ESAME DI FONDAMENTI DI INFORMATICA T-2 del 9/1/2019

Proff. E. Denti – R. Calegari – G. Zannoni   Tempo: 4 ore

NOME PROGETTO ECLIPSE: CognomeNome-matricola (es. RossiMario-0000123456)
NOME CARTELLA PROGETTO: CognomeNome-matricola (es. RossiMario-0000123456)
NOME ZIP DA CONSEGNARE: CognomeNome-matricola.zip (es. RossiMario-0000123456.zip)

NB: l'archivio ZIP da consegnare deve contenere l'intero progetto Eclipse

È richiesto di sviluppare un semplice ma efficace software, denominato *SafeRepo*, per la gestione di un repository di documenti *versionati*: l'obiettivo è cioè quello di mantenere tutte le versioni di un documento, consentendo di recuperare in ogni momento una versione precedente (undo), ripristinarla, etc., in modo pratico e sicuro.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

Un classico "spazio di archiviazione" – su disco o nel cloud, come Dropbox, Google Drive, etc. – consente di caricare file di ogni tipo, ma ne mantiene un'unica copia (l'ultima): caricando un nuovo file con lo stesso nome di uno già esistente, il precedente va perso.

Un ***repository versionato***, invece, non cancella mai nulla: quando si carica una nuova versione di un documento, vengono comunque mantenute anche tutte le precedenti, con la relativa storia (numero di versione, data e ora di caricamento di ognuna). Ciò consente di:

- aggiungere una nuova versione senza mai perdere le precedenti;
- recuperare la versione corrente del documento;
- recuperare una qualunque versione precedente del documento, o indicandone il numero (ad esempio, "la versione n.3"), o, in alternativa, indicando l'istante (data/ora) che interessa (ad esempio, "la versione in essere al 30 dicembre 2018 alle ore 16:15")
- cancellare virtualmente un elemento dal repository semplicemente aggiungendo una nuova versione vuota, senza con ciò perdere l'accesso a tutte le versioni precedenti.

Concretamente, occorre distinguere la ***gestione logica*** dalla ***gestione fisica:***

- a livello logico, per ogni documento (identificato da un ID univoco) il sistema mantiene in memoria la lista delle sue versioni che però, per ovvi motivi di spazio, *non* incorporano fisicamente altrettante copie del file;
- a livello fisico, il sistema mantiene, in una cartella su disco, le varie copie dei file, opportunamente rinominate in modo da esprimere direttamente sia il numero di versione sia la data/ora corrispondente: da sottolineare che il sistema produce e manipola sempre e solo copie del file indicato dall'utente, mai l'originale.

È compito di *SafeRepo* rendere tutta questa gestione trasparente all'utente, che dovrà poter inserire e recuperare i suoi documenti senza preoccuparsi di dove siano fisicamente archiviati i file e come si chiamino.

MODELLO DEI DATI

Un ***documento*** è caratterizzato da un identificativo univoco (es. "doc1") e dal percorso assoluto che identifica il corrispondente file su disco (ad esempio, "C:/enrico/doc1").

Un ***documento versionato*** accoppia un documento a un numero di versione (intero crescente nel tempo) e un timestamp in formato ISO che ne caratterizza l'instante di caricamento (ad esempio, la versione 1 di "doc1", caricata il 31/12/2018 alle 15:39:02.154254, ha come numero di versione 1, come timestamp 2018-12-31T15:39:02.154254 ed è associata al documento "C:/myrepo/doc1_1_2018-12-31T15_39_02.154254" – vedere oltre per i dettagli).

