ESAME DI FONDAMENTI DI INFORMATICA T-2 del 15/07/2015
Proff. E. Denti – G. Zannoni
Tempo a disposizione: 4 ore MAX

NB: il candidato troverà nell'archivio ZIP scaricato da Esamix anche il software "Start Kit"

NOME PROGETTO ECLIPSE: CognomeNome-matricola (es. RossiMario-0000123456)

L'azienda *ZannoLift,* produttrice di ascensori, richiesto lo sviluppo di un simulatore per i suoi ascensori.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA.
Un ascensore serve un singolo *edificio*, costituito da un certo numero di *piani*, sia sopra terra (identificati da numeri positivi) sia sotto terra (identificati da numeri negativi); per convenzione, il piano terra è sempre il piano 0.

Ogni piano è dotato di un pulsante di chiamata e di un display che indica dove si trova l'ascensore. I pulsanti possono assumere diversi colori: *verde* se l'ascensore è a quel piano, *rosso* se l'ascensore è stato chiamato a quel piano, *grigio (spento)* se l'ascensore è altrove e non è stato chiamato a quel piano.

Analoghi pulsanti sono presenti all'interno dell'ascensore, raggruppati in una elegante *pulsantiera*.

Gli ascensori prodotti da *ZannoLift* possono operare in varie modalità:

- in modalità ***Base***, l'ascensore serve una singola chiamata per volta: se viene chiamato a un piano, si muove verso di esso ignorando eventuali pressioni sui pulsanti fino a destinazione; quindi, non consente di "prenotarsi" per dopo, né effettua mai alcuna fermata intermedia (questa modalità è la tipica installazione presente nei condomìni);
- in modalità ***Evoluta***, invece, l'ascensore consente di effettuare *prenotazioni*: se, mentre si sta muovendo verso un piano, viene chiamato a un altro piano, esso prosegue per la destinazione originaria ma ricorda la nuova richiesta, che viene *messa in coda* per essere servita successivamente. (Ogni piano può ovviamente fare una sola prenotazione per volta: perciò, la lunghezza massima della coda è pari al numero di piani esistenti meno 1.) Da notare che anche in questo caso non si effettuano fermate intermedie: l'unica differenza rispetto alla modalità base è la possibilità di ricordare le prenotazioni e servirle in ordine di ricezione.

Come più oltre specificato, il file di testo Edifici.txt contiene la descrizione di alcuni edifici, ciascuno caratterizzato da un nome, dai piani da servire (piano minimo, piano massimo) e dalla modalità di funzionamento del relativo ascensore (base o evoluta).

Parte 1 (punti: 17)

Dati (namespace ascensore.model) *(punti: 11)*

Il modello dei dati è organizzato secondo il diagramma UML più sotto riportato.

SEMANTICA:

a) la classe ***Edificio*** (fornita) modella un edificio con le sue proprietà;

b) l'enumerativo ***Modo*** (fornito) definisce le due modalità di funzionamento dell'ascensore, ***BASE*** ed ***EVOLUTO***;

