Práctica 5 - Programación Funcional

Pruebas unitarias para los métodos de orden superior

Dados los métodos de orden superior vistos en clase (`Map`, `Filter`, `Reduce` y `Zip`), implementa pruebas unitarias para probar su funcionamiento en los siguientes casos:

- Aplica `Map` utilizando una función con el mismo tipo de origen y destino: Ej. `x => x*x` (cuadrado de un número).
- Aplica `Map` utilizando una función con un tipo destino diferente al de la colección original: Ej. `"hello" => 2` (contar vocales).
- Filtra con una condición que cumplan **todos los elementos** de la colección de entrada.
- Filtra con una condición que cumplan **algunos elementos**.
- Filtra con una condición que no cumpla **ningún elemento**.
- Aplica `Reduce` generando el mismo tipo destino que la colección: Ej. `x => sum(x)` (sumatorio de la colección).
- Aplica `Reduce` generando el mismo tipo destino **con semilla**: Ej. `x => min(x)` (mínimo de la colección).
- Aplica `Reduce` generando un tipo de destino diferente: Ej. `"hello" => sum(length(s))` (sumatorio de las longitudes de las cadenas de texto).
- Aplica `Reduce` generando un tipo de destino diferente **con semilla**: Ej. `"hello" => d['e'] += 1; d['o'] += 1` (diccionario con el número de ocurrencias de las vocales).
- Aplica un `Zip` a dos colecciones del mismo tipo y misma longitud.
- Aplica un `Zip` a dos colecciones con diferentes tipos y longitudes.
- Operaciones donde se combinen varios métodos: Ej. `Zip` + `Map`, `Map` + `Filter` + `Reduce`, ...

Resolución de problemas combinando `Map`, `Reduce`, `Filter` y `Zip`

Una vez hayas completado las pruebas unitarias y comprobado que funcionan correctamente, implementa los siguientes ejercicios:

- Calcula el número de ventas no confirmadas en Norteamérica.
- Calcula el importe total de las ventas confirmadas realizadas en Europa.
- Obtén la región con mayor facturación neta. Debes devolver el nombre de la región y su importe de facturación neta.
- Dado un conjunto de rangos de importe (tupla `min`, `max`), utiliza `Reduce` para agrupar las ventas en un `Dictionary<string, List<Sale>>` donde cada clave represente

un rango (`"min-max"`) y su valor sea la lista de ventas cuyo Amount esté dentro de ese intervalo `[min, max)`.

```
var ranges = new[]{(0m, 100m),(0m, 500m),(500m, 2000m)};
var result = Reduce(ranges, ...);
foreach (var item in result)
  Console.WriteLine($"Range: {item.Key}, Count: {item.Value.Count}");
// Range: 0-100, Count: 1
// Range: 0-500, Count: 6
// Range: 500-2000, Count: 2
```