Università degli Studi di Napoli Federico II
Esame di Advanced Computer Programming

Proff. De Simone, Della Corte

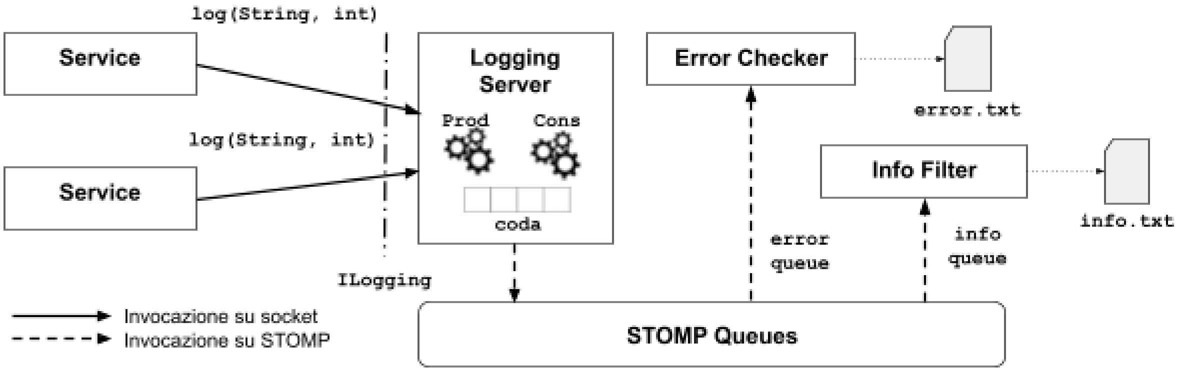
Prova pratica del giorno 04/07/2023
Durata della prova: 120 minuti

°°°

Lo studente legga attentamente il testo e produca il programma ed i casi di test necessari per dimostrarne il funzionamento. La mancata compilazione dell'elaborato, la compilazione con errori o l'esecuzione errata daranno luogo alla valutazione come ***prova non superata****.*
Al termine della prova lo studente dovrà far verificare il funzionamento del programma ad un membro della Commissione.

°°°

Testo della prova



Il candidato implementi un sistema distribuito in **Python** per la gestione di eventi di log basato su **Socket** e **STOMP**. Il sistema è caratterizzato dai seguenti componenti.

Service. E' un client che genera le entry di log da inviare al **Logging Server**. L'invio di una entry di log consiste nella invocazione del metodo *void log(String, int)* specificato nell'interfaccia **ILogging**. La richiesta è caratterizzata da 1) **messaggioLog** (*String*), ossia il messaggio della entry di log, 2) **tipo** (*int*), ossia la tipologia di entry di log (0 = DEBUG, 1 = INFO, 2 = ERROR). Il Client genera 10 entry di log, invocando il metodo *log* per ogni entry (attendendo 1 secondo tra le invocazioni). Per ciascuna entry, *tipo* è generato in maniera casuale scegliendo un interno tra 0 e 3 (estremi inclusi), mentre *messaggioLog* è generato in maniera casuale scegliendo tra *success* o *checking* se il tipo è 0 o 1, e tra *fatal* o *exception* se il tipo è pari a 2.

Logging Server. Fornisce l'interfaccia *ILogging* e il relativo metodo *void log(String, int)*. Il metodo *log* avvia un processo produttore, il quale inserisce in una coda (process-safe e che implementi il problema del produttore/consumatore) una stringa che concatena sia la stringa del parametro *messaggioLog* che l'intero del parametro *tipo* (ad es., `fatal-2`). I dati inseriti dalla coda, sono consumati da un processo consumatore avviato al lancio del Logging Server. Quando un nuovo dato è disponibile nella coda, il processo consumatore, preleva la stringa e la inserisce in un messaggio STOMP. Il messaggio viene scritto nella **STOMP Queue *error*** se il messaggio contiene il tipo pari a 2, nella **STOMP Queue *info*** negli altri casi.

Error Checker. Implementa la ricezione sulla STOMP Queue *error* e prevede come parametro di avvio (da terminale) una stringa tra *fatal* o *exception*. Alla ricezione di ciascun messaggio, il listener STOMP di *Error Checker* estrae il contenuto del messaggio, verifica se esso contiene la stringa ricevuta in input e, in caso affermativo, scrive su file (*error.txt*) e stampa a video il messaggio appena ricevuto.

Info Filter. Implementa la ricezione sulla STOMP Queue *info*. Alla ricezione di ciascun messaggio, il listener STOMP di *Info Filter* estrae il contenuto del messaggio, se esso contiene il valore 1, allora scrive il contenuto del messaggio sul file *info.txt* (oltre che visualizzarla a video).

Il candidato utilizzi proxy-skeleton con socket TCP per la comunicazione tra Service e Logging Server e Queue STOMP per quella tra Logging Server ed Error Checker/Info Filter. A tal fine, il candidato predisponga le opportune interfacce e le classi Proxy-Skeleton. Si utilizzi inoltre skeleton per ereditarietà per il Logging Server.