SISTEMI OPERATIVI

(MODULO DI INFORMATICA II)

Sincronizzazione in Java (**Monitor in Java**)

Prof. Gherardi Luca

Prof.ssa Patrizia Scandurra (anni precedenti)

Università degli Studi di Bergamo

a.a. 2012-13

La gestione dei thread in Java

- Creazione e avvio
- Terminazione e cancellazione dei thread
- La gestione dei segnali
- **Sincronizzazione**
  - **Monitor**
    - **Esempio lettori-scrittori**
- Dati specifici dei thread
- Schedulazione
- Gruppi di thread

La sincronizzazione indiretta (o di mutua esclusione)

- Tramite **blocchi sincronizzati**:

```
Object mutexLock = new Object();
. . .
public void someMethod() {
    nonCriticalSection();

    synchronized(mutexLock) {
       criticalSection();
    }

    remainderSection();
}
```

- oppure tramite interi **metodi sincronizzati**

La sincronizzazione diretta (per i segnali!)

- Tramite il meccanismo **wait()/notify() all'interno di blocchi/metodi sincronizzati**:

```
Object mutexLock = new Object();
. . .
synchronized(mutexLock) {
    try {
                wait();
    }
    catch (InterruptedException ie) { }
}

synchronized(mutexLock) {
              notify();
}
```

Soluzioni esercizi precedenti

- Bank account
- Conto

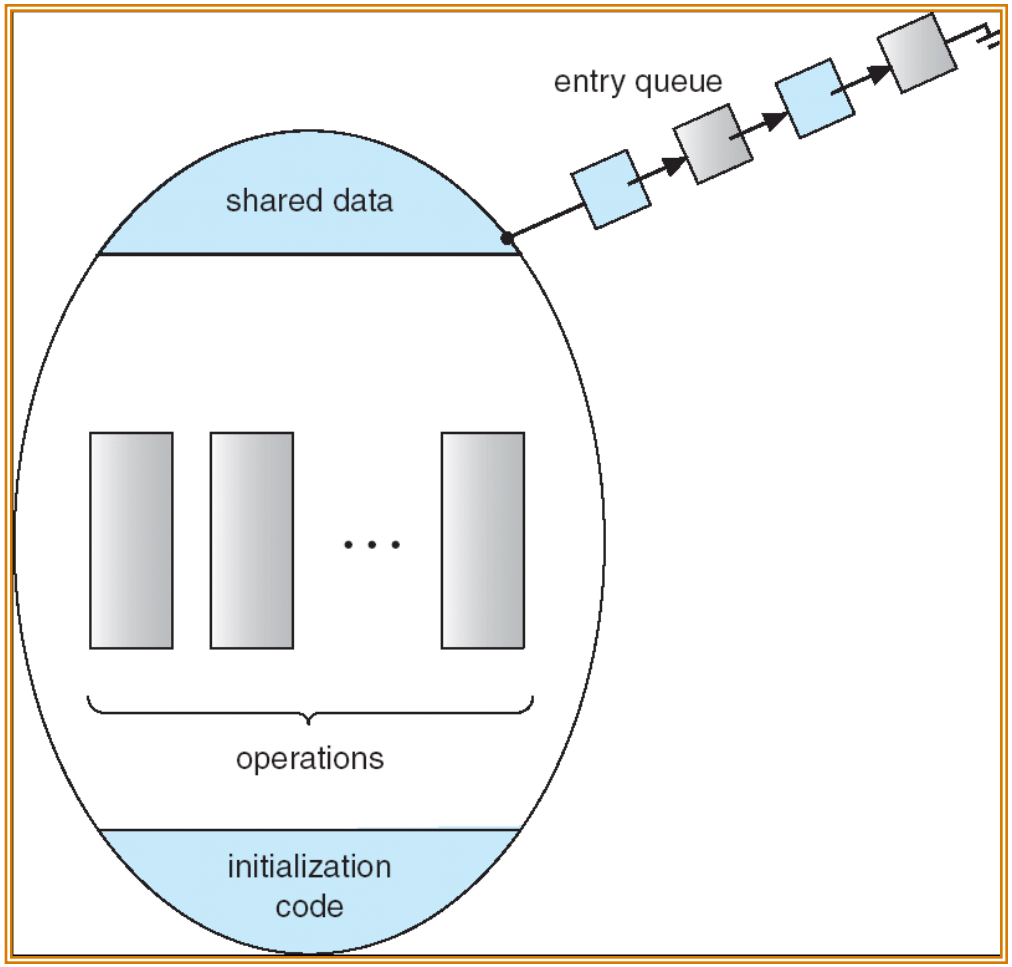
Osservazioni sulla sincronizzazione in Java

