Ascensori

Progetto di Programmazione del 18 settembre 2018

Introduzione

In questo esame vi chiederemo di scrivere un insieme di classi per la gestione di un sistema di ascensori.

Il progetto verrà valutato prima di tutto in base al suo corretto funzionamento rispetto ai requisiti qui descritti; suggeriamo di commentare con cura il codice, in particolare anche scrivendo i commenti `javadoc`.

Realizzazione

Classe Piano

Le istanze di questa classe rappresentano i piani di un edificio. Più precisamente, ogni istanza ha un nome (e.g., "Terra", "P1", "P2", "Garage"...), e ha associati altri due piani, chiamati sopra e sotto. Tali due piani (due attributi della classe) possono avere come valore un piano oppure `null` (nel caso che il piano considerato sia l'ultimo in alto o l'ultimo in basso, rispettivamente).

La classe ha solo un costruttore

- `public Piano(String nome, Piano sopra, Piano sotto)` : costruisce un piano con un dato nome e due dati piani superiore e inferiore (uno o entrambi tali piani possono essere `null`, come spiegato)

e i seguenti metodi

- `public String toString()` : che restituisce il nome del piano
- `public Piano sopra()` : restituisce il piano sopra quello su cui è invocato
- `public void sopra(Piano p)` : modifica il piano sopra, rendendolo uguale a `p`
- `public Piano sotto()` : restituisce il piano sotto quello su cui è invocato
- `public void sotto(Piano p)` : modifica il piano sotto, rendendolo uguale a `p`
- `public boolean ultimoInAlto()` : restituisce true se e solo se il piano su cui è invocato è il più alto dell'edificio
- `public boolean ultimoInBasso()` : restituisce true se e solo se il piano su cui è invocato è il più basso dell'edificio.

Realizzate inoltre il seguente metodo:

- `public int distanza(Piano p)`

Questo metodo restituisce la distanza fra il piano su cui è invocato e `p`; tale distanza sarà un numero positivo se `p` è più in alto di quello su cui è invocato, sarà un numero negativo se è più in basso e sarà `Integer.MAX_VALUE` se il piano `p` non fa parte dello stesso edificio. Il metodo va implementato come segue:

- un ciclo fa partire una variabile `x` dal piano corrente (quello su cui il metodo è invocato) e una variabile intera `d` da zero; ad ogni esecuzione del ciclo, la variabile `x` passa dal piano su cui si trova al piano che gli sta sopra, e incrementa la variabile `d` di uno; tale ciclo termina o quando `x==p` oppure quando `x==null`. Nel primo caso, siamo arrivati al piano che ci interessa e dobbiamo solo restituire `d`. Nel secondo caso...
- ...un secondo ciclo analogo al precedente parte sempre dal piano corrente (quello su cui il metodo è invocato), ma nel corso del ciclo passa a piani sempre più in basso. Anche questo ciclo termina o con `x==p` oppure quando `x==null`. Stabilite voi, sulla base della logica del metodo, che cosa si deve restituire nei due casi indicati.

Aggiungete infine il metodo:

- `public int confronta(Piano p)`: confronta il piano su cui è invocato con `p`, resituendo +1 se `p` è più in alto, -1 se è più in basso, 0 se i due piani sono uguali, o `Integer.MAX_VALUE` se i due piani non fanno parte dello stesso edificio.c

Classe Ascensore

Le istanze di questa classe rappresentano degli ascensori. Ogni ascensore, in ciascun istante, si trova a un certo piano e può essere in uno dei seguenti stati: *fermo*, *in salita*, *in discesa*. I possibili stati sono rappresentati da una variabile d'istanza (attributo) il cui valore sarà 0 se l'ascensore è fermo, +1 se è in salita, -1 se è in discesa. Inoltre, una terza variabile d'istanza (attributo) contiene il piano di destinazione (questa variabile ha senso solo se l'ascensore non è fermo; in caso contrario, il contenuto della variabile è irrilevante).

