Programmazione concorrente

Si desidera realizzare una applicazione concorrente, in cui operano due tipi di thread:

- Provider, che aggiorna un dato con cadenza casuale;
- Reader, che legge il dato.

Nel sistema esiste un unico Provider e diverse istanze di Reader.
Un reader che desidera leggere il dato si trova in due condizioni possibili:

- Il dato non è disponibile: in questo caso si mette in attesa del dato;
- Il dato è disponibile: in questo caso lo legge, rendendolo indisponibile per gli altri Reader.

Dunque si vuole che il dato venga letto da un solo Reader ogni volta che viene aggiornato.
Il dato è un'istanza della classe DataObject.
NB: il sistema è diverso dal classico produttore-consumatore in quanto il Provider non va mai in attesa: se un dato non è stato letto, viene comunque sostituito dal nuovo dato.

Per realizzare l'applicazione si può modificare il codice dato.
Attenzione: nel codice dato manca solo l'implementazione della classe DataObject, che deve contenere tutto quel che serve per la corretta sincronizzazione dei thread.

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 funge da Provider, mentre i Reader 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 funge da Provider, mentre i Reader 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.