- I meccanismi di sincronizzazione visti finora sono il modo in cui Java permette di implementare la nozione di **monitor**
- Keyword **`synchronized`**
  - **Ogni oggetto ha un lock che funziona come monitor**
  - Un solo thread alla volta può essere attivo all'interno di un blocco/metodo **`synchronized`**
    - *Locking*
  - Eventuali altri thread rimangono in attesa (nel *entry set*) fin quando il thread attivo nel monitor "finisce"
    - Approccio *segnala-e-continua*
  - Il successivo thread ad entrare nel monitor è solitamente il thread a più alta priorità
  - Metodi costruttori non possono essere synchronized

Visione schematica dei monitor

- **MONITOR** è un costrutto linguistico

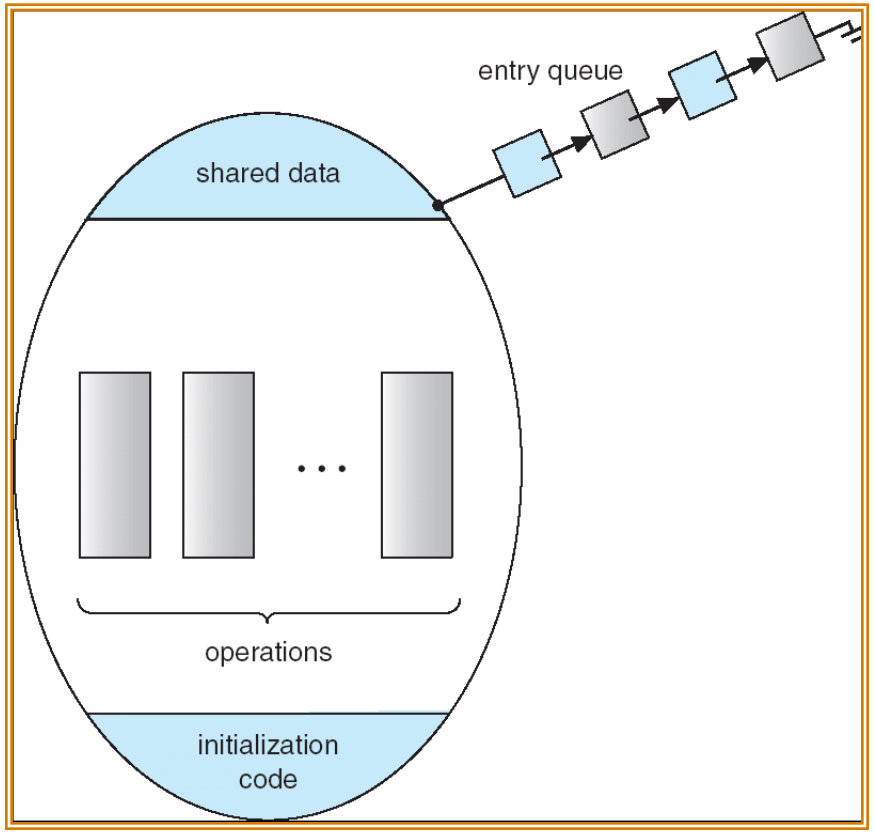
```
monitor monitor-name {
  // variable declarations
     public entry p1(...) {
       ...
     }
     public entry p2(...) {
       ...
     }
}
```



Visione schematica dei monitor

- **MONITOR** è un costrutto linguistico
- **in Java:** classi con metodi sincronizzati

```
class monitor-name {
  // variable declarations
     synchronized public
     entry p1(…) {
       …
     }
     synchronized public
     entry p2(…) {
       …
     }
}
```



Il problema dei lettori-scrittori

- Una base di dati è condivisa fra vari thread concorrenti
  - **Lettori**: accedono solo in lettura; non modificano la base di dati
  - **Scrittori**: possono sia leggere che scrivere

- **Problema:**
  - **permettere a lettori multipli** di leggere contemporaneamente (non crea interferenze)
  - **permettere ad un solo scrittore alla volta** di accedere alla base di dati (accesso esclusivo)

- Diverse varianti…
  - Variante 1 (*priorità ai lettori*): nessun lettore deve essere tenuto in attesa, a meno che uno scrittore abbia già ottenuto l'accesso alla base di dati condivisa
    - È **soggetta però al problema della starvation dei scrittori**

Il problema dei lettori-scrittori (variante 1) – soluzione tramite monitor

- **L'applicazione è organizzata:**
  - Un'interfaccia **RWLock** per le operazioni di read-write con lock che invocheranno i thread lettori e scrittori
  - Una classe **Database** che implementa RWLock

- Vedi contenuto dell'archivio **readwrite**