Progetto di Programmazione ad oggetti: MercurialInterface

Svolto da:

Filippo Solazzi e

Silvia Gasparroni

Deadline: E

Indice

Analisi

Il programma che abbiamo implementato vuole riprodurre un'interfaccia grafica per Mercurial, che si occupa di facilitare l'utilizzo di quest'ultimo grazie alla pressione di vari bottoni.

Per prima cosa ci siamo occupati di cercare un modo per fare comunicare le classi java con il prompt dei comandi.

Dopo una lunga ricerca abbiamo deciso di utilizzare le classi Process e Runtime che ci hanno permesso di creare un nuovo processo al fine di inviare informazioni al prompt.

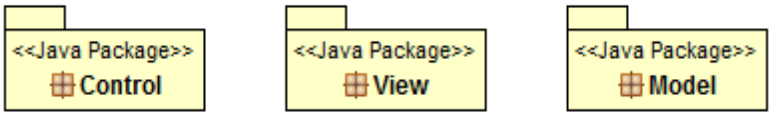
Per rendere il codice più fluido e ordinato si è scelto di utilizzare il Pattern MVC perché così facendo possiamo ottenere una buona programmazione e se in futuro si vorrà modificare il programma lo si può fare in modo rapido e senza complicazioni.

Mentre per una buona programmazione il model è stato suddiviso in più classi per ottenere una buona distinzione delle attività svolte.

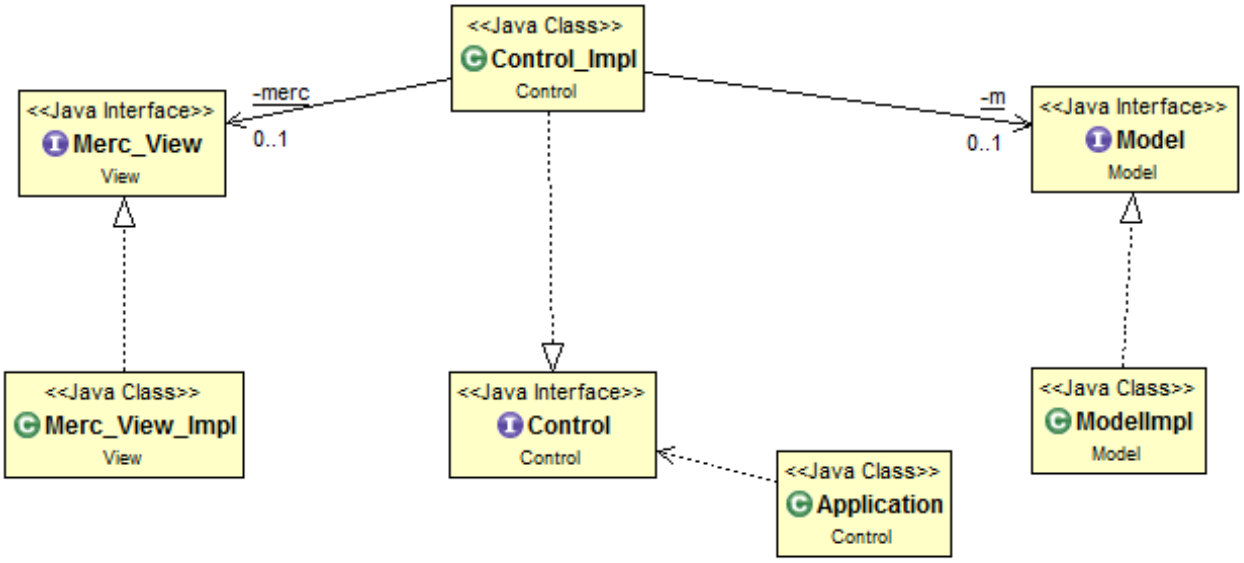
Inoltre si sono fatte diverse considerazioni lungo la progettazione che verranno spiegate durante la lettura della relazione.