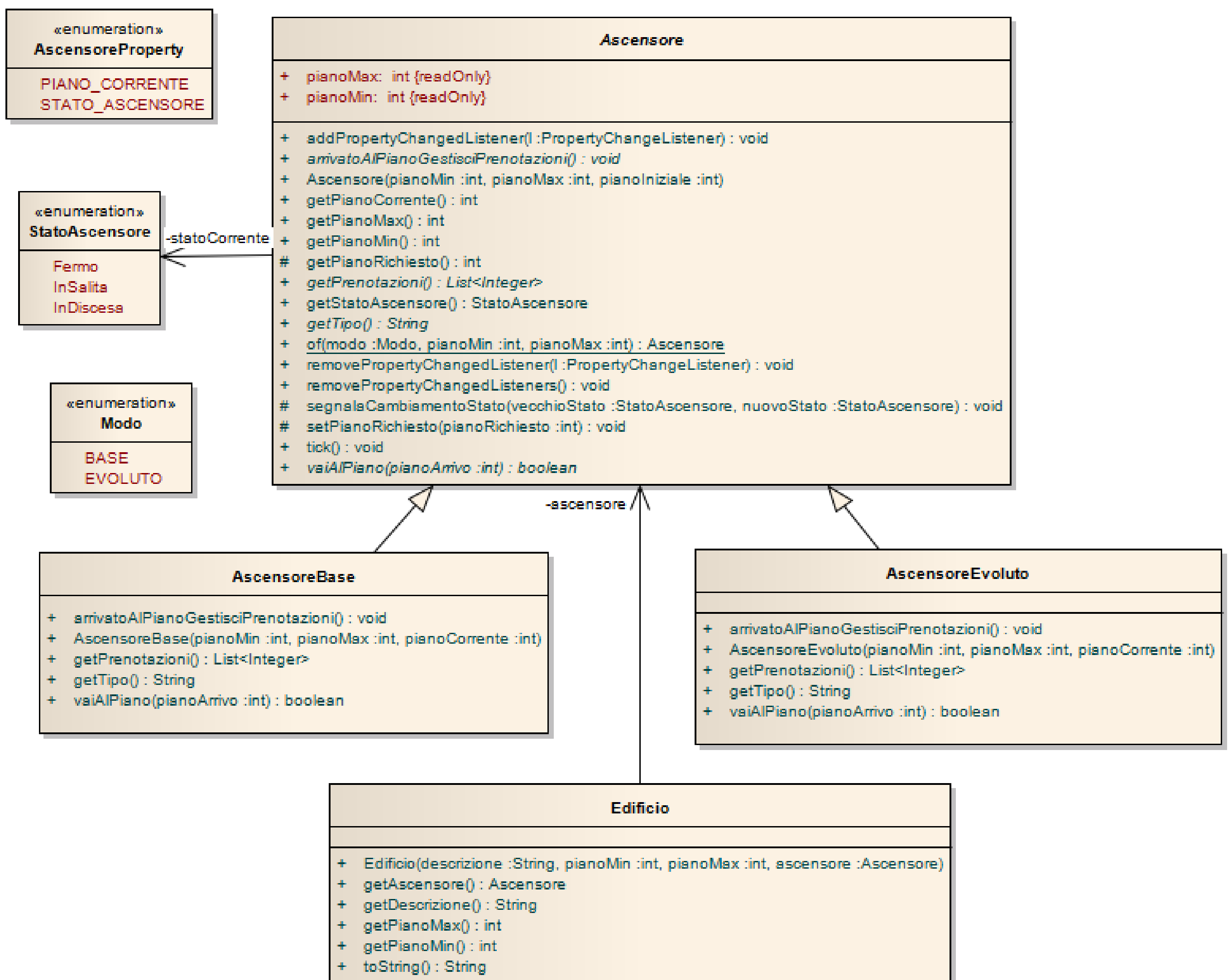
c) l'enumerativo ***StatoAscensore*** (fornito) definisce i tre stati possibili dell'ascensore – ***Fermo***, ***InSalita*** e ***InDiscesa***;

d) l'enumerativo ***AscensoreProperty*** (fornito) definisce le due proprietà ***PIANO_CORRENTE*** e ***STATO_ASCENSORE***;

e) la classe astratta ***Ascensore*** (fornita) fattorizza la caratteristiche comuni a tutti gli ascensori e implementa tutta la logica generale di simulazione. In particolare:

   - il costruttore accetta tre argomenti interi: il piano minimo, il piano massimo e il piano iniziale;
   - il metodo statico ***of*** incapsula una factory di ascensori, costruendo l'ascensore opportuno in base al modo di funzionamento richiesto.
   - gli accessor ***getPianoMin, getPianoMax, getPianoCorrente, getTipo*** e ***getStatoAscensore*** restituiscono lo stato attuale di tali proprietà (***tipo*** è una stringa che può valere "Base" o "Evoluto" – v. oltre);

