DietroLeQuinte

Specifiche v1.1.1 del progetto di Metodologie di Programmazione 2015

Corso di Laurea Triennale in Informatica

Prof. Roberto Navigli, Dott. Igor Melatti

Il progetto consiste nello sviluppo di un piccolo sistema di Log Analyzing (LA) chiamato DietroLeQuinte. Esempi avanzati di tali sistemi sono confrontati in questa pagina.
Il sistema da progettare è, ovviamente, più semplice e consiste nei seguenti moduli:

1. sistema di lettura dei file in input e caricamento delle informazioni in memoria (Loader);
2. sistema di interrogazione (Searcher);
3. interfaccia da riga di comando, che accetti comandi predefiniti ed è quindi, dal punto di vista dell'utente finale, il sistema di LA vero e proprio (DietroLeQuinte).

Il sistema dovrà essere in grado di analizzare i seguenti 2 tipi di file di log, che saranno descritti nel seguito:

- log di chat del tipo IRC
- log delle interrogazioni sottomesse ai motori di ricerca AOL

Obbligatorio per gruppi da 2 studenti: il sistema dovrà essere progettato in modo da prevedere l'estensibilità (mediante reflection) a qualsiasi altro tipo/formato di log.

Download e formato dei dati in input

Due file ZIP, AOL.zip e IRC.zip, sono disponibili da http://robertonavigli.com/metodologie2015/progetto_metodologie2015.zip. La prima directory, AOL (risultante dalla decompressione di AOL.zip), contiene le interrogazioni su AOL, mentre la seconda directory, IRC (risultante dalla decompressione di IRC.zip), contiene i log della chat. Tutti i file contenuti in queste directory sono di testo (estensione .txt o .log), oppure sono file di testo compressi (doppia estensione .log.gz o .txt.gz). Ogni nome di file contiene come suo prefisso la tipologia del log in esso contenuto. Ad esempio, `query.collection.txt` specifica con il prefisso "query." che si tratta di un log di interrogazioni. Nel seguito dettagliamo il formato di tali file di testo; nel caso di file compressi, si tratta ovviamente del file di testo risultante dalla decompressione.

Formato dei log AOL (interrogazioni)

AOL (America on Line) gestisce un motore di ricerca, come Google o Yahoo!. I log di AOL.zip si riferiscono alle ricerche effettuate da utenti presi a caso e anonimizzati.

Contenuto dei file: Il formato è auto-esplicativo: si tratta di colonne separate dal carattere TAB, in cui la prima riga è di intestazione (spiega succintamente il significato delle colonne) e le righe successive alla prima contengono le seguenti informazioni:

- l'ID numerico di un utente (prima colonna),
- l'interrogazione fatta dall'utente (seconda colonna),
- la data ed l'ora al secondo dell'interrogazione (terza colonna),
- il link sul quale l'utente ha cliccato, tra quelli mostrati da AOL in risposta all'interrogazione (quinta colonna) e la posizione occupata da tale link tra i link mostrati in risposta all'interrogazione (quarta colonna). Quarta e quinta colonna possono essere vuote, se l'utente non ha cliccato nulla.

Formato del nome di file: I nome dei file di tipo interrogazione iniziano con il prefisso *query* (ad es. `query.collection.txt`). Si noti che, per rendere il progetto generale, è necessario poter gestire con classi differenti formati differenti di log di interrogazioni.

Formato dei log IRC (chat)

Una piccola precisazione per iniziare: le IRC (Internet Relay Chat) erano il sistema di chat più popolare agli inizi di Internet e consistono in "camere" virtuali separate in cui si riuniscono un certo numero di utenti (0, 1 o più utenti) per discutere di un qualche tema. Gli utenti di una camera, oltre che ad inviare messaggi e poter vedere chi entra/esce, possono scrivere delle azioni come messaggi "speciali" (ad esempio, se l'utente N scrive "/va a farsi una doccia", allora viene visualizzato "N va a farsi una doccia"). In IRC.zip vengono forniti i log di una sola di queste camere, denominata "evergreen". Tuttavia il progetto delle classi deve funzionare su un nomi di camere arbitrari (sulla base del prefisso del nome del file log prima del primo carattere '.').

