Esercizio 1. Programmazione concorrente

Si consideri il programma multi-thread dato, in cui ogni thread rappresenta un giocatore che, iterativamente

1) aspetta che sia possibile fare una mossa;
2) fa la sua mossa;
3) attende che tutti i giocatori abbiano mosso, e che sia dunque disponibile l'esito della mossa;
4) legge l'esito della mossa.

NB: lo scopo del gioco è irrilevante. Tuttavia, per fissare le idee si può pensare al gioco della carta più alta: ad ogni mano ciascun giocatore mette sul tavolo una carta coperta, e quando tutti hanno giocato si scoprono le carte, in modo che ciascuno possa vedere il risultato. Quando tutti hanno visto il risultato, si gioca la mano successiva.

Nel codice dato, i thread giocatori effettuano i passi 1) e 3) leggendo la situazione e se non possono proseguire dormono per un po' prima di riprovarci.

Si modifichi il codice dato, in modo che i thread non debbano gestire in proprio le attese ai punti 1) e 3): i thread giocatori devono essere sospesi ogni volta che e` necessario, ed essere svegliati immediatamente, appena le condizioni lo consentono.
Occorre evitare i problemi tipici della programmazione concorrente (corse critiche, deadlock, starvation, ecc.).
Si ricorda che bisogna caricare i file .java, NON i .class, raggruppati in un unico file zip.

Esercizio 2. Programmazione distribuita / socket

Si chiede di trasformare il programma sviluppato come soluzione dell'esercizio di programmazione concorrente in un sistema distribuito, in cui il server gestisce il gioco e i client operano come giocatori.
Si realizzi il sistema mediante l'uso di socket.
Come sempre, occorre evitare i problemi tipici della programmazione concorrente (corse critiche, deadlock, starvation, ecc.).
Si ricorda che bisogna caricare i file .java, NON i .class, raggruppati in un unico file zip.

Esercizio 3. Programmazione distribuita / RMI

Si chiede di trasformare il programma sviluppato come soluzione dell'esercizio di programmazione concorrente in un sistema distribuito, in cui il server gestisce il gioco e i client operano come giocatori.
Si realizzi il sistema mediante l'uso di RMI.
Come sempre, occorre evitare i problemi tipici della programmazione concorrente (corse critiche, deadlock, starvation, ecc.).
Si ricorda che bisogna caricare i file .java, NON i .class, raggruppati in un unico file zip.