Índice

Ejercicios Unidad 20 - Programación Funcional - Consultas

Descargar estos ejercicios

> **Antes de empezar**
>
> Para realizar estos ejercicios, deberás descargar los recursos del enlace de proyecto_poo. Como puedes ver, la solución está compuesta de varios proyectos. Cada uno de ellos corresponde con un ejercicio, deberás implementar todo el código, tanto de la Main como de los métodos que se piden en cada ejercicio. Cada proyecto contiene el test correspondiente, que deberás pasar para comprobar que has hecho el ejercicio correctamente.

Ejercicio 1. Extension Biblioteca

A partir del ejercicio 1 de la entrega anterior se va a crea un método extensor de la clase `Biblioteca`

Requisitos

- Deberás añadir un nuev fichero BibliotecaExtensiones.cs en el que crearás un namespace llamado `BibliotecaExtensions`.
- Dentro de ese namespace crearás una clase BibliotecaExtensiones de la forma que se explicó en temas anteriores, para este tipo de clases.
- Esta clase incluirá el método de extensión `ISBNS` que devolverá un array con los ISBN de los libros en nuestra biblioteca ordenados.
- Se deberá usar métodos de System.Linq para resolver la funcionalidad.

```
Ejercicio 1. Extensión Biblioteca

Prestando ... { DNI = 22111333, Titulo = Cien años de soledad, ISBN = 9788420471839 }
Prestando ... { DNI = 22111333, Titulo = Los mejores cuentos de Clarín, ISBN = 97884315
True
False
Título: Los mejores cuentos de Clarín, Autor: Leopoldo Alas Clarín, Editorial: De Vecch

3
Título: El camino, Autor: Miguel Delibes, Editorial: Espasa, ISBN: 9788467023664, Nº Pá

TituloAutor { Titulo = Los mejores cuentos de Clarín, Autor = Leopoldo Alas Clarín }

Listado de ISBNs ordenados:
9788420471839
9788431533441
9788441405298
9788484326977

Pulsa una tecla para finalizar...
```

Ejercicio 2. Operaciones Linq con lista de números

Vamos ha realizar una serie de operaciones funcionales usando funciones-λ con el patrón **Map – Filter – Fold** (Select – Where – Aggregate en C#)

```
Ejercicio 2.  Coincidencia en lista de cadenas

Elementos:  0,5 1,6 2,8 3,9 4,1 5,2 6,3 7,4 8,1 9,2

Número elementos con parte decimal < 0,5 = 6

Suma elementos con parte entera múltiplo de 3 = 19,4

Máximo cuya parte decimal > 0,5 = 3,9

Elementos parte entera es primo:  2,8 3,9 5,2 7,4

Pulsa una tecla para finalizar...
```

Requisitos:

- Partiremos de la siguiente lista de números reales:

```
List<double> reales = new List<double>
{
        0.5, 1.6, 2.8, 3.9, 4.1, 5.2, 6.3, 7.4, 8.1, 9.2
};
```

- Vamos a realizar las siguientes operaciones, cada una en el método que se proporciona para ello:
  i. Mostrar la lista usando el método `ForEach(Action<T> action)` de lista. Pasando a la función-λ action, una **clausura** de la variable string texto, en la que iremos componiendo su contenido separado por un espacio en blanco.
  ii. Cuenta aquellos elementos cuya **parte decimal es menor que 0.5**
    - **Map:** Paso del valor real a su parte decimal. **Ej: 2.8 → 0.8**
    - **Filter:** Filtro aquellas partes decimales que cumplen el predicado: **d < 0.5**
    - **Fold:** Contar los elementos en la secuencia resultante.
  iii. Calcular la suma de todos los valores de la secuencia cuya **parte entera sea múltiplo de 3**.
    - **Map:** Mapea el valor real a un objeto anónimo con la parte entera y el propio valor real de la secuencia. **Ej:** `2.8 → new { e = 2, r = 2.8 }`