- il metodo <u>astratto</u> ***vaiAlPiano*** esprime l'ordine di muoversi verso il piano indicato: le sue implementazioni concrete racchiuderanno la specifica logica di movimento desiderata; *restituisce un boolean* che informa se la richiesta è stata o meno accettata;
- il metodo <u>astratto</u> ***getPrenotazioni*** restituisce la lista delle prenotazioni esistenti: le sue implementazioni concrete rifletteranno la logica di funzionamento desiderata, popolando la lista di conseguenza;
- il metodo <u>astratto</u> ***arrivatoAlPianoGestisciPrenotazioni*** racchiude in sé la logica di gestione delle prenotazioni (quando presente: può non esserci alcuna logica), da applicare quando l'ascensore arriva e si ferma ad un piano: viene invocato automaticamente all'arrivo al piano; anche qui, le sue implementazioni concrete rifletteranno la logica di funzionamento desiderata;
- la coppia di accessor <u>protetti</u> ***getPianoRichiesto*** /***setPianoRichiesto*** mediano l'accesso alla proprietà "piano richiesto", in cui viene registrata la **richiesta** di andare a un dato piano prima di essere messa in atto.



f) la classe ***AscensoreBase* (da realizzare: punti 4)** concretizza ***Ascensore*** nel caso semplice in cui l'ascensore non accetti prenotazioni ma solo una singola richiesta:

- il costruttore deve avere la medesima *signature* di quello della classe base;
- il metodo ***getTipo*** deve restituire la stringa **"Base"**;
- Il metodo ***getPrenotazioni***, poiché l'ascensore base non accetta prenotazioni ma gestisce una sola chiamata per volta, restituisce:
    - la lista vuota, se l'ascensore è fermo al piano richiesto;
    - una lista contenente il singolo piano richiesto (da ***getPianoRichiesto***), altrimenti;

- Il metodo ***arrivatoAlPianoGestisciPrenotazioni*** non fa assolutamente niente;
- Il metodo ***vaiAlPiano*** deve dapprima verificare gli argomenti, lanciando ***IllegalArgumentException*** se il piano di destinazione è fuori range; dopo di che:
  - se l'ascensore è già in movimento, oppure il piano richiesto è diverso dal piano corrente, oppure l'ascensore è già fermo al piano di arrivo, respinge la richiesta (restituendo *false*)
  - se invece è fermo, accetta la richiesta (tramite ***setPianoRichiesto***) e restituisce *true*.

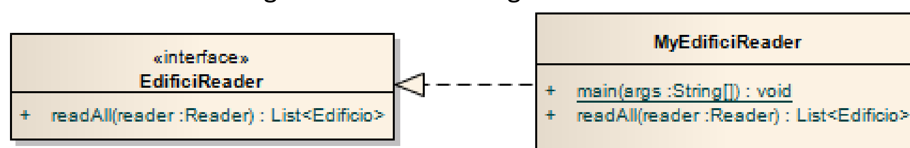
g) la classe ***AscensoreEvoluto*** **(da realizzare: punti 7)** concretizza ***Ascensore*** nel caso più complesso dell'ascensore evoluto. La differenza è che deve gestire la <u>*coda delle prenotazioni*</u> (si suggerisce l'uso di una ***LinkedList***, che può essere gestita o come ***List*** o come ***Queue*** a piacimento del candidato: importante è che sia applicata la politica a coda, ossia che i nuovi elementi siano aggiunti in fondo, mentre le estrazioni avvengano dalla testa).