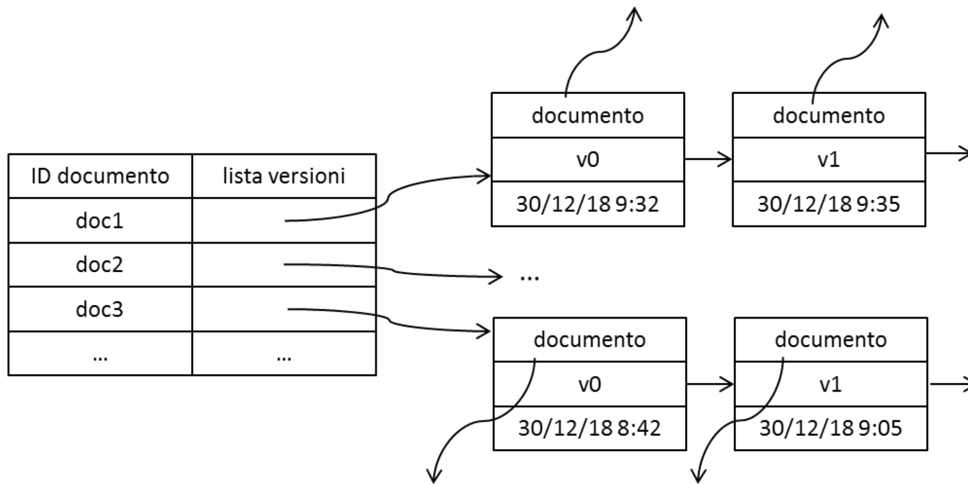
GESTIONE INTERNA DEL REPOSITORY

A livello logico, si desidera che il repository sia organizzato con una mappa che associ a ogni identificativo di documento la lista dei corrispondenti ***documenti versionati***, come illustrato nella figura sottostante. Le nuove versioni vengono aggiunte in coda alla lista, così che i timestamp siano automaticamente ordinati in senso crescente,. Di conseguenza:

- la versione corrente è l'ultimo elemento della lista
- la versione N-ma è recuperabile accedendo all'N-mo elemento della lista (N≥0, N<size della lista)
- la versione in essere a un certo istante X è recuperabile filtrando solo gli elementi della lista con timestamp T≤X, e prendendo poi quella corrispondente al massimo valore del timestamp.

Ad esempio, se del documento "doc1" esistono le 7 versioni v0 del 2018-12-30T09:39:02, v1 del 2018-12-30T09:39:03, v2 del 2018-12-30T09:40:04, v3 del 2018-12-30T09:40:05, v4 del 2018-12-30T09:42:06, v5 del 2018-12-30T09:42:07 e v6

del 2018-12-30T09:44:08, chiedendo di recuperare la versione valida all'istante X=2018-12-30T09:40:06 si deve ottenere in risposta la v3, in quanto le successive v4, v5 e v6 sono state inserite solo successivamente all'istante X.



GESTIONE PERSISTENZA NEL REPOSITORY

IMPORTANTE: per prevenire incompatibilità **utilizzare sempre le barre standard '/'** nei percorsi, anche su piattaforma Windows, evitando le barre inverse '\' (che comunque devono, nel caso, essere espresse con la notazione '\\').

Directory di lavoro: il repository mantiene le sue copie private dei file in una sua cartella interna, configurata all'atto della creazione del repository stesso.

Aggiunta file: ogni volta che l'utente aggiunge al repository una nuova versione di un file, il repository effettua una nuova copia del file dell'utente e la colloca nella sua cartella interna: il nuovo file viene chiamato ***IDdoc_versione_timestamp***, dove il carattere ':' presente nel timestamp in formato ISO (che non sarebbe accettato dal file system) è sostituito dal carattere '_' (underscore). Così, ad esempio, la versione 2 del documento "doc1", inserita il 31/12/2018 alle ore 15:39:03.318918, viene copiata in un file interno di nome "doc1_2_2018-12-31T15_39_03.318918": il *path documento* della versione 2 di doc1 è perciò "....../doc1_2_2018-12-31T15_39_03.318918", essendo "......" la directory di lavoro del repository (prefissata, come detto, all'atto della creazione del repository stesso).

Eliminazione file: se l'utente elimina un documento, le sue versioni precedenti vengono mantenute: viene quindi creata una nuova versione con un nuovo documento il cui *path* è posto a null. Ad esempio, se l'utente elimina il documento "doc2", di cui erano presenti (supponiamo) tre versioni, viene creata una nuova versione v4 in data/ora corrente, il cui documento ha identificativo "doc2" ma path null.

