Appello TAP del 14/06/2018

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

Esercizio 1 (10 punti)

Scrivere l'extension-method `FibonacciProvider` che data una sequenza di interi produce una sequenza di serie di Fibonacci di cui l'i-esima usa come valori di partenza gli elementi di posto i e $(i+1)$ della sequenza data.

Per chi non si ricorda, la sequenza di Fibonacci con valori di partenza x e y, $F_{x,y}$, è definita induttivamente da $F_{x,y}(0) = x$, $F_{x,y}(1) = y$, $F_{x,y}(n+2) = F_{x,y}(n) + F_{x,y}(n+1)$.

Ad esempio $F_{1,1} = 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765 \ldots$.

Il metodo dovrà prendere come unico parametro "this" `intSeq`, la sequenza sorgente (una sequenza di interi **potenzialmente infinita**) e produrre come output una sequenza di sequenze di Fibonacci. Ad esempio sulla sequenza $1, 4, -6, 0$ produrrà (una rappresentazione di) $F_{1,4}, F_{4,-6}, F_{-6,0}$.

Il metodo `FibonacciProvider` deve sollevare l'eccezione

- `ArgumentNullException` se `intSeq` è `null`.
- `ArgumentException` se `intSeq` ha meno di due elementi.

Si noti che l'argomento di `FibonacciProvider`, il suo risultato e ogni elemento del risultato possono essere infinite.

Esercizio 2 ([2+3+5] = 10 punti)

Implementare, usando NUnit e/o Moq, i seguenti test relativi al metodo `FibonacciProvider`, dell'esercizio 1.

1. Input della chiamata sotto test: `intSeq` deve essere la sequenza vuota.

   Output atteso: deve essere sollevata l'eccezione `ArgumentException`.

2. Input della chiamata sotto test: `intSeq` deve essere la sequenza $1, 1$.

   Output atteso: la sequenza che contiene un solo elemento, la rappresentazione di $F_{1,1}$ (verificare che effettivamente sia una rappresentazione corretta della sequenza di Fibonacci controllando solo i primi 10 elementi).

3. Test parametrico con parametri interi `approx` e `whichSeries`.

   Input della chiamata sotto test: `intSeq` deve essere una sequenza infinita a vostra scelta che generi come risultato di `FibonacciProvider` una sequenza di serie di Fibonacci tutte diverse fra loro.

   Il test deve verificare la correttezza dei primi `approx` elementi della serie di Fibonacci di posizione `whichSeries` nel risultato. Ad esempio se `intSeq` $=7, 23, 11, 2, 1, 1, 4$, `whichSeries` $= 0$ e `approx` $= 3$, il test dovrà verificare che il primo elemento del risultato di `FibonacciProvider` restituisca come primi tre elementi $7, 23, 30$, mentre se whichSeries$= 2$ e `approx` $= 4$, il test dovrà verificare che il terzo elemento del risultato di `FibonacciProvider` restituisca come primi quattro elementi $11, 2, 13, 15$.

   Ipotesi semplificativa: si assuma che `approx` non possa essere superiore a 20.

   Una proprietà delle serie di Fibonacci che potreste trovare utile è $F_{x,y}(n+2) = x \cdot F_{1,1}(n) + y \cdot F_{1,1}(n+1)$, che permette di calcolare i valori non banali delle serie generiche a partire da quelli della serie di Fibonacci classica (di cui avete i primi valori nel testo dell'esercizio precedente).

Esercizio 3 ([3+7] = 10 punti)

1. Definire un custom attribute che si possa applicare (al più una volta) solo a interfacce e permetta di associare ad un'interfaccia il nome (cioè una `String`) della sua implementazione di default, ed eventualmente ulteriori possibili implementazioni. In particolare, definire costruttore e meccanismi per memorizzare ed accedere ai nomi delle classi indicate come implementazione.

2. Scrivere un metodo che verifica che tutti gli usi del custom attribute al punto precedente, in un assembly passato come parametro, siano corretti, cioè che se un'interfaccia `I`, contenuta nell'assembly è annotata con tale attributo indicando `C` come sua implementazione di default [e $\texttt{C}_1, \ldots \texttt{C}_n$ come altre implementazioni alternative], allora nell'assembly compare una classe `C` che implementa `I` [e classi $\texttt{C}_1, \ldots, \texttt{C}_n$ che implementano `I`].

   Ricordate che fra i membri di `Type` (e di `TypeInfo`) vi sono metodi `Get<Tipologia>s` per le varie tipologie di membri (es. `GetMethods`, `GetInterfaces`, `GetCostructors`...), ove sensato anche generici (es. `GetCustomAttributes<T>`). Analogamente `Assembly` ha alcune property e metodi che potreste trovare utili:

   - `public virtual IEnumerable<TypeInfo> DefinedTypes;` Gets a collection of the types defined in the assembly
   - `public virtual Type GetType(string name)`: Gets the Type object with the specified name in the assembly instance (null if the class is not found)
   - `public virtual Type GetType(string name,bool throwOnError)` Gets the Type object with the specified name in the assembly instance and optionally throws an exception if the type is not found
   - `public virtual Type[] GetTypes()` Gets the types defined in this assembly