Appello TAP del 23/01/2015

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	$\frac{30}{6+9}$
6CFU	no	2, 3 e 4	1h 30'	$\frac{30}{6+9+9}$
8CFU	sì	1, 2 e 3	1h 30'	$\frac{30}{5+6+9}$
8CFU	no	tutti (1, 2, 3 e 4)	2h	1

Esercizio 1 (punti 5)

- Dare l'implementazione del metodo asincrono `TaskMaxAsync` che, preso in input un array di `Task<int>`, restuisca un `Task<int>` che corrisponde a trovare il massimo fra gli interi dispari prodotti dai task dell'array (sollevando `InvalidOperationException` se non ce ne sono e `ArgumentNullException` se l'array è null).
- Dare un esempio di invocazione di `TaskMaxAsync` che non sollevi eccezioni.

Esercizio 2 (punti 6)

Scrivere l'extension-method generico `IncrementallyCompose<T>` che, presa una sequenza `funcs` (potenzialmente infinita) di elementi di tipo `Func<T, T>` ed un elemento `seed` di tipo `T`, restituisce una sequenza (potenzialmente infinita) i cui elementi sono ottenuti applicando man mano le funzioni in `funcs`. Ad esempio, se `funcs` == `f1`, `f2`, `f3`, `f4`,..., restituisce `f1(seed)`, `f2(f1(seed))`, `f3(f2(f1(seed)))`, `f4(f3(f2(f1(seed))))`,... Il metodo dovrà prendere come parametri:

1. (come parametro "this") `funcs`, la sequenza sorgente. Nota: questa sequenza può anche essere infinita;
2. `seed`, l'elemento di partenza da cui cominciare ad applicare le funzioni in `funcs`.

Per esempio, il seguente frammento di codice scrive sullo standard output i numeri: 1, 5, 9.

```
Func<int, int> func1 = (s => s+1);
Func<int, int> func2 = (s => s*5);
Func<int, int> func3 = (s => s+(s-1));
new Func<int, int>[] { func1, func2, func3 }
    .IncrementallyCompose(seed).ToList().ForEach(Console.WriteLine);
```

Mentre , il seguente frammento di codice scrive sullo standard output le potenze di 2: 2, 4, 8, 16, 32, 64,...

```
private IEnumerable<Func<T, T>> InfiniteCopies<T>(Func<T, T> f)
{
    while (true)
       yield return f;
}
//.....
foreach (var i in InfiniteCopies<int>(x => x*2).IncrementallyCompose(1))
    Console.WriteLine(i);
```

Il metodo deve sollevare l'eccezione...

- `ArgumentNullException` se `funcs` è null;
- `InvalidOperationException` se `funcs` è vuota.

Eventuali eccezioni sollevate dall'esecuzione di un elemento in `funcs` non devono essere gestite.

Esercizio 3 (punti 9) Si consideri la seguente interfaccia per descrivere (in modo molto semplificato) prodotti da porre in vendita.

```
public interface IProduct
{
    string Name { get; }
    string Description { get; }
    double PriceInEuro { get; set; }
}
```

Definire una classe generica `Shop<T>` che, per i soli tipi `T` che realizzano l'interfaccia `IProduct`, definisca negozi monotematici, ovvero negozi in cui si vendono solo prodotti di tipo `T`.

La classe `Shop<T>` dovrà fornire metodi (pubblici) per

- aggiungere un certo numero di esemplari di un prodotto al magazzino;
- verificare quanti esemplari di un certo prodotto sono presenti in magazzino;
- vendere un certo numero di esemplari di un dato prodotto, limitatamente alle scorte presenti a magazzino (ed aggiornando lo stato del magazzino);
- stampare a video lo stato del magazzino.

Dovrà inoltre avere proprietà per alcuni dati, quali nome e partita IVA, nonché costruttori appropriati.

Documentate con brevi commenti come vengono segnalati all'utente gli errori e che cosa succede quando un'operazione non può andare a buon fine (ad esempio perché si cerca di vendere un numero di esemplari di un prodotto superiore a quanto presente in magazzino, o perché il numero non è positivo).

Esercizio 4 (punti 3+3+2+1 = 9 punti)

- Elencare, descrivendoli a parole, una lista di test significativi per il metodo `IncrementallyCompose<T>`, 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 `IncrementallyCompose<T>` vada a buon fine) e uno che vada a testare un caso "cattivo" (ovvero, dove ci si aspetta che l'invocazione di `IncrementallyCompose<T>` sollevi un'eccezione).
- Implementare, usando NUnit, un test significativo in cui `funcs` è instanziato su una lista infinita.
- Entro quali limiti si riesce a testare il comportamento della vostra implementazione dell'Esercizio 2 su uno specifico input corretto?