Esercizio 1

Il programma permette la lettura e la modifica di una HashMap condivisa tra più thread. In particolare, le operazioni di modifica di una HashMap (ad esempio, put() e remove()) potrebbero sollevare ConcurrentModificationException quando proviamo ad eseguire la modifica.

Per questo è possibile procedere per risolvere il problema in due modi diversi:

* Utilizzare blocchi synchronized dove avviene una qualsiasi operazione sulla mappa (lettura o scrittura) utilizzando la tecnica del client side locking, sfruttando la mappa stessa come argomento di synchronized
* Utilizzare ConcurrentHashMap, una HashMap creata appositamente per supportare l’accesso concorrente di più thread.

Tra le due soluzioni, è preferibile la seconda: la prima infatti, acquisendo synchronized su tutte le operazioni (anche quelle di scorrimento), fa calare drasticamente le performance, rendendo il programma quasi seriale e non sfruttando adeguatamente le possibilità offerte dal multi-threading.

Esercizio 2

Il programma soffre di problemi di concurrent modification: ci sono infatti più thread che richiedono l’accesso ad una lista condivisa tra di loro da poter scorrere attraverso un iteratore, mentre il main-thread aggiunge nuovi elementi alla lista.

È possibile implementare 3 tipi di soluzione:

* Utilizzare blocchi synchronized, con la tecnica del client side locking (come prima)
* Utilizzare la synchronized collection (wrapper intorno alle normali collection che impediscono alcune operazioni sulla collection che potrebbero causare concurrent modification)
* Utilizzare CopyOnWriteArrayList, collection sviluppata per supportare operazioni concorrenti effettuate da più thread.

Per comprendere la differenza di performance tra le 3 versioni, bisogno osservare il numero di comparazioni presenti nei log scritti dal programma: con i blocchi Synchronized abbiamo un numero di confronti molto bassi (circa 65000), sintomo di scarsa efficienza del programma. Il numero di confronti sala utilizzando le SynchronizedCollection, mentre con l’utilizzo di CopyOnWriteArrayList i confronti vengono sestuplicati rispetto alle SynchronizedCollection (circa 600\_000), indicando che i thread hanno spesso trovato l’oggetto condiviso utilizzabile senza incappare in una ConcurrentModification.

Esercizio 3

Versione 1

Per rendere thread-safe il programma con l’utilizzo di un comune ArrayList, è necessario gestire l’accesso alla collection mediante l’utilizzo di metodi synchronized, che regolano l’accesso alla casella di posta (oggetto condiviso da ogni Amico con gli altri) per le operazioni di aggiunta di nuova posta, controllo della presenza di posta e rimozione delle vecchie lettere.

Utilizzando invece un CopyOnWriteArrayList, la classe Amico è sollevata dalla gestione dell’accesso concorrente alla casella di posta, essendo CopyOnWriteArrayList progettata per questo.