Prova di Laboratorio di Programmazione — Edizione 2

26 settembre 2014

ATTENZIONE: Non è possibile usare le classi del package `prog.io` del libro di testo

Lo scopo è realizzare un programma che permetta di svolgere alcune semplici operazioni su dei conti correnti bancari.

Oltre ai metodi richiesti in ciascuna classe, è opportuno implementare altri metodi che si ritengono utili per svolgere l'esercizio nel modo più corretto ed elegante, per esempio il metodo `toString` e i metodi di accesso ai campi.

Il codice esterno ad una classe non deve mai accedere direttamente ai campi della classe.

Le operazioni di stampa vanno effettuate esclusivamente nelle classi che definiscono il metodo `main`.

Esercizio 1

Si implementino le seguenti classi.

Classe `ContoCorrente`

Un oggetto della classe `ContoCorrente` rappresenta un conto corrente bancario caratterizzato da:
- numero del conto,
- nome dell'intestatario,
- cifra del saldo attuale,
- lista di movimenti effettuati.

Ogni conto corrente è univocamente identificato dal suo numero. Questo è un intero progressivo, assegnato a ogni nuova istanza di `ContoCorrente` al momento della creazione. Al primo conto creato viene quindi assegnato il numero 0, al secondo il numero 1, eccetera.

Nota: uno stesso intestatario può avere più conti correnti distinguibili dal numero di conto. Il nome dell'intestatario è specificato dal comando di creazione, come spiegato in seguito.

Quando un oggetto `ContoCorrente` viene creato, la lista movimenti contiene un unico movimento effettuato il giorno `0` e con saldo pari a un saldo iniziale (che deve necessariamente essere positivo; in caso contrario il conto viene creato con saldo pari a 0 euro). Ogni successivo movimento effettuato cambia il valore del saldo incrementandolo (versamento) o decrementandolo (prelievo).

Classe `Movimento`.

Un oggetto della classe `Movimento` rappresenta un movimento (versamento o prelievo) eseguito su un oggetto `ContoCorrente`. Un movimento è caratterizzato dal giorno di esecuzione e dalla somma (positiva se si tratta di un versamento e negativa se si tratta di un prelievo, per semplicità trascuriamo i centesimi). I giorni sono contati a partire dal giorno di attivazione del conto, che è convenzionalmente indicato con `0`.

La classe `ContoCorrente` definisce il metodo seguente:

`public double giacenzaMedia(int oggi)`
Ritorna il valor medio del saldo dal giorno zero fino al giorno precedente a quello corrente, indicato dal valore dell'argomento `oggi`.
Esempio: il conto `myConto` ha due movimenti, cioè un saldo iniziale di 100 euro il giorno 0 e un versamento di 20 euro il giorno 4. Quindi, se `oggi` vale 12, il conto nei 12 giorni (dal giorno 0 al giorno 11) aveva 100 euro per i primi 4 giorni e 120 euro per i rimanenti 8 giorni. La giacenza media è quindi pari a $(100 \times 4 + 120 \times 8)/12$ euro.

Scrivere un programma `Es1` che legge da **standard input** una sequenza di linee con i seguenti formati:

```
C,nome
C,nome,somma
D,giorno,somma
P,giorno,somma
S
S,nome
M,giorno
```

Il comando `C,nome` richiede la creazione di un conto corrente col nome indicato e con un deposito iniziale di 0 euro; se il comando è invece `C,nome,somma` il saldo iniziale è pari a `somma` (in caso sia positivo) o 0 (in caso `somma` sia negativo).

Le linee che iniziano con `D` richiedono l'esecuzione, nel giorno indicato, di un deposito sull'ultimo conto corrente creato della somma indicata.

Le linee che iniziano con `P` richiedono l'esecuzione, nel giorno indicato, di un prelievo sull'ultimo conto corrente creato della somma indicata, sempre che questa sia inferiore al saldo attuale nel conto; in caso contrario il comando viene ignorato e viene stampata la linea "Saldo insufficiente per effettuare il prelievo".

Le linee che contengono il comando `S` richiedono la stampa di numero di conto, intestatario e saldo dell'ultimo conto corrente creato.

Le linee che contengono il comando `S,nome` richiedono la stampa di numero di conto, intestatario e saldo di tutti i conti intestati a `nome`.

Le linee che iniziano con `M` richiedono la stampa della giacenza media al giorno indicato dell'ultimo conto corrente creato.

