Appello TAP del 13/07/2015

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") e se avete consegnato i test durante l'anno e intendete usufruire del bonus così conseguito.

Chi deve sostenere TAP da 6 CFU non deve svolgere l'esercizio 1 e chi ha consegnato i test durante l'anno (e intende usufruire del bonus conseguito) non deve svolgere l'esercizio 4; per loro il punteggio indicato nel testo sarà scalato, di conseguenza, in modo che il massimo conseguibile sia sempre 30. Avete a disposizione mezzora per esercizio. In sintesi:

Tipo TAP	Bonus Test	Esercizi da svolgere	Tempo a disposizione	Fattore di normalizzazione
6CFU	sì	2 e 3	1h	$\frac{30}{13+6}$
6CFU	no	2, 3 e 4	1h 30'	$\frac{30}{13+6+6}$
8CFU	sì	1, 2 e 3	1h 30'	$\frac{30}{5+13+6}$
8CFU	no	tutti (1, 2, 3 e 4)	2h	1

Esercizio 1 (punti 5)

- Dare l'implementazione del metodo asincrono `ScalarProductAsync` che, presi in input due array di `Task<double>` di pari lunghezza, restuisca un `Task<double>` che corrisponde a calcolare il prodotto scalare degli array contenenti i risultati dei task.

  Ad esempio, sugli array `[t0,t1,t2]` e `[tt0,tt1,tt2]` il task risultato di `ScalarProductAsync` calcolerà `t0.Result*tt0.Result + t1.Result*tt1.Result + t2.Result*tt2.Result`

  Si sollevino opportune eccezioni per segnalare i casi di errore.

- Dare un esempio di invocazione del metodo implementato nel punto precedente.

Esercizio 2 (punti 3+10 = 13)

- Definire un metodo statico e generico che presi come argomenti due predicati `P1` e `P2` di tipo `Func<T, bool>` restituisca il predicato che ne calcola l'and, ovvero un valore di tipo `Func<T, bool>` che su un elemento di tipo `T` vale `true` se e solo se entrambi `P1` e `P2` valgono `true` su di esso.

- Scrivere l'extension-method `IncrementallyFilter` generico sul tipo `T` che, presa una sequenza di `T` ed una sequenza di predicati su `T`, restituisce la sequenza di sequenze di `T` il cui elemento i-esimo contiene tutti gli elementi della sequenza iniziale su cui sono veri tutti i primi $i+1$ predicati.

  Ad esempio,

  - sulla sequenza di stringhe: "paperino",null,"pluto", "topolino", "minnie"
  - e sulla sequenza dei predicati: (`s =>s!=null`), (`s=>s.First()=='p'`), (`s=>s.Length<4`),

  il metodo dovrà restituire la sequenza:

  - primo elemento: "paperino","pluto", "topolino", "minnie".
  - secondo elemento: "paperino","pluto".
  - terzo elemento: sequenza vuota.

  Il metodo dovrà prendere come parametri

  - ("this") `source`, la sequenza da filtrare, di tipo `IEnumerable<T>`
  - `filters`, la sequenza dei predicati da usare, di tipo `IEnumerable<Func<T, bool>>`

  solleverà l'eccezione `ArgumentNullException` se `source`, `filters` o uno dei suoi elementi sono `null` e avrà come tipo di ritorno `IEnumerable<IEnumerable<T>>`.

  Si noti che tutte le sequenze citate, tanto come argomenti quanto come risultato (e suoi elementi), sono potenzialmente infinite.

  Ad esempio, se

– `source` corrisponde ai numeri naturali e
– filters contiene la sequenza dei predicati $\texttt{P}_i$ per $i \in \{k | k \in \mathbb{N} \wedge k \geq 2\}$, dove $\texttt{P}_i$ vale true su n se e solo se n **non** è multiplo di i,

allora il risultato è la sequenza infinita `seq` di sequenze infinite di cui l'i-esima contiene solo i numeri che sono co-primi con tutti i primi $i+2$ numeri (come nel crivello di Eratostene), cioè `seq`[0] contiene i numeri dispari, `seq`[1] contiene i numeri dispari che non sono multipli di 3 e così via.

Esercizio 3 (punti 6) Applicare i principi della dependency injection per fare refactoring della seguente classe `C` eliminando le dipendenze indesiderate. Introdurre i tipi necessari e modificare `C` di conseguenza.

Dire se è necessario modificare anche altre classi e, in caso positivo, descrivere le modifiche necessarie.

```
public class C
    {
        public D MyD { get; private set; }
        public E MyE { get; private set; }
        public int[] MyArray { get; private set; }

        public C()
        {
            this.MyArray = new int[42];
            this.MyD = new D();
            this.MyE = new E();
        }

        public string M(bool x, int y)
        {
            return this.MyE.H(this.MyArray, this.MyD.F(x), this.MyD.G(y));
        }
    }
```

Esercizio 4 (punti 2+2+2 = 6 punti)

- Elencare, descrivendoli a parole, una lista di test significativi per il metodo `IncrementallyFilter`, dell'esercizio 2.
- 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 `IncrementallyFilter` vada a buon fine) e uno che vada a testare un caso "cattivo" (ovvero, dove ci si aspetta che l'invocazione di `IncrementallyFilter` sollevi un'eccezione).
- Implementare, usando NUnit, un test in cui `IncrementallyFilter` venga invocato su un argomento "infinito".