**Programmazione concorrente**

Si consideri il codice dato, in cui due thread Giocatore fanno mosse su un tavolo da gioco (TavoloGioco). TavoloGioco garantisce la mutua esclusione (i metodi sono synchronized) ma non la sincronizzazione tra thread: è quindi possibile che due mosse consecutive siano fatte dal medesimo giocatore. NB: sono state fatte alcune semplificazioni: non si gestisce la fine della partita (cioè i due giocatori fanno mosse all'infinito) e non si comunicano a ciascun giocatore le mosse dell'altro.

Si chiede di modificare il codice dato in modo che i due giocatori muovano alternativamente (come negli scacchi, nella battaglia navale, ecc.), Attenzione: non e` consentito modificare la classe TavoloGioco. Bisogna pertanto che i giocatori si "mettano d'accordo" sui turni senza coinvolgere il tavolo da gioco, che resta ignaro dell'esistenza dei turni.

NB: ci sono diversi modi, anche molto diversi tra loro, per ottenere l'effetto desiderato. Verranno premiate le soluzioni che non implicano il blocco dei giocatori in attesa del turno, in modo che i giocatori possano impiegare utilmente il tempo durante il quale non sono di turno (ad es., pensando alla mossa successiva).

Evitare i problemi tipici della programmazione concorrente (corse critiche, starvation, deadlock, ecc.).
Si ricorda che bisogna caricare il file .java (NON i .class) in un unico file zip.

**Programmazione distribuita / socket**

Si consideri il codice dato (simile a quello dell'esercizio di programmazione concorrente, ma con tavolo che gestisce turni e attese).

Si modifichi il codice dato in modo da ottenere un sistema distribuito in cui il server gestisce il tavolo da gioco e i client hanno il ruolo di giocatori. I client giocatori si connettono al server, poi aspettano che il server li inviti a fare una mossa. Un client fa la sua mossa solo quando riceve l'invito dal server. Il server chiede ai due client giocatori di fare la loro mossa alternativamente, in modo che non vengano mai fatte due mosse consecutive da parte dello stesso giocatore.

Realizzare il sistema usando socket. NB: il gioco prevede due soli giocatori, quindi il sistema dovrà gestire due (soli) client.

Evitare i problemi tipici della programmazione concorrente (corse critiche, starvation, deadlock, ecc.).
Si ricorda che bisogna caricare il file .java (NON i .class) in un unico file zip.

**Programmazione distribuita / RMI**

Si consideri il codice dato (simile a quello dell'esercizio di programmazione concorrente, ma con tavolo che gestisce turni e attese).

Si modifichi il codice dato in modo da ottenere un sistema distribuito in cui il server gestisce il tavolo da gioco e i client hanno il ruolo di giocatori. I client giocatori si connettono al server, poi aspettano che il server li inviti a fare una mossa. Un client fa la sua mossa solo quando riceve l'invito dal server. Il server chiede ai due client giocatori di fare la loro mossa alternativamente, in modo che non vengano mai fatte due mosse consecutive da parte dello stesso giocatore.

Realizzare il sistema usando RMI. NB: il gioco prevede due soli giocatori, quindi il sistema dovrà gestire due (soli) client.

Evitare i problemi tipici della programmazione concorrente (corse critiche, starvation, deadlock, ecc.).
Si ricorda che bisogna caricare il file .java (NON i .class) in un unico file zip.