Es.1

Il testo chiedeva “di modificare il codice dato in modo che un client venga bloccato quando l'operazione richiesta non e` possibile, cioe` quando il deposito da cui prelevare e` vuoto o il deposito in cui aggiungere e` pieno.”

Questo si puo` fare molto semplicemente rendendo il metodo synchronized a poi implementando la condizione esattamente come indicato nel testo.

```
public synchronized void switchElement(boolean AtoB) {
        if(AtoB) {
                while(DA.isEmpty() || DB.size()==MC) {
                        try {
                                wait();
                        } catch (InterruptedException e) { }
                }
        . . .
```

Prima di uscire dal metodo occorre ricordarsi di fare notifyAll();

Il codice ottenuto puo` andare in deadlock quando uno dei due depositi e` vuoto e tutti i client richiedono di trasferire dati da quel deposito. La condizione illustrata sopra blocca tutti i client, e non ce n’e` alcuno che facendo l’operazione inversa ne possa sbloccare almeno uno.

Esistono diversi modi per evitare questa situazione. Naturalmente quale sia meglio usare dipende da considerazioni sull’uso reale che si vuole fare del sistema. Un modo semplice per evitare il deadlock consiste nel non svuotare mai completamente un deposito: quando un client tenta di asportare l’ultimo elemento, l’operazione fallisce immediatamente, e il client riceve un risultato false. Questa e` la soluzione illustrata nel codice disponibile.
Si noti che questa soluzione evita che il sistema si blocchi, ma non garantisce che il sistema funzioni come desiderato: se un client che riceve un risultato false riprova ostinatamente a eseguire la stessa azione, avremo un sistema “funzionante” in cui i client eseguono all’infinito un ciclo di operazioni che falliscono tutte regolarmente.

Es. 2 e 3

La trasformazione del sistema da programma multi-thread a insieme di programmi distribuiti puo` essere fatta come gia` visto in molti temi d’esame precedenti. Si rimanda pertanto a quelli.