Traccia A: Socket & Concurrent Programming

Titolo

Car Parks

Contesto

Car Parks vuole essere una piattaforma che simuli una sosta di un'automobile in un parcheggio manuale.

Per parcheggio manuale si intendono quei parcheggi gestiti da uno o più parcheggiatori che prendono in consegna un'automobile, la parcheggiano all'interno della struttura, e la restituiscono su richiesta.
I parcheggi manuali si differenziano dai parcheggi automatici per l'assenza di automazione nell'ingresso, nel pagamento e nell'uscita dal parcheggio stesso.

In un parcheggio manuale è inoltre possibile prenotare un posto auto, comunicando l'orario di arrivo, in modo che venga riservato e risulti libero.

Un'ultima funzionalità della piattaforma dà la possibilità all'automobilista di essere a conoscenza di tutti i parcheggi che aderiscono alla piattaforma e di poterne scegliere uno da utilizzare.

Descrizione

Versione base - fino ad un massimo di 2 punti [concorrenza]

Realizzare una piattaforma che permetta a un automobilista di parcheggiare la propria auto in un parcheggio tra quelli gestiti dalla piattaforma.

Tenere conto che più automobilisti potrebbero voler parcheggiare allo stesso momento e che un parcheggio ha a disposizione parcheggiatori per un numero limitato. Quindi, se dovessero presentarsi X automobilisti e fossero presenti Y parcheggiatori (dove X > Y) ci saranno automobilisti in attesa.

Più precisamente realizzare le seguenti classi (con attributi caratterizzanti):

- **Automobilista**: In grado di poter parcheggiare la propria automobile e di poterla ritirare dopo un determinato lasso di tempo;

- **Parcheggio**: Avente una determinata capacità di posti auto e Y oggetti di classe Parcheggiatore. La classe deve ritirare l'automobile che vuole essere parcheggiata (ritornando all'automobilista un codice univoco, ticket) e restituire l'automobile al suo legittimo proprietario (con il giusto ticket). Parcheggio è una classe che si comporta come un arbitro e quindi delegherà le proprie funzioni di ritiro e riconsegna dell'auto ad un parcheggiatore.

- **Parcheggiatore**: Gli oggetti di classe parcheggiatore riceveranno l'ordine di prendere in consegna o di restituire una determinata automobile. Simulare le operazioni con un tempo di attesa in secondi che potrebbe corrispondere alla reale esecuzione dell'operazione.

- **Automobile**: Classe che permette di istanziare gli oggetti che passeranno da automobilista al parcheggio e viceversa durante la sosta.

Upgrade 1 - incremento fino ad un massimo di 1 punto [socket]

Realizzare un server remoto con il quale un'automobilista possa comunicare mediante connessione Socket.
L'automobilista può far richiesta, al server remoto, della lista dei parcheggi disponibili. Un parcheggio si considera disponibile se ha ancora posti auto liberi.

In seguito alla scelta è possibile utilizzare il parcheggio come meta dell'automobilista.

Se un parcheggio dovesse diventare indisponibile, dovrà essere comunicato al server il cambio di stato in modo che alle successive richieste da parte degli automobilisti non compaia nell'elenco. Prevedere il caso reciproco se un parcheggio dovesse tornare disponibile.

Upgrade 2 - incremento fino ad un massimo di 1 punto [socket & concorrenza]

L'automobilista deve essere in grado di prenotare la propria sosta in un parcheggio. Il parcheggio ha una lista delle prenotazioni (per esempio uno slot ogni mezz'ora).

Utilizzare il server socket creato in precedenza come intermediario tra il parcheggio e l'automobilista.
Un esempio di caso d'uso potrebbe essere il seguente:
Un automobilista richiede al server gli slot disponibili di un parcheggio prescelto. Il server riceve la richiesta e la inoltra al parcheggio. Il parcheggio risponderà alla richiesta con la propria lista di slot disponibili ed il server la recapiterà all'automobilista.

In seguito alla scelta, l'automobilista comunica al server l'orario della prenotazione che a sua volta comunicherà al parcheggio la richiesta di prenotazione.

La lista che, in seguito alla richiesta, l'utente riceve deve essere al netto delle prenotazioni già presenti in modo da visualizzare solamente gli slot disponibili.

Quando la prenotazione viene effettuata, il parcheggio deve concedere lo slot solamente al primo richiedente e deve restituirgli un ticket valido sia per l'ingresso sia per l'uscita.
Prestare attenzione al caso in cui più richiedenti vogliano prenotare un determinato slot contemporaneamente.

In seguito alla prenotazione deve essere riservato un posto auto e quando l'automobilista si presenta con una prenotazione effettuata l'auto viene parcheggiata senza emettere un nuovo ticket.

Tips&Tricks

1. Per favorire l'esecuzione e la correzione dei tre punti del progetto, è **necessario** creare una classe Main per ognuno di essi.
2. Realizzare interfacce interattive per le varie scelte che l'automobilista potrebbe fare (es: scelta dello slot di prenotazione, scelta del parcheggio, …). Non si richiede la realizzazione di un'interfaccia grafica ma è sufficiente un'interfaccia "a riga di comando".

```
[1] prima scelta
[2] seconda scelta
[3] terza scelta
Premere il numero corrispondente alla scelta:
```

3. Le parole sottolineate potrebbero essere alcuni (non necessariamente tutti) dei metodi richiesti nell'implementazione. L'assenza di sottolineatura non implica che un metodo non sia necessario.
4. Utilizzare server concorrenti, come visto nel primo laboratorio, e prestare attenzione anche alla possibile necessità di gestire la concorrenza anche in alcuni metodi di determinati client.
5. Prima di procedere con la realizzazione delle diverse funzionalità, chiarirsi bene i flow delle esecuzioni e delle richieste. Aiutarsi con carta e penna, molte volte, può salvare la vita.
6. Nell'*Upgrade 1*, potrebbe è necessario scambiarsi tramite socket oggetti complessi. Accennato, ma non approfondito, nella lezione di laboratorio. Una semplice ricerca in internet e le slide di lezione potrebbero tornare utili.

Nell'*Upgrade 2*, se si ritiene necessario, utilizzare un formato per lo scambio delle informazioni e per richiamare i giusti metodi. JSON potrebbe essere un esempio, ma sono ammessi anche semplici formati custom, per semplificare la stesura di un piccolo parser.