Esercitazioni di Laboratorio

8. Laboratorio 8

Come anticipato questo laboratorio e', in termini di numero di esercizi (non necessariamente in termini di tempo richiesto per la loro soluzione), piu' "leggero" del precedente. Tuttavia, se dovesse risultare troppo "veloce" potete approfittarne per terminare i laboratori precedenti o per sostenere il **quiz di Autovalutazione 3**.

I primi 2 esercizi sono sulla realizzazione di una classe (e' estremamente probabile, anzi certo) che vi venga chiesto di realizzare una classe, quindi bisogna continuare ad "allenarsi" a scrivere classi.

Il terzo e quarto esercizio sono sugli algoritmi di ordinamento (e' altrettanto estremamente probabile che all'esame venga chiesto un algoritmo di ordinamento). **Sono da svolgersi, ovviamente, dopo che sono stati presentati in classe.**

Esercizio 1

Argomento: Classi e variabili statiche; gestione di input ed eccezioni, "tokenizzazione" di stringhe
Modificare la classe BankAccount vista a lezione, con le seguenti diverse specifiche:

- i metodi deposit e withdraw restituiscono un valore di tipo logico, true se e solo se l'operazione è ammissibile e va a buon fine, ma non devono visualizzare nessun messaggio d'errore nel caso in cui non vada a buon fine (semplicemente, restituiscono false);
- se l'operazione non è ammissibile, il saldo del conto non deve essere modificato;
- un versamento va a buon fine se e solo se l'importo che viene versato è maggiore di zero;
- un prelievo va a buon fine se e solo se l'importo che viene prelevato è maggiore di zero e minore o uguale al saldo del conto.

Scrivere il programma BankAccountTester che crea un conto bancario (esemplare della classe appena progettata) e accetta ripetutamente comandi dall'utente, introdotti da tastiera, finché non viene introdotto il comando **Q** di terminazione del programma.

I comandi disponibili sono:

Q	Quit: termina il programma
B	Balance: visualizza il saldo del conto
D x	Deposit: versa nel conto la somma x
W x	Withdraw: preleva dal conto la somma x
A x	Add interest: accredita sul conto gli interessi, calcolati in base alla percentuale x del saldo attuale (x deve essere positivo)

Nei comandi che necessitano di un argomento, questo è separato dal comando mediante uno o più spazi. La lettera che indica il comando può essere maiuscola o minuscola. I comandi errati o con un numero errato di parametri devono essere semplicemente ignorati dal programma, chiedendo un nuovo comando.

Un esempio di esecuzione del programma e` il seguente (in corsivo i comandi introdotti dall'utente, in grassetto cio` che viene scritto dal programma):

```
Comando? (Q, B, D, W, A)
B
Saldo attuale: 0.0
Comando? (Q, B, D, W, A)
w  10
Prelievo non autorizzato
Comando? (Q, B, D, W, A)
D    10
Versamento effettuato: 10.0
Comando? (Q, B, D, W, A)
D    0
Versamento non corretto
Comando? (Q, B, D, W, A)
W 5
Prelievo effettuato: 5.0
Comando? (Q, B, D, W, A)
b
Saldo attuale: 5.0
Comando? (Q, B, D, W, A)
Z2
Comando? (Q, B, D, W, A)
A 5
Interessi calcolati e accreditati: 0.25
Comando? (Q, B, D, W, A)
B
Saldo attuale: 5.25
Comando? (Q, B, D, W, A)
q  5
Comando? (Q, B, D, W, A)
Q
Arrivederci
```

Tale esempio va anche inteso come specifica dei messaggi che devono essere visualizzati dal programma.
Il programma deve gestire tutte le possibili condizioni di dati in ingresso senza essere mai interrotto da eccezioni; nei casi in cui manchino specifiche di comportamento del programma, assumere ipotesi ragionevoli.

Esercizio 2

Argomento: uso di array riempiti solo in parte, ricerca lineare in array, gestione delle eccezioni

Scrivere una classe eseguibile **StudentManager** che gestisce un elenco di studenti.
In una prima fase il programma riceve dall'input standard oppure da file (decidete liberamente) e memorizza un elenco di dati che rappresentano

- i nomi di un insieme di studenti
- il voto della prova scritta
- il voto della prova orale

I dati di ciascuno studente sono inseriti in una riga separati da uno spazio (prima il nome, poi il voto scritto, poi il voto orale). I dati sono terminati da una riga vuota.
Al termine dell'inserimento, il programma chiede all'utente di inserire un comando per identificare l'elaborazione da svolgere.

