Ejercicios

Sobrecarga y encapsulamiento

Ejercicios de sobrecarga y encapsulamiento

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.

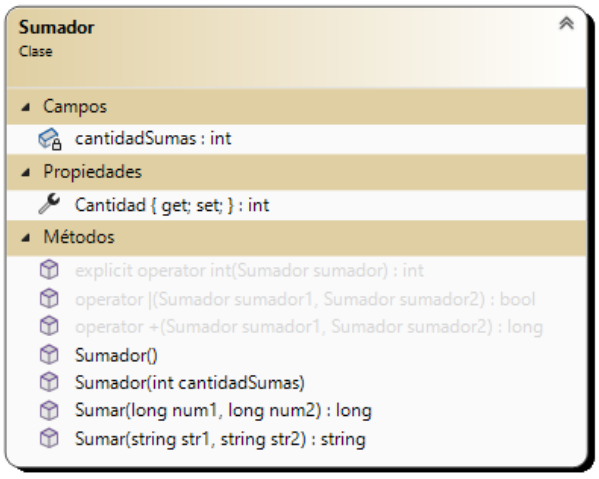
- Del mismo modo se deberán nombrar los proyectos de consola.

  Por ejemplo:
  Ejercicio_##.

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

I01. Sumador

Crear un proyecto de tipo biblioteca de clases y agregar la clase `Sumador`.



1. Crear dos constructores:
   a. `Sumador(int)` inicializa `cantidadSumas` en el valor recibido por parámetro.
   b. `Sumador()` inicializa `cantidadSumas` en cero. Reutilizará al primer constructor.
2. El método `Sumar` incrementará `cantidadSumas` en 1 y adicionará sus parámetros con la siguiente lógica:
   a. En el caso de `Sumar(long, long)` sumará los valores numéricos
   b. En el de `Sumar(string, string)` concatenará las cadenas de texto.

Crear un proyecto de consola y agregar un objeto del tipo `Sumador` en el método `Main` y probar el código.

1. Generar una conversión explícita que retorne `cantidadSumas`.
2. Sobrecargar el operador *+* (suma) con dos operadores de tipo Sumador. El resultado será un *long* correspondiente al resultado de la suma del atributo `cantidadSumas` de cada argumento.
3. Sobrecargar el operador / (pipe) con dos operadores de tipo `Sumador`. Deberá retornar *true* si ambos sumadores tienen igual valor en el atributo `cantidadSumas`.

Agregar un segundo objeto del tipo `Sumador` en el método `Main` y probar el código.

