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 (10 punti)

Scrivere l'extension-method `Slice` che, presa una sequenza `s` di stringhe ed un intero `position`, restituisce la sequenza dei caratteri che si trovano in posizione `position` nelle varie stringhe di `s`, sollevando opportune eccezioni nel caso un elemento di `s` non sia sufficientemente lungo. Per esempio, il seguente frammento di codice

```
var names = new string[] { "Duckburg", "Topolinia is Mouseton", "???...aha!" }
foreach (var ii in names.Slice(0))
    Console.WriteLine(ii);
```

stampa, nell'ordine, `D`, `T`, `?`.

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

Il metodo deve sollevare l'eccezione...

- `ArgumentNullException` se `s` o uno dei suoi elementi è `null`;
- `ArgumentOutOfRangeException` se `position` è (strettamente) negativo;
- `ArgumentException` se uno degli elementi di `s` contiene meno di `position` + 1 caratteri.

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

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

1. Test parametrico con parametri interi `pos` e `howMany`.

   Input della chiamata sotto test: `s` deve essere una sequenza di `howMany` stringhe tutte di lunghezza non inferiore a `pos` +1 e aventi in posizione `pos` il carattere con codice ASCII 32, 33, 34...126, 32, 33... (cioè ripetitivamente i caratteri stampabili)

   Output atteso: la sequenza di `howMany` caratteri con codice ASCII 32, 33, 34...126, 32, 33...

2. Input della chiamata sotto test: `s` deve essere la sequenza (illimitata) della stringhe `"Iniziamo!!!!"`, `"a"`, `"aa"`,`"aaa"`... e `position` deve essere 3.

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

3. Input della chiamata sotto test: `s` deve essere una sequenza infinita di stringhe tutte di lunghezza 42 e aventi in posizione 21 il carattere con codice ASCII 32, 33, 34...126, 32, 33... e `position` deve essere 21.

   Il test deve verificare la correttezza dei primi 7000 elementi del risultato, ovvero che vengano restituiti i caratteri con codice ASCII 32, 33, 34...126, 32, 33... con un numero sufficiente di ripetizioni.

Esercizio 3 (10 punti)

Si assumano date le seguenti classi, che possono essere modificate a piacere ("..." indica le parti non rilevanti ai fini dell'esercizio) e che implementano un sistema giocattolo per la compra-vendita di titoli.

Di ciascun titolo (`Stock`) si conosce il prezzo corrente. L'agente di borsa (`Broker`) può comprare o vendere un certo numero di un dato titolo. Inoltre l'agente ha per ciascun titolo un elenco di operazioni (acquisti o vendite) che intende fare (`ToDo`) quando le condizioni di borsa diventeranno favorevoli e può inserire in o eliminare da tale elenco alcune operazioni.

```
public class Stock {
    /*...*/
    public double Price { get; set; }
}

struct Trade {
    internal double DesiredPrice { get; set; }
    internal int Quantity { get; set; }
    //If true we want to sell, otherwise we want to buy
    internal bool ToBeSold { get; set; }
    internal Trade(double desiredPrice, int quantity, bool toBeSold) {
        DesiredPrice = desiredPrice;
        Quantity = quantity;
        ToBeSold = toBeSold;
    }
}
public class Broker {
    /*...*/
    private Dictionary<Stock, List<Trade>> ToDo { set; get; }
    public void BuyStock(Stock stock, int howMany) {/*...*/}
    public void SellStock(Stock stock, int howMany) {/*...*/}
    public void QueueIntent(Stock stock, bool toBeSold, int quantity, double desiredP) {
        if (ToDo.ContainsKey(stock))
            ToDo[stock].Add(new Trade(desiredP, quantity, toBeSold));
        else
            ToDo[stock] = new List<Trade> {new Trade(desiredP, quantity, toBeSold)};
    }
    public void DequeueIntent(Stock stock, bool toBeSold, int quantity, double desiredP) {
        const double precision = 0.00000001;
        if (!ToDo.ContainsKey(stock))
            return;
        var i = ToDo[stock].FindIndex(t =>
             t.ToBeSold == toBeSold && t.Quantity == quantity &&
                  Math.Abs(t.DesiredPrice - desiredP) < precision);
        if (i < 0) return;//not found
        ToDo[stock].RemoveAt(i);
        if (ToDo[stock].Any()) return;
        ToDo.Remove(stock);//cleaning up empty list
    }
}
```

Utilizzando un meccanismo di comunicazione basato sugli eventi, migliorare queste classi così da permettere ad un agente di borsa di effettuare automaticamente le operazioni nel proprio elenco dei `ToDo` quando un cambiamente del prezzo di un titolo rende vantaggiosa l'operazione, ovvero se il nuovo prezzo è superiore al prezzo di vendita desiderato deve venire invocato il metodo `SellStock` con i corretti parametri e viceversa, se il nuovo prezzo è inferiore al prezzo di acquisto desiderato deve essere invocato il metodo `BuyStock` con i corretti parametri.