Appello TAP del 16/06/2016

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)

Scrivere l'extension-method `Pack` generico sul tipo reference `T` che, invocato su `elemSeq`, una sequenza di elementi di tipo `T` e un ulteriore parametro intero, `packageSize`, suddivide gli elementi in `elemSeq` in blocchi tutti della stessa lunghezza `packageSize`, li *impacchetta* in array e ne restituisce la sequenza.

Se il numero degli elementi in `elemSeq` (è finito, ma) non è un multiplo esatto di `packageSize`, l'ultimo array del risultato dovrà essere completato con dei `null`.

Per esempio, il seguente frammento di codice

```
var mySequence = Enumerable.Range(1, 20).Select(i => i.ToString());
foreach (var a in mySequence.Pack(8)){
      Console.Write('[');
      foreach (var s in a)
            Console.Write((s ?? "null")+", ");
     Console.WriteLine(']');
}
```

stampa:

```
[1, 2, 3, 4, 5, 6, 7, 8, ]
[9, 10, 11, 12, 13, 14, 15, 16, ]
[17, 18, 19, 20, null,  null, null,]
```

Il metodo dovrà prendere come parametro "this" `elemSeq`, la sequenza sorgente, e come altro parametro l'intero `packageSize`. Nota: `elemSeq` e, di conseguenza, il risultato potrebbero anche essere sequenze infinite.

Il metodo deve sollevare l'eccezione...

- `ArgumentNullException` se `elemSeq` è `null`;
- `ArgumentOutOfRangeException` se `packageSize` non è strettamente positivo.

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

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

- Input della chiamata sotto test: `elemSeq` deve essere la sequenza `"1"`, `"2"`...`"17"` e `packageSize` deve essere 5.

  Output atteso: la sequenza (di 4 array): `["1", "2", "3", "4", "5"]`, `["6", "7", "8", "9", "10"]`, `["11", "12", "13", "14", "15"]`, `["16", "17", null, null, null]`.

- Input della chiamata sotto test: `elemSeq` deve essere una sequenza di stringhe inifinita e `packageSize` deve essere 0.

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

- Input della chiamata sotto test: `elemSeq` deve essere la sequenza infinita delle stringhe composte da `"pippo"` seguito da un intero (a partire da 0 a crescere con passo 1), quindi `"pippo0"`, `"pippo1"`, `"pippo2"`, `"pippo3"`, `"pippo4"`... e `packageSize` deve essere 10.

  Il test deve essere parametrico con un parametro intero `howMany` scelto a caso fra 100 e 1000 e verificare che i primi `howMany` del risultato della chiamata siano corretti. Ovvero che l'i-esimo elemento del risultato sia l'array `["pippo"+(i*10), "pippo"+(i*10+1), "pippo"+(i*10+2), "pippo"+(i*10+3), "pippo"+(i*10+4)..., "pippo"+(i*10+9)]`

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

Utilizzando un meccanismo di comunicazione basato sugli eventi, implementare le parti essenziali di un sistema (sovra-semplificato) di allarme automatico anti-effrazione.

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, nel cercare di aprire la porta, viene inserito un numero di codice scorretto per tre volte di seguito venga invocato il metodo `CallSecurity` con i corretti parametri.

```
public class Flat{/*...*/
    private bool VerifyCode(string code){/*...*/}
    public void OpenDoor(string code){/*...*/}
}

public class SecurityService{/*...*/
    public void CallSecurity(Flat toBeChecked, DateTime when){/*...*/}
}
```