- il costruttore deve avere la medesima *signature* di quello della classe base;
- il metodo ***getTipo*** deve restituire la stringa **"Evoluto"**;
- Il metodo ***getPrenotazioni*** restituisce la lista delle prenotazioni;
- Il metodo ***vaiAlPiano*** deve anche qui verificare in primo luogo che il piano di destinazione non sia fuori range, lanciando ***IllegalArgumentException*** in caso contrario; dopo di che:
  - se il piano di destinazione è già presente nella lista prenotazioni, oppure l' ascensore è già fermo al piano di arrivo e questi coincide col piano richiesto, respinge la richiesta (restituendo *false*), perché non ha senso prenotare lo stesso piano più volte;
  - se invece il piano indicato non è già presente nella lista prenotazioni:
    - se la lista prenotazioni è vuota, accetta la richiesta (tramite ***setPianoRichiesto***)
    - poi, comunque aggiunge alla lista prenotazioni la nuova richiesta e restituisce *true* [NB: ciò è coerente con l'ascensore base, che restituisce sempre una lista prenotazioni fatta dal singolo piano dove sta andando]
- il metodo ***arrivatoAlPianoGestisciPrenotazioni*** deve gestire la coda di prenotazioni nel momento in cui l'ascensore arriva e si ferma ad un piano e pertanto:
  - rimuove dalla lista prenotazioni il piano dove si è appena arrivati (dopo aver controllato quindi che il piano corrente sia effettivamente quello di destinazione originariamente prenotato);
  - recupera la prossima prenotazione in coda (se esiste) e la accetta (tramite ***setPianoRichiesto***)

Persistenza (package ascensore.persistence) *(punti 6)*

Come già anticipato, il file di testo Edifici.txt contiene la descrizione di alcuni edifici, uno per riga, i cui campi sono **separati da virgole**. Più precisamente ogni riga contiene nell'ordine una *descrizione testuale* dell'edificio (che può contenere spazi, tabulazioni, e ogni altro carattere diverso da virgola e terminatore di riga), i piani minimo e massimo (interi), e una stringa indicante il modo di funzionamento desiderato per l'ascensore (base o evoluto): tale stringa può essere scritta *con qualunque combinazione di caratteri minuscoli e/o maiuscoli.*

L'architettura software è illustrata nel diagramma UML che segue:



SEMANTICA:

a) l'interfaccia ***EdificiReader*** (fornita) dichiara il metodo ***readAll*** che restituisce una lista di ***Edificio***, lanciando, oltre alla "naturale" ***IOException***, una ***BadFileFormatException*** nel caso di errore nel formato del file;

b) la classe ***MyEdificiReader*** **(da realizzare)** concretizza ***EdificiReader*** implementando ***readAll*** secondo il formato del file sopra descritto, effettuando le puntuali verifiche di formato richieste. Non sono previsti costruttori.

Parte 2 (punti: 13)

L'interfaccia grafica deve permettere all'utente di scegliere un edificio fra quelli disponibili in una combo e simulare il funzionamento del relativo ascensore.

Appena si sceglie l'edificio, l'applicazione visualizza sulla sinistra la pulsantiera dell'ascensore, tarata sul giusto numero di piani, e svuota l'area di testo; il pulsante del piano corrente acquista lo sfondo verde [la colorazione è fatta automaticamente dalla pulsantiera] e il numero del piano corrente viene mostrato nel display in alto a destra (Fig. 1).

Premendo un pulsante, il pulsante del piano richiesto si colora di rosso (ad esempio, il 4° in Fig. 2; di nuovo, è la pulsantiera a occuparsi automaticamente di questo aspetto) e l'ascensore inizia a muoversi verso di esso, come mostrato dalla progressione dei colori dei pulsanti e dal numero mostrato dal display (Fig. 2).