Non è necessario fare controlli sulla correttezza dell'input. Non vanno fatte assunzioni sul numero di linee in input. L'input va letto da **standard input**, utilizzando la redirezione fornita da shell (si veda l'esempio sotto).

Esempio

Il file `in1.txt` contiene le linee seguenti

```
C,Carlo
D,5,100
D,10,80
P,15,150
S
M,10
M,20
P,30,50
```

Si noti che il comando `M,10` richiede di stampare la giacenza media al giorno 10. Dato che i giorni interessati sono dal giorno 0 al giorno 9, la giacenza media risulta essere $(0 \times 5 + 100 \times 5)/10 = 50$, in quanto il deposito di 80 Euro il giorno 10 non viene considerato. Il comando `java Es1 < in1.txt` deve quindi stampare

```
>java Esercizio1 < in1.txt
Numero: 1, intestatario: Carlo, saldo: 30
Giacenza media: 50.0
Giacenza media: 77.5
Saldo insufficiente per effettuare il prelievo.
```

Esercizio 2

Si aggiunga al programma la classe seguente.

Classe `Banca`.

Un oggetto della classe `Banca` rappresenta una banca caratterizzata da una lista di conti correnti.

La classe `Banca` definisce il metodo seguente:

`public int bonifico(int giorno, int numFrom, int numTo, int somma)`
Se `somma` è maggiore del saldo del conto `numFrom` allora non viene eseguita nessuna operazione e il metodo ritorna il valore zero. Altrimenti, viene eseguito —con data nel giorno indicato— un prelievo dal conto `numFrom` di un ammontare pari a `somma` e un versamento sul conto `numTo` dello stesso ammontare.

Scrivere un programma `Es2` che crea una banca (un'istanza della classe `Banca`) e quindi legge da **standard input** una sequenza di linee con i seguenti formati:

```
C,nome,somma
C,nome
S,nome
S
D,numConto,giorno,somma
P,numConto,giorno,somma
B,giorno,numFrom,numTo,somma
```

I comandi `C,nome` e `C,nome,somma` sono gli stessi dell'Esercizio 1.

Il comando `S,nome` è lo stesso dell'Esercizio 1.

Il comando `S` richiede la stampa di tutti i conti correnti ordinati alfabeticamente rispetto al nome dell'intestatario.

Il comando `B` richiede l'esecuzione, nel giorno indicato, di un bonifico della somma indicata dal conto con numero `numFrom` al conto con numero `numTo`.

I comandi `D` e `P` sono simili all'Esercizio 1 con la differenza che contengono il numero di conto invece che il nome dell'intestatario. Precisamente: `D,numConto,giorno,somma` e `P,numConto,giorno,somma` richiedono il deposito o il prelievo dal conto `numConto` di una quantità di euro pari a `somma` nel giorno indicato da `giorno`.

Non è necessario fare controlli sulla correttezza dell'input. Non vanno fatte assunzioni sul numero di linee in input. L'input va letto da **standard input**, possibilmente utilizzando la redirezione fornita da shell (si veda l'esempio sotto).

Esempio

Il file `in2.txt` contiene le linee seguenti

```
C,Carlo,100
C,Beppe,50
C,Carlo
S
D,1,10,550
D,3,16,700
P,1,22,450
D,3,22,400
S,Carlo
S,Beppe
B,24,3,2,350
S
```

e il comando `java Es2 < in2.txt` deve stampare

```
Numero: 2, intestatario: Beppe, saldo: 50
Numero: 1, intestatario: Carlo, saldo: 100
Numero: 3, intestatario: Carlo, saldo: 0
Numero: 1, intestatario: Carlo, saldo: 200
Numero: 3, intestatario: Carlo, saldo: 1100
Numero: 2, intestatario: Beppe, saldo: 50
Numero: 2, intestatario: Beppe, saldo: 400
Numero: 1, intestatario: Carlo, saldo: 200
Numero: 3, intestatario: Carlo, saldo: 750
```

Istruzioni per la consegna

Consegnare soltanto i file seguenti (**non** i file `.class`):

```
Movimento.java
ContoCorrente.java
Banca.java
Es1.java
Es2.java
```

come un unico file compresso creato col comando

```
> zip provalab Movimento.java ContoCorrente.java Banca.java Es1.java Es2.java
```

Eseguire l'upload del file all'indirizzo `upload.di.unimi.it`.

Non consegnare file che non compilano.