Appello TAP del 10/07/2018

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 `LineProvider` che date due sequenze di interi produce una sequenza di funzioni da $\mathbb{Z}$ in $\mathbb{Z}$, l'i-esima delle quali rappresenta la retta avente come pendenza e termine noto l'i-esimo elemento rispettivamente dei due parametri, ovvero una rappresentazione della funzione $y = a_i * x + b_i$, dove a_i è la pendenza (in inglese *slope* o *gradient*) e b_i è il termine noto (in inglese *y-intercept*).

Il metodo dovrà prendere come parametro "this" `slopes`, la sequenza delle *pendenze* delle rette da costruire (una sequenza di interi **potenzialmente infinita**) e un ulteriore parametro `yIntercepts`, la sequenza dei *termini noti* delle rette da costruire (una sequenza di interi **potenzialmente infinita**), per produrre come output una sequenza di rette. Ad esempio sulle sequenze `slopes` $=1, -6, 7$ e `yIntercepts` $= 0, -5, 7$ produrrà (una rappresentazione di) $y = x, y = -6 * x - 5, y = 7 * x + 7$.

In caso `yIntercepts` abbia un numero di elementi strettamente minore di `slopes`, i termini noti mancanti verranno sostituiti con 0. Quindi, ad esempio sulle sequenze `slopes` $=7, 21, 42$ e `yIntercepts` $= 11$ produrrà (una rappresentazione di) $y = 7 * x + 11, y = 21 * x, y = 42 * x$. Viceversa, qualora `slopes` abbia un numero di elementi strettamente minore di `yIntercepts` si dovrà sollevare un' eccezione.

Il metodo `LineProvider` deve sollevare l'eccezione

- `ArgumentNullException` se `slopes` o `yIntercepts` è `null`.
- `ArgumentException` se `slopes` ha un numero di elementi strettamente minore di `yIntercepts`.

Si noti che le sequenze rappresentate da entrambi gli argomenti di `LineProvider` (e quindi il suo risultato) possono essere infinite.

Esercizio 2 ([2+3+5] = 10 punti)

Implementare, usando NUnit e/o Moq, i seguenti test relativi al metodo `LineProvider`, dell'esercizio 1.

1. Input della chiamata sotto test: `slopes` deve essere la sequenza vuota, `yIntercepts` deve essere la sequenza che contiene solo un intero strettamente positivo a vostra scelta.

   Output atteso: deve essere sollevata l'eccezione `ArgumentException`.

2. Input della chiamata sotto test: `slopes` deve essere la sequenza $1, 0, -7$, `yIntercepts` deve essere la sequenza $5, 4$.

   Output atteso: la sequenza che contiene tre elementi, la rappresentazione di $y = x + 5$, $y = 4$ e $y = -7 * x$. Verificare che l'elemento i-esimo effettivamente sia una rappresentazione corretta della funzione richiesta controllando solo i 4 punti per cui $x = 0$, $x = 100$, $x = 1$ e $x = -1$.

3. Test parametrico con parametri:

   whichLine: un intero positivo che indica quale elemento del risultato della chiamata si intende verificare;

   minX: un intero, corrispondente al primo valore delle ascisse su cui si intende verificare l'elemento scelto;

   maxX: un intero, corrispondente all'ultimo valore delle ascisse su cui si intende verificare l'elemento scelto;

   expected: un numero arbitrario di parametri interi, in cui verranno memorizzati i valori attesi come ordinate della retta da verificare in corrispondenza delle ascisse indicate

   Input della chiamata sotto test: `slopes` deve essere una sequenza **infinita** e `yIntercepts` una qualsiasi sequenza a vostra scelta, purché generino come risultato di `LineProvider` una sequenza di rette tutte diverse fra loro.

   Il test deve verificare (parzialmente) la correttezza della funzione di posizione `whichLine` nel risultato della chiamata, controllando che sugli interi in [`minX`, `maxX`] la funzione restituisca i valori in `expected`.