Design

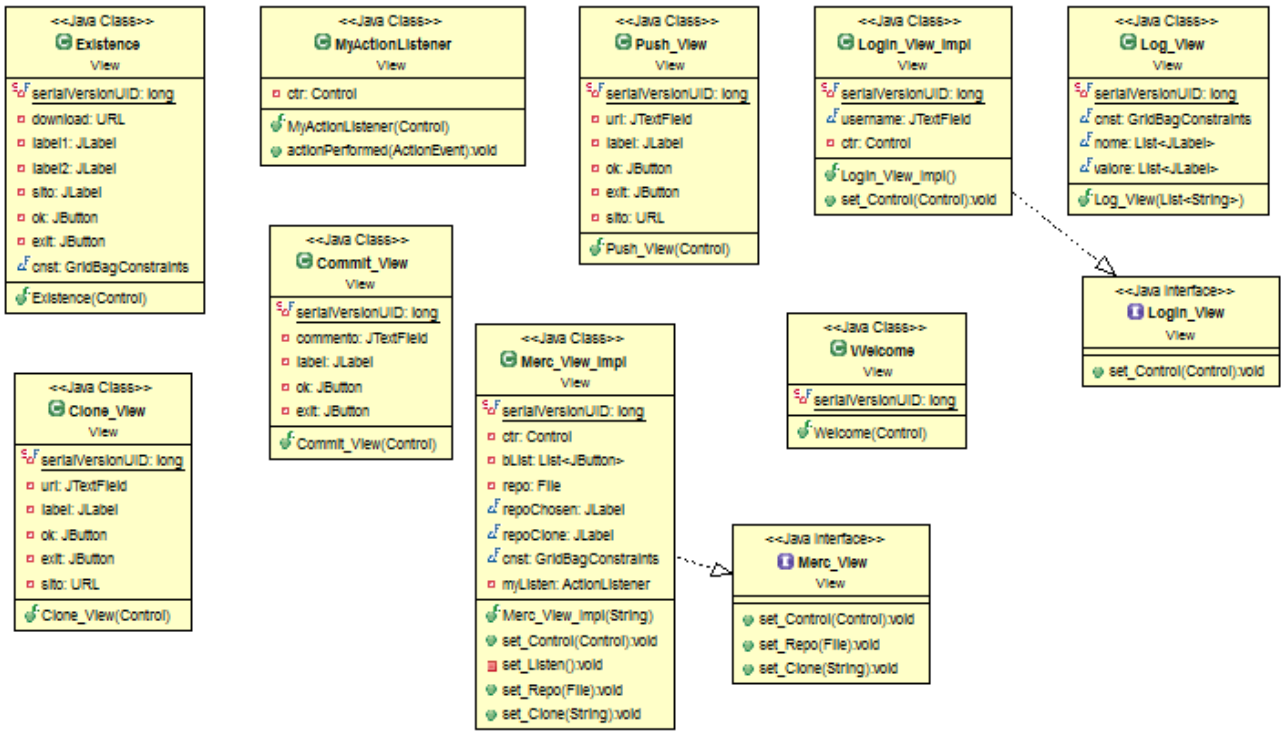
Per rendere il programma più ordinato si è fatta anche una suddivisione di Package.



Come già detto, per la realizzazione del nostro progetto, abbiamo utilizzato il pattern MVC e in particolare Filippo Solazzi si è occupato della gestione della View mentre Silvia Gasparroni del Model mentre il Control è stato svolto in collaborazione.



View



Come si può notare dall'immagine tutte le View non hanno nessuna dipendenza ne tra di loro ne dalla View principale solo per il semplice fatto che vengono gestite dagli eventi generati dalla Merc_View_Impl quindi vengono intercettati dal Control e generate a loro volta per essere utilizzate dall'utente e dopo ciò verranno distrutte tramite il comando dispose() per non andare a riempire la memoria inutilmente.

Ogni View si aspetta come parametro un Control() che verrà utilizzato per intercettare gli eventi dell'interfacce. Ogni interfaccia ha una funzione diversa dalle altre, anche se alcune possono sembrare simili tra loro.

