ni menos que crear una clase completa y luego agrupar métodos y propiedades en dos o más archivos.

Problema 1: (Ejercicio 11)

Plantear una clase Rectángulo, definir dos propiedades: Lado1 y Lado2. Definir dos métodos RetornarSuperficie y RetornarPerimetro. Dividir la clase en dos archivos utilizando el concepto de "partial class".

Para codificar este proyecto procedemos de la siguiente forma:

1. Seleccionamos desde el menú de opciones Archivo -> Nuevo proyecto...
2. En el diálogo definimos el nombre del proyecto: ClaseParcial1
3. Ahora tenemos que agregar los otros archivos. Presionamos el botón derecho del mouse en la ventana del "Explorador de soluciones" sobre el nombre del proyecto ("ClaseParcial1") y seleccionamos la opción Agregar -> Nuevo elemento.
4. Seleccionamos en el diálogo la plantilla "Clase" y en la parte inferior del diálogo definimos el nombre del archivo, en nuestro caso lo llamamos "archivo1.cs".
5. En este archivo planteamos la clase "partial class Rectangulo" que define las dos propiedades y atributos.
6. En forma similar seguimos los pasos para crear el archivo2.cs y codificar la clase "partial class Rectángulo" que define los dos métodos.
7. Finalmente codificamos la clase principal en el archivo Program.cs. En la main creamos un objeto de la clase Rectángulo e inicializamos las propiedades y llamamos a sus métodos.