Relación de Ejercicios de Clases y Objetos (5): Dinero y Moneda

En esta relación, implementaremos la clase **Dinero** que nos permitirá guardar información sobre una cantidad de dinero expresada en una moneda concreta (p.ej.: 23,45 dólares). Para que todo funcione como es debido, mantendremos una lista con los cambios de moneda (p.ej.: cuántos dólares corresponden a un euro) que se mantendrá de manera estática para todos los objetos de la clase Dinero, de manera que todos usaran la misma tabla para las conversiones.

Para que todo esto funcione, necesitaremos los siguientes elementos:

1) Un **enum** que se llamará **TipoMoneda** en el que tendremos una lista de las monedas con las que trabajaremos. El hecho de que esté en un *enum* le facilita al usuario el poder escoger la moneda fácilmente.

2) Una clase **Moneda** que nos servirá para mantener la información sobre cada moneda (es decir, euros, dólares, etc.) que usemos. De cada moneda guardaremos el símbolo, el número de decimales que usa y el cambio en euros.
   - Atributos:
     - **tMoneda**, de tipo *TipoMoneda*, que usaremos para identificar de qué moneda estamos guardando los datos.
     - **decimales**, de tipo *int*, en el que guardaremos cuántos decimales se muestran para esta moneda (p.ej.: en el euro tenemos dos decimales, los céntimos de euro, mientras que los yenes no tienen decimales). Estos decimales lo usaremos solamente a la hora de mostrar las cantidades por pantalla. Internamente, guardaremos todos los decimales para que no haya problema con los redondeos.
     - **simbolo**, de tipo *string*, en el que almacenamos el símbolo que se usa para representar la moneda (€, $, etc.).
     - **cambioEuro**, de tipo *decimal*, en el que guardaremos el cambio con respecto al euro de cada moneda. Más concretamente, guardaremos a qué cantidad de esta moneda corresponde un euro (p.ej.: 1€ = 136.0497¥).
   - Constructor:
     - Tendrá un único constructor que leerá los cuatro valores y los pondrá en los cuatro atributos. Haremos las comprobaciones pertinentes: que el número de decimales sea sensato (entre 0 y 4 es lo normal), que la cadena del símbolo no esté vacía y que el cambio con el euro no sea negativo.
   - Propiedades:
     - Las propiedades **TMoneda**, **Decimales** y **Simbolo** serán de sólo lectura, ya que una vez creada una moneda no vamos a cambiar estos valores.
     - La propiedad **CambioEuro** será de lectura y escritura. A la hora de modificarlo, habrá que tener cuidado de que el valor no sea negativo.

Una vez tengamos lista la clase **Moneda** y el enum **TipoMoneda**, ya podremos hacer nuestra clase **Dinero**. Esta clase tendrá los siguientes elementos:

- Atributo estático:
  - Tendremos un atributo estático **listaMonedas** que será una lista de la clase *Moneda*. Será estático para que una vez rellena con datos la lista, nos sirva para todas las cantidades de dinero (es independiente de cada objeto). Además, si actualizamos esta lista, se actualizarán automáticamente todos los objetos *Dinero* que la usan.
- Constructor estático:
  - A continuación, escribiremos un constructor **static** para inicializar esa lista de monedas con unos datos iniciales.
    Por cada valor que tengamos en *TipoMoneda*, tendremos que añadir una entrada a esa lista de monedas con los datos de esa moneda concreta. Este constructor se ejecuta automáticamente una sola vez cuando vayamos usar el primer objeto de la clase *Dinero*, así que tendremos la lista rellena cuando nos haga falta.
    Por ejemplo, el primer dato podría insertarse así:
    ```
    listaMonedas.add(new Moneda(TipoMoneda.euro, 2, "€", 1m));
    ```
- Métodos estáticos:
  - **ActualizaCambio**(*TipoMoneda* t, *decimal* cambio), al que le pasaremos un *TipoMoneda* y un nuevo valor para el cambio en euros de esa moneda y lo actualizará en la lista.
  - Como nos va a hacer falta en varios métodos consultar datos sobre nuestra moneda, también vendría bien hacer un método (en este caso privado) **BuscaMoneda**(*TipoMoneda* t) que nos devuelva la moneda de la lista.

