**Esercizio 1. Programmazione concorrente**

Si consideri un programma multi-thread in cui

- un thread gestisce risorse;
- altri thread richiedono e poi usano le risorse.

Le risorse sono di tipo A e B.
Ogni thread utilizzatore esegue un programma tale per cui si può trovare in una delle seguenti situazioni:

- ha bisogno di una risorsa di tipo A per proseguire l'elaborazione
- ha bisogno di una risorsa di tipo B per proseguire l'elaborazione
- ha bisogno di una risorsa di tipo A e di una di tipo B per proseguire l'elaborazione
- non ha bisogno di alcuna risorsa per proseguire l'elaborazione

NB: in nessun caso un thread utilizzatore richiede più di una risorsa di tipo A e una di tipo B.

Nel codice dato i thread utilizzatori provano a ottenere le risorse e se non ci riescono si sospendono per un po' prima di riprovarci.

**Domanda 1:**
Spiegare perché il codice dato può causare deadlock.

**Domanda 2:**
Si modifichi il codice dato, in modo che non si verifichino casi di deadlock. A questo scopo, si può usare la tecnica che si preferisce. Occorre però garantire un certo livello di concorrenza tra i thread.
Come sempre, occorre evitare i problemi tipici della programmazione concorrente (corse, critiche, deadlock, starvation, ecc.).
Si ricorda che bisogna caricare i file .java, NON i .class, raggruppati in un unico file zip.

**Esercizio 2. Programmazione distribuita / socket**

Si chiede di trasformare il programma sviluppato come soluzione dell'esercizio di programmazione concorrente in un sistema distribuito, in cui il server gestisce le risorse e i client operano come utilizzatori delle risorse.
Si realizzi il sistema mediante l'uso di socket.
Come sempre, occorre evitare i problemi tipici della programmazione concorrente (corse, critiche, deadlock, starvation, ecc.).
Si ricorda che bisogna caricare i file .java, NON i .class, raggruppati in un unico file zip.

**Esercizio 3. Programmazione distribuita / RMI**

Si chiede di trasformare il programma sviluppato come soluzione dell'esercizio di programmazione concorrente in un sistema distribuito, in cui il server gestisce le risorse e i client operano come utilizzatori delle risorse.
Si realizzi il sistema mediante l'uso di RMI.
Come sempre, occorre evitare i problemi tipici della programmazione concorrente (corse, critiche, deadlock, starvation, ecc.).
Si ricorda che bisogna caricare i file .java, NON i .class, raggruppati in un unico file zip.