Appello TAP del 20/01/2017

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 (3+4+3=10 punti)

- Definire un custom-attribute `Complexity` che permetta di specificare, solo su metodi, qual è la complessità di lettura del codice, espressa mediante un intero non negativo (in un'opportuna scala che non ci interessa dettagliare) e il nome di chi ha espresso questa valutazione. La complessità di uno stesso metodo può essere valutata da più persone e quindi l'attributo deve poter essere usato più volte su uno stesso metodo.

  Esempio d'uso:

```
[Complexity(42, "Capitan Harlock")]
[Complexity(7, "Jacob Frey")]
public void SomeMethod() { /* ... */ }
```

- Utilizzare il custom-attribute definito al punto precedente per implementare il metodo `GetComplexities` che dato un assembly restituisce un `Dictionary` che associa ad ogni classe dell'assembly la media delle complessità presenti in una valutazione dei suoi metodi (si ignorino i metodi non pubblici), o 0 se non sono presenti valutazioni per i suoi metodi.

- Utilizzare il custom-attribute definito al primo punto per implementare il metodo `AuthorMinLevel` che dato un assembly e una stringa contenente il nome del valutatore restituisce il minimo delle complessità dei metodi presenti nell'assembly e annotati dal valutatore (si ignorino i metodi non pubblici).

  Se nessun metodo nell'assembly è annotato dal valutatore deve essere sollevata una `ArgumentException`.

Esercizio 2 (10 punti)

Scrivere l'extension-method `ExtractLetters` che traduce sequenze di elementi di tipo `bool` in sequenze di elementi di tipo `char`. Il metodo `ExtractLetters` invocato su `s` ne estrae gli elementi a blocchi di `8`, interpreta ciascun blocco come un numero espresso su 8 bit, lo converte nel carattere corrispondente e restituisce tale carattere.

Per esempio, il seguente frammento di codice

```
foreach (var d in new [] {false, true, true, false, false, true, true, false,
                          false, true, true, false, false, false, false, true}.ExtractLetters())
    Console.Write("{0}, ",d);
Console.WriteLine("finito");
```

stampa:

```
f, a, finito
```

Il metodo dovrà prendere come parametro "this" `s`, la sequenza sorgente. Nota: la sequenza può anche essere infinita.

Il metodo deve sollevare l'eccezione...

- `ArgumentNullException` se `s` è `null`;
- `ArgumentOutOfRangeException` se si verifica una delle seguenti condizioni:
  - `s` è finita e la sua dimensione non è un multiplo intero di `8`
  - uno dei caratteri estratti non è una lettera.

Alcune informazioni che potrebbero essere utili:

- la conversione implicita da `char` a `int` è sempre corretta; viceversa, un cast da `int` a `char` è staticamente corretto, ma *può* causare errori dinamici (in contesti `checked`);
- `char` ha un metodo statico `IsLetter` che, dato un carattere, restituisce vero se e solo se il carattere è una lettera.
- per chi preferisce lavorare direttamente sulle codifiche dei caratteri, i valori numerici corrispondenti ad alcune lettere "critiche" sono i seguenti:

  'A': 65; 'Z': 90; 'a': 97; 'z': 122

  e le lettere sono fra loro contigue e ordinate (cioè se 'a'==97, allora 'b'==98, 'c'==99...)

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

1. Implementare il metodo

```
public static IEnumerable<bool> String2Bits(string s)
```

che prende in input una stringa s (di caratteri ASCII qualsiasi, non necessariamente lettere) e restituisce la sequenza dei bit corrispondenti alla rappresentazione in base 2 (in 8 bit) dei caratteri che compaiono nella stringa.

Ad esempio sulla stringa `"AA"` restituirà

```
false true false false false false false true false true false false false false false true
```

(ovvero 01000001 01000001).

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

   (a) Input della chiamata sotto test: `s` deve essere la sequenza la sequenza dei bit corrispondenti alla rappresentazione in base 2 dei caratteri che compaiono nella stringa `"gatto"`.

   Output atteso: la sequenza *gatto*.

   (b) Input della chiamata sotto test: `s` deve essere la sequenza dei bit corrispondenti alla rappresentazione in base 2 dei caratteri che compaiono nella stringa `"pollivendolo"` seguiti da ulteriori due bit.

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

   (c) Il test deve essere parametrico con un parametro intero `howMany` e un parametro di tipo stringa `theWord`.

   Input della chiamata sotto test: `s` deve essere la sequenza (infinita) dei bit corrispondenti alla rappresentazione in base 2 dei caratteri che compaiono nel parametro `theWord` ripetuti all'infinito.

   Il test deve verificare che il risultato della chiamata inizi con `howMany` copie dei caratteri che compaiono in `theWord`.