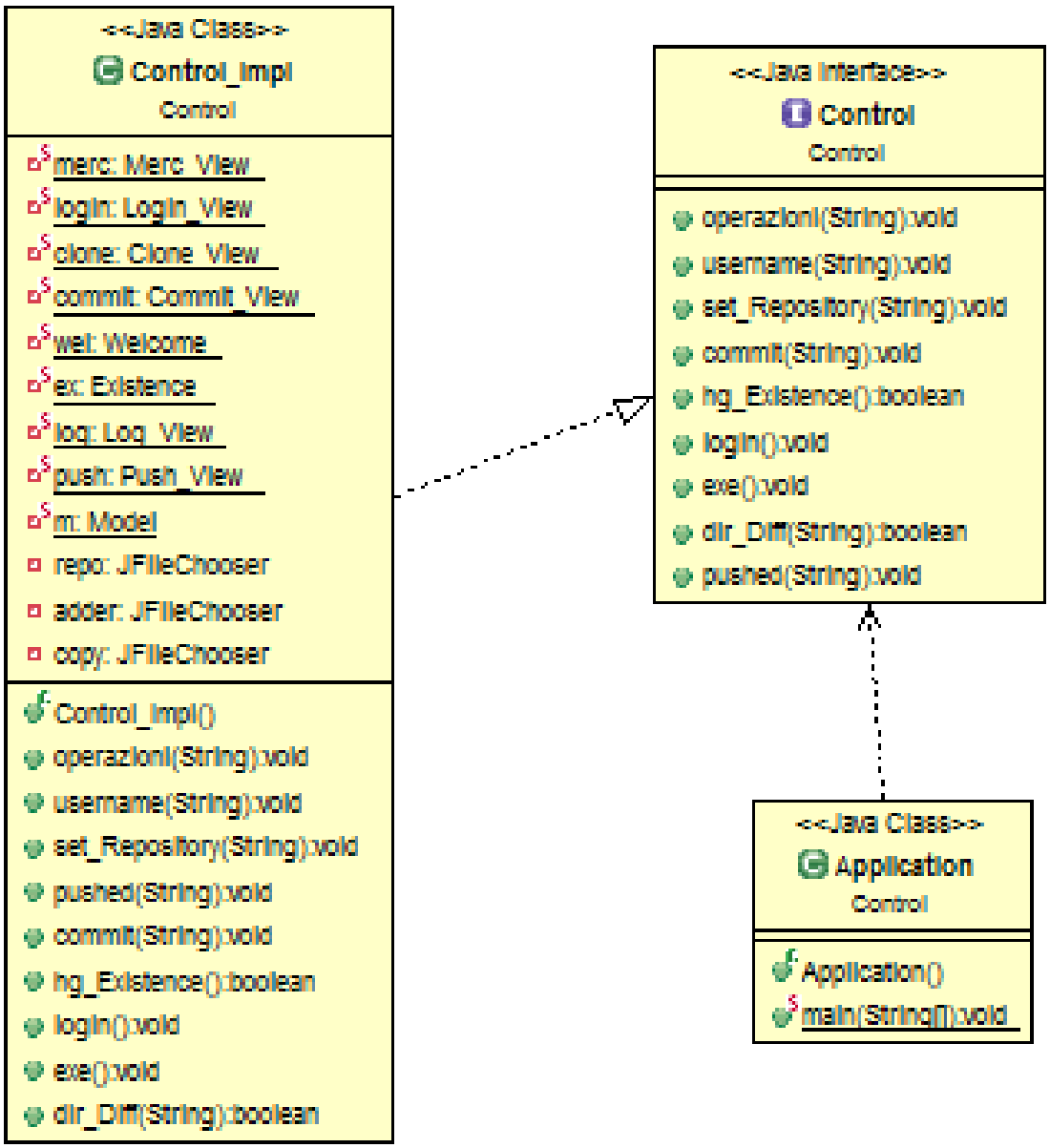
Partendo dalla prima Existence(): questa View viene utilizzata per aiutare l'utente ad installare Mercurial nel caso non fosse già presente. Essa viene richiamata da Control, dopo la chiusura del Welcome(), dopo aver interrogato il Model(), che tramite altri metodi, fa i propri controlli e restituisce al Control() l'esito della richiesta e così facendo l'Interfaccia Existence() può essere inizializzata o meno.

La MyActionListener() non ha bisogno di grandi presentazioni in quanto è l'implementazione dell'Interfaccia ActionListener() ed essa non fa altro che intercettare gli eventi della pressione dei bottoni nella Merc_View() per poi passarli al Control().

Push_View(), Clone_View(), Commit_View() e Log_View() sono delle GUI che vengono generate con la pressione dei bottoni ed essi vengono di volta in volta instanziate dal Control(), che come già detto è lui ad intercettare gli eventi della View, per poi essere distrutte. Ogni View si spiega da se con il proprio nome ma per aiutare l'utente si ha un ToolTip sopra ogni bottone, sempre della View principale, per ottenere delle informazioni che lo aiutino.

