

Si deve implementare un piccolo programma che simuli e visualizzi il funzionamento di un ascensore a N piani con meccanismo di prenotazione. Ad ogni piano sono normalmente presenti due pulsanti: il primo pulsante viene premuto dagli utenti quando viene prenotato l'ascensore per salire rispetto al piano corrente (UP), e un pulsante da usare quando viene richiesto l'ascensore per scendere rispetto al piano corrente (DOWN).

All'esterno della cabina, ad ogni piano, è inoltre presente un indicatore che mostra se l'ascensore è attualmente *in salita* (gli utenti in discesa in questo caso evitano di salire a bordo se l'ascensore dovesse transitare al piano), o *in discesa* (gli utenti in discesa in questo caso evitano di salire a bordo se l'ascensore dovesse transitare al piano).

Assumeremo invece, per semplicità, che sia disponibile ad ogni piano una tastiera, posta all'esterno della cabina, dove si possa indicare immediatamente il piano di destinazione desiderato. Assumiamo inoltre che ad ogni piano ci sia al più un utente in attesa, e che la cabina dell'ascensore non abbia limiti di peso/num. di persone.

A tal proposito si deve implementare una classe `SimulazioneAscensore` che includa tutte le strutture dati necessarie alla avvio e alla gestione della simulazione, e in particolare i seguenti campi:

1. Un thread di tipo `ControlloAscensore`: questo thread implementa la logica decisionale che controlla l'ascensore. L'ascensore viene mosso periodicamente di un piano alla volta in alto o in basso rispetto al piano corrente; il `ControlloAscensore` può tener conto delle prenotazioni correnti fatte dagli utenti in attesa ai vari piani, nel momento in cui deve decidere che tipo di movimento fare;
2. Un thread di tipo `GeneraUtenti`: questo thread simula il comportamento degli utenti dell'ascensore, generando, a intervalli casuali, dei thread di tipo `Utente` provenienti da un certo piano scelto casualmente e diretti verso un piano scelto altrettanto casualmente; i thread `Utente` terminano una volta che la richiesta di spostarsi di piano è stata soddisfatta.
3. Un thread `Visualizzatore`: questo thread visualizza periodicamente lo stato dell'ascensore, indicando la posizione dell'ascensore e quella di eventuali utenti in attesa.
4. Un monitor di tipo `Ascensore`: questa classe contiene i metodi per acquisire e memorizzare le richieste prodotte dagli utenti; tiene aggiornato e memorizzato lo stato dell'ascensore, la posizione delle varie prenotazioni al piano, e fornisce i metodi per muovere l'ascensore.

Nella classe `Ascensore` devono essere implementati (almeno) i metodi

`bool prenotaEViaggia(int pianoS, int pianoD).`

Invocato da un thread `Utente` che si trova al piano `pianoS` e intende muoversi al piano `pianoD` facendo uso dell'ascensore. Il metodo è bloccante fintanto che l'ascensore non raggiunge il `pianoS`, l'utente sale, e infine viene raggiunto il `pianoD`. Il metodo ritorna immediatamente *false* se al `pianoS` c'è già un altro utente in attesa.

void muovi(bool direzione, bool prossDirezione).

Il metodo viene invocato dal thread `ControlloAscensore` per muoversi in alto (`direzione=true`) o in basso (`direzione=false`) rispetto al piano corrente.

L'effetto di ogni mossa deve essere quello di rimuovere dall'ascensore i passeggeri destinati al piano appena raggiunto e di caricare il passeggero eventualmente in attesa, ma solo se diretto nella direzione futura dell'ascensore (`prossDirezione`). Ad esempio, quando l'ascensore si trova al penultimo piano e vuole muoversi verso l'ultimo, quasi certamente il thread `ControlloAscensore` invocherà il metodo con i parametri `void muovi(true,false)`.

void visualizza()

Questo metodo, invocato dal thread `visualizzatore`, mostra a video, su due linee orizzontali, lo stato attuale dell'ascensore. Ad esempio l'output

```
----3----  
2  7
```

Indica che l'ascensore si trova, nel momento della visualizzazione, al quinto piano con tre persone a bordo, mentre al primo piano e al quarto si trovano due utenti, rispettivamente diretti al secondo e settimo piano.

Devono essere inoltre previsti (con definizione del prototipo a carico dello studente) tutti gli opportuni metodi per consentire al thread `ControlloAscensore` di leggere la posizione corrente dell'ascensore, e a quali piani risultano esserci utenti prenotati e dove sono diretti.

Nel progetto complessivo si deve tenere conto di (in ordine di priorità):

1. di garantire che il thread `Visualizzatore (V)`, `ControlloAscensore (CA)` e `GeneraUtenti (GU)` accedano alla struttura dati `Ascensore` con la dovuta mutua esclusione.
2. CA deve invocare il metodo *muovi* con la garanzia che nel frattempo lo stato dell'Ascensore non è cambiato (es. nuovi utenti ai piani, che potrebbero cambiare la scelta della direzione in cui muoversi).
3. Tra V, CA e GU non ci devono essere condizioni di deadlock e/o starvation;
4. **Bonus 1 (opzionale):** Gli **utenti in attesa** non devono soffrire di condizioni di starvation: CA deve prima o poi servire tutti gli utenti in attesa su tutti i piani (attenzione: la condizione 4 non ha nulla a che vedere con la condizione 3 e non è garantita da quest'ultima);
5. **Bonus 2 (opzionale):** si modifichi il thread `visualizzatore`, ed eventualmente alcune parti della classe `Ascensore` per garantire che la visualizzazione a video venga effettuata solo nel momento in cui sono rilevate modifiche allo stato dell'ascensore.

