Appello TAP del 5/6/2023

Scrivere nome, cognome e matricola sul foglio protocollo. Avete a disposizione due ore e mezza.

Esercizio 1 (9 punti)

Scrivere l'extension-method `MultipleApply` che, data una sequenza di funzioni unarie sul tipo generico `T` a valori interi, `s`, un valore di tipo `T`, `v`, e un parametro intero `n`, produce una sequenza di array di interi, tutti di lunghezza `n`, contenenti l'applicazione delle funzioni in `s` al valore `v`.

Il primo array conterrà l'applicazione a `v` delle prime `n` funzioni in `s`, il secondo delle successive `v` e così via. In generale, iniziando a contare gli elementi di `s` da 0, l'array i-esimo conterrà l'applicazione a `v` delle funzioni in posizione da i`*n` a $(i+1)$`*n` escluso.

Ad esempio, sulla sequenza di 6 funzioni da interi a interi:

f(x)=2*x | f(x)=3*x | f(x)=4*x | f(x)=5*x | f(x)=6*x | f(x)=7*x

e i parametri `v` = 2 e `n` =3 produce la sequenza di due array di tre interi | 4, 6, 8 | | 10, 12, 14 |

Il metodo dovrà sollevare:

- `ArgumentNullException` se `s` è null
- `ArgumentOutOfRangeException` se `n` non è strettamente positivo
- `InconsistentSourceException` se la sorgente è finita e la sua dimensione non è un multiplo di `n`, dove `InconsistentSourceException` è definita da

```
public class InconsistentSourceException : Exception {
    public InconsistentSourceException() { }
    public InconsistentSourceException(string message) : base(message) {}
    public InconsistentSourceException(string message, Exception inner) :
        base(message, inner) { }
}
```

Esercizio 2 (8 punti)

Implementare, usando NUnit, i seguenti test relativi a `MultipleApply`, dell'esercizio 1.

1. Implementare l'esempio dato nell'esercizio 1.
2. Input della chiamata sotto test: `s` è una sequenza di nove funzione da stringhe a interi a vostra scelta, `v` vale "boom", `n` vale 2

   Output atteso: una `InconsistentSourceException`.
3. Input della chiamata sotto test: `s` è una qualsiasi sequenza **infinita**, `n` vale 0

   Output atteso: una `ArgumentOutOfRangeException` sollevata **senza enumerare la sorgente** neppure parzialmente

Esercizio 3 (9 punti)

1. Dato il seguente frammento di codice, indicare quali test avranno successo

```
public static IEnumerable<bool> M(this IEnumerable<bool>? s) {
    if (null == s) throw new ArgumentNullException();
    return M1();

    IEnumerable<bool>? M1() {
        foreach (var n in s) {
            if (n) throw new ArgumentException();
            yield return !n;
        }
    }
}
[TestFixture]
public class Test {
    static IEnumerable<bool>? S() {
       for (int i = 0; i < 10; i++) yield return false;
       while (true) yield return true;
    }
    static IEnumerable<bool>? Null() { return null; }
    [Test]
    public void TestA() { Assert.That(() => S().M().Take(15),
        Throws.TypeOf<ArgumentException>()); }
    [Test]
    public void TestB() { Assert.That(() => Null().M().Take(15),
        Throws.TypeOf<ArgumentNullException>()); }
    [Test]
    public void TestC() { Assert.That(() => S().M(),
        Throws.TypeOf<ArgumentException>()); }
    [Test]
    public void TestD() { Assert.That(() => Null().M(),
        Throws.TypeOf<ArgumentNullException>()); }
    [Test]
    public void TestE() { Assert.That(() => S().M().ToArray(),
        Throws.TypeOf<ArgumentException>()); }
    [Test]
    public void TestF() { Assert.That(() => Null().M().ToArray(),
        Throws.TypeOf<ArgumentNullException>()); }
    [Test]
    public void TestG() { Assert.That(S().M(),
        Throws.TypeOf<ArgumentException>()); }
    [Test]
    public void TestH() { Assert.That(Null().M(),
        Throws.TypeOf<ArgumentNullException>()); }
    [Test]
    public void TestI(){ Assert.That(S().M().First(), Is.True); }
}
```

Vero	Falso	
☐	☐	TestA
☐	☐	TestB
☐	☐	TestC
☐	☐	TestD
☐	☐	TestE
☐	☐	TestF
☐	☐	TestG
☐	☐	TestH
☐	☐	TestI

2. Supponete di rappresentare in memoria il risultato di una query effettuata usando l'EF mediante una variabile `q` di tipo `IQueryable<E>` per una qualche entità `E`.

Vero	Falso	
☐	☐	Gli elementi di `q` si possono modificare e questo si riflette *immediamente* sui corrispondenti elementi nella base di dati
☐	☐	Accedendo due volte all'i-esimo elemento di `q` si ottiene sempre lo stesso valore
☐	☐	Il codice `q.ElementAt(1); q.ElementAt(7);` non causa multiple enumerazioni della query
☐	☐	`var x = q.ToArray(); var y = q.ToArray();` genera sempre errore statico
☐	☐	`var x = q.ToArray(); var y = q.ToArray();` può essere corretta dinamicamente anche se `q` contiene elementi
☐	☐	Valutare `q.ToArray();` può richiedere più RAM di quella disponibile
☐	☐	Siccome il tipo di `q` non è una specializzazione di `IDbSet`, la sua dichiarazione causa un errore statico
☐	☐	È sempre indifferente usare `IQueryable` o `IEnumerable` come tipo di `q`
☐	☐	Enumerare `q` dopo che il contesto usato per popolarla è stato *disposed* causa sempre un errore dinamico, anche se staticamente il codice è corretto