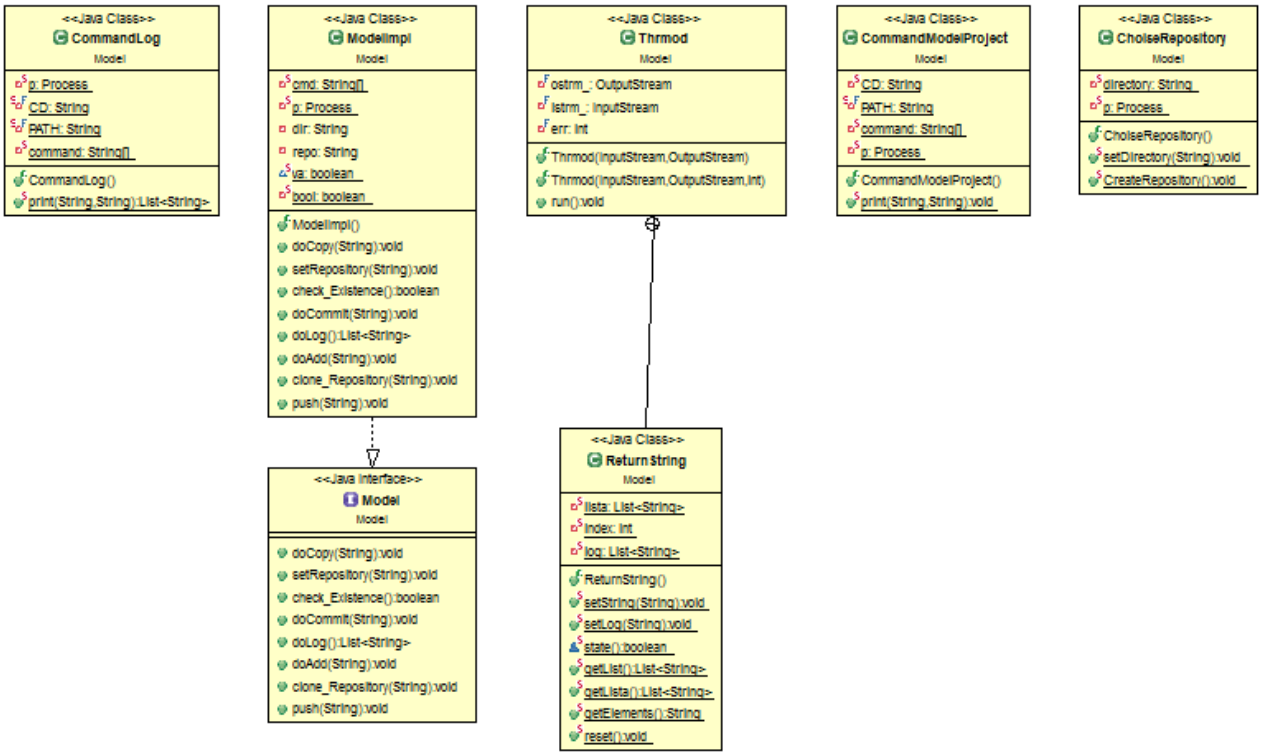
Tutte le volte che si compiono degli eventi in queste View secondarie vengono inviati al Control() per effettuare dei controlli e/o passare delle informazioni al Model che le gestirà.

Control



Il Control è il nucleo del progetto perché è lui ad effettuare i controlli sulla maggior parte delle informazioni ed è lui che fa “interagire” il Model con la View, più precisamente, intercetta gli eventi delle View e passa le informazioni al Model che le elabora ed in caso restituisce altre informazioni da trascrivere nella View.

Model



Come già detto il ruolo de Model è quello di trascrivere i comandi che gli vengono passati dal Control nel Prompt. Il Control passa tutte le sue informazioni al ModelImpl che esegue lui stesso delle operazioni o le passa alle altre classi interne al package Model.

La classe CommandLog si occupa del salvataggio del log, la classe CommandModelProject si occupa di eseguire quasi tutti i comandi di hg che vengono passati a ModelImpl, ChoiseRepository si occupa di creare la repository.

Infine c’è una classe thrmod che estende runnable che permette di intercettare ciò che passa attravero il Prompt.

Sviluppo

View

Nella progettazione della View per prima cosa si è dovuto cercare di capire come poteva essere il modo migliore per ottenere una GUI semplice ed immediata senza troppe complicazioni. Per fare ciò si sono fatte più View. La prima View di Welcome è solo scenica e mostra il logo ed un pulsante per iniziare il programma. Dopo di che si potrebbe avere una seconda GUI per l'installazione del programma Mercurial. Quest'ultima è stata fatta per offrire un link all'utente per l'installazione di Mercurial e finché non verrà installato non si potrà procedere con il programma.