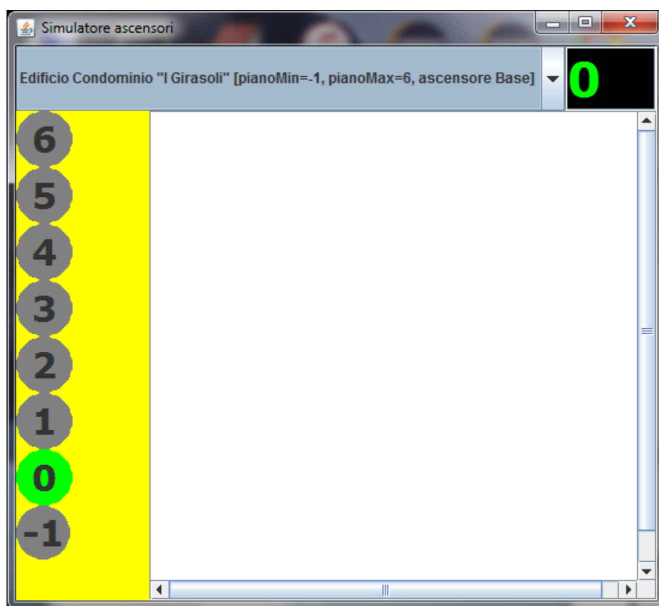


Figura 1: ascensore base, situazione iniziale

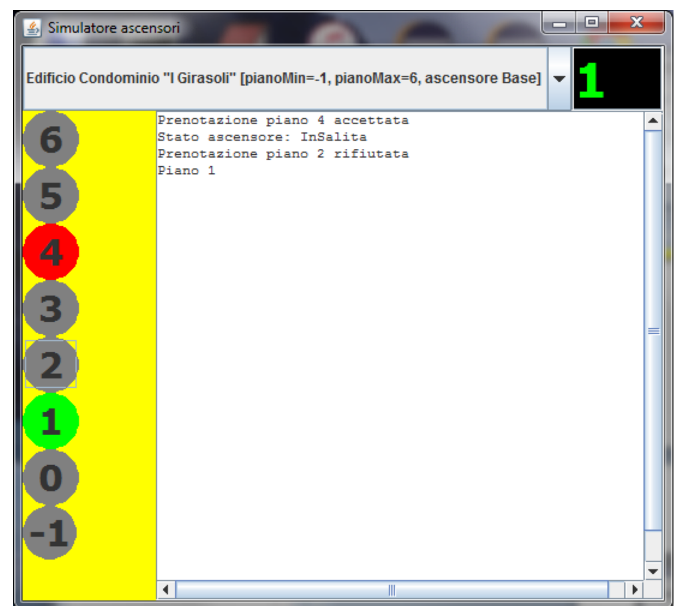


Figura 2: ascensore base in moto verso 4° piano

Nel caso dell'ascensore base, che serve un'unica chiamata per volta, ogni ulteriore tentativo di prenotazione mentre l'ascensore è in movimento viene rigettato (vedere l'output "Prenotazione piano 2 rifiutata" in Fig. 2).

L'ascensore evoluto (Fig. 3 – pagina seguente), invece, accetta la prenotazione e la mette in coda (Fig. 4): questo ascensore serve le richieste in ordine di arrivo, ovvero – nel caso mostrato in figura – prima il piano "-1" e poi, risalendo, il 4° (Figg. 4-5-6). Da notare che *non si effettua alcuna ottimizzazione degli spostamenti* (ossia, non si inseriscono fermate intermedie per "risparmiare giri"), perché non si tratta di un ascensore "smart", ma soltanto di un ascensore che ricorda le prenotazioni dei diversi utenti in attesa ai vari piani e le serve in ordine di prenotazione.

L'architettura software complessiva è illustrata nel diagramma UML a pagina 6.