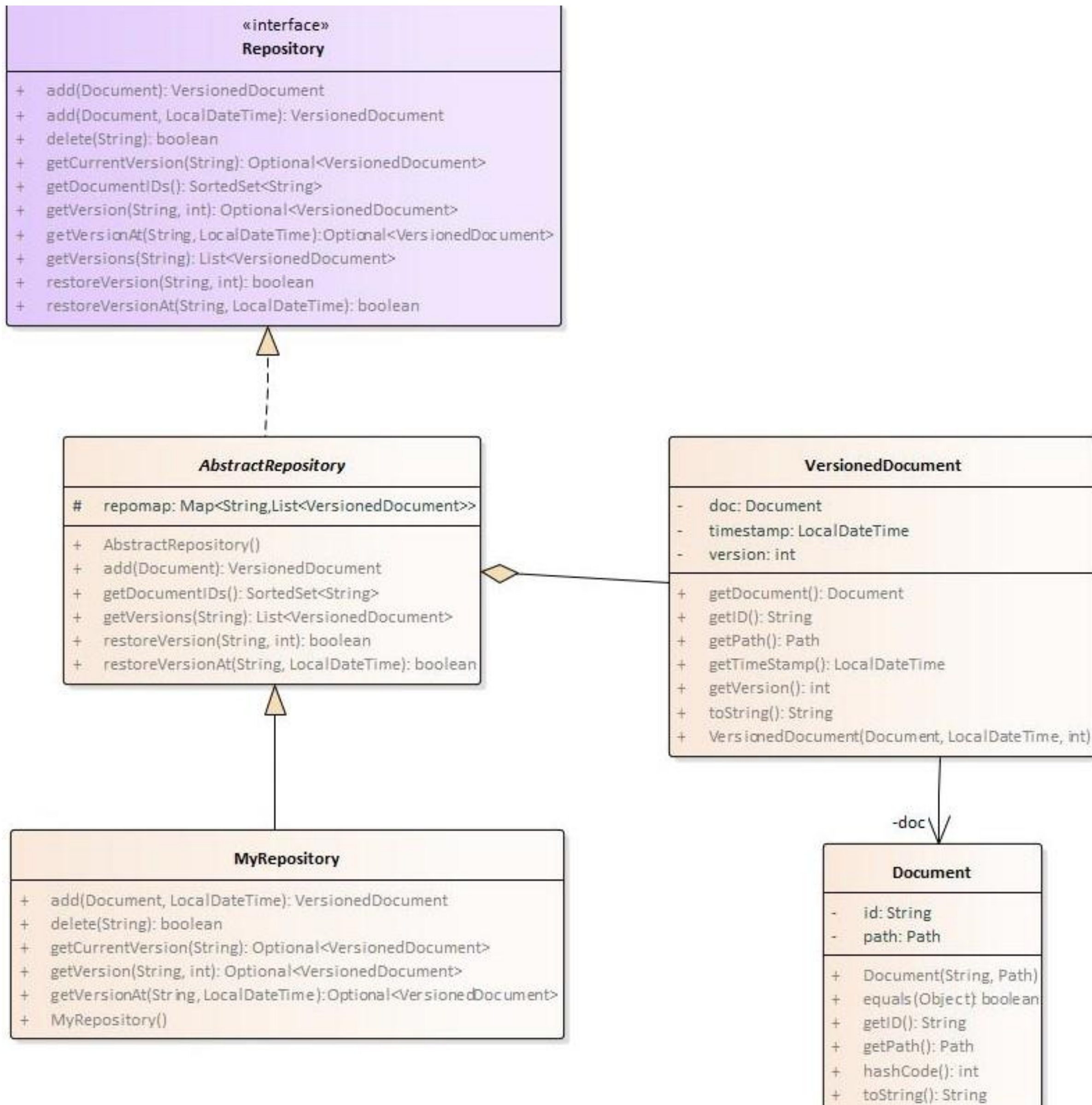
Ripristino file: se l'utente ripristina una versione precedente (ossia riporta logicamente il documento allo stato in cui era in un tempo precedente), viene semplicemente creata una nuova versione il cui *documento* punta allo stesso file a cui puntava la versione ripristinata. Ad esempio, se l'utente ripristina la versione 4 del documento "doc1" (che portava la data del 2018-12-30T09:42:06), viene creata una nuova versione 7 con timestamp corrente, il cui *documento* punta però allo stesso file a cui puntava la versione 4, ossia al file "doc2_4_2018-12-30T09_42_06".

