Esercizio 1. Programmazione concorrente

Si desidera un programma multi-thread che funzioni come descritto nel seguito.

I thread svolgono uno dei seguenti ruoli:
- thread “richiedenti”
- thread “esecutori”

I thread richiedenti operano in due fasi:
1. depositano una richiesta nel deposito delle richieste
2. leggono la risposta dal deposito delle risposte

I thread esecutori operano in tre fasi:
1. leggono una richiesta dal deposito delle richieste
2. elaborando una risposta
3. depositano la risposta nel deposito delle risposte

I depositi forniscono i seguenti metodi
- Lettura. La lettura comporta la cancellazione dell’elemento letto (richiesta o risposta) dal deposito. La lettura deve essere bloccante quando il deposito e` vuoto. Ci sono due tipi di letture:
  - Lettura di un elemento qualunque.
  - Lettura di una risposta specifica, cioe` della risposta avente l’identificatore passato come parametro.
- Aggiunta. L’aggiunta non e` mai bloccante: si suppone che ci sia sempre spazio per aggiungere una nuova richiesta o risposta.

NB: le richieste hanno un identificatore univoco (cioe` non esistono due richieste col medesimo identificatore). Ciascuna risposta contiene l’identificatore della richiesta cui si riferisce.

I richiedenti specificano l’identificatore della richiesta quando leggono la relativa risposta. Gli esecutori invece leggono una richiesta qualunque.

Il codice dato
- non implementa letture bloccanti (sia thread richiedenti che thread esecutori fanno polling)
- non implementa alcuna sincronizzazione tra thread.

Si modifichi il codice dato, in modo che
- Le letture risultino sospensive come indicato sopra.
- Ogni richiesta venga letta e consumata da un unico esecutore.
- Non si verifichino corse critiche, deadlock, starvation, ecc.

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

Esercizio 2. Programmazione distribuita / socket

Si consideri il problema di programmazione concorrente dato.
Si chiede di trasformare il programma in un sistema distribuito, in cui i processi richiedenti ed esecutori possono trovarsi su macchine diverse. Un ulteriore processo server gestisce il deposito delle richieste e quello delle domande. Si consiglia di creare un unico server che gestisce entrambi i depositi.

Si realizzi il sistema mediante l’uso di socket.
NB: deve essere implementata la soluzione dell’esercizio di programmazione concorrente, non il codice dato.

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

Esercizio 3. Programmazione distribuita / RMI

Si consideri il problema di programmazione concorrente dato.
Si chiede di trasformare il programma in un sistema distribuito, in cui i processi richiedenti ed esecutori possono trovarsi su macchine diverse. Un ulteriore processo server gestisce il deposito delle richieste e quello delle domande. Si consiglia di creare un unico server che gestisce entrambi i depositi.

Si realizzi il sistema mediante l'uso di RMI.
NB: deve essere implementata la soluzione dell'esercizio di programmazione concorrente, non il codice dato.

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