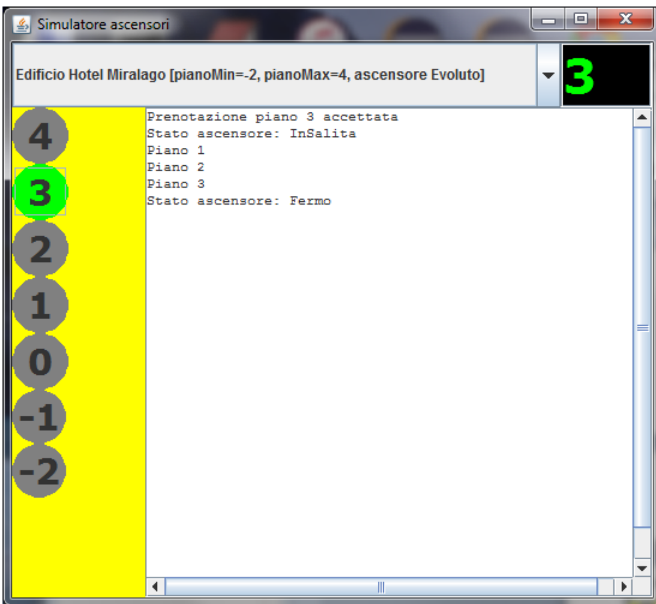


Figura 3: ascensore evoluto dopo salita al 3° piano

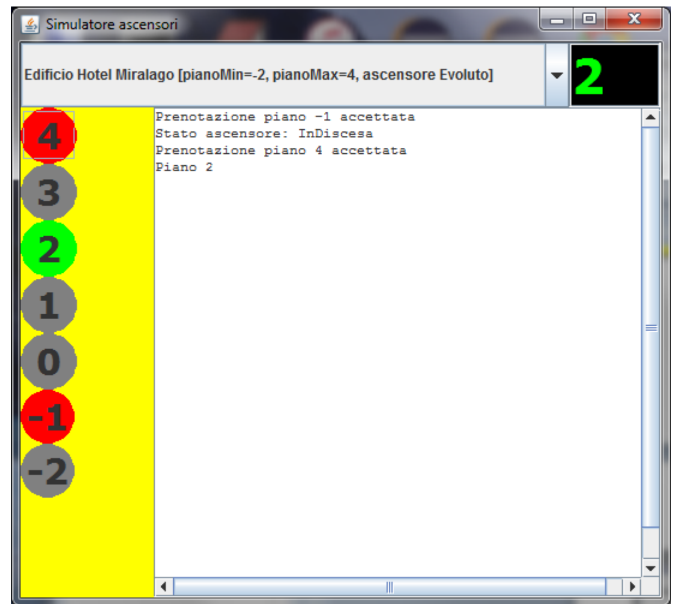


Figura 4: ascensore evoluto in discesa al -1…

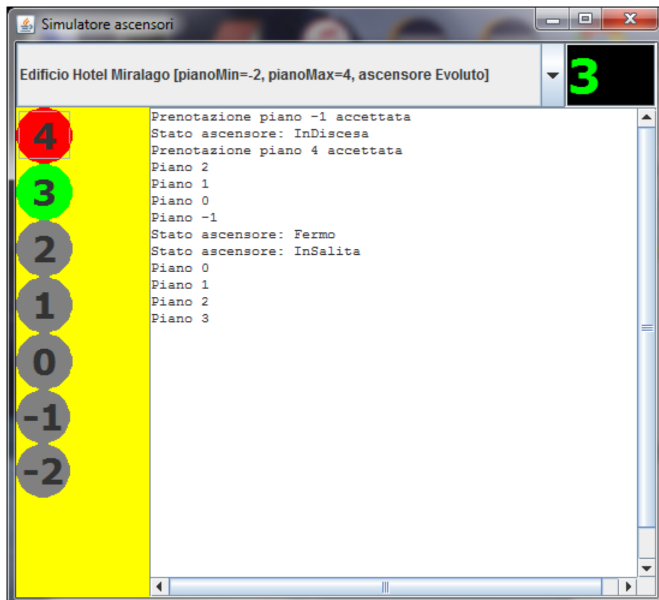


Figura 5: ..con successiva risalita vero il 4° piano

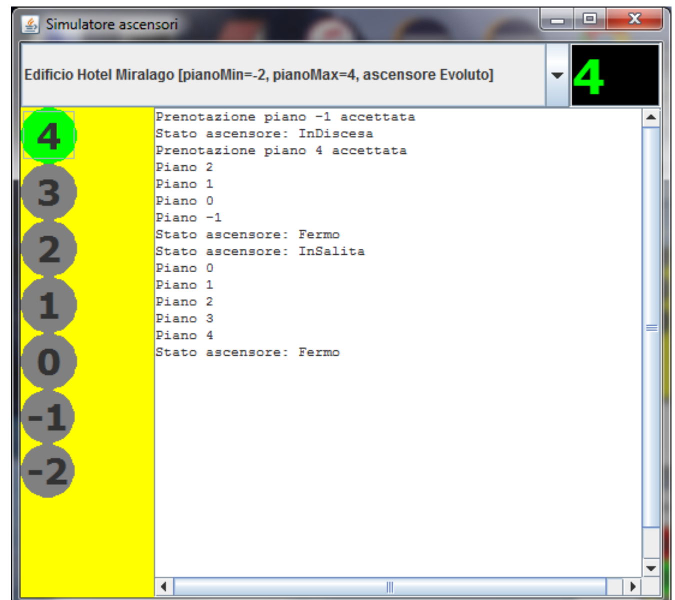


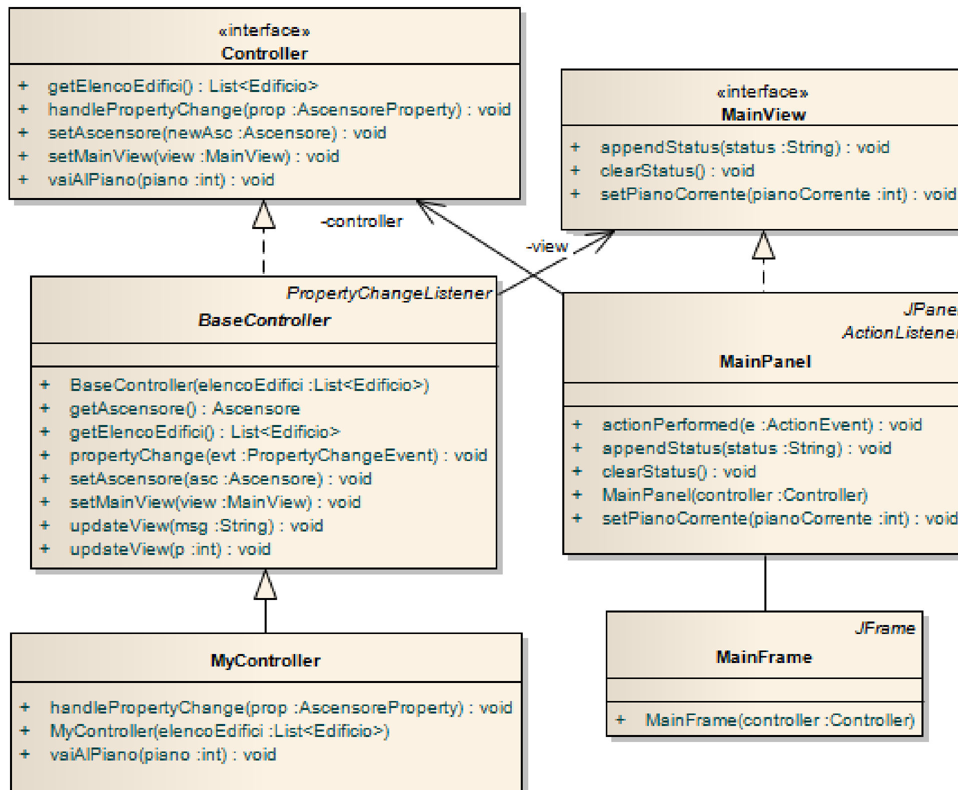
Figura 6: ascensore evoluto giunto al 4° piano

SEMANTICA:

a) la classe ***Program*** (fornita) contiene il main di partenza dell'intera applicazione;

b) il package ***ascensore.ui.pulsantiera*** (fornito) implementa la pulsantiera dell'ascensore. L'unica classe che è necessario conoscere è la classe ***Pulsantiera***

- seppur composta di vari pulsanti, si comporta verso l'esterno come un unico "maxi-bottone": quando uno dei suoi pulsanti viene premuto essa emette un ***ActionEvent***, da ascoltare tramite un normale ***ActionListener*** – agganciabile, secondo l'usuale schema, tramite ***addActionListener.***
- la pulsantiera viene costruita *indipendentemente dall'ascensore che dovrà servire*: per questo l'unico costruttore è quello di default, senza argomenti;
- la pulsantiera viene poi *configurata per servire uno specifico ascensore* tramite il metodo ***setAscensore***, che ne determina il numero di pulsanti e l'apparenza grafica; per cambiare ascensore basta reinvocare questo metodo – la pulsantiera si adeguerà subito di conseguenza (per completezza, il metodo duale ***getAscensore*** restituisce l'ascensore attualmente associato alla pulsantiera).