Prosegue alla pagina successiva

Parte 1 (punti: 20)

Dati (namespace saferepo.model) *(punti: 13)*

Il modello dei dati deve essere organizzato secondo il diagramma UML di seguito riportato:



SEMANTICA:

a) la classe ***Document*** (fornita) rappresenta un documento, caratterizzato da identificativo univoco (stringa) e percorso (Path).

b) la classe ***VersionedDocument*** (fornita) rappresenta un documento versionato: incapsula un documento, ed è caratterizzata anche da timestamp (LocalDateTime) e numero di versione (intero non negativo).

c) l'interfaccia ***Repository*** (fornita) descrive l'interfaccia d'uso del repository dal punto di vista logico (non gestisce quindi alcun file: ciò è compito della persistenza descritta più oltre) e consente di:

- recuperare la versione corrente o precedente di un documento, di cui sia noto l'identificativo univoco: il risultato è un ***Optional< VersionedDocument >***;
- ripristinare una versione precedente di un documento di cui sia noto l'identificativo univoco, specificando o il numero di versione, o il timestamp corrispondente all'istante di tempo che interessa;
- aggiungere un documento (***Document***) al repository, eventualmente specificando il timestamp da associare (default: ***now***); i metodi restituiscono il ***VersionedDocument*** del nuovo documento inserito;
- eliminare (logicamente) dal repository un documento di cui sia noto l'identificativo univoco.

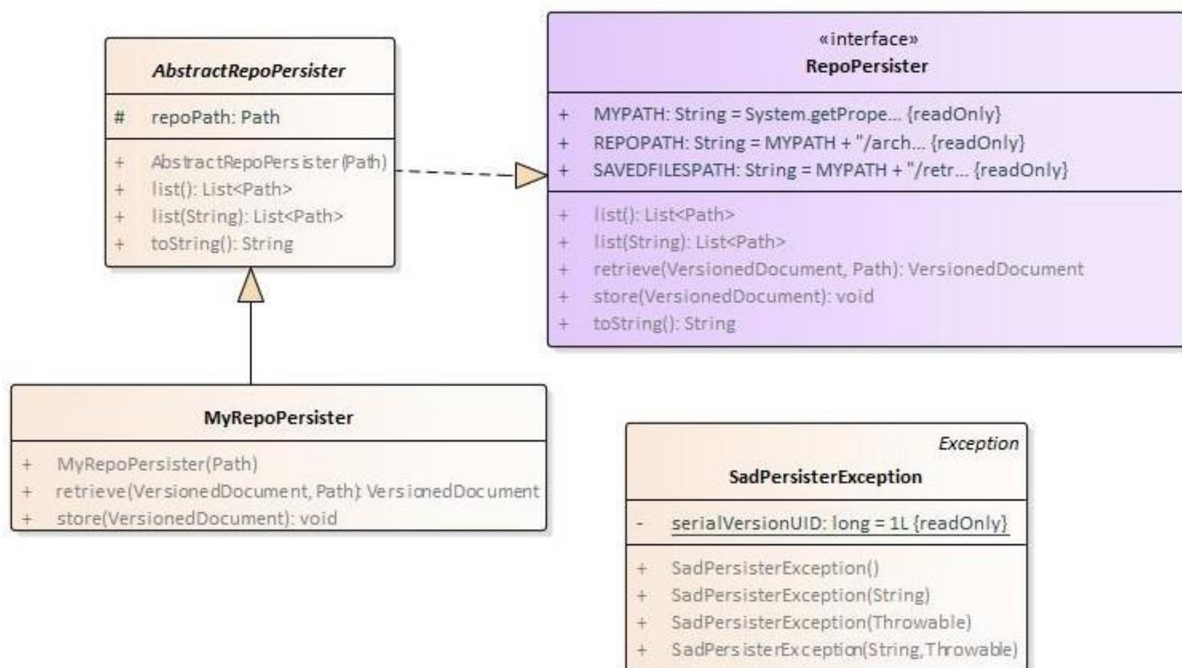
d) la classe astratta ***AbstractRepository*** (fornita) implementa parzialmente ***Repository***, definendo la struttura dati e i metodi che non catturano direttamente specifiche politiche – ovvero ***getVersions, getDocumentIDs, add/1*** (che è un banale shortcut per un caso particolare di ***add/2***) - nonché ***restoreVersion*** e ***restoreVersionAt*** (in quanto non direttamente coinvolti nella gestione della successiva applicazione grafica).

e) la classe **MyRepository** (da realizzare) completa l'implementazione di ***Repository*** estendendo la precedente classe ***AbstractRepository***: a tal fine implementa i metodi ***add, delete, getCurrentVersion, getVersion, getVersionAt*** nel modo specificato in precedenza. Più precisamente, tutti i metodi che restituiscono una versione possono restituire un optional *empty* nel caso la versione richiesta non esista: in più, ***getVersion*** lancia ***IllegalArgumentException*** in caso sia invocata con un numero di versione negativo, incoerente con l'idea stessa di "numero di versione".

