Gestione Concorrente Lettori-Scrittori in Java

Giordano Iaquinta e Guido Testi

November 27, 2025

1 Introduzione

Il presente lavoro affronta il problema della gestione concorrente di una risorsa condivisa, ispirato al classico problema dei Lettori-Scrittori. La risorsa condivisa è rappresentata da una lavagna su cui più persone possono leggere o scrivere parole.

La scelta di non implementare classi separate Lettore e Scrittore nasce dalla volontà di rendere il modello più vicino a un caso reale, in cui ciascuna persona può leggere e, occasionalmente, scrivere, secondo un turno casuale.

2 Obiettivo

L'obiettivo è gestire correttamente l'accesso concorrente alla lavagna, garantendo che:

- Tutti i thread leggano la parola corrente prima che qualcun altro possa scrivere.
- Solo il thread di turno scriva una nuova parola.
- L'ultimo giro della lezione permetta a tutti di leggere l'ultima parola.
- Si evitino race condition o conflitti logici.

3 Classi principali

La soluzione implementa tre classi principali:

3.1 Applicazione

Classe principale che:

- Gestisce l'input utente (numero di persone, eventuali nomi).
- Crea la lavagna condivisa e un thread per ciascuna persona.
- Imposta la durata della lezione e avvia i thread.

3.2 Persona

Rappresenta un thread che:

- Controlla se è il suo turno di scrivere.
- Legge la parola corrente se non l'ha già letta.
- Scrive una parola quando è il suo turno.
- Continua fino alla fine della lezione, rispettando l'ultimo giro.

3.3 Lavagna

Gestisce la risorsa condivisa:

- Tiene traccia di chi ha letto (`haLetto`) e di quanti lettori hanno completato la lettura (`letti`).
- Memorizza la parola corrente (`valore`) e il turno corrente (`turnoCorrente`).
- Coordina l'ultimo giro (`ultimoGiro`) e la fine della lezione (`lezioneFinita`).
- Metodi principali:
  - `leggi(nome, numeroElenco)`
  - `scrivi(nome)`
- Usa `synchronized`, `wait()` e `notifyAll()` per garantire accesso corretto.

4 Gestione dell'ultimo giro e della fine lezione

La logica dell'ultimo giro e della fine della lezione è gestita tramite due variabili:

- `lezioneFinita`: impostata quando il tempo della lezione è scaduto. Serve come segnale per fermare la creazione di nuove parole.
- `ultimoGiro`: attivata quando tutti i thread devono leggere l'ultima parola scritta prima di terminare. Garantisce che:
  - Tutti i thread leggano la parola finale.
  - Nessun thread esca prima di completare la lettura dell'ultimo valore.

In pratica:

1. Il thread che nota che il tempo è scaduto imposta `lezioneFinita = true` e `ultimoGiro = true`, quindi chiama `notifyAll()`.
2. Tutti i thread eseguono ancora il metodo `leggi()` fino a leggere l'ultima parola.
3. Dopo aver letto l'ultimo valore, ciascun thread termina e stampa il messaggio di uscita.

Questo meccanismo evita che alcuni thread terminino senza leggere l'ultima parola e mantiene la coerenza dell'output.

5 Motivazioni della sincronizzazione

- **Metodo leggi() sincronizzato:** necessario per controllare correttamente le variabili `haLetto` e `letti` e evitare stampe multiple.
- **NotifyAll:** consente di svegliare tutti i thread quando lo stato della lavagna cambia (nuova parola o ultimo giro).

6 Flusso della soluzione

1. Avvio: creazione dei thread, lavagna condivisa, impostazione della durata della lezione.
2. Lettura: ogni thread legge la parola corrente se non l'ha già letta.

3. Scrittura: il thread di turno scrive una nuova parola, resetta le letture e notifica tutti.
4. Ultimo giro: tutti leggono l'ultima parola.
5. Fine: tutti i thread terminano e viene stampata la campanella di fine lezione.

7 Vantaggi e svantaggi

7.1 Vantaggi

- Corretta gestione della concorrenza tra thread.
- Tutti i lettori leggono la parola corrente prima di una nuova scrittura.
- Ultimo giro garantisce lettura finale completa.
- Modello realistico, vicino a un esempio concreto di lezione.

7.2 Svantaggi

- Lettura seriale, non parallela.
- Ordine di lettura variabile ad ogni esecuzione.
- La sincronizzazione può causare attese dei thread.
- Non è una simulazione astratta del problema Lettori-Scrittori, ma un esempio pratico.

8 Esempio di output

```
Il professore chiama alla lavagna: Persona2
Persona2 scrive: muro

Persona1 legge: muro
Persona3 legge: muro
Persona2 legge: muro
...
Persona1 smette di seguire la lezione
Persona2 smette di seguire la lezione
...
>>> CAMPANELLA: fine lezione <<<
```

9 Conclusioni

Questa soluzione mostra come gestire l'accesso concorrente a una risorsa condivisa in Java, usando thread, sincronizzazione e notifiche. La scelta di non implementare classi Lettore e Scrittore separate permette di avere un esempio realistico di lezione, più vicino a un caso concreto, pur rispettando le regole di lettura e scrittura e risolvendo il problema.