- **Filter:** Filtro aquellas partes enteras que cumplen el predicado: `o.e % 3 == 0`
- **Fold:** Suma todos los `o.r` de la secuencia resultante.

iv. Calcular el máximo valor de la secuencia cuya parte decimal es mayor que **0.5**

- **Map:** Mapea el valor real a un objeto anónimo con la parte decimal y el propio valor real de la secuencia. **Ej:** `2.8 → new { d = 0.8, r = 2.8 }`
- **Filter:** Filtro aquellas partes decimales que cumplen el predicado: `o.d > 0.5`
- **Fold:** Me quedo con el máximo de todos los `o.r` de la secuencia resultante.

v. **Ampliación**: Muestra aquellos elementos de la secuencia cuya **parte entera es un valor primo**.

> 💡 **Idea:** Seguramente has de hacer más de un `Filter` e incluso otro `Filter` dentro de uno de ellos. Además, para crear un predicado que me diga si un número primo de forma funcional (pero poco eficiente) puedes hacer lo siguiente ...

a. Generar una **secuencia entre 2 y la parte entera menos uno** con `Enumerable.Range()`

b. Filtrar aquellos valores divisibles por la parte entera.

c. Preguntar si la secuencia resultante tiene 0 elementos.

Una vez completadas todas las operaciones. Al ejecutar el programa la salida por pantalla del programa deberá ser ...

Ejercicio 3. Auditoría de Pedidos (FlatMap y Zip)

En este ejercicio vamos a simular un proceso de auditoría sobre una lista de pedidos. Usaremos `SelectMany` (FlatMap) para aplanar la estructura de pedidos y `Zip` para asignar códigos de auditoría únicos.

```
Ejercicio 3. Auditoría de Pedidos

AUD-001: Pedido 1 - Monitor (2 uds)
AUD-002: Pedido 1 - Ratón (1 uds)
AUD-003: Pedido 2 - Teclado (1 uds)
AUD-004: Pedido 3 - Impresora (1 uds)
AUD-005: Pedido 3 - Toner (4 uds)

Pulsa una tecla para finalizar...
```

Requisitos:

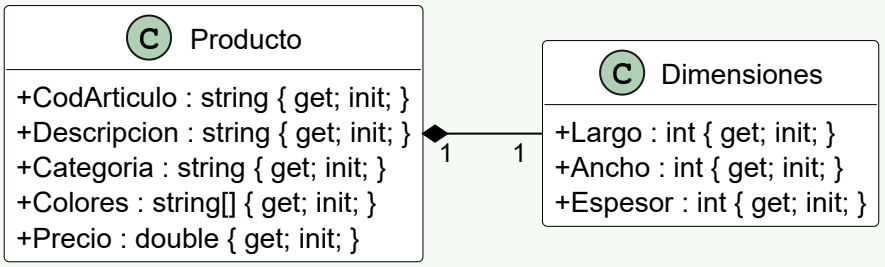
- Define las siguientes clases (o records):

```
public record LineaPedido(string Producto, int Cantidad);
public record Pedido(int Id, string Cliente, List<LineaPedido> Lineas);
```

- En el programa principal, crea una lista de pedidos con datos de prueba (al menos 3 pedidos con varias líneas cada uno).
- **Paso 1: Aplanado (FlatMap/SelectMany)**
  Obtén una secuencia plana de todas las líneas de pedido vendidas.
  Usa `SelectMany` para proyectar cada pedido en sus líneas.
  Ampliación: Al proyectar, incluye el Id del pedido en un objeto anónimo: `{ IdPedido, Producto, Cantidad }`.
- **Paso 2: Generación de Códigos (Zip)**
  Queremos asignar un código de auditoría secuencial a cada línea procesada (ej: "AUD-001", "AUD-002"...).
  i. Genera una secuencia de identificadores usando `Enumerable.Range`.
  ii. Usa la función `Zip` para combinar la secuencia plana del paso 1 con la secuencia de identificadores.
- **Paso 3: Informe**
  Muestra por consola el resultado final combinando los datos.