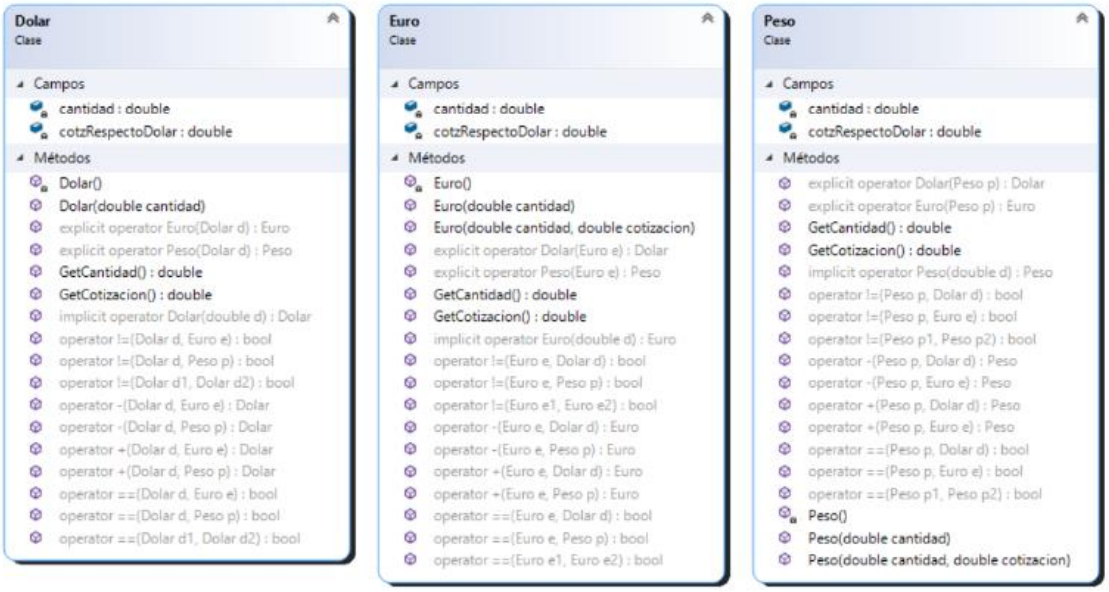
I02. Cotizador

Crear un proyecto de tipo biblioteca de clases. Construir tres clases dentro de un namespace llamado `Billetes`: `Pesos`, `Euro` y `Dolar`.

Se debe lograr que los objetos de estas clases se puedan sumar, restar y comparar entre sí con total normalidad como si fueran tipos numéricos, teniendo presente que 1 Euro equivale a 1,17 Dólares y 1 Dólar equivale a 102,65 Pesos.

1. El atributo `cotizacionRespectoDolar` y el método `GetCotizacion` son estáticos.
2. Los constructores estáticos le darán a las clases una cotización por defecto respecto del dólar.
3. Sobrecargar los operadores explicit y/o implicit para lograr compatibilidad entre los distintos tipos de datos.
4. Los operadores de comparación == comparan cantidades.
5. Reutilizar el código. Sólo realizar las conversiones dentro de los operadores para dicho uso.
6. Crear un proyecto de consola y colocar dentro del método `Main` el código necesario para probar todas las funcionalidades.
7. Agregar enumeradores para representar las monedas (Pesos, Euro y Dolar) y propiedades para acceder a los atributos de cada clase.

Nota: Para operar dos tipos de monedas distintos, primero se deberá convertir todo a una misma moneda y luego realizar la operación. Por ejemplo, si quiero sumar `Dolar` y `Euro`, deberé convertir el `Euro` a `Dólar` y luego sumarlos.



I03. Conversor binario recargado

Partiendo del ejercicio I03. Conversor binario de la clase 02 , se pide agregar las clases:

`NumeroBinario`:

1. Único atributo `numero` de tipo `string`.
2. Único constructor privado (recibe un parámetro de tipo `string`).

`NumeroDecimal`:

1. Único atributo `numero` de tipo `double`.
2. Único constructor privado (recibe un parámetro de tipo `double`).

Utilizando los métodos de la clase `Conversor` donde corresponda, agregar las sobrecargas de operadores:

`NumeroBinario`:

1. `string +` (NumeroBinario, NumeroDecimal)
2. `string –` (NumeroBinario, NumeroDecimal)
3. `bool ==` (NumeroBinario, NumeroDecimal)
4. `bool !=` (NumeroBinario, NumeroDecimal)

`NumeroDecimal`:

1. `double +` (NumeroDecimal, NumeroBinario)
2. `double –` (NumeroDecimal, NumeroBinario)
3. `bool ==` (NumeroDecimal, NumeroBinario)
4. `bool !=` (NumeroDecimal, NumeroBinario)

Agregar conversiones implícitas para poder ejecutar:

```
NumeroBinario objBinario = "1001";
NumeroDecimal objDecimal = 9;
```

