PROGRAMMAZIONE ORIENTATA AGLI OGGETTI (DM 270-9 e 12 CFU)
Appello del 09-01-2020 – Allievi Informatici

Cognome e Nome:	Matr:	CFU:	Email (stampatello):	Durata: 3 ore

Esercizio 1 [10 punti] Con riferimento alla classe AlberoEspressione studiata a lezione, in cui si elabora sotto forma di albero binario un'espressione aritmetica intera (operandi interi senza segno) con i quattro operatori +, -, * e /, sviluppare un metodo:

```
void build( String rpn )
```

che riceve una stringa contenente una versione postfissa dell'espressione aritmetica, e costruisce il corrispondente albero. Si ricorda che in una espressione postfissa un operatore segue immediatamente ai due operandi cui si applica. Nella stringa parametro i simboli sono separati uno dall'altro da spazi bianchi. Il metodo build(…) deve validare la stringa ricevuta mediante un'espressione regolare. Inoltre, deve sollevare un'eccezione runtime in tutti i casi in cui l'espressione risulta malformata.
[*Suggerimento*: basare la costruzione dell'albero su uno stack di nodi. Quando arriva un operando, si costruisce un nodo operando e lo si pone sullo stack. Quando arriva un nodo operatore, si costruisce il corrispondente nodo operatore, si estraggono i suoi due nodi operandi in cima allo stack, si legano opportunamente come figlio sinistro e figlio destro del nodo operatore, e si pone sullo stack il nodo operatore creato etc.]
Sviluppare altresì il metodo:

```
void inOrderIte( List<String> lv )
```

che visita in ordine simmetrico l'albero di espressione e memorizza gli elementi visitati sulla lista lv ricevuta parametricamente. Il metodo va ottenuto in veste iterativa.
Mostrare, infine, un semplice metodo main(…) aggiunto alla classe, che invoca in un caso i metodi richiesti.

Esercizio 2 [10 Punti] Sviluppare una classe Permutazioni che riceve a tempo di costruzione un array `a` di interi (supposti distinti), ed espone il metodo `void risolvi()` che lancia il processo di generazione di tutte le possibili permutazioni di `a`. Le permutazioni vanno ottenute con la tecnica backtracking e vanno visualizzate su standard output.
[*Suggerimento*: utilizzare un secondo array `b` della stessa dimensione di `a`, e strutturare la classe Permutazioni attorno ai seguenti metodi privati:

```
void permuta(int ps)
boolean assegnabile(int s, int ps)
void assegna(int s, int ps)
void deassegna(int s, int ps)
void scriviSoluzione()
```

in cui si considerano punti di scelta gli indici di elementi di `b`, e come scelte gli indici di elementi di `a`.]

Esercizio 3 [10 Punti] È definita la seguente interfaccia Cruciverba per elaborare uno schema risolto di parole crociate (matrice bidimensionale di caratteri), nel quale le caselle nere sono rappresentate da spazi bianchi:

```
public interface Cruciverba{
  int getNumeroRighe();
  int getNumeroColonne();
  boolean contains( String parola );
  List<String> paroleOrizzontali();
  List<String> paroleVerticali();
}
```

Il metodo `contains()` ritorna true se la parola ricevuta parametricamente è presente nel cruciverba, come parola orizzontale o verticale. I metodi `paroleOrizzontali()` e `paroleVerticali()`, rispettivamente, costruiscono e ritornano una lista concatenata di `java.util`, con le parole orizzontali del cruciverba, e le parole verticali. In ciascuna lista, la successione delle parole è per lunghezza crescente e, a parità di lunghezza, in ordine alfabetico.

Si chiede di scrivere una classe concreta `Schema` che implementa `Cruciverba` e riceve a tempo di costruzione una matrice bidimensionale di caratteri `schema`, contenente uno schema risolto di parole crociate. La classe `Schema` deve esportare anche i metodi `toString()`, `equals()` e `hashCode()`, basati sul contenuto costante della matrice `schema`.