NB: come da specifica, si ricorda che la mappa (*protected* in ***AbstractRepository***) ha come chiave l'ID del documento e come valore una lista di ***VersionedDocument***, ordinata per costruzione per *timestamp* crescente.

Persistenza (namespace saferepo.persistence) *(punti: 7)*

Questo componente è la <u>controparte fisica</u> del ***Repository*** precedente: suo compito è gestire le copie del file su disco con i relativi rename, tramite la coppia di metodi ***store***/***retrieve*** (NB: non tutte le funzionalità offerte da Repository a livello logico sono disponibili qui a livello fisico: in particolare, non viene offerto il servizio di ripristino di versioni precedenti, pertanto *non* sono utilizzati i metodi ***restoreVersion*** e ***restoreVersionAt*** del repository), secondo il diagramma UML di seguito riportato:



SEMANTICA:

a) l'interfaccia ***RepoPersister*** (fornita) dichiara **le costanti che definiscono le cartelle di lavoro**:

- `MYPATH`, la cartella principale di lavoro del persister: la posizione di default è la <u>home dell'utente</u>
- `REPOPATH` e `SAVEDFILESPATH`, sottocartelle di `MYPATH`, corrispondono alla directory in cui il persister copia i file versionati (metodo ***store***) e a quella in cui rende disponibili i file estratti (metodo ***retrieve***)

e i metodi per la manipolazione dei ***VersionedDocument***:

- ***store*** memorizza un ***VersionedDocument*** nella cartella interna del sistema *SafeRepo*, generando il nome di file e il percorso corrispondenti, ed effettuando la <u>copia fisica</u> del file, come da specifiche iniziali;
- ***retrieve*** recupera un ***VersionedDocument*** dalla cartella interna del sistema *SafeRepo*, generando il nome di file e il percorso corrispondenti ed effettuando la <u>copia fisica</u> del file da restituire all'utente: tale file viene collocato nella cartella specificata dall'utente come secondo argomento (Path);

- ***list*** restituisce <u>la lista di tutti i file</u> (Path) attualmente presenti nella cartella interna del sistema *SafeRepo*;
- ***list(documentID)*** restituisce <u>la lista dei soli file</u> (Path) disponibili nel sistema *SafeRepo* <u>corrispondenti al documento specificato</u>;
- ***toString*** restituisce una frase che specifica il <u>numero di file</u> attualmente presenti nella cartella interna del sistema *SafeRepo*.

b) la classe astratta ***AbstractRepoPersister*** (fornita) implementa parzialmente ***RepoPersister***, gestendo nel costruttore la cartella di lavoro e concretizzando i metodi ***list*** e ***toString.*** I metodi ***list*** incapsulano l'eventuale ***IOException*** in una ***SadPersisterException*** (fornita) con opportuno messaggio d'errore, mentre ***toString*** la cattura e restituisce "ERROR".

c) la classe **MyRepoPersister** (da realizzare) completa l'implementazione di ***RepoPersister*** estendendo la classe ***AbstractRepoPersister***: a tal fine essa implementa i due metodi ***store*** e ***retrieve*** effettuando le copie dei file come da specifica, gestendo opportunamente i nomi dei percorsi. Anche questi metodi devono incapsulare l'eventuale ***IOException*** in una ***SadPersisterException***. <u>NB: è necessario evitare di cablare nel codice i percorsi delle cartelle di lavoro, utilizzando sempre le costanti definite nell'interfaccia ***RepoPersister***</u>