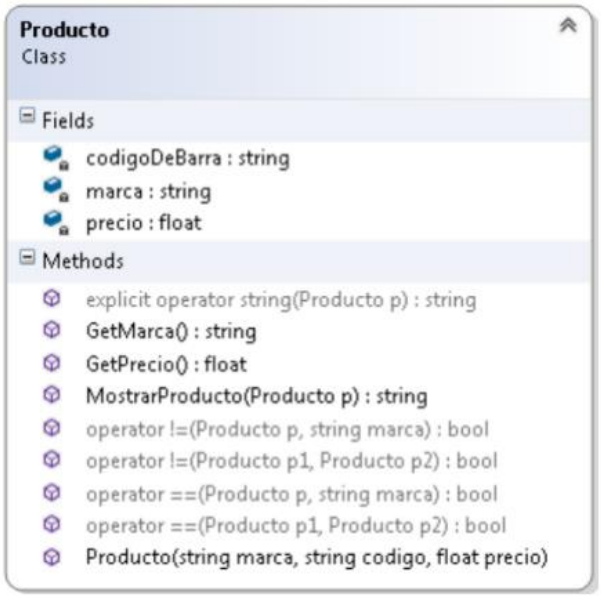
Agregar conversiones explícitas para poder ejecutar:

```
string binario = (string)objBinario;
double numeroDecimal = (double)objDecimal;
```

Generar el código en el método `Main` para instanciar un objeto de cada tipo y operar entre ellos, imprimiendo cada resultado por pantalla.

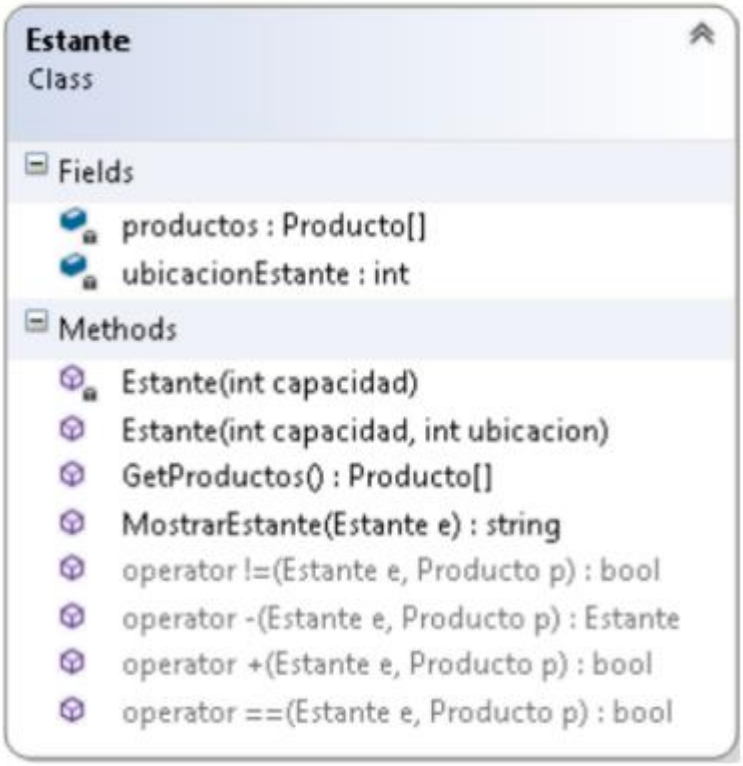
I03. La estantería

Crear un proyecto de tipo biblioteca de clases que contenga la clase `Producto`.



1. Todos sus atributos son privados.
2. Posee sólo un constructor de instancia.
3. El método `GetMarca`, retornará el valor correspondiente al atributo marca.
4. También poseerá el atributo `codigoDeBarras`, el cual se publicará sólo a través de la conversión explícita nombrada más adelante.
5. El método de clase (estático) `MostrarProducto` es público y retornará una cadena detallando los atributos de la clase.
6. Posee las siguientes sobrecargas de operadores:
   a. `explicit`: Realizará la conversión de un objeto `Producto` a string. Sólo retorna el atributo `codigoDeBarras` del producto.
   b. `== (Producto, Producto)`: Retornará *true* si las marcas y códigos de barra son iguales, *false* caso contrario.
   c. `== (Producto, string)`: Retornará *true* si la marca del producto coincide con la cadena pasada como argumento, false caso contrario.