Contenuto dei file: Il formato dei log delle chat IRC è un po' meno ovvio del precedente, dal momento che ci sono due diversi tipi di righe: righe che identificano login/logout (entrata in evergreen/uscita da evergreen o da IRC) e righe che descrivono una riga di commento immessa da un utente (alcune potrebbero essere immesse automaticamente). Differentemente da AOL, non esiste una riga di intestazione.
Per entrambi i tipi di righe del log IRC, la prima colonna contiene data ed ora dell'evento descritto dalla riga stessa (**attenzione:** il formato è diverso da quello usato per la data di AOL). Dopodiché si possono verificare i seguenti casi (la separazione dalla prima colonna è costituita da 1 o 2 spazi):

1. `*** U has quit IRC`
2. `-!- U [X] has quit [C]`
3. `*** U has joined #NOME_CAMERA`
4. `-!- U [X] has joined #NOME_CAMERA`

```
5. *** U has left #NOME_CAMERA
6. * U A
7. <U> M
```

Nel primo, nel secondo e nel quinto caso, si tratta di un logout fatto dall'utente U (gli utenti sono identificati da nickname), nel terzo e nel quarto caso si registra un login dell'utente U, nel sesto caso si tratta della descrizione dell'azione A da parte dell'utente U e nel settimo caso si tratta dell'invio del messaggio M da parte dell'utente U. Notare che:

1. nel settimo caso, il nickname dell'utente è racchiuso da <>;
2. nel settimo caso, il messaggio è terminato dalla fine della riga (quindi non si possono inviare messaggi contenti il carattere di a capo...);
3. nel secondo caso, X e C contengono l'indirizzo (solitamente un IP) che si è usato per accedere ad IRC e la causa del logout, rispettivamente;
4. nel quarto caso, X contiene l'indirizzo (solitamente un IP) da cui si accede ad IRC;
5. nel terzo caso, sono possibili anche altre configurazioni della riga (ad es., un utente "is now known as" un altro utente). Ai fini di questo progetto, tali configurazioni possono essere ignorate.

Formato del nome di file: I nome dei file di tipo chat iniziano con il prefisso *chat* (ad es. `chat.evergreen.05.03-Sun-2015.log`). Si noti che, per rendere il progetto generale, è necessario poter gestire con classi differenti formati differenti di log di chat.

Moduli del progetto

Il progetto consta di 3 parti, ciascuna implementata in un package separato:

- `it.uniroma1.lcl.dietrolequinte.loader`
- `it.uniroma1.lcl.dietrolequinte.search`
- `it.uniroma1.lcl.dietrolequinte.cmdline`

Il package `it.uniroma1.lcl.dietrolequinte` deve contenere le classi `Interrogazione, Utente` e relative sottoclassi. Tutti gli oggetti che rappresentano un'interrogazione o un messaggio devono poter essere rappresentati sotto forma di stringa secondo il formato seguente:

```
Utente: U
Stringa immessa: S
Tempo: T
```

Nel caso di una interrogazione di tipo AOL, se applicabile, occorre che la stringa restituita termini con due righe ulteriori:

```
Link cliccato: L
Posizione link cliccato: N
```

Modulo 1: Caricatore o Loader

Questo modulo prende in input una directory e legge tutti i file di log al suo interno. Ciascuno di questi file sarà in uno dei due formati visti sopra.

Nel progetto, deve esistere la classe eseguibile `it.uniroma1.lcl.dietrolequinte.loader.Loader`.

All'esecuzione di:

```
java -cp matricola.jar it.uniroma1.lcl.dietrolequinte.loader.Loader
logsDir
```

la classe deve leggere l'elenco dei file di log contenuti in logsDir e, per ciascun file, richiedere alla classe specializzata un'analisi del file in questione. I loader specializzati per un determinato tipo di log devono essere contenuti nel package `it.uniroma1.lcl.dietrolequinte.loader.tipo`. Ad esempio, un loader di log di interrogazioni sarà contenuto nel package `it.uniroma1.lcl.dietrolequinte.loader.query`, mentre un loader di log di chat sarà contenuto nel package `it.uniroma1.lcl.dietrolequinte.loader.chat`. Il `Loader` generale deve essere in grado di gestire anche formati diversi della stessa tipologia (ad es. con due classi differenti per il tipo query) e nuove tipologie, sconosciute al momento della consegna del progetto, semplicemente mediante l'aggiunta di nuovi package. Il `Loader` colleziona tutte le informazioni risultanti dall'analisi dei singoli file e scrive su standard output quanto segue:

```
Numero di file in LOGS_DIR: N
Numero di file di tipo chat: M1
Numero di file di tipo query: M2
Numero totale di byte: B
Numero totale di righe: R
File1 Tipo di file: T1
File1 Numero totale di byte: B1
File1 Numero totale di righe: R1
File1 Numero totale di interrogazioni: I1
File1 Numero totale di login/logout: J1
File1 Numero totale di messaggi: K1
File1 Numero totale di azioni: A1
....
FileN Tipo di file: TN
```

```
FileN Numero totale di byte: BN
FileN Numero totale di righe: RN
FileN Numero totale di interrogazioni: IN
FileN Numero totale di login/logout: JN
FileN Numero totale di messaggi: KN
FileN Numero totale di azioni: AN
```

dove `N, B, R, Bi, Ri, M1, M2, Ii, Ji, Ki, Ai` sono numeri, `Ti` è chat o query (o un altro tipo di log, se presente e leggibile tramite un loader apposito), `Filei` è il nome del file esaminato. Le righe in grassetto sono presenti solo in corrispondenza del tipo di log appropriato (ovvero: interrogazioni per le query, login/logout, messaggi e azioni per le chat).

