Esercitazione sulla ricorsione e le collezioni
(Fondamenti di Informatica Emilio Di Giacomo)

Esercizio 1 – Si vuole scrivere un programma che genera tutti gli anagrammi di una stringa data. Gli anagrammi di una stringa possono essere generati secondo il seguente procedimento ricorsivo:

```
crea una sequenza di anagrammi seq vuota
se s ha lunghezza 0
     aggiungi s a seq
altrimenti
     per ogni carattere c di s
          sia s' la stringa ottenuta rimuovendo c da s
          calcola la sequenza seq' degli anagrammi di s'
          per ogni s'' nella sequenza seq'
               aggiungi c+s'' a seq
return seq
```

Scrivere una classe **Anagramma** con il seguente scheletro:

```
public class Anagramma{

     /* rimuove il carattere i-esimo dalla stringa s */
     private static String rimuovi(String s, int i){…}

     /* restituisce una lista contenente tutti gli anagrammi della
     stringa s */
     public static List<String> anagrammi(String s){…}

}
```

Dopo aver scritto la classe **Anagramma** scrivere una classe **TestAnagramma** dotata del solo metodo **main** che chiede all'utente di inserire una stringa e mostra tutti gli anagrammi della stringa inserita.

Esercizio 2 – Si modifichi la classe **Anagramma** dell'esercizio precedente dotandola del seguente metodo:

```
public static List<String> anagrammiInDizionario(String s){…}
```

che restituisce soltanto gli anagrammi di **s** che sono parole di senso compiuto. A tale scopo si utilizzi la lasse **Dizionario** che viene fornita già implementata.

La classe **Dizionario** estende la classe **TreeSet<String>** e rappresenta quindi un insieme di stringhe ordinate alfabeticamente. La classe è dotata soltanto del costruttore senza argomenti che crea un dizionario contenente 660000 parole della lingua italiana (lette dal file 660000_parole_italiane.txt.

Dopo aver modificato la classe **Anagramma** modificare la classe di **TestAnagramma** in maniera che visualizzi soltanto gli anagrammi di senso compiuto.

Esercizio 3. Siano date le classi **Studente** ed **Esame** le cui istanze rappresentano rispettivamente studenti ed esami universitari. Ogni studente ha un nome, un cognome e una matricola (tutti di tipo **String**). La classe **Studente** ha un costruttore che riceve come parametri il nome, il cognome e la matricola dello studente, metodi per leggere tali valori e il metodo **toString()** che restituisce una rappresentazione testuale dello studente. Ogni esame ha una materia (di tipo **String**), una data (di tipo **String**) ed un voto (di tipo **int**). La classe **Esame** ha un costruttore che riceve come parametri la materia, la data, e il voto, metodi per leggere tali valori e il metodo **toString()** che restituisce una rappresentazione testuale dell'esame.
Scrivere una classe **Registro** che permette di memorizzare gli esami sostenuti da un insieme di studenti. Per ogni studente viene memorizzata una lista di esami sostenuti. La classe è dotata dei seguenti metodi:

- **void iscriviStudente (Studente s)** che aggiunge lo studente **s** al registro;
- **Studente trovaStudente (String matricola)** che restituisce lo studente con la matricola data; se non esiste studente con la matricola data il metodo restituisce **null**;
- **boolean aggiungiEsame (String matricola, Esame e)** che aggiunge l'esame **e** alla lista degli esami sostenuti dallo studente **s**; il metodo restituisce **true** se l'inserimento è avvenuto con successo; l'inserimento fallisce se non esiste nessuno studente con la matricola data;
- **int numeroEsami(String matricola)** che restituisce il numero di esami sostenuti dallo studente con la matricola data; se non esiste studente con la matricola data il metodo restituisce 0;
- **double media(String matricola)** che restituisce la media dello studente con la matricola data; se non esiste studente con la matricola data il metodo restituisce 0;
- **String toString()** che restituisce una descrizione testuale dell'intero registro.

Nota: Si suggerisce di memorizzare gli studenti in una mappa utilizzando come chiave la matricola e di memorizzare le liste degli esami in una seconda mappa usando ancora la matricola dello studente come chiave.

Esercizio 4 – Un oggetto della classe **`FileSystem`** rappresenta una porzione del File System del calcolatore con radice in una cartella specificata. La classe **`FileSystem`** ha il seguente scheletro:

```
class FileSystem{

     private File root; // radice del file system

     /* crea un oggetto FileSystem a partire dalla cartella il cui
     percorso è passato come parametro */
     public FileSystem(String root){...}

     /* restituisce una stringa che descrive la porzione di file
     system rappresentata dall'oggetto corrente*/
     public String toString(){...}

     /* cerca un file (o directory) con il nome dato nella porzione
     di file system rappresentata dall'oggetto corrente.
     Restituisce il percorso completo del file o null se il file
     non viene trovato*/
     public String cerca(String nome){...}
}
```

Scrivere la classe **`FileSystem`** e la classe **`ProvaFileSystem`** per testarla.

Nota: per scrivere la classe **`FileSystem`** si utilizzi la classe **`File`** di Java (presente nel package **`java.io`**) che permette di rappresentare file e directory. I metodi della classe **`File`** che interessano sono i seguenti:

- costruttore **`File(String percorso)`**: crea un file con il percorso specificato.
- **`String getName()`**: restituisce il nome del file
- **`boolean isDirectory()`**: restituisce **`true`** se l'oggetto **`File`** rappresenta una directory
- **`File[] listFiles()`**: restituisce un array di oggetti **`File`** che rappresentano i file contenuti nella directory corrente (se l'oggetto su cui è invocato il metodo è una directory), **`null`** altrimenti

Il metodo stampa della classe **`FileSystem`** deve stampare la porzione di file system secondo il formato seguente:

```
Radice
	File0.txt
	SottoCartella1
		File1.txt
		File2.txt
	SottoCartella2
		File3.txt
		File4.txt
	SottoCartella3
		File5.txt
```