Esercizio 1

Versione 1 – Wait/Notify

I thread Sommatore eseguono la somma della riga a loro assegnata, aggiungono il risultato ad un array condiviso, incrementano il contatore dei thread che hanno terminato la somma e, utilizzando il metodo notify(), svegliano il main-thread: quest'ultimo controlla il contatore e verificare la condizione del while: se vera, significa che i thread non hanno ancora terminato tutte le somme e si rimetterà in attesa con wait(), altrimenti proseguirà ed eseguirà la somma totale delle righe. Nel frattempo, i thread Sommatore avranno iniziato la somma delle colonne. Non sarà però necessario ricorrere di nuovo ai metodi wait() e notify(): il main-thread eseguirà un join e rimarrà in attesa della fine dei sommatori per poi eseguire la somma delle colonne e concludere.

Versione 2 – Conditions

L'utilizzo delle conditions cambia di poco l'implementazione rispetto alla precedente. Troviamo la sostituzione dei blocchi synchronized con l'utilizzo di ReentrantLock e l'utilizzo dei metodi offerti dalla condition (await() e signal()) al posto di quelly di wait() e notify() offerti dai singoli oggetti. Come nel caso precedente, l'utilizzo degli oggetti per garantire la sincronizzazione è necessario solamente per la somma delle righe.

Versione 3 – Synchronizer

L'utilizzo dei Synchronizer (in questo caso di un Phaser) cambia invece alcune parti dell'implementazione: intanto non è più necessario l'utilizzo di blocchi synchronized o di lock espliciti e cambia anche l'utilizzo dell'oggetto che permette la coordinazione, in quanto il Phaser prevede la registrazione dei thread che partecipano alle operazioni di coordinazione/schedulazione. In particolare, è necessario registrare un thread quando esso viene creato e deregistrarlo quando esso finisce le proprie attività. Anche in questo caso, come nei precedenti, le operazioni di coordinazione sono necessarie solamente per la somma delle righe.

Esercizio 2

Versione 1 – ConcurrentLinkedQueue

In questa versione, rispetto alla precedente, si fa utilizzo una ConcurrentLinkedQueue al posto di un CopyOnWriteArrayList: con la coda è possibile unire due metodi (per il recupero e la rimozione di una lettera) nell'unico poll().

Versione 2 – LinkedBlockingQueue

Rispetto alla prima versione che utilizza le code, in questo caso al posto di poll() si utilizza take(): il metodo take(), a differenza di poll(), nel caso in cui trovi la coda vuota, rimane in attesa fino a quando non viene inserito un elemento, che provvederà subito a ritornare e a cancellare dalla coda. In questo modo, è possibile eliminare dal metodo run() la condizione che si occupa di controllare se la coda è vuota.

```
if (isCasellaVuota())
        continue;
System.out.println("Utente " + id + " ricevuto messaggio " + casella_posta.poll());

System.out.println("Utente " + id + " ricevuto messaggio " + casella_posta.take());
```