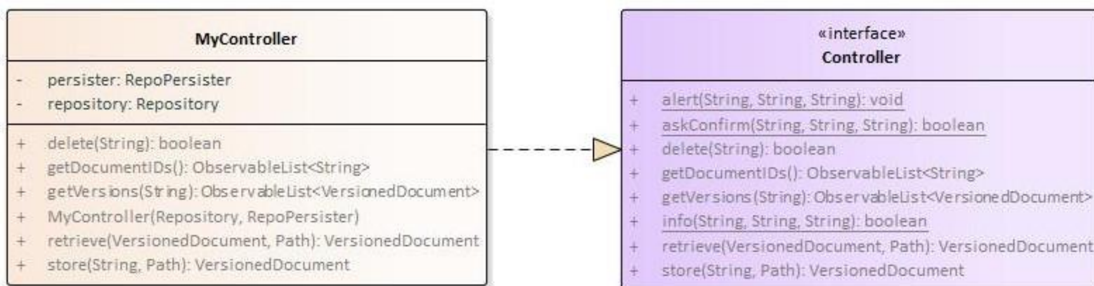
SUGGERIMENTO 1: ricordare i metodi di utilità della classe ***Files*** (in particolare ***copy***) e della classe ***Path*** (in particolare ***resolve*** per comporre percorsi). Per copiare fisicamente un file, si suggerisce la sintassi:

Path copiedFile = Files.copy(*pathSorgente*, *pathDestinazione*, StandardCopyOption.REPLACE_EXISTING);

Parte 2 (punti: 10)

Controller (namespace saferepo.controller)

Il controller – articolato in interfaccia e implementazione – è fornito già pronto: i suoi metodi fanno da ponte verso quelli opportuni del repository e del persister. L'interfaccia ***Controller*** offre altresì tre metodi statici ***alert***, ***askConfirm*** e ***info*** per far comparire le finestre di dialogo utili a segnalare errori (Fig.10), chiedere conferma dell'intenzione dell'utente (Fig. 8) o fornire informazioni (Fig. 7).



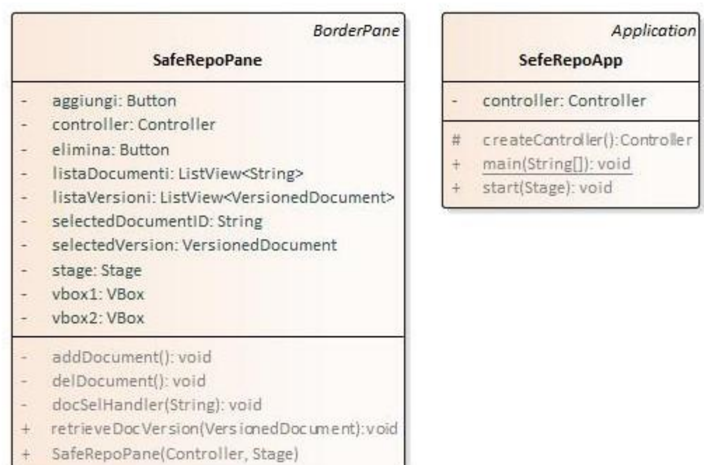
GUI (namespace saferepo.ui) (punti: 10)

L'interfaccia grafica dev'essere simile (non necessariamente identica) a quella sotto illustrata.

La classe ***SefeRepoApp*** (fornita) contiene il main di partenza dell'applicazione.

La classe ***SefeRepoPane*** (da realizzare) contiene il pannello principale, organizzato come segue:

- Due componenti ***ListView***, inizialmente vuoti, mostrano rispettivamente i documenti disponibili nel repository e le rispettive versioni; nella riga sottostante, due pulsanti (***Aggiungi file / Elimina***



file – quest'ultimo inizialmente disabilitato) consentono le omonime operazioni (Fig. 1).

