Appello TAP del 21/01/2016

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)

Sia `T` un tipo *ordinato*, cioè che implementa l'interfaccia `IComparable` e che quindi ha il metodo `CompareTo(Object)` che restituisce un numero intero minore di 0 se l'istanza corrente precede l'argomento della chiamata, 0 se sono uguali e un numero maggiore di 0 se lo segue (rispetto all'ordine). Scrivere l'extension-method `Min` che, invocato su `leftSeq`, una sequenza di elementi di tipo `T` e un ulteriore parametro, `rightSeq`, anch'esso una sequenza di elementi di tipo `T`, restituisce la sequenza di elementi di tipo `T` il cui elemento in posizione i è il minimo fra gli elementi in posizione i delle due sequenze.

Per esempio, il seguente frammento di codice

```
foreach (var d in new [] { 1, 3, -25, 42 }.Min(new[] {-11, 4, 7, 6, 77, 99, 1024 }))
    Console.Write("{0}, ",d);
Console.Writeln("finito");
```

stampa:

```
-11, 3, -25, 6, finito
```

Il metodo dovrà prendere come parametro "this" `leftSeq`, la sequenza sorgente, e come altro parametro la sequenza `rightSeq`. Nota: entrambe le sequenze possono anche essere infinite.

Il metodo deve sollevare l'eccezione...

- `ArgumentNullException` se `rightSeq` o `leftSeq` sono `null`;
- `ArgumentOutOfRangeException` se `rightSeq` contiene meno elementi di `leftSeq`.

Esercizio 2 (3+3+4 = 10 punti)

Implementare, usando NUnit ed eventualmente Moq, i seguenti test relativi al metodo `Min`, dell'esercizio precedente.

- Input della chiamata sotto test: `leftSeq` deve essere la sequenza $1, 7, 5, 4, 2, -3, 0$ e `rightSeq` deve essere la sequenza $5, 4, 11, 2, -2, 4, -1, 7$.

  Output atteso: la sequenza $1, 4, 5, 2, -2, -3, -1$.

- Input della chiamata sotto test: `leftSeq` deve essere una sequenza di stringhe inifinita e `rightSeq` deve essere la sequenza `"pippo"`, `"pluto"`, `"paperino"`.

  Output atteso: deve essere sollevata un'eccezione di tipo `ArgumentOutOfRangeException`.

- Input della chiamata sotto test: `leftSeq` deve essere la sequenza infinita dei numeri multipli di 4 (a partire da 0) e `rightSeq` deve essere la sequenza infinita delle potenze di 2.

  Il test deve essere parametrico con un parametro intero `howMany` e verificare che i primi `howMany` del risultato della chiamata siano i primi `howMany` elementi della sequenza $0, 2, 4, 8, 16, 20, 24, 28, 32 \ldots$

Esercizio 3 (3+4+6=13 punti)

- Si implementi un attributo `RangeChecked` con proprietà `Maximum` e `Minimum` di tipo `int` e una proprietà `ParamName` di tipo `string`; si definisca un costruttore che presi i valori necessari inizializzi le proprietà. Il costruttore dovrà verificare solo che la stringa non sia nulla né vuota e che i valori dati come massimo e minimo siano coerenti fra loro.

  `RangeChecked` (`ParamName`, `Minimum`, `Maximum`) rappresenta l'annotazione che richiede che il parametro `ParamName` sia compreso fra `Minimum` e `Maximum`.

  `RangeChecked` dovrà poter essere usato solo su metodi e dovrà essere possibile inserire più annotazioni.

  Si ignorino i problemi relativi ad annotazioni sbagliate (nome di parametro non esistenti, annotazioni relative a parametri di tipo non intero...).

- Implementare un metodo `VerifyRangeCheckedAttributes` che preso in input un metodo `m` verifica che tutte le annotazioni di `m` di tipo `RangeChecked` siano corrette, ovvero i nomi dei parametri nelle annotazioni corrispondano effettivamente a parametri di `m` di tipo `int`.

- Utilizzare `RangeChecked` per implementare il metodo `CheckedIntParamInvocation` generico sul proprio tipo di ritorno `T`, che, presi in input un metodo `m`, un oggetto su cui invocarlo e gli argomenti per chiamarlo, verifica che tutti gli argomenti di tipo intero (se ce ne sono) siano nei range richiesti dalle annotazioni di tipo `RangeChecked` e se così è effettua la chiamata del metodo e ne restituisce il risultato, altrimenti solleva l'eccezione `ArgumentOutOfRangeException`.

  Si assuma che gli argomenti passati a `CheckedIntParamInvocation` siano corretti (cioè che `m` abbia come tipo di ritorno `T`, l'oggetto su cui effettuare la chiamata sia non nullo ed abbia `m` fra i suoi metodi, e infine che gli altri argomenti siano del tipo richiesto da `m`) e non si facciano le verifiche relative.

Per accedere alle informazioni relative ai metodi ed ai loro parametri, tipi di ritorno etc., si ricorda che il tipo `MethodInfo` ha proprietà, come ad esempio `Name`, `ReturnType`, `DeclaringType` e metodi come ad esempio `GetCustomAttributes` (o la sua variante generica `GetCustomAttributes<T>`), `GetParameters`, ed analogamente `ParameterInfo` ha proprietà `ParameterType`, `Name` e così via.