Ejercicios

Herencia

Herencia

Antes de empezar

- Deberá asignarle a la propiedad 'Title' de la clase Console, el número de ejercicio.

  Por ejemplo:
  Console.Title = “Ejercicio Nro ##” donde ## será el número del ejercicio.

- Del mismo modo se deberán nombrar los proyectos de consola.

  Por ejemplo:
  Ejercicio_##.

- Para visualizar los valores decimales de los ejercicios, deberá dar el siguiente formato de salida al método Write/WriteLine: “#,###.00”.

I01. El viajar es un placer

Crear un proyecto de biblioteca de clases con las clases `Automovil`, `Moto` y `Camion`.

1. Crear un enumerado:

   a. `Colores` { Rojo, Blanco, Azul, Gris, Negro }

2. `Furgon` tendrá los atributos:
   a. `cantidadRuedas: short`
   b. `cantidadPuertas : short`
   c. `color : Colores`
   d. `cantidadMarchas : short`
   e. `pesoCarga : int.`
3. `Automovil` tendrá los atributos:
   a. `cantidadRuedas : short`
   b. `cantidadPuertas : short`
   c. `color : Colores`
   d. `cantidadMarchas : short`
   e. `cantidadPasajeros : int.`
4. `Ciclomotor` tendrá los atributos:
   a. `cantidadRuedas : short`
   b. `cantidadPuertas : short`
   c. `color : Colores`
   d. `cilindrada : short`
5. Crearle a cada clase un constructor que reciba todos sus atributos.
6. Crear la clase `VehiculoTerrestre` y abstraer la información necesaria de las clases anteriores. Luego generar una relación de herencia entre ellas, según corresponda.
7. `VehiculoTerrestre` tendrá un constructor que recibirá todos sus atributos.
8. Modificar las clases que heredan de ésta para que lo reutilicen.
9. Crear propiedades para acceder a los atributos de cada clase.

Crear un proyecto de consola y generar el código necesario para probar las clases.

I02. Lavadero

Crear en un proyecto de tipo Class Library la siguiente jerarquía de clases:

Clase `Vehiculo` que posea como atributos protegidos:

- `patente : string` (con una propiedad sólo lectura)
- `cantidadRuedas : Byte`
- `marca : EMarcas` (con los siguientes enumerados: Honda, Ford, Zanella, Scania, Iveco y Fiat). Crear propiedad de sólo lectura.

Y los siguientes métodos:

- `(~) Mostrar() : string`
- `(+) Vehiculo (string, Byte, EMarcas)` (sin sobrecargas)

Sobrecarga de operadores:

- `(+) == (Vehiculo, Vehiculo) : bool.` Si las patentes y marcas son iguales, retorna TRUE.
- `(+) != (Vehiculo, Vehiculo) : bool.` Si las patentes y marcas son distintas, retorna TRUE.

Además, se pide:

Crear tres clases (`Auto, Camion y Moto`) que hereden de `Vehiculo` y que posean: `cantidadAsientos (int)`, para auto, `tara (float)`, para camión y `cilindrada (float)`, para moto. Todos atributos protegidos.
Cada una de estas clases deberá implementar el método `MostrarAuto, MostrarCamion` y `MostrarMoto` (reutilizando código de la clase base) para poder retornar un *string* con todos sus atributos.
Generar un constructor en cada clase para inicializar cada uno de los atributos.

Por último, se desea construir la clase `Lavadero` que tendrá como atributos:

- `(-) vehiculos : List<Vehiculo>`
- `(-) precioAuto : float`
- `(-) precioCamion :float`
- `(-) precioMoto :float`

Todos los atributos se inicializarán desde su constructor con parámetros. El constructor por default, que será privado, será el único encargado de inicializar la lista genérica.

Tendrá una propiedad de sólo lectura (`Detalle : string`) que retornará la información completa del lavadero: precios, vigentes y el listado completo de los vehículos que contiene. Reutilizar código.

También poseerá una propiedad de sólo lectura `Vehiculos`, asociada a la lista genérica.

Los métodos que tendrá Lavadero son:

- `MostrarTotalFacturado`: devolverá la ganancia total del lavadero (`Double`), dicho método tendrá una sobrecarga que reciba como parámetro la enumeración `EVehiculos` (con `Auto`, `Camión` y `Moto` como enumerados) y retornará la ganancia del `Lavadero` por tipo de vehículo.
- Sobrecarga == entre un lavadero y un vehículo, retornara TRUE, si el vehículo se encuentra en el lavadero.
- Sobrecarga del operador +, que agregara un vehículo siempre y cuando el vehículo no se encuentre en el lavadero. *Ej. miLavadero += unAuto*;
- Sobrecarga del operador -, que quitara al vehículo del lavadero, siempre y cuando este dicho vehículo. *Ej. miLavadero -= unaMoto*;
- Generar un método estático (`OrdenarVehiculosPorPatente : int`) que reciba dos vehículos y retorne un 0 (cero), si ambas patentes son iguales, si la primera patente es 'mayor' que la segunda, retornará un 1 (uno) y si no, retornará un -1 (menos uno).
- Generar un método de instancia (`OrdenarVehiculosPorMarca : int`) que reciba dos vehículos retorne un 0 (cero), si ambas marcas son iguales, si la primera marca es 'mayor' que la segunda, retornará un 1 (uno) y si no, retornará un -1 (menos uno).

La aplicación debe poder ingresar vehículos de distintos tipos y marcas al lavadero, quitarlos, obtener las ganancias totales o por tipo de vehículo y mostrar los vehículos ingresados al lavadero ordenados por los distintos criterios.

Referencias:

(+) → public
(~) → protected
(-) → private