- L'utente può aggiungere un documento al sistema *SafeRepo* premendo il pulsante ***Aggiungi file***: compare allora un apposito ***FileChooser*** (Fig. 2) per scegliere i file da caricare. Confermando, l'elenco documenti viene aggiornato (Fig. 3) con il documento appena inserito. Continuando a inserire documenti, essi compaiono via via nell'elenco, in ordine alfabetico (Fig. 4).
- Cliccando su una voce dell'elenco documenti, la lista a fianco viene immediatamente popolata con le rispettive versioni (Fig. 5): cliccando su una di queste voci, il sistema offre la possibilità di scaricare il corrispondente documento, tramite un apposito ***DirectoryChooser*** (componente analogo al ***FileChooser***, che consente di scegliere cartelle anziché file – Fig. 6): il titolo del chooser indica quale versione si sta scaricando (ad es. "Scaricamento versione #1 di doc2"). Se l'utente conferma, il corrispondente file viene copiato nella cartella di destinazione scelta, e viene mostrato un apposito messaggio di conferma (Fig. 7).
- Se, infine, l'utente desidera eliminare (logicamente) un documento, deve selezionarlo e premere il pulsante Elimina documento: comparirà un apposito dialogo di richiesta conferma (Fig. 8), dopo il quale verrà effettuata la cancellazione, ossia verrà creata la nuova versione collegata al documento vuoto, come da specifica (Fig. 9). Nel caso si tenti di scaricare tale versione, un apposito messaggio d'errore (Fig. 10) informerà della situazione, specificando altresì che rimangono comunque disponibili tutte le versioni precedenti.

NB: evitare di cablare nel codice i percorsi delle cartelle di lavoro del persister: agire sempre tramite controller.

Per comodità, può convenire invece impostare i file/directory chooser in modo che si posizionino sulle cartelle predefinite nel progetto – in particolare sulla sottocartella "testfiles" nel caso del file chooser "aggiungi file".
Ciò può essere fatto chiamando il metodo `setInitialDirectory` con opportuni argomenti, ad esempio:

```
chooser.setInitialDirectory(Paths.get(System.getProperty("user.dir")).toFile());
```

