TAP: Appello del 8/9/2022

Scrivere nome, cognome e matricola sul foglio protocollo e sul foglio con le risposte al quiz. Avete a disposizione due ore e mezza.

Esercizio 1 (9 punti)

Data la classe statica

```
public static class ExtraMath {/*...*/}
```

Arricchirla di un metodo statico generico `GeneralizedTartaglia <T>` che prenda in input:

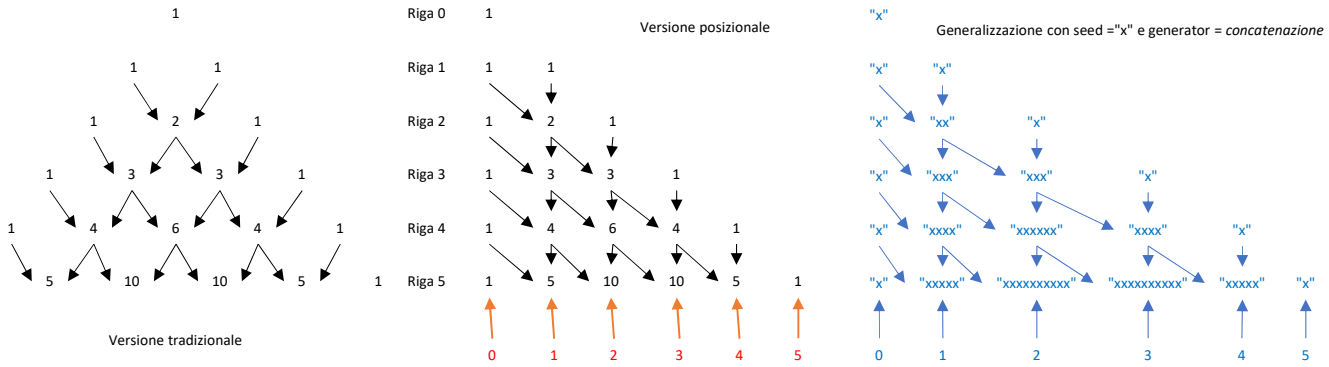
- un elemento `seed` di tipo `T`
- una funzione, `generator`, che dati due argomenti di tipo `T` restituisce un valore di tipo `T`

e restituisca una sequenza **infinita** di array di tipo `T` i cui elementi rappresentano le linee del triangolo di Tartaglia generalizzato usando, nella sua costruzione, `seed` invece di 1 e la funzione `generator` invece della somma.

Per chi non si ricorda, il triangolo di Tartaglia si costruisce induttivamente partendo con la prima riga (riga 0) contenente 1, usando le seguenti regole:

- la lunghezza di ciascuna riga è uno più di quella della riga precedente
- il primo e l'ultimo elemento di ciascuna riga sono 1
- gli altri elementi sono la somma degli elementi nella stessa posizione e nella posizione precedente nella riga precedente

Per capire meglio guardate la figura sottostante, dove avete a sinistra la rappresentazione standard del triangolo di Tartaglia, al centro una versione più informatica in cui vengono allineati gli elementi con la stessa posizione in righe diverse e a destra un esempio della generalizzazione richiesta.



Se nel corso della costruzione di una riga la funzione `generator` solleva eccezione, dovete sollevare una `AggregateException` che raccolga nella sua property `InnerExceptions` *tutte* le eccezioni sollevate nel computo di quella riga. Ad esempio se la concatenazione sollevasse una eccezione quando la stringa risultato contiene più di 3 caratteri, la costruzione della quarta riga non potrebbe andare a buon fine e dovreste sollevare una `AggregateException` contenente le 3 eccezioni generate cercando di costruire gli elementi in posizione 1, 2 e 3.

Hint: `AggregateException` mette a disposizione i costruttori `AggregateException(Exception[])` e `AggregateException(IEnumerable<Exception>)`, mentre `InnerExceptions` è una property di sola lettura.

Esercizio 2 (7 punti)

Implementare, usando NUnit, i seguenti test relativi a `GeneralizedTartaglia`, dell'esercizio 1.

1. Input della chiamata sotto test: `seed` vale 1 e `generator` è la somma (triangolo di Tartaglia standard).
   Output atteso: verificare che i primi 5 elementi del risultato coincidano con le righe da 0 a 4 del triangolo di Tartaglia (potete recuperare i valori dalla figura).
2. Input della chiamata sotto test: `seed` vale `"x"` e `generator` è la concatenazione di stringhe quando entrambi i suoi argomenti hanno lunghezza diversa da 4, altrimenti solleva un'eccezione di tipo `MyException`, dove

```
public class MyException : Exception {
  private static int _count;
  public int Index { get; } = ++_count;
  public MyException() { }
  public MyException(string message) : base(message) { }
  public MyException(string message, Exception inner) :
    base(message, inner) { }
}
```

   Output atteso: una eccezione di tipo `AggregateException` la cui property `InnerExceptions` contiene 4 eccezioni, tutte di tipo `MyException`, le cui property `Index` sono {1, 2, 3, 4}.
3. Test parametrico con parametro `lineNumber` di tipo intero. Se `lineNumber` non è strettamente positivo il test dovrà risultare *inconclusive*.
   Input della chiamata sotto test: a vostra scelta, purché la funzione usata come `generator` non sollevi mai eccezione
   Output atteso: verificate che enumerando `lineNumber` elementi del risultato la funzione `generator` venga chiamata (`lineNumber`-1)*(`lineNumber`-2)/2 volte.

