

**INFORMATICA INDUSTRIALE E SISTEMI OPERATIVI IDI**  
**Seconda prova scritta in itinere, 9 aprile 2004**

Nome: \_\_\_\_\_

Cognome: \_\_\_\_\_

Esercizio 1

Siano date 4 classi di risorse A, B, C e D, di cui esistono rispettivamente 3, 3, 4 e 2 istanze. Siano dati anche 3 processi P1, P2 e P3, le cui richieste massime in termini delle suddette risorse sono riportate nella tabella seguente:

	A	B	C	D
P1	0	3	2	1
P2	1	0	3	2
P3	3	2	1	0

Ad un certo istante di tempo, l'allocazione di risorse ai processi è quella di seguito indicata:

	A	B	C	D
P1	0	1	1	1
P2	0	0	2	1
P3	2	1	0	0

Applicando l'algoritmo del banchiere, si valuti se, nelle condizioni indicate,

- il sistema si trova in uno stato sicuro;
- la richiesta da parte di P2 di una richiesta di una risorsa di tipo C e di una risorsa di tipo D può essere soddisfatta;
- la richiesta da parte di P1 di una richiesta di una risorsa di tipo C può essere soddisfatta;
- la richiesta da parte di P1 di una richiesta di una risorsa di tipo B e di due risorse di tipo C può essere soddisfatta.

## Esercizio 2

Supponendo di non conoscere come la JVM gestisce la coda dei threads bloccati a seguito dell'invocazione del metodo `wait()` su un determinato oggetto, si sviluppi una classe Java chiamata `Semaforo` che realizza un semaforo con contatore, in cui i threads bloccati sono gestiti in modalità FIFO.

Per la realizzazione della coda FIFO si consiglia di utilizzare la classe `LinkedList` del package `java.util`, che dispone dei seguenti metodi:

`boolean addLast( Object o )`, che aggiunge un oggetto in coda alla lista,  
`Object getFirst()`, che restituisce l'oggetto che si trova in testa alla lista,  
`Object removeFirst()`, che l'oggetto che si trova in testa alla lista, e lo restituisce al chiamante,  
`int size()`, che restituisce il numero di oggetti contenuti nella lista.

Lo schema della classe da sviluppare potrebbe quindi essere il seguente:

```
public class Semaforo {
    /* costruttore, cui si passa il valore iniziale del contatore
    public Semaforo( int vic ) { /* ... */ }
    /* realizza l'operazione di WAIT sul semaforo
    public void attendi() { /* ... */ }
    /* realizza l'operazione di SIGNAL sul semaforo
    public void segnala() { /* ... */ }
    /* il contatore associato al semaforo
    protected int contatore;
    /* la coda per gestire i threads bloccati
    protected LinkedList coda;
}
```

Si completi la classe con le implementazioni del costruttore e dei metodi indicati, con gli eventuali attributi non specificati nello schema riportato sopra ma necessari al corretto funzionamento, e con le opportune soluzioni per garantire la corretta sincronizzazione tra i threads.

Suggerimento: ad ogni invocazione del metodo `segnala()` si sblocchino tutti gli eventuali threads bloccati; ciascuno di essi determini quindi se è abilitato o meno a procedere, ed in caso negativo si riblocchi.

Si ricorda infine che il riferimento al thread corrente può essere ottenuto mediante l'invocazione del metodo statico `currentThread()` della classe `Thread`, che restituisce un oggetto di tipo `Thread`.