L'obbiettivo di questa serie d'esercizi è di permettervi di familiarizzare con l'utilizzo di wait/notify, delle Conditions e dei Synchronizers.

Esercizio 1

Copiate il codice sorgente 'S8Esercizio1.java' e analizzatelo nel dettaglio. Il programma esegue semplici operazioni di calcolo su una matrice. L'esecuzione del programma è di tipo sequenziale. L'obiettivo dell'esercizio è quello di modificare il programma per renderlo multi-thread, in modo che le operazioni di calcolo vengano eseguite concorrentemente. Il thread principale (quello di main()) dovrà inizializzare la matrice con numeri random e poi lanciare gli altri threads. Ogni nuovo thread dovrà occuparsi di calcolare la somma dei valori di una riga della matrice (in questo momento quest'operazione viene fatta da un'iterazione del for loop) e poi aspettare che gli altri threads abbiano finito. Il thread main dovrà quindi calcolare la somma di tutte le somme e visualizzarla. Nel frattempo gli altri threads potranno cominciare a calcolare la somma dei valori di una colonna ciascuno. Anche in questo caso, dovranno aspettare che tutti abbiano terminato e, successivamente, il thread principale potrà visualizzare la somma delle somme delle colonne.

Implementate tre versioni del programma utilizzando: wait/notify, Conditions e Synchronizers.

Esercizio 2

Riprendete l'esercizio 3 della serie precedente (Corrispondenza tra amici) e provate ora ad implementarla usando le code di Java per simulare il passaggio delle lettere. Usate la prima volta una ConcurrentLinkedQueue ed in seguito una LinkedBlockingQueue. In che modo vi si semplifica il codice nel caso della BlockingQueue.

Esercizio 3

Scrivete un programma che simuli la corsa a staffetta tra 4 squadre. Ogni squadra è composta da 10 corridori che si passano a turno il testimone. Prima dell'inizio della gara il primo corridore di ogni squadra si mette in attesa del segnale di partenza, mentre tutti gli altri corridori restano in attesa del testimone. Al segnale di partenza, il corridore con il testimone inizia a correre, passa il testimone al corridore successivo e termina la propria corsa. Il primo corridore che raggiunge il traguardo scrive a schermo "*Vittoria!!!*", mentre gli atleti seguenti scrivono "*Perso.*". Ogni corridore stampa a schermo il tempo impiegato per il suo tratto di corsa, a chi passa il testimone e quando riceve il testimone. Si stampi infine il tempo totale impiegato da ogni squadra. Usate il metodo Thread.sleep() con valore casuale tra 100 ms – 150 ms per simulare la corsa di un corridore e create 1 thread per ogni corridore.

Potete risolvere il problema utilizzando le Conditions oppure i Synchronizers di Java.

Esempio di un possibile output del programma:

```
Corridore0_Squadra0: in attesa del segnale di partenza!
Corridore0_Squadra1: in attesa del segnale di partenza!
Corridore8_Squadra1: in attesa del testimone!
Corridore6_Squadra0: in attesa del testimone!
...
Pronti... Partenza... Via!
...
Corridore0_Squadra3: corro per 141 ms
Corridore0_Squadra1: corro per 114 ms
...
Corridore0_Squadra0: passo testimone t0 a Corridore1_Squadra0
```

```
Corridore1_Squadra0: testimone ricevuto!
Corridore1_Squadra0: corro per 127 ms
Corridore0_Squadra2: passo testimone t2 a Corridore1_Squadra2
Corridore1_Squadra2: testimone ricevuto!
Corridore1_Squadra2: corro per 132 ms
...
Corridore9_Squadra0: VITTORIA!!!!
Corridore9_Squadra2: ...perso...
Corridore9_Squadra3: ...perso...
-----------------------------------
Classifica finale:
Squadra0 elapsedTime: 1196 ms
Squadra2 elapsedTime: 1228 ms
Squadra3 elapsedTime: 1292 ms
Squadra1 elapsedTime: 1306 ms
```

Esercizio 4

Sviluppate un programma che simuli la partenza di 10 fantini al palio di Siena. Tutti i fantini arrivano in tempi diversi (da generare a random tra i 1000 e 1050 ms) alla linea di partenza e devono aspettare l'arrivo dell'ultimo fantino. Quando anche l'ultimo fantino arriva alla linea di partenza la gara può cominciare.

Il programma deve misurare e stampare a schermo il tempo di attesa di ogni fantino.

Sviluppatene due versioni: la prima che faccia uso degli explicit locks in combinazione delle conditions, la seconda che usi i synchronizers di java.

Esempio di un possibile output del programma:

```
Fantino1: arrivo alla linea di partenza
Fantino3: arrivo alla linea di partenza
Fantino0: arrivo alla linea di partenza
Fantino9: arrivo alla linea di partenza
Fantino2: arrivo alla linea di partenza
Fantino6: arrivo alla linea di partenza
Fantino7: arrivo alla linea di partenza
Fantino8: arrivo alla linea di partenza
Fantino5: arrivo alla linea di partenza
Fantino4: arrivo alla linea di partenza
Fantino4: ha atteso 0 ms
Fantino1: ha atteso 57 ms
Fantino0: ha atteso 43 ms
Fantino9: ha atteso 39 ms
Fantino6: ha atteso 39 ms
Fantino3: ha atteso 55 ms
Fantino5: ha atteso 26 ms
Fantino8: ha atteso 35 ms
Fantino7: ha atteso 38 ms
Fantino2: ha atteso 39 ms
```