Esercitazione su thread in Java

E' richiesta la realizzazione di una libreria di classi per la gestione delle ordinazioni e della preparazione dei piatti in un piccolo ristorante.

In particolare, la libreria deve permettere di gestire la registrazione da parte di uno o più camerieri del dettaglio delle ordinazioni, che riguardano il tipo di piatto ordinato, la quantità e il tavolo a cui dovrà essere servito. Le ordinazioni vengono prese in carico da un cuoco che si occupa della preparazione dei piatti. Inoltre, la libreria prevede un sistema di backup su file delle ordinazioni a intervalli di tempo regolari.

Una singola ordinazione è gestita con un oggetto di tipo **Ordinazione**. La classe Ordinazione dovrà avere la seguente interfaccia:

Ordinazione
+ Ordinazione(String piatto, int tavolo, int quantita) + setPiatto(String piatto) : void + getPiatto() : String + setTavolo(int tavolo) : void + getTavolo () : int + setQuantita(int quantita) : void + getQuantita() : int + toString() : String

Il metodo toString() dovrà restituire una stringa nel seguente formato: "Piatto: [piatto], Tavolo: [tavolo], Quantità: [quantita]", ad esempio "Piatto: Pasta al pomodoro, Tavolo: 1, Quantità: 3".

La gestione delle ordinazioni deve essere fatta tramite la classe **Comande** che ha la seguente interfaccia.

Comande
+ Comande(String filename, Boolean leggiBackup) + <<synchronized>> aggiungiOrdinazione(Ordinazione ordinazione) : void + <<synchronized>> consegnaOrdinazione() : Ordinazione + <<synchronized>> salvaOrdinazioni() : void

Le ordinazioni sono gestite secondo una politica First In First Out (FIFO), quindi verranno servite dal cuoco del ristorante nello stesso ordine in cui vengono create.
Il costruttore della classe Comande riceve due parametri. Il primo (String) indica il nome del file da utilizzare per salvare e/o caricare il backup. Il secondo (boolean), se true, indica che l'insieme iniziale di ordinazioni deve essere letto da file, altrimenti deve essere creata una lista vuota.
Il metodo *aggiungiOrdinazione* dovrà gestire l'aggiunta dell'ordinazione alla coda delle ordinazioni.
Il metodo *consegnaOrdinazione* dovrà gestire l'accesso alla coda di ordinazioni e restituire la prossima Ordinazione da servire, se c'è almeno una Ordinazione in coda. Se non ci sono ordinazioni in coda, il metodo deve rilasciare il mutex e restare in attesa che venga aggiunta un'ordinazione alla coda.

Il metodo *salvaOrdinazioni* si occupa di salvare l'attuale insieme di ordinazioni su file attraverso il sistema di serializzazione degli oggetti di Java. Dovranno essere implementate opportunamente le interfacce per la gestione del meccanismo di serializzazione.
Il salvataggio avviene solo se ci sono ordinazioni in coda altrimenti si deve restare in attesa che venga aggiunta un'ordinazione alla coda.

La classe **Cameriere**, che implementa l'interfaccia *Runnable*, ha accesso all'oggetto Comande che gli viene passato come argomento del costruttore. Cameriere si occupa della creazione delle ordinazioni. Ogni cameriere ha un nome che viene passato come parametro al costruttore.

Cameriere *implements* Runnable
+ Cameriere(String nome, Comande comande) + run()

Nel metodo run(), la classe Cameriere svolge le seguenti operazioni:

- Per semplicità di svolgimento non è richiesta l'interazione con l'utente ma la creazione automatica di un'Ordinazione generando randomicamente il numero del tavolo (intero compreso tra 1 e 5) e la quantità (intero compreso tra 1 e 4) e selezionando un piatto dal menu utilizzando il metodo *getPiatto* della classe **Menu** allegata alla traccia.
- attesa di un tempo random compreso tra 5 e 10 secondi
- aggiunta dell'ordinazione alla lista
- stampa a video del messaggio "Ordinazione presa da [nome cameriere]: [dettaglio ordinazione attraverso il metodo toString della classe ordinazione]", ad esempio "Ordinazione presa da Pippo: Piatto: Tiramisu – Tavolo: 1 – Quantità: 2".

La classe **Cuoco**, che implementa l'interfaccia *Runnable*, ha accesso all'oggetto Comande che gli viene passato come argomento del costruttore.

Cuoco *implements* Runnable
+ Cuoco(Comande comande) + run()

Nel metodo run(), la classe Cuoco svolge le seguenti operazioni, ciclicamente:

- preleva la prossima Ordinazione invocando il metodo *consegnaOrdinazioni* dell'oggetto Comande
- attende un tempo random compreso tra 5 e 10 secondi
- stampa a video del messaggio "Piatto pronto: [dettaglio ordinazione attraverso il metodo toString della classe ordinazione]"

Per la gestione del backup deve essere creata un'ulteriore classe **BackupAutomatico** che implementa l'interfaccia *Runnable* e che ha accesso all'oggetto Comande che gli viene passato come argomento del costruttore.

BackupAutomatico *implements* Runnable
+ BackupAutomatico(Comande comande) + run()

Nel metodo run(), la classe BackupAutomatico svolge le seguenti operazioni, ciclicamente:

- Richiedere il salvataggio della coda di ordinazioni su file attraverso il metodo *salvaOrdinazioni* della classe Comande.
- Visualizzare a video il messaggio “BACKUP EFFETTUATO”
- Attendere 20 secondi prima di effettuare un nuovo salvataggio.

E’ fornita una classe TestRistorante, contenente il main dell’applicazione, che dovrà essere utilizzata per testare il programma sviluppato. E’ fornita una classe Menu per la selezione (randomica) del piatto da aggiungere alle ordinazioni.