Ejercicio 4. Consultas sobre Productos

A partir del código que se da en este ejercicio, donde verás que hay definidas las siguientes clases ...



```
Ejercicio 5. Consultas sobre lista de productos


------------------------------------------------------------

Consulta 1: Usando las funciones Where y Select.
Mostrar CodArticulo, Descripcion y Precio .
de productos con Precio entre 10 y 30 euros

{ CodArticulo = A01, Descripcion = Uno, Precio = 15,05 }
{ CodArticulo = A02, Descripcion = Dos, Precio = 25,95 }
{ CodArticulo = A04, Descripcion = Cuatro, Precio = 18,45 }

------------------------------------------------------------

Consulta 2: Usando las funciones Select, OrderByDescending y Take.
Muestra CodArticulo, Descripcion y Precio de los 3 productos.
más caros (ordenando por Precio descendente)

{ CodArticulo = A03, Descripcion = Tres, Precio = 30,25 }
{ CodArticulo = A02, Descripcion = Dos, Precio = 25,95 }
{ CodArticulo = A04, Descripcion = Cuatro, Precio = 18,45 }

------------------------------------------------------------

Consulta 3: Usando las funciones GroupBy, OrderBy y First.
Muestra el precio más barato por categoría

{ Categoria = C1, PrecioMasBarato = 15,05 }
{ Categoria = C2, PrecioMasBarato = 18,45 }

------------------------------------------------------------

Consulta 4: Usando las funciones GroupBy, Count.
¿Cuántos productos hay de cada categoría?

{ Categoria = C1, NumeroProductos = 3 }
{ Categoria = C2, NumeroProductos = 1 }

------------------------------------------------------------

Consulta 5: Usando las funciones GroupBy, Count, Where y Select
Mostrar las categorías que tengan más de 2 productos

C1

------------------------------------------------------------

Consulta 6: Usando la función Select
Mostrar CodArticulo, Descripcion, Precio y Descuento redondeado a 2 decimales,
siendo Descuento el 10% del Precio
```

```
{ CodArticulo = A01, Descripcion = Uno, Precio = 15,05, Descuento = 1,5 }
{ CodArticulo = A02, Descripcion = Dos, Precio = 25,95, Descuento = 2,6 }
{ CodArticulo = A03, Descripcion = Tres, Precio = 30,25, Descuento = 3,03 }
{ CodArticulo = A04, Descripcion = Cuatro, Precio = 18,45, Descuento = 1,84 }

------------------------------------------------------------

Consulta 7: Usando las funciones Where, Contains y Select.
Mostrar CodArticulo, Descripcion y Colores
de los productos de color verde o rojo
(es decir, que contengan alguno de los dos)

{ CodArticulo = A02, Descripcion = Dos, Colores = [blanco, gris, rojo] }
{ CodArticulo = A03, Descripcion = Tres, Colores = [rojo, gris, verde] }
{ CodArticulo = A04, Descripcion = Cuatro, Colores = [verde, rojo] }

------------------------------------------------------------

Consulta 8: Usando las funciones Where, Count y Select.
Mostrar CodArticulo, Descripcion y Colores.
de los productos que se fabrican en tres Colores

{ CodArticulo = A01, Descripcion = Uno, Colores = [blanco, negro, gris] }
{ CodArticulo = A02, Descripcion = Dos, Colores = [blanco, gris, rojo] }
{ CodArticulo = A03, Descripcion = Tres, Colores = [rojo, gris, verde] }

------------------------------------------------------------

Consulta 9: Usando las funciones Where, Select.
Mostrar CodArticulo, Descripcion y Dimensiones
de los productos con espesor de 3 cm

{ CodArticulo = A01, Descripcion = Uno, Dimensiones = L:4 x A:4 x E:3 }
{ CodArticulo = A03, Descripcion = Tres, Dimensiones = L:5 x A:5 x E:3 }

------------------------------------------------------------

Consulta 10: Usando las funciones SelectMany, Distinct y OrderBy.
Mostrar los colores de productos ordenados y sin repeticiones

blanco
gris
negro
rojo
verde

------------------------------------------------------------

Consulta 11: Usando las funciones SelectMany, GroupBy, OrderByDescending.
Mostrar TotalProductos que hay de cada Color ordenando de mayor a menor cantidad
```

```
{ Color = gris, TotalProductos = 3 }
{ Color = rojo, TotalProductos = 3 }
{ Color = blanco, TotalProductos = 2 }
{ Color = verde, TotalProductos = 2 }
{ Color = negro, TotalProductos = 1 }

Pulsa una tecla para finalizar...
```