Cognome e nome:

Esercizio 3 (9 punti)

Per ciascuna delle seguenti affermazioni, indicate se è vera o falsa

1. Anche se il server `Git` di riferimento per un repo (l'*origin* del repo) non è raggiungibile, i seguenti comandi possono ugualmente avere successo:

Vero	Falso	
☐	☐	`git commit`
☐	☐	`git rebase`
☐	☐	`git pull`

2. Confrontando le interfacce `IQueryable` e `IEnumerable`

non è detto...IQueryable estende IEnumerable

Vero	Falso	
☐	☒	se un metodo si può chiamare su un `IEnumerable` si può chiamare anche su un `IQueryable`
☒	☐	se un metodo si può chiamare su un `IQueryable` si può chiamare anche su un `IEnumerable`
☒	☐	un `IQueryable` ha il metodo `GetEnumerator`

3. Date le dichiarazioni

```
public class Parent {public void M(){Console.WriteLine("Parent");}}
public class Child: Parent {public void M() {Console.WriteLine("Child");}}
```

Vero	Falso	
☐	☒	Il codice `Parent o = new Child(); o.M();` genera un errore statico
☒	☐	Il codice `Parent o = new Child(); o.M();` stampa `Parent`
☐	☒	Il codice `Parent o = new Child(); o.M();` stampa `Child`

eseguito online :)

4. Data la dichiarazione `public class C<T>{/*...*/}`

Vero	Falso	
☒	☐	`C<3> c;` causa errore statico
☐	☒	`C<3> c;` causa errore dinamico
☐	☒	Se `A` è sottoclasse di una classe `B`, allora `C<A>` è sottoclasse di `C<B>`

errore di compilatore :)

La relazione di sottoclasse non viene ereditata dai tipi generici in modo diretto. La relazione di sottoclasse è specifica per i tipi concreti e non viene estesa ai tipi generici. Quindi, non possiamo fare assunzioni sulla relazione di sottoclasse tra tipi generici basandoci sulla relazione di sottoclasse tra i loro parametri di tipo

5.

Vero	Falso	
☒	☐	Le dichiarazioni `var v = 3;` e `int v = 3;` sono equivalenti
☐	☒	Le dichiarazioni `var v = 3;` e `dynamic v = 3;` sono equivalenti
☐	☒	Le dichiarazioni `var v = 3;` e `object v = 3;` sono equivalenti

In sintesi, mentre var determina staticamente il tipo della variabile in fase di compilazione, dynamic consente il tipo dinamico a tempo di esecuzione

object è un tipo esplicito :)

6. In un progetto in cui sia stato settato `<Nullable>enable</Nullable>`

Vero	Falso	
☐	☒	`string s = null;` causa errore dinamico
☒	☐	`string s = null;` può al più generare un warning del compilatore
☒	☐	`string s = null!;` è corretto sia staticamente che dinamicamente e non genera warning

testato su rider :)

7. Data una classe `C` con metodi `public void M(in int a1) { /*... */ }` e `public void F(out int a0) {/*... */}` e costruttore di default

Vero	Falso	
☐	☒	`new C().M(in 88);` è staticamente corretto
☒	☐	`new C().F(out var zzz);Console.WriteLine(zzz);` causa errore statico
☐	☒	`new C().F(out var zzz);Console.WriteLine(zzz);` causa errore dinamico nei casi in cui `zzz` non è inizializzato dalla chiamata di `F()`

compiler error

stampa a0

compiler error, non dinamico

8. Data `var q = new[] { 3, 5, −7, 0, 2 }.Select(i => i > 0 ? 2 * i : throw new Exception());`

Vero	Falso	
☐	☒	la dichiarazione di `q` causa errore dinamico
☐	☒	`q.Take(3);` causa errore dinamico
☐	☒	`q.Count();` causa errore dinamico

error CS1955: Non-invocable member 'ArgumentException' cannot be used like a method

se si toglie il throw e si mette un metodo, funge. Altrimenti static error

9. Applicando la *dependency injection by constructor*, se una classe `C` ha un campo di tipo `A`

Vero	Falso	
☒	☐	i costruttori di `C` devono avere almeno un parametro
☐	☒	`C` deve avere un metodo `public void Inject()` usato dal DI container per inizializzare il campo automaticamente
☐	☒	serve una *factory* per `A`, da usare nel costruttore di `C`

non è obbligatorio