- se l'utente inserisce il comando Q l'esecuzione termina
- se l'utente inserisce il comando S il programma
    - chiede all'utente di inserire il cognome di uno studente
    - se lo studente è presente nell'insieme, ne visualizza il voto finale (calcolato come media aritmetica tra i voti della prova scritta e della prova orale), altrimenti segnala che il cognome e` errato
    - chiede un nuovo inserimento di comando (Q o S)
- se il comando inserito è diverso da Q o S, il programma segnala che il comando è errato e chiede un nuovo inserimento di comando

Suggerimenti:

1. Scrivere una classe **Student**, che rappresenta il tipo di dato "studente" (specificato da nome, voto scritto, e voto orale), e usare un array

di oggetti **Student** per memorizzare l'insieme di studenti nel programma.
2. La parte piu` impegnativa di questo esercizio e` la scrittura della classe **StudentManager**. Si consiglia di individuare le principali operazioni che la classe deve realizzare (ad esempio, creazione di un oggetto **Student**, ricerca di un oggetto **Student** nell'array, stampa della media dei voti) e per ciascuna di esse scrivere un metodo statico ausiliario invocato dal metodo **main**.
3. Scrivere la classe in due fasi: in una prima fase realizzare le operazioni richieste senza curarsi di precondizioni e situazioni inaspettate; solo in una seconda fase affrontare il problema della gestione delle eccezioni (in particolare i metodi scritti potranno sia lanciare che catturare eccezioni).

Esercizio 3

Argomento: algoritmi di ordinamento

Implementare gli algoritmi di ordinamento visti (selection sort, mergesort e insertionsort) per gli interi come metodi statici di una classe ArrayAlgs. Anche se avete gia' provato a implementare questi algoritmi nei giorni scorsi, io vi consiglio di ripetere l'esercizio. Questi algoritmi bisogna conoscerli "a memoria". Attenzione che questo non significa ricordarsi ogni riga di codice a memoria, ma ricordarsi come questi algoritmi funzionano ed essere capaci di implementarli da zero. Infatti questi algoritmi usano array, cicli e/o ricorsione: tutti concetti che ora dovreste essere in grado di padroneggiare in modo autonomo. Se non vi ricordate gli algoritmi, ripassateli prima e poi provate a implementarli.

Esercizio 4

Argomento: analisi sperimentale degli algoritmi

Implementare una classe SortingTester che acquisisca in ingresso un parametro n (come args[0]) e:

1. costruisca un array A di numeri interi random da 1 a 100 di lunghezza n
2. testi ciascuno degli algoritmi visti (selectionsort, mergesort, insertionsort) su tale sequenza e se n<=20 verifichi che l'ordinamento sia corretto stampando in uscita l'array originale e i tre array ordinati
3. stampare in output i tempi di esecuzione di ciascun algoritmo attraverso l'utilizzo del metodo System.currentTimeMillis() chiamato subito prima e subito dopo la chiamata di ciascun metodo di ordinamento
4. Eseguire il programma per valori di n da 1000 a 10000 (con passo 1000) e da 10000 a 100000 (con passo 10000) e osservare l'andamento dei risultati
5. modificare il codice in modo che l'array di lunghezza n non sia generato casualmente ma sia ordinato in ordine crescente con gli interi da 0 a n-1. Eseguire il programma con n che varia da 10000 a 100000 (con passo 10000) e osservare l'andamento dei risultati
6. modificare il codice in modo che l'array di lunghezza n non sia generato casualmente ma sia ordinato in ordine decrescente con gli interi da n-1 a 0. Eseguire il programma con n che varia da 10000 a 100000 (con passo 10000) e osservare l'andamento dei risultati

Attenzione:
Fare attenzione, dopo aver effettuato il primo ordinamento, a non passare come argomento al secondo algoritmo l'array gia' ordinato! L'input deve essere uguale per tutti e tre gli algoritmi.