*E' **parte integrante** della prova di esame completare le specifiche date nei punti non esplicitamente definiti, introducendo eventuali campi e metodi ausiliari, e resolvendo*

eventuali ambiguità. Non è consentito modificare il prototipo dei metodi se questo è stato fornito.

Si deve implementare un piccolo programma che simuli e visualizzi il funzionamento di una testina che si muove leggendo da un disco con N tracce. In ogni istante la testina è posizionata su una certa traccia. Al sistema di gestione del disco possono pervenire delle prenotazioni per richieste di lettura che indicano il numero di traccia da leggere. Assumeremo, per semplicità, che 1) le tracce vengano lette nella loro interezza (non è possibile chiedere la lettura di singoli settori), 2) le operazioni richieste siano di sola lettura.

Il sistema di controllo del disco, per evitare continui movimenti inutili della testina, smaltisce le richieste di lettura solo nel momento in cui se ne siano accumulate 10. Eventualmente, le richieste vengono ordinate in maniera da ridurre il movimento della testina. Ad esempio se la testina si trova correntemente sulla traccia 5, e ci sono prenotazioni per leggere la traccia 3,4 e 9, si potrebbe pensare di muovere la testina prima sulla traccia 4, poi sulla 3, e infine sulla 9.

A tal proposito si deve implementare una classe `SimulazioneDisco` che includa tutte le strutture dati necessarie alla avvio e alla gestione della simulazione, e in particolare i seguenti campi:

5. Un thread di tipo `ControlloDisco`: questo thread implementa la logica decisionale che controlla la testina. Il disco può essere mosso di una traccia alla volta o in una direzione o in quella opposta rispetto alla traccia corrente; il `ControlloDisco` può tener conto delle prenotazioni correnti fatte e poste in attesa, nel momento in cui deve decidere che tipo di movimento fare; il `ControlloDisco` sposta la testina solo se ci sono 10 o più richieste di lettura in attesa.
6. Un thread di tipo `GeneraRichiesta`: questo thread simula il comportamento dei programmi che fanno uso del disco, generando, a intervalli casuali, dei thread di tipo `Requester` che richiedono la lettura di una certa traccia scelta casualmente, e terminano una volta che la richiesta è stata smaltita.
7. Un thread `Visualizzatore`: questo thread visualizza periodicamente lo stato del disco, indicando la posizione della testina e quella di eventuali richieste in attesa.
8. Un monitor di tipo `Disco`: questa classe contiene i metodi per acquisire e memorizzare le richieste prodotte dai `Requester`; tiene aggiornata e memorizzata la posizione corrente della testina, la posizione delle varie richieste di lettura prenotate, e fornisce i metodi per muovere la testina.

Nella classe `Disco` devono essere implementati (almeno) i metodi

`bool prenotaedEseguiLetture(int traccia).`

Invocato da un thread `Requester` che intende leggere la traccia `traccia` del disco. Il metodo è bloccante fintantochè la testina del disco non raggiunge la traccia richiesta, viene atteso un tempo casuale che simula il tempo di lettura e viene infine ritornato il valore `true`.

`void muovi(bool direzione)`

Il metodo viene invocato dal thread `ControlloDisco` per muoversi a destra (`direzione=true`) o a sinistra (`direzione=false`) rispetto alla posizione corrente della testina. L'effetto di ogni mossa deve essere quello di "smaltire" tutte le richieste di lettura in attesa sulla traccia appena raggiunta.

`void visualizza()`

Questo metodo, invocato dal thread visualizzatore, mostra a video, su due linee orizzontali, lo stato attuale del disco. Ad esempio l'output

```
-----*-----  
2    7
```

Indica che la testina si trova, nel momento della visualizzazione, sulla traccia 4, mentre sulla traccia 0 e sulla traccia 3 si trovano rispettivamente 2 e 7 richieste di lettura.

Devono essere inoltre previsti (con definizione del prototipo a carico dello studente) tutti gli opportuni metodi per consentire al thread `ControlloDisco` di leggere la posizione corrente della testina, e su quali tracce risultano esserci richieste.

Nel progetto complessivo si deve tenere conto di (in ordine di priorità):

1. di garantire che il thread Visualizzatore (V), `ControlloDisco` (CD) e `GeneraRichieste` (GR) accedano alla struttura dati Disco con la dovuta mutua esclusione.
2. CD deve invocare il metodo *muovi* con la garanzia che nel frattempo lo stato del Disco non è cambiato (es. nuove richieste che potrebbero cambiare la scelta della direzione in cui muoversi).
3. Tra V, CD e GR non ci devono essere condizioni di deadlock e/o starvation;
4. **Bonus 1 (opzionale):** I Requester **in attesa** non devono soffrire di condizioni di starvation: CD deve prima o poi servire tutte le richieste in attesa su tutte le tracce (attenzione: la condizione 4 non ha nulla a che vedere con la condizione 3 e non è garantita da quest'ultima);
5. **Bonus 2 (opzionale):** si modifichi il thread visualizzatore, ed eventualmente alcune parti della classe Disco per garantire che la visualizzazione a video venga effettuata solo nel momento in cui sono rilevate modifiche allo stato del disco.

*E' **parte integrante** della prova di esame completare le specifiche date nei punti non esplicitamente definiti, introducendo eventuali campi e metodi ausiliari, e risolvendo eventuali ambiguità. Non è consentito modificare il prototipo dei metodi se questo è stato fornito.*