Esercizio 1 Programmazione concorrente

Si consideri il codice dato.
Una classe Deposito gestisce un'informazione, costituita da una descrizione testuale e da un timestamp (un valore che indica quando e` avvenuto l'ultimo aggiornamento dell'informazione).
Un certo numero di thread Updater creati dal main accede al deposito per aggiornare l'informazione ivi contenuta; contestualmente viene aggiornato il timestamp.
Un certo numero di thread Client creati dal main accede al deposito per leggere l'informazione.

Il codice dato non effettua alcuna sincronizzazione. In particolare e` possibile che un Client legga piu` volte la stessa informazione.
Si richiede di modificare il codice dato in modo che un Client venga bloccato quando l'informazione disponibile non e` piu` recente dell'ultima che ha letto; cioe` il client aspetta di leggere un'informazione per lui nuova. Affinche' si possa valutare questa condizione, il client associa alla richiesta di lettura il timestamp dell'ultima informazione letta (come parametro del metodo read).

Si ricorda che bisogna caricare i file .java, NON i .class, meglio se raggruppati in un unico file zip.

Esercizio 2 Programmazione distribuita con socket

Si consideri il codice prodotto come soluzione all'esercizio di programmazione concorrente.
Si modifichi in modo che i client, gli updater e il server che gestisce il deposito siano processi che possono girare su macchine diverse.
Si utilizzino i socket per la comunicazione.

Si ricorda che bisogna caricare i file .java, NON i .class, meglio se raggruppati in un unico file zip.

Esercizio 3 Programmazione distribuita con RMI

Si consideri il codice prodotto come soluzione all'esercizio di programmazione concorrente.
Si modifichi in modo che i client, gli updater e il server che gestisce il deposito siano processi che possono girare su macchine diverse.
Si utilizzi RMI per la comunicazione.

Si ricorda che bisogna caricare i file .java, NON i .class, meglio se raggruppati in un unico file zip.