Università di Ferrara
Laurea Triennale in Informatica
A.A. 2021-2022
Sistemi Operativi e Laboratorio

Lab-08. I Thread in Java

Prof. Carlo Giannelli

http://www.unife.it/scienze/informatica/insegnamenti/sistemi-operativi-laboratorio
http://docente.unife.it/carlo.giannelli
https://ds.unife.it/people/carlo.giannelli

Esercizio 3: terminazione thread

Si implementi un programma Java che crea N thread (dove N è passato come argomento) e stampa a video i loro id. Ciascun thread esegue in un ciclo la stampa del proprio id, poi attende un secondo e verifica se ne è stata richiesta la terminazione.

Il main thread deve mettersi in attesa sullo standard input (stdin) per ricevere dall'utente l'id del Thread da terminare. Ricevuto tale id, il main thread si occuperà di terminare il Thread con id corrispondente.

Si implementi la terminazione del thread utilizzando:

1) soluzione con verifica periodica
2) soluzione con risveglio del thread, con chiamata a interrupt() su thread dalla classe del thread stesso (si veda https://www.baeldung.com/java-thread-stop)

Esercizio 3: traccia

- Definire una classe WorkerThread che contiene nel metodo run() il la stampa richiesta nella consegna. La classe WorkerThread deve anche esporre un metodo stop() per gestire in modo corretto la terminazione del Thread stesso.
  - Per implementare il metodo stop() si faccia riferimento agli esempi visti a lezione.
- Definire una classe pubblica e il rispettivo metodo main in cui vanno creati e lanciati N WorkerThread.
- Creare un ciclo in cui viene chiesto all'utente di inserire l'id del thread da terminare. Ricevuto questo id, il main thread invoca sul thread corrispondente il metodo stop().

- Per semplicità si supponga che l'id del Thread sia un numero compreso tra 0 e N-1.
- Suggerimento: il main deve tener traccia dei thread che ha già terminato in un opportuno array di Boolean.

Esercizio 4: accesso a risorse condivise

Si realizzi un'applicazione Java multithread e **thread-safe** che gestisca la modifica di oggetti all'interno di un magazzino, considerando che più operatori possono interagire in modo concorrente col sistema. Ad esempio, un operatore può creare un nuovo tipo di bullone, aggiungere una certa quantità e poi eliminarne un'altra quantità.

La soluzione deve essere costituita da:

- una classe **Magazzino** che permetta di: 1) verificare se esiste un tipo di oggetto, 2) creare un nuovo tipo di oggetto (con quantità iniziale 0), 3) aggiungere/rimuovere quantità a tale oggetto. Ciascuna azione richiede un tempo di 500 ms.
- una classe **Operatore** che esegua in sequenza alcune attività, ad esempio verifica esistenza "bulloni", creazione se non esiste, aggiunta di 1000 pezzi, eliminazione di 500 pezzi.
- una classe con metodo **main** che crei N classi Operatore e le esegua in modo concorrente. Al termine delle operazioni viene stampata la quantità di oggetti presenti in magazzino.

Si confronti il risultato ottenuto con una soluzione non thread-safe.

Esercizio 4: traccia

- Definire una classe **Magazzino** che espone i metodi richiesti dalla consegna. Per semplicità si utilizzi un'istanza di *Map<String, Integer>* per memorizzare gli oggetti e le rispettive quantità nel magazzino. Si ridefinisca il metodo toString() per permettere di visualizzare la quantità degli oggetti presenti in magazzino.
- Definire una classe **Operatore** che implementa la sequenza di istruzioni: verifica bulloni, inserisci 1000 bulloni, rimuovi 500 bulloni.
- Definire il metodo **main** che crea e manda in esecuzione N thread Operatore e poi ne attende la terminazione.

- Al termine dell'operazione stampare a video gli oggetti presenti nel magazzino a video e verificare la correttezza del risultato.
- Risultato corretto? Perché? Gestione della concorrenza?

- Opzionale: si provi ad aggiungere delle pause random tra le operazioni dell'operatore