E' da notare che i file potrebbero essere compressi; in tal caso, avranno un'estensione ".gz". Per la lettura di questi file, si consiglia l'uso della classe GZIPInputStream:

```
FileInputStream fin = new FileInputStream(FILENAME);
GZIPInputStream gzis = new GZIPInputStream(fin);
InputStreamReader isr = new InputStreamReader(gzis);
BufferedReader br = new BufferedReader(isr);
```

Modulo 2: Searcher

Questo modulo legge la struttura dati creata in RAM dal modulo loader e risponde ad una determinata (sequenza di) query. La classe `it.uniroma1.lcl.dietrolequinte.search.Searcher` è il punto di accesso. Di tale classe può esistere una sola istanza. La classe espone i seguenti metodi:

- `getIstanza`(): deve avere un argomento di tipo `String`, contenente la directory dei logs sui quali effettuare la ricerca. Deve ritornare l'unica istanza ammessa della classe `Searcher`.
- `getUsers`(): può avere o un argomento o nessuno. In quest'ultimo caso, deve restituire una collezione iterabile di oggetti `Utente`, contenente tutti e soli gli utenti (senza ripetizioni) che compaiono nei log. Se un argomento f1 (di tipo `String`) viene fornito in input, allora occorre considerare solo gli utenti che compaiono nel file f1.
- `search`(): dato in input il tipo di informazione ricercata (`String`) e un oggetto `Utente`, deve restituire una collezione iterabile di oggetti `SearchResult`, ciascuno rappresentante un risultato della ricerca. Per ciascuno di tali oggetti deve esistere una rappresentazione tramite stringa, che nel caso delle interrogazioni e dei messaggi si comporti come descritto nella sezione "moduli del progetto". Come sopra, è possibile restringere la ricerca ad un singolo file, fornendo un'ulteriore stringa in input. Infine, dev'essere possibile restringere la ricerca ad un intervallo di tempo e fornire solo i risultati relativi a tale intervallo. L'intervallo dovrà essere rappresentato da 2 oggetti di tipo `java.time.LocalDateTime` (dall'API di Java 8). Pertanto, questo metodo potrà

prendere da un minimo di 2 ad un massimo di 5 argomenti (nell'ordine: 1) il tipo di informazione richiesta (String), 2) l'Utente, 3) il file cui restringersi, 4) data di inizio (inclusa), 5) data di fine (inclusa), con le seguenti quattro combinazioni di intestazioni: 1, 2; 1, 2, 3; 1, 2, 4, 5; 1, 2, 3, 4, 5). Il metodo solleva un'eccezione proprietaria nel caso in cui il tipo di informazione richiesta non sia supportato.

- L'unico tipo di informazione ammesso dal Loader di tipo query è "interrogazione", mentre i `Loader` di tipo chat ammettono i seguenti tipi di informazione: "loginout", "messaggio", "azione". Nel caso di ulteriori tipologie di log (ulteriori package non conosciuti al momento della consegna del progetto), saranno le classi Loader corrispondenti a conoscere i tipi di informazione ammissibili (e quindi le stringhe da usare con il metodo `Searcher.search`), utilizzando correttamente il meccanismo dell'ereditarietà o delle interfacce e del polimorfismo.

Nel progetto, deve esistere la classe
`it.uniroma1.lcl.dietrolequinte.search.Searcher`
che espone i metodi di cui sopra. La classe utilizza il `Loader` per caricare le informazioni dai log.

Nota bene: la richiesta del modulo `Searcher` serve per offrire delle API a programmi Java che interroghino il Log Analyzer, senza passare per le modalità interattive o batch (descritte nel modulo 3, vedere più avanti).
Ovvero, un utente-programmatore di DietroLeQuinte dev'essere in grado di scrivere un programma Java come ad esempio il seguente:

```
import it.uniroma1.lcl.dietrolequinte.search.Searcher;

public class TestDietroLeQuinte {
	public static void main(String args[]) {
		Searcher searcher = Searcher.getIstanza(args[0]);
		System.out.println(searcher.getUsers().size());
	}
}
```

che dovrà stampare il numero di utenti presenti nei log nella directory passata come argomento.