La classe ha solo un costruttore

- `public Ascensore(Piano p)`: crea un ascensore, inizialmente collocato al piano `p` e nello stato *fermo*

e ha i metodi

- `public Piano piano()`: restituisce il piano a cui l'ascensore si trova
- `public Piano destinazione()`: restituisce il piano destinazione (il valore restituito è privo di significato se l'ascensore è fermo)
- `public int stato()`: restituisce 0, +1 o -1 a seconda che l'ascensore sia fermo, stia salendo o

stia scendendo
- `public void cambia(int x, Piano d)` : modifica lo stato dell'ascensore; `x` deve valere 0, +1 o -1. Se `x` vale 0 vuol dire che l'ascensore si ferma (e in tal caso il valore di `d` è irrilevante); se `x` ha un valore diverso da 0, l'ascensore inizia a salire (o scendere, a seconda che `x` sia positivo o negativo) e `d` diventa la sua destinazione.

L'ascensore ha il seguente metodo

- `public void step()`

che cambia il piano e lo stato dell'ascensore come indicato:

- se l'ascensore è *fermo* questo metodo non fa niente
- se l'ascensore è *in salita* ma il piano a cui si trova è il più in alto dell'edificio oppure è il piano di destinazione, lo stato dell'ascensore diventa *fermo*
- se l'ascensore è *in salita* e il piano a cui si trova non è il più in alto dell'edificio e non è il piano destinazione, il piano diventa il piano superiore (lo stato rimane lo stesso)
- se l'ascensore è *in discesa* ma il piano a cui si trova è il più in basso dell'edificio oppure è il piano destinazione, lo stato dell'ascensore diventa *fermo*
- se l'ascensore è *in discesa* e il piano a cui si trova non è il più in basso dell'edificio e non è il piano destinazione, il piano diventa il piano inferiore (lo stato rimane lo stesso).

Classe Edificio

Le istanze di questa classe rappresentano degli edifici.

Ogni edificio ha associato un insieme di piani, che viene specificato all'atto della costruzione e che non può più cambiare, e un insieme di ascensori ognuno dei quali si trova inizialmente fermo ad uno dei piani dell'edificio.

La classe ha solo un costruttore

- `public Edificio(Piano[] p, Ascensore[] a)` : crea un edificio con i piani indicati nell'array `p` e gli ascensori indicati nell'array `a` .

Non è importante in che ordine i piani compaiano nell'array, ma si può dare per scontato che quando si crea un edificio tutti i piani siano presenti nell'array.

La classe ha inoltre i seguenti metodi:

- `public void step()` : esegue il metodo `step()` su tutti gli ascensori
- `public boolean ceAscensore(Piano p)` : restituisce true se e solo se c'è (almeno) un ascensore fermo al piano `p`
- `public Ascensore chiama(Piano p)` : è il metodo che implenta ciò che avviene se un utente al piano `p` chiama un ascensore; il metodo funziona così:

- se c'è un ascensore fermo al piano `p`, restituisce quell'ascensore
- altrimenti, considera tutti gli ascensori che sono in salita e si trovano a un piano inferore a `p` o sono in discesa e si trovano a un piano superiore a `p`; se questo insieme di ascensori non è vuoto, fra tutti questi ascensori ne prende uno che minimizza la distanza (in valore assoluto), e lo restituisce; inoltre se `p` è più in alto dell'attuale destinazione dell'ascensore, `p` diventa l'attuale destinazione;
- altrimenti, nel caso che l'insieme di ascensori descritto sopra sia vuoto, considera tutti gli ascensori fermi (a qualunque piano si trovino); se questo insieme di ascensori non è vuoto, fra tutti questi ascensori ne prende uno che minimizza la distanza (in valore assoluto), e lo restituisce; inoltre l'ascensore inizia a salire o scendere (a seconda che `p` sia più in alto o più in basso di dove si trova l'ascensore) e `p` diventa la sua destinazione;
- altrimenti, l'unica possibilità è che non ci siano ascensori fermi e che tutti gli ascensori in salita si trovino a un piano superiore a `p` e tutti quelli in discesa si trovino a un piano inferiore a `p`: in tal caso, il metodo deve restituire `null`.