Controller (package ascensore.ui) *(punti 6)*

a) l'interfaccia ***Controller*** (fornita) dichiara i seguenti metodi:

- ***getElencoEdifici*** restituisce la lista degli Edificio disponibili;
- ***setMainView*** imposta il controller per governare la ***MainView*** passatagli come argomento;
- ***setAscensore*** imposta il controller per governare l'***Ascensore*** passatogli come argomento;
- ***vaiAlPiano*** chiede al controller di far muovere l'ascensore al piano passatogli come argomento;
- ***handlePropertyChange*** che esprime come reagire al cambiamento di una ***AscensoreProperty.***

b) la classe astratta ***BaseController*** **(fornita)** concretizza ***Controller*** implementando gran parte delle sue funzionalità: rimangono astratti due soli metodi, ***vaiAlPiano*** e ***handlePropertyChange.*** La classe inoltre introduce nuovi metodi di utilità per aggiornare l'interfaccia utente:

- ***updateView(int piano)*** aggiorna l'indicazione del piano corrente nella GUI
- ***updateView(String msg)*** aggiorna la text area che descrive l'avanzare della simulazione, appendendo un nuovo messaggio.

c) la classe ***MyController*** **(da realizzare)** deve dunque estendere ***BaseController*** implementando i rimanenti metodi astratti, sfruttando sagacemente le funzionalità ereditate; a tal fine:

- il costruttore riceve la lista di ***Edificio*** disponibili e delega ogni azione a quello della classe superiore;
- ***vaiAlPiano*** deve invocare l'omonimo metodo di ***Ascensore*** e, in base all'esito della richiesta, emettere sulla GUI il messaggio "**Prenotazione piano X accettata**" o "**Prenotazione piano X rifiutata**"
- ***handlePropertyChange*** deve reagire al cambiamento di una delle due ***AscensoreProperty*** e precisamente:
  - se cambia ***PIANO_CORRENTE***, aggiornare sia l'indicazione del piano corrente sia la text area, emettendo su quest'ultima il messaggio "**Piano X**" (vedere figure sopra);
  - se cambia ***STATO_ASCENSORE*** aggiornare solo la text area, emettendo su quest'ultima il messaggio "**Stato ascensore: XXX**" (vedere figure sopra).

Interfaccia utente (package ascensore.ui) ***(punti 7)***

SEMANTICA:

a) La classe ***MainFrame*** (fornita) rappresenta la finestra principale: incorpora un pannello di tipo ***MainPanel***, che a sua volta implementa l'interfaccia ***MainView***.

b) L'interfaccia ***MainView*** (fornita) espone i tre metodi che governano l'interazione fra UI e controller:

- ***appendStatus*** appende un messaggio sulla text area della GUI;
- ***clearStatus*** ripulisce la text area della GUI;
- ***setPianoCorrente*** visualizza il piano corrente sul display della GUI.

c) La classe ***MainPanel*** **(da realizzare)** implementa ***MainView***: è organizzata con BorderLayout, con in alto un box orizzontale contenente la combo con gli edifici disponibili e un campo di testo a due caratteri, usato come display del piano corrente (font Verdana 40 bold, colore verde su fondo nero); sotto al centro è mostrata la pulsantiera (dimensione preferita: 100x60; si ricordi che è anch'essa è JPanel!), mentre sulla destra vi è un'area di testo (30 x 60), racchiusa in un opportuno scroll pane.

Come già anticipato, la GUI consente di scegliere un edificio e simulare poi le chiamate ai vari piani del relativo ascensore, vedendo sull'area di testo i messaggi che descrivono cosa stia accadendo. Pertanto:

- la reazione alla scelta di un nuovo edificio deve reimpostare controller e pulsantiera sul nuovo ascensore da simulare;
- la pressione di un pulsante sulla pulsantiera deve delegare al controller la gestione della richiesta di chiamata a tale piano.