Modulo 3: Log Analyzer

Modalità 1 - interattiva

Il terzo modulo è l'interfaccia di Log Analyzer verso l'utente. Esso rende disponibile una classe DietroLeQuinte e deve essere possibile avviarlo da riga di comando come segue:

```
java -cp matricola.jar
it.uniroma1.lcl.dietrolequinte.cmdline.DietroLeQuinte logsDir
```

In questo caso, il metodo entra in un ciclo infinito: richiede un input da tastiera, ovvero la domanda da sottoporre al sistema, la elabora, restituisce la risposta all'utente e torna alla richiesta successiva. Se l'utente immette "exit", il metodo esce dal ciclo infinito e mostra statistiche sull'uso dell'applicazione. Occorre che il sistema sappia rispondere alle seguenti domande:

- num_login u: riporta il numero di login effettuati dall'utente u
- num_logout u: riporta il numero di logout effettuati dall'utente u
- num_messaggi u: riporta il numero di messaggi inviati dall'utente u
- num_azioni u: riporta il numero di azioni descritte dall'utente u
- num_login u f t: riporta il numero di login effettuati dall'utente u tra la data f e la data t
- num_logout u f t: riporta il numero di logout effettuati dall'utente u tra la data f e la data t
- num_messaggi u f t: riporta il numero di messaggi inviati dall'utente u tra la data f e la data t
- num_azioni u f t: riporta il numero di azioni descritte dall'utente u tra la data f e la data t
- num_query u: riporta il numero di interrogazioni effettuate dall'utente u
- num_query u i: riporta il numero di interrogazioni effettuate dall'utente u, in risposta alle quali l'utente ha cliccato sull'i-esimo risultato
- num_query u i f t: riporta il numero di interrogazioni effettuate dall'utente u tra la data f e la data t, in risposta alle quali l'utente ha cliccato sull'i-esimo risultato
- data_di_login u n: riporta l'n-esima data di login dell'utente u
- data_di_logout u n: riporta l'n-esima data di logout dell'utente u
- messaggio u n: riporta l'n-esimo messaggio dell'utente u
- data_di_login u n f t: riporta l'n-esima data di login dell'utente u, tra quelle fatte tra la data f e la data t
- data_di_logout u n f t: riporta l'n-esima data di logout dell'utente u, tra quelle fatte tra la data f e la data t
- messaggio u n f t: riporta l'n-esimo messaggio dell'utente u, tra quelli inviati tra la data f e la data t
- azione u n f t: riporta l'n-esima azione dell'utente u, tra quelle descritte tra la data f e la data t
- query u n: riporta l'n-esima interrogazione dell'utente u
- query u n i: riporta l'n-esima interrogazione dell'utente u, tra quelle in risposta alle quali l'utente ha cliccato sull'i-esimo risultato
- query u n i f t: riporta l'n-esima interrogazione dell'utente u, tra quelle in risposta alle quali l'utente ha cliccato sull'i-esimo risultato e fatte tra la data f e la data t

Il formato delle date dev'essere in ISO 8601. Se la ricerca non restituisce risultati, allora le richieste di numeri rispondono con "0" e le altre con "Nessuno".

Modalità 2 - batch

```
java -cp matricola.jar
it.uniroma1.lcl.dietrolequinte.cmdline.DietroLeQuinte logsDir
cmdsFile
```

In questa seconda modalità, i comandi sono listati nel file di testo cmdsFile, con lo stesso formato di cui sopra, e sono eseguiti in sequenza, come se venissero immessi uno dopo l'altro dall'utente (modalità batch).

Valutazione

Prima di tutto, il progetto dev'essere corretto. Verrà testato con dei file di input che sono un sottoinsieme di quelli contenuti in IRC.zip ed AOL.zip. Un possibile esempio di questo test è nella directory `esempio`, che contiene anche il risultato atteso dell'esecuzione del programma (nella versione non interattiva). La parte specificata in giallo è richiesta ai gruppi di 2 studenti, mentre è opzionale per gli studenti che consegnano individualmente (e può portare fino a **+5 punti** per questi ultimi).
Qualora l'output non sia quello atteso, il progetto sarà valutato come insufficiente e sarà necessario correggere gli errori per la sessione seguente. Nel caso in cui il test di correttezza venga passato, verranno valutati i seguenti elementi:

1) l'architettura e l'organizzazione del codice in termini di correttezza ed estensibilità, soprattutto in relazione all'uso appropriato di naming, incapsulamento e visibilità di campi e metodi, ereditarietà e polimorfismo (**fino 25 punti**).
2) documentazione del codice mediante javadoc (su tutte le intestazioni di classe e sui metodi richiesti da queste specifiche, inclusa la documentazione dei parametri, **fino 5 punti**)
3) la gestione della reflection per permettere la specifica di più classi per lo stesso tipo di log e di nuovi package per l'aggiunta di nuovi Loader (**fino a 5 punti per gli studenti che consegnano individualmente, obbligatoria per i gruppi di 2 studenti**)

Per un totale massimo di: 30 punti base + 5 punti se si è gestita la reflection (studenti individuali) + 5 punti per le interrogazioni più avanzate (studenti individuali).