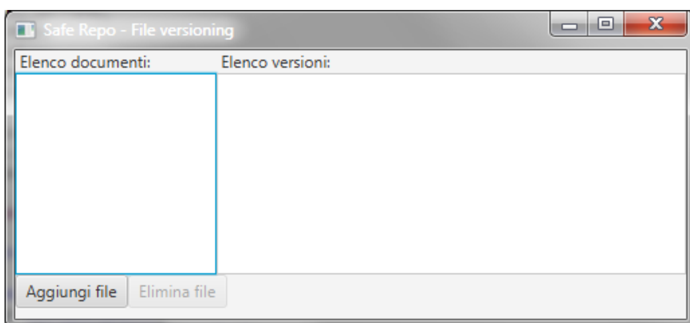


Figura 1

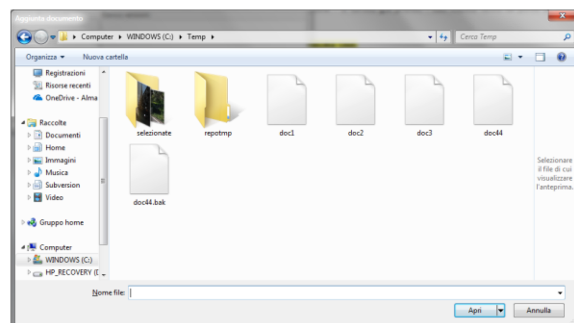


Figura 2

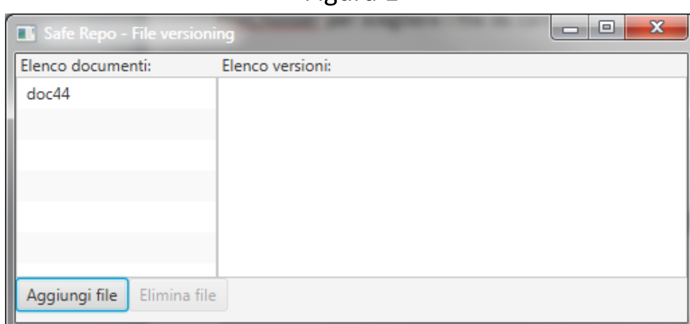


Figura 3

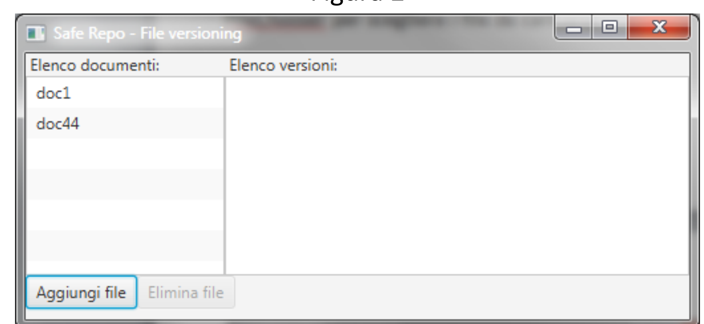


Figura 4

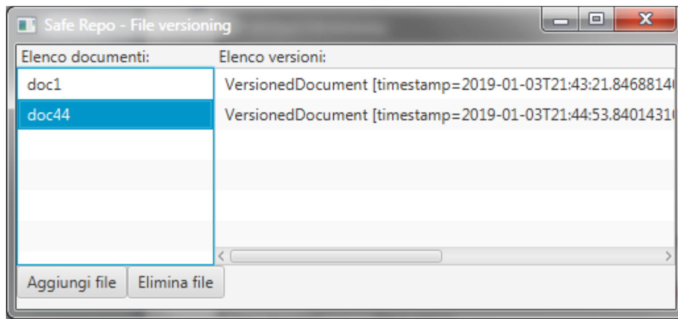


Figura 5

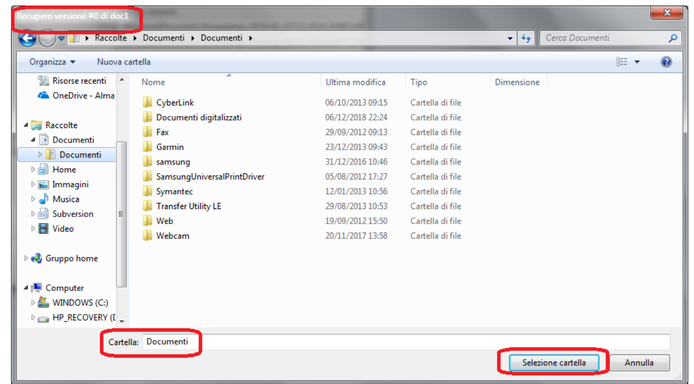


Figura 6

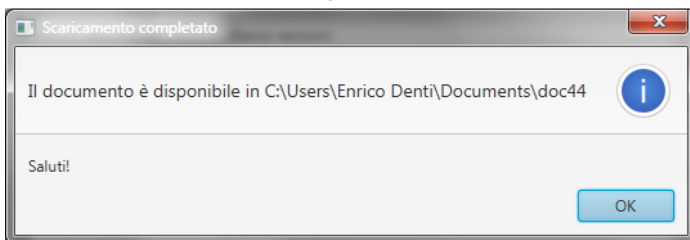


Figura 7

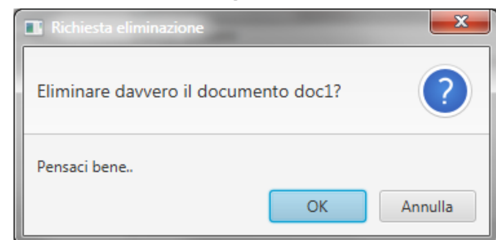


Figura 8

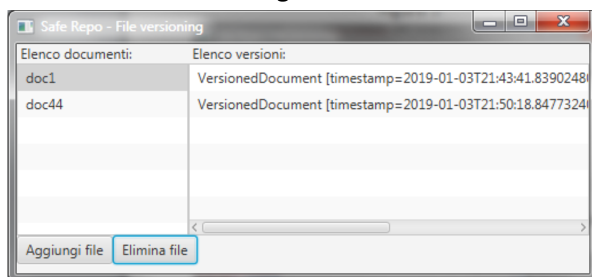


Figura 9

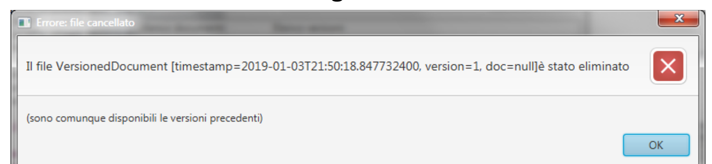


Figura 10

I tre metodi statici ***Controller***.*alert*, ***Controller***.*askConfirm* e ***Controller***.*info* consentono di far comparire le finestre di dialogo rispettivamente per segnalare errori (Fig.10), chiedere conferma dell'intenzione dell'utente (Fig. 8) o fornire informazioni (Fig. 7).