Hasta aquí sería la parte estática, que es la menos habitual. Con esto, ya tendríamos una lista con las diferentes monedas y algunas funciones para acceder a ella. A continuación, escribiremos los atributos y métodos normales, en los que usaremos esa listaMonedas.

- Atributos:
  - **cantidad**, será un *decimal* en el que se guardará la cantidad exacta de dinero, independientemente del número de decimales que use la moneda concreta.
  - **tMoneda**, será de *TipoMoneda* y nos guardará en qué moneda está representada la cantidad (ej.: euros).
- Constructores:
  - Un constructor con dos parámetros: cantidad (*decimal*) y tMoneda (*TipoMoneda*), que inicializará los atributos.
  - Otro constructor igual al que le pasamos un *int* en lugar de un *decimal*.
  - Otro constructor igual al que le pasamos un *double* en lugar de un *decimal*.
- Propiedades:
  - Propiedades de lectura y escritura para ambos atributos.
- Métodos:
  - **ToString**(), nos devolverá un *string* con la representación en texto de nuestro dinero (ej.: “4.25$”). El símbolo correspondiente a nuestra moneda

lo cogeremos de la lista estática listaMonedas, así como el número de decimales que tendremos que escribir. Para redondear a los decimales deseados, usaremos Math.Round().

- **ValorEn**(*TipoMoneda*) nos devolverá un *double* que corresponderá al valor de nuestro dinero en la moneda que nos pasen por parámetro (p.ej.: si nuestro dinero es "2€" y pedimos el valor en yenes, nos devolverá "251.53"). Para ello, tendremos que buscar la conversión en euros de cada moneda en la lista estática listaMonedas.
- **ConvierteEn**(*TipoMoneda*) es similar al método anterior pero nos devuelve un objeto de tipo Dinero con la cantidad y la moneda correspondiente (p.ej.: si nuestro dinero es "2€" nos devolverá otro dinero que será "251.53¥").
- **ToString**(*TipoMoneda*) será como el ToString() anterior pero nos escribirá la cantidad representada en la moneda que nosotros le digamos (con sus correspondientes decimales y su simbolito).

- Métodos (operadores):
  - **operator +**(*Dinero* d1, *Dinero* d2) y **operator -** que nos permiten sumar y restar cantidades de dinero y nos devuelven un objeto *Dinero* con el resultado. Si las cantidades están en diferentes monedas, habrá que convertir el segundo *Dinero* a nuestra moneda para poder hacer las operaciones.
  - **operator ***(*Dinero* d1, *double* d2) y **operator /** nos multiplican o dividen nuestro *Dinero* entre el número que le pasamos y nos devuelve otro *Dinero* con el resultado. Para que funcione la propiedad conmutativa, habrá que crear también el **operator***(*double* d2, *Dinero* d1)
  - **operator -**(*Dinero* d) nos devolverá nuestro *Dinero* en negativo.
  - **operator ==**(*Dinero* d1, *Dinero* d2), **operator !=**, **operator >**, **operator <**, **operator >=**, y **operator <=** nos devolverán un bool y nos comparán dos *Dinero*s. Si no están en la misma *Moneda*, convertiremos el segundo a la *Moneda* del primero para compararlos.

Ejercicios adicionales (opcional)