   Ad esempio se `slopes` $=0, 1, 2, 3, 4, ...$, `yIntercepts` $= 0, 10, 20, 30, 40....$, `whichLine` $= 10$, `minX` $= 0$, `maxX` $= 7$, allora il test dovrà verificare che l'elemento di posizione 10 (contando `whichLine` a partire da 0) del risultato di `LineProvider` rappresenti la funzione $10 * x + 100$ controllando che il suo risultato su x in $[0, 7]$ coincida con `expected` (che dovrà essere $[100, 110, 120, 130, 140, 150, 160, 170]$ dati i valori degli altri parametri e delle sequenze scelte come input della chiamata sotto test).

   Si noti che l'esempio è molto più complicato di quanto necessario a scrivere il test richiesto ed è stato scelto al solo scopo di far capire il significato dei parametri.

Esercizio 3 (10 punti)

Usare le seguenti interfacce

```
interface IPeople {
    int Id { get; }
    string Name { get; }
 }

interface IThing {
    int Id { get; }
    string Description { get; }
    int OwnerId { get; }
}
```

che collegano fra loro persone ed oggetti da loro posseduti per per rappresentare una *comunità di prestiti*. L'idea base è che un gruppo di persone specifiche, ad esempio studenti, sub, cuochi dilettanti, . . . , si organizza in un gruppo fra cui condividere in prestito una particolare categoria di oggetti, ad esempio libri di testo, apparecchiature per immersioni, strumenti di cucina e teglie,. . . . Quindi bisogna modellare un sistema che possa memorizzare quali oggetti possono essere presi a prestito, da quali persone e gli effettivi prestiti avvenuti in passato o ancora attivi. Solo chi offre oggetti da prendere a prestito ha diritto di prenderne a prestito a sua volta.

Nei seguenti punti si dettaglia meglio quanto richiesto.

1. Definire un'interfaccia *generica* per rappresentare la *comunità di prestiti*. I parametri di tipo dovranno poter essere instanziati solo su tipi che implementano/estendono rispettivamente `IThing` e `IPeople`. Le operazioni richieste sono le seguenti.

   (a) Aggiungere un oggetto a quelli che si possono prendere a prestito

   (b) Eliminare uno degli oggetti dal possibile prestito

   (c) Prendere a prestito un oggetto da parte di una persona

   (d) Restituire un oggetto che si era preso a prestito

   (e) Elenco di tutte le persone che partecipano alla comunità di prestito

   (f) Elenco dei prestiti attualmente in corso

   (g) Elenco dei prestiti (presenti e/o passati) che riguardano un dato oggetto

   (h) Ricerca di oggetti che possono essere presi a prestito data la descrizione di quello che si sta cercando

   Se utile/necessario, si possono definire interfacce ausiliarie. Come stile di segnalazione di errori (impossibilià ad eseguire operazioni), si usino sistematicamente eccezioni.

2. I parametri nell'interfaccia al punto precedente possono essere dichiarati co/contro varianti? perché?

3. Dare un'implementazione dell'interfaccia richiesta al primo punto, limitatamente ai metodi 1a, 1c, 1e e 1g e a eventuali metodi, campi o properties necessarie alla loro implementazione.

   N.B. L'implementazione richiesta deve lavorare solo in memoria (non, ripeto non vi preoccupate di salvare i dati in maniera permanente).

   L'`Id` deve agire come chiave e non ci devono essere duplicati negli elenchi.

   Alcune precisazioni sulle singole funzioni richieste.

   1a Aggiungere un oggetto a quelli che si possono prendere a prestito deve dare errore se esiste già un oggetto con lo stesso `Id` fra quelli noti. Se l'aggiunta va a buon fine, il proprietario acquisisce (se non lo aveva già) il diritto di prendere a prestito oggetti.

   1c Prendere a prestito un oggetto da parte di una persona deve dare errore se la persona non ha diritto di prendere a prestito, l'oggetto non è fra quelli che possono essere presi a prestito, oppure è già in prestito. Il prestito decorre dal momento in cui viene effettuata l'aggiunta.

   1e L'elenco di tutte le persone che partecipano alla comunità di prestito non deve contenere duplicazioni, ovvero non deve contenere due oggetti aventi lo stesso `Id`.

   1g Non è richiesta l'implementazione del tipo di ritorno di questo metodo, se diversa da tipi standard.