Appello TAP del 26/06/2014

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 gli ultimi tre esercizi; per loro il punteggio indicato nel testo sarà scalato, di conseguenza, di $\frac{\sum_{i=1}^{4} PuntiEs_i}{\sum_{i=2}^{4} PuntiEs_i}$ 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 (5 punti)

Si assuma data un'interfaccia `I`, con un metodo `M()` senza parametri che restituisce un intero.

Scrivere l'extension-method `IterateM` che, preso in input una sequenza di `I` ed un predicato p su `I`, produce la somma dell'esecuzione di `M()` su tutti gli elementi che soddisfano p, sfruttando per quanto possibile eventuale hardware multicore, assumendo che tutti i metodi e predicati descritti possano essere eseguiti in parallelo.

Si sollevino le eccezioni che si ritiene opportuno.

Esercizio 2 (9 punti)

Si assuma data la seguente interfaccia, che rappresenta i gruppi su un insieme di valori di tipo `T`, mediante le operazioni che caratterizzano il gruppo (attenzione: un oggetto di tipo `IGroup<T>` rappresenta l'intero gruppo, non un singolo elemento del gruppo)

```
public interface IGroup<T>
    {
        T Zero ();
        T Sum (T x, T y);
        T Inverse (T x);
    }
```

di cui un esempio di implementazione potrebbe essere il seguente

```
public class IntGroup : IGroup<int>
    {
        private readonly int _zero;
        private readonly Func<int, int> _inverse;
        private readonly Func<int, int, int> _sum;
        public IntGroup(int zero, Func<int, int> inverse, Func<int, int, int> sum)
        {
            _zero = zero;
            _inverse = inverse;
            _sum = sum;
        }
        public IntGroup() : this(0, x => -x, (x, y) => x + y) { }

        public int Zero() { return _zero; }
        public int Sum(int x, int y) { return _sum(x, y); }
        public int Inverse(int x) { return _inverse(x); }
    }
```

Scrivere l'extension-method `Fibonacci` che, preso un gruppo `group` su valori di tipo `T` ed un valore iniziale `init` di tipo `T`, restituisce la sequenza (infinita) dei valori della serie di Fibonacci originata da `init`, secondo la definizione classica: $a_0 = \texttt{Zero}()$, $a_1 = \texttt{init}$, $a_{n+2} = \texttt{Sum}(a_n, a_{n+1})$.

Il metodo dovrà prendere come parametro "this" `group`, il gruppo, e come parametro ulteriore `init` di tipo `T` e dovrà sollevare l'eccezione

- `ArgumentNullException` se `group` o `init` sono nulli;
- `ArgumentOutOfRangeException` se `init` è `Zero`.

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

- Elencare, descrivendoli a parole, i test significativi per il metodo `Fibonacci`, dell'esercizio precedente.
- Implementare, usando NUnit ed eventualmente Moq, due test della lista precedente; uno che vada a testare un caso "buono" (ovvero, dove ci si aspetta che l'invocazione di `Fibonacci` vada a buon fine) e uno che vada a testare un caso "cattivo" (ovvero, dove ci si aspetta che l'invocazione di `Fibonacci` sollevi un'eccezione).
- Implementare, usando NUnit ed eventualmente Moq, un test che verifica che le operazioni dell'interfaccia `IGroup<T>` siano invocate il numero atteso di volte (in un caso "buono").

Esercizio 4 (7 punti)

Utilizzando un meccanismo di comunicazione basato sugli eventi, implementare le parti essenziali di un sistema (futuribile) che emette automaticamente le multe per eccesso di velocità.

Si assumano date le seguenti classi, che possono essere modificate a piacere ("..." indica le parti non rilevanti ai fini dell'esercizio), in modo da ottenere che quando la velocità di una macchina cambia, se la nuova velocità è superiore al limite previsto in quel punto venga invocato il metodo `IssueSpeedTicket` con i corretti parametri.

```
public class Person {/*...*/}

public class GPSLocation
{
    /*...*/
    public int SpeedLimit { get; private set; }
}

public class Car
{
    /*...*/
    public Person Owner { get; set; }
    public GPSLocation Location { get; set; }
    public int Speed { get; set; }
}

public class Police
{
    /*...*/
    public void IssueSpeedTicket(Car culprit, DateTime when, int speed) { /*...*/}
    public void RegisterCar(Car car) { /*...*/}
}
```