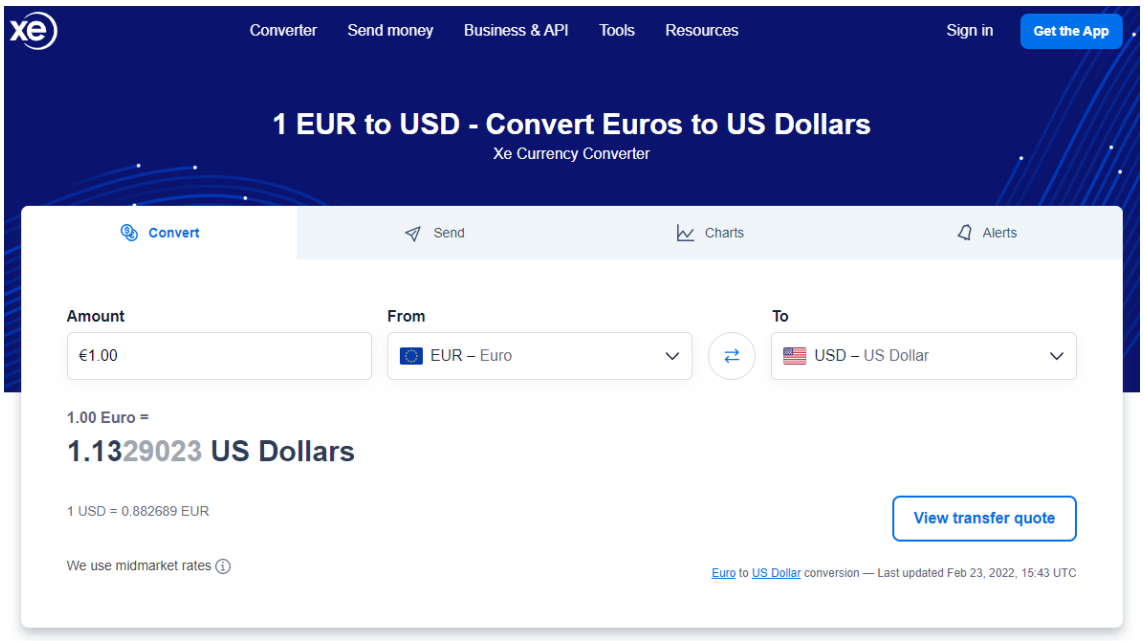
Estaría chulo que la lista de cambio de monedas se actualizara automáticamente de internet. Para ello, usaremos las clases *URL* e *InputStream* para descargarnos los valores de una página web. Aquí tenéis un ejemplo:

```
WebClient wc = new WebClient();
string pagina = wc.DownloadString("http://www.stackoverflow.com");
Console.WriteLine(pagina);
```

Para descargar los valores podemos usar, por ejemplo, la página https://ww.xe.com/, que nos muestra en una página web el cambio simplemente añadiendo parámetros en la URL.

P.Ej.: https://www.xe.com/currencyconverter/convert/?Amount=1&From=EUR&To=USD

nos devolvería:



y de la página HTML que nos descarguemos tendríamos que extraer el cambio.

O podemos usar esta otra página que nos da en JSON todos los cambios con respecto al euro:

Ej.: https://api.exchangerate.host/base=EUR

{"motd":{"msg":"If you or your company use this project or like what we doing, please consider backing us so we can continue maintaining and evolving this project.","url":"https://exchangerate.host/#/donate"},"success":true,"historical":true,"base":"EUR","date":"2022-02-23","rates":{"AED":4.160324,"AFN":104.125614,"ALL":121.193652,"AMD":543.547019,"ANG":2.042903,"AOA":565.502832,"ARS":121.406542,"AUD":1.566652,"AWG":2.038955,"AZN":1.926025,"BAM":1.953927,"BBD":2.265125,"BDT":97.4581,"BGN":1.957546,"BHD":0.427184,"BIF":2265.923521,"BMD":1.133233,"BND":1.526067,"BOB":7.804756,"BRL":5.729173,"BSD":1.133007,"BTC":0.00003,"BTN":84.730827,"BWP":13.043817,"BYN":2.997841,"BZD":2.285304,"CAD":1.443973,"CDF":2268.622323,"CHF":1.043653,"CLF":0.033744,"CLP":897.725876,"CNH":7.162429,"CNY":7.165567,"COP":4463.198428,"CRC":724.406109,"CUC":1.133716,"CUP":29.161179,"CVE":110.868574,"CZK":24.545325,"DJF":201.778286,"DKK":7.437906,"DOP":64.183199,"DZD":159.291488,"EGP":17.794134,"ERN":16.987942,"ETB":57.66416,"EUR":1,"FJD":2.410443,"FKP":0.833385,"GBP":0.833518,"GEL":3.369633,"GGP":0.833134,"GHS":7.396852,"GIP":0.833 71,"GMD":60.36049,"GNF":10183.867437,"GTQ":8.740657,"GYD":237.168577,"HKD":8.836575,"HNL":27.91238,"HRK":7.534596,"HTG":117.785497,"HUF":355.388713,"IDR":16258.146102,"ILS":3.654387,"IMP":0.833169,"INR":84.588742,"IQD":1654.440341,"IRR":47874.788092,"ISK":141.172901,"JEP":0.833251,"JMD":176.132869,"JOD":0.80341,"JPY":130.286751,"KES":128.818143,"KGS":96.031334,"KHR":4610.168286,"KMF":491.970245,"KPW":1019.215467,"KRW":1350.49028,"KWD":0.343596,"KYD":0.945796,"KZT":495.650407,"LAK":12974.102129,"LBP":1713.900466,"LKR":229.832413,"LRD":174.541141,"LSL":17.166151,"LYD":5.194396,"MAD":10.684203,"MDL":20.297292,"MGA":4504.732245,"MKD":61.557978,"MMK":2015.369938,"MNT":3245.187477,"MOP":9.109927,"MRU":40.809138,"MUR":49.715484,"MVR":17.509086,"MWK":911.036297,"MXN":22.969746,"MYR":4.740518,"MZN":72.306226,"NAD":17.044195,"NGN":471.557382,"NIO":40.190981,"NOK":10.067969,"NPR":135.570149,"NZD":1.675591,"OMR":0.436896,"PAB":1.133546,"PEN":4.264472,"PGK":4.015496,"PHP":58.053866,"PKR":200.104138,"PLN":4.547706,"PYG":7884.80552,"QAR":4.127617,"RON":4.946168,"RSD":117.538769,"RUB":89.060844,"RWF":1178.741368,"SAR":4.250004,"SBD":9.151899,"SCR":14.889666,"SDG":503.380194,"SEK":10.568228,"SGD":1.524315,"SHP":0.833398,"SLL":12977.998391,"SOS":655.712405,"SRD":23.407556,"SSP":147.51468,"STD":23947.235851,"STN":24.631619,"SVC":9.919729,"SYP":2844.742829,"SZL":17.163608,"THB":36.771804,"TJS":12.792685,"TMT":3.963286,"TND":3.269479,"TOP":2.588397,"TRY":15.538025,"TTD":7.697347,"TWD":31.570927,"TZS":2622.721984,"UAH":32.798238,"UGX":3986.851595,"USD":1.133555,"UYU":48.559337,"UZS":12289.270948,"VES":4.91285,"VND":25855.275106,"VUV":129.011239,"WST":2.971277,"XAF":655.817552,"XAG":0.047618,"XAU":0.001324,"XCD":3.060478,"XDR":0.808507,"XOF":655.817139,"XPD":0.000474,"XPF":119.306812,"XPT":0.001164,"YER":283.455838,"ZAR":17.032593,"ZMW":19.988329,"ZWL":364.652419}}

y de ahí habría que sacar los cambios que nos interesen.

Una vez descargada la información, habrá que extraer el cambio y actualizarlo en la lista.

Como vamos a necesitar saber el código de divisa, se recomienda añadir a las clases que ya tenemos los siguientes elementos:

- Clase Moneda
  - Añadir un atributo **codigo** en cada moneda con el código de la moneda (ej.: euro = EUR, libra = GBP, dólar = USD, etc.).
  - Modificar el constructor para que haya que pasarle el código de la moneda y lo guarde en el atributo.
  - Añadir la propiedad de sólo lectura correspondiente.
- Clase Dinero
  - En el constructor estático, añadir los códigos de todas las monedas a la lista.
  - Añadir un método **ActualizaListaInternet** que automáticamente vaya actualizando todas las monedas que tenemos en la lista bajándose los cambios de divisa actualizados de Internet.