Appello TAP del 19/02/2014

Scrivere nome, cognome e matricola sul foglio protocollo, indicando anche se avete nel piano di studi TAP da 8 CFU (quello attuale) o da 6 CFU (quello "vecchio").

Chi deve sostenere TAP da 6 CFU dovrà svolgere solo i primi tre esercizi; per loro il punteggio indicato nel testo sarà scalato, di conseguenza, di $\frac{\sum_{i=1}^{4} PuntiEs_i}{\sum_{i=1}^{3} PuntiEs_i}$

Avete a disposizione mezzora per esercizio (quindi, un'ora e mezza per chi deve sostenere TAP da 6 CFU e due ore per TAP da 8 CFU).

Esercizio 1 (10 punti)

Scrivere l'extension-method generico `Jumping<T>` che, presa una sequenza `s` di elementi di tipo `T`, restituisce una sequenza infinita i cui elementi sono ottenuti visitando in modo circolare `s` e saltando a blocchi di dimensione prefissata `step`. Ad esempio, se `step` == 0, restituisce tutti i valori di `s` e poi ricomincia da capo infinite volte. Se `step` == 1 restituisce tutti i valori di `s` in posizione pari e poi ricomincia da capo infinite volte. Il metodo dovrà prendere come parametri:

1. (come parametro "this") `sequence`, la sequenza sorgente. Nota: questa sequenza può anche essere infinita;
2. `step`,la dimensione dei salti da effettuare, cioè dei gap da lasciare fra un valore scelto ed il successivo.

Per esempio, il seguente frammento di codice scrive sullo standard output i numeri: 1, 5, 9, 2, 6, 10, 3, 7, 11, 4, 8, 1, 5, 9, 2, 6, 10, 3, 7, 11, 4, 8, 1, 5, 9, 2, 6, 10, 3, 7, 11, 4, 8,...

```
new [] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
    .Jumping (3)
    .ToList()
    .ForEach(Console.WriteLine);
var list = new List<int>(){1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
            foreach (var i in list.Jumping(3))
                Console.WriteLine(i);
```

Il metodo deve sollevare l'eccezione...

- `ArgumentNullException` se `sequence` è null;
- `ArgumentOutOfRangeException` se `step` è strettamente negativo.

Esercizio 2 (3+3+2+1 = 9 punti)

- Elencare, descrivendoli a parole, una lista di test significativi per il metodo `Jump<T>`, dell'esercizio precedente.
- Implementare, usando NUnit, due test della lista precedente; uno che vada a testare un caso "buono" (ovvero, dove ci si aspetta che l'invocazione di `Jump` vada a buon fine) e uno che vada a testare un caso "cattivo" (ovvero, dove ci si aspetta che l'invocazione di `Jump` sollevi un'eccezione).
- Implementare, usando NUnit, un test significativo in cui `sequence` è instanziato su una lista infinita.
- Entro quali limiti si riesce a testare il comportamento della vostra implementazione dell'Esercizio 1 su uno specifico input corretto?

Esercizio 3 (3+3=6 punti)

- Definire un custom-attribute `EffortAttribute` che permetta di specificare, solo su metodi, qual è stato lo sforzo necessario per implementarlo, espresso mediante un intero, in un'opportuna scala (unità di tempo/uomo). Esempio d'uso:

```
[Effort(27)]
public void VeryDifficultMethod() { /* ... */ }
```

- Utilizzare il custom-attribute definito al punto precedente per calcolare l'effort complessivo di sviluppo di una classe/interfaccia (si ignorino i metodi non pubblici).

Esercizio 4 (5 punti)

- Dare l'implementazione del metodo asincrono `InitialsAsync` che, preso in input un array di `Task<string>`, restuisca un `Task<string>` che corrisponde a concatenare le iniziali di tutte le stringhe non vuote e non nulle prodotte dai task dell'array in un ordine qualsiasi.
- Dare un esempio di invocazione del metodo implementato nel punto precedente.