Requisitos:

- En la propiedad estática `Productos` de la clase `Datos` te devolverá una secuencia de productos (`IEnumerable<Producto>`) sobre la que realizar las consultas.
- Además, en el programa principal tienes un '*esqueleto*' a completar con descripción de cada consulta. Por ejemplo, para la primera consulta tendríamos ...

```
Console.WriteLine(SeparadorConsulta);
Console.WriteLine(
    "Consulta 1: Usando las funciones Where y Select.\n" +
    "Mostrar CodArticulo, Descripcion y Precio .\n" +
    "de productos con Precio entre 10 y 30 euros\n");
var consulta1 = "";
Console.WriteLine(string.Join("\n", consulta1));
```

- Nosotros deberemos rellenar la consulta de acuerdo a las especificaciones de la descripción, cuidando la presentación y sangría para que sean legibles. Por ejemplo ...

```
...
var consulta1 = Productos.Where(p => p.Precio >= 10 && p.Precio <= 30)
                         .Select(p => new
                         {
                             CodArticulo = p.CodArticulo,
                             Descripcion = p.Descripcion,
                             Precio = p.Precio
                         });
...
```

Ejercicio 5. Recursividad con Lambdas (Anexo I - No evaluable)

Programa que nos permite iniciarnos con la recursividad con aplicación directa de funciones lambda.

```
Ejercicio 5. Recursividad con Lambdas
Ejercicio 5. Recursividad con Lambdas

Sumatorio de 1 a "5": 15
Suma de dígitos del número "543": 12
Cuenta de vocales en la cadena "Hola MUNDO": 4
```

Requisitos

Implementa **tres** funciones recursivas sencillas utilizando expresiones lambda (`Func<...>`) para resolver los siguientes problemas. Recuerda que para que una lambda pueda llamarse a sí misma, primero debes declarar la variable delegada (asignándole `default` o `null`) y luego asignarle la expresión lambda.

1. **Sumatorio de un número**:
   Crea una función lambda `sumatorio` que reciba un entero `n` y devuelva la suma de todos los enteros desde 1 hasta `n` .
   - Caso base: si `n` es 0, devuelve 0.
   - Caso recursivo: devuelve `n + sumatorio(n-1)` .
2. **Suma de sus dígitos**:
   Crea una función lambda `sumaDigitos` que reciba un entero positivo y devuelva la suma de sus dígitos individualmente.
   - Caso base: si `n` es menor que 10, devuelve `n` .
   - Caso recursivo: devuelve el último dígito (`n % 10`) más la llamada recursiva con el número sin ese dígito (`n / 10`).
3. **Contar vocales**:
   Crea una función lambda `contarVocales` que reciba una cadena (string) y devuelva el número de vocales que contiene (mayúsculas o minúsculas).
   - Caso base: si la cadena está vacía (o su longitud es 0), devuelve 0.
   - Caso recursivo: comprueba si el primer carácter (`str[0]`) es una vocal. Si lo es, suma 1; si no, suma 0. Luego añade el resultado de la llamada recursiva con el resto de la cadena (puedes usar el operador de rango `str[1..]`).

Nota: Intenta usar el operador ternario `condición ? valor_true : valor_false` para que la expresión lambda quede concisa.