Generar la clase `Estante`:



1. Posee dos atributos privados. Uno será un entero que indicará la ubicación del estante y el otro es un array de tipo Producto.
2. El constructor de instancia privado será el único que incializará el array. La sobrecarga pública del constructor inicializará la ubicación del estante, recibiendo como parámetros la capacidad y la ubicación. Reutilizar código.
3. El método público `GetProductos`, retornará el array de productos.
4. El método público de clase (estático) `MostrarEstante`, retornará una cadena con toda la información del estante incluyendo el detalle de cada uno de sus productos. Reutilizar código.
5. Posee las siguientes sobrecargas de operadores:
    a. ==: Retornará true si es que el producto ya se encuentra en el estante, false caso contrario.
    b. +: Retornará *true* y agregará el producto si el estante posee capacidad de almacenar al menos un producto más y dicho producto no se encuentra en el estante, *false* caso contrario. Reutilizar código.
    c. −: Retornará un estante sin el producto, siempre y cuando el producto se encuentre en el listado. Reutilizar código.

Crear un proyecto de consola y agregar en el método Main el siguiente código:

```
// Crear un estante
Estante estante = new Estante(3, 1);
```

```
// Crear 4 productos
Producto p1 = new Producto("Pepsi", "PESDS97413",
(float)18.5);
Producto p2 = new Producto("Coca-Cola", "COSDS55752",
(float)11.5);
Producto p3 = new Producto("Manaos", "MASDS51292",
(float)20.5);
Producto p4 = new Producto("Crush", "CRSDS54861",
(float)10.75);

// Agregar los productos al estante
if (estante + p1)
{
  Console.WriteLine("Agregó {0} {1} {2}", p1.GetMarca(),
(string)p1,
  p1.GetPrecio());
}
else
{
  Console.WriteLine("¡NO agregó {0} {1} {2}!",
p1.GetMarca(), (string)p1,
  p1.GetPrecio());
}

if (estante + p1)
{
  Console.WriteLine("Agregó {0} {1} {2}", p1.GetMarca(),
(string)p1,
  p1.GetPrecio());
}
else
{
  Console.WriteLine("¡NO agregó {0} {1} {2}!",
p1.GetMarca(), (string)p1,
  p1.GetPrecio());
}

if (estante + p2)
{
  Console.WriteLine("Agregó {0} {1} {2}", p2.GetMarca(),
(string)p2,
  p2.GetPrecio());
}
else
{
  Console.WriteLine("¡NO agregó {0} {1} {2}!",
p2.GetMarca(), (string)p2,
  p2.GetPrecio());
}

if (estante + p3)
{
  Console.WriteLine("Agregó {0} {1} {2}", p3.GetMarca(),
(string)p3,
  p3.GetPrecio());
```

```
}
else
{
  Console.WriteLine("¡NO agregó {0} {1} {2}!",
p3.GetMarca(), (string)p3,
  p3.GetPrecio());
}

if (estante + p4)
{
  Console.WriteLine("Agregó {0} {1} {2}", p4.GetMarca(),
(string)p4,
  p4.GetPrecio());
}
else
{
  Console.WriteLine("¡NO agregó {0} {1} {2}!",
p4.GetMarca(), (string)p4,
  p4.GetPrecio());
}

// Mostrar todo el estante
Console.WriteLine();
Console.WriteLine("<----------------------------------------
-------->");
Console.WriteLine(Estante.MostrarEstante(estante));
```

I03. Fahrenheit 451

Crear un proyecto de consola y un proyecto de biblioteca de clases. Agregar al último tres clases `Fahrenheit`, `Celsius` y `Kelvin`.

Realizar la implementación necesaria para poder convertir una temperatura en sus distintas unidades de medida (Fahrenheit, Celsius y Kelvin).

Utilizar sobrecargas de métodos, operadores y/o conversiones.

F = C * (9/5) + 32
C = (F-32) * 5/9
F = K * 9/5 – 459.67
K = (F + 459.67) * 5/9