Programmazione concorrente

Si desidera realizzare una applicazione concorrente, in cui operano tre thread: due istanze della classe Giocatore e uno istanza della classe Arbitro.
Ogni giocatore aspetta il proprio turno, poi effettua una mossa. Ogni volta che viene fatta una mossa, l'arbitro ne verifica la regolarità: se la mossa non risulta regolare il giocatore deve muovere nuovamente; se la mossa è regolare, il turno passa all'altro giocatore.

Per realizzare l'applicazione si può modificare il codice dato.
Attenzione:

- Nel codice dato manca solo l'implementazione della classe Gioco, che deve contenere tutto quel che serve per la corretta sincronizzazione dei thread.
- Non occorre modificare la logica del gioco, che è banale e irrilevante agli effetti della sincronizzazione.

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 programma realizzato come soluzione dell'esercizio di programmazione concorrente. Se ne modifichi il codice, in modo da ottenere un sistema distribuito in cui il server gestisce il gioco e giocatori ed arbitro sono client.

Realizzare il sistema usando socket.

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 programma realizzato come soluzione dell'esercizio di programmazione concorrente. Se ne modifichi il codice, in modo da ottenere un sistema distribuito in cui il server gestisce il gioco e giocatori ed arbitro sono client.

Realizzare il sistema usando RMI.

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.