Una volta che si è passato questo primo controllo si visualizzerà la View di login dove potrà essere inserito tramite una JTextField un utente e finché non si fa ciò non si potrà continuare. Dopo di che si giungerà alla View principale la Merc_View_Impl. Come prima cosa si potrà notare subito che affiancato al nome del progetto si potrà visualizzare il nome utente (per non scordarselo) e anche la presenza di molti bottoni ed una JLabel dove verrà visualizzata la scritta "No Repository". Questa JLabel ci fa capire che per iniziare il progetto si dovrà per prima cosa inserire una repository o nuova o già esistente, infatti, solo il bottone con scritto Repository sarà abilitato mentre tutti gli altri saranno disabilitati. Andando con il mouse sopra il tasto si visualizzerà un ToolTip che ci mostrerà un piccolo suggerimento. Premendo il tasto si aprirà uno sfoglia e da lì si potrà andare a scegliere la propria repository locale.

Una volta selezionata la repository apparirà un messaggio che ci dirà se è stata creata una nuova repository oppure è una già esistente. Dopo questo primo passaggio si attiveranno tutti gli altri i bottoni e ovviamente anche il tasto repository rimarrà attivo perché se si vorrà creare una

nuova repository o si vorrà cambiare repository lo si potrà fare in qualsiasi momento.

In ordine vedremo i seguenti bottoni: “Copy”, “Add”, “Commit”, “Log”, “Clone”, “Push” e “Repository”.

Il primo bottone servirà semplicemente per copiare un qualsiasi file o directory da un qualsiasi punto del Pc dentro la nostra directory. Per fare ciò, nel momento della pressione del bottone, si aprirà uno sfoglia e da lì si potrà selezionare quello che si vuole. Una volta aggiunto apparirà un messaggio per avvertire all’utente che l’operazione è andata a buon fine.

Il secondo bottone aprirà sempre uno sfoglia ma questa volta per aiutare l’utente la prima schermata che si vedrà sarà all’interno della repository in modo tale da facilitare l’utente a scegliere quale programma vuole aggiungere alla directory. Anche qui una volta fatto ciò apparirà un messaggio che avverte l’utente della buona riuscita dell’operazione.

Il bottone “Commit” invece aprirà una nuova View dove sarà possibile aggiungere un commento e poi premendo in tasto “OK” si compirà il comando e si avrà un messaggio o di buona riuscita o di fallimento e si spiegherà anche il motivo della NON riuscita.

Il “Clone” sarà molto simile al “Commit” infatti sarà sempre una View per inserire qualcosa ma questa volta non sarà un commento ma un URL. La GUI ha anche al suo interno dei controlli per far sì che l’Utente non vada a inserire testi a caso o URL non esistenti, inoltre ci sarà anche qui un messaggio di buona o cattiva riuscita del processo.

Il “Push” avrà lo stesso funzionamento del Clone solo che invece l’URL questa volta verrà utilizzato in un altro modo dal Model e ci saranno dei differenti controlli per la correttezza dell’URL.

Infine il “Log”. Anche qui il bottone aprirà una nuova View ma questa volta sarà diversa dalle altre. Il “Log” mostrerà una specie di tabella dove

si potranno vedere a sinistra ed in rosso il nome delle “etichette” mentre a destra ed in blu si visualizzeranno i valori delle rispettive. Per fare questa View si è impiegato molto tempo perché inizialmente la suddivisione degli elementi la si voleva fare con dei rettangoli disegnati su schermo di colore, posizione e dimensione differente ma per un miglior ordinamento del codice e per mancanza di tempo si è preferito cambiare approccio anche se ormai di tempo se ne era già speso abbastanza.

Tutti questi bottone hanno un ToolTip per descrivere brevemente all’utente che cosà si andrà a fare o ad ottenere con la pressione di questi. Anche nella View principale vi voleva andare a modificare la forma ed il colore dei bottoni ma sempre per mancanza di tempo e per non renderlo fluido si è scelto di lasciare i classici bottoni.

Inoltre nella View principale si hanno 2 JLabel. La prima è nascosta e verrà visualizzato solo l’URL da dove si è effettuato il copy della directory, in modo da aiutare l’utente, mentre la seconda che è visibile, inizialmente si leggerà il messaggio “No Repository” ma una volta selezionata la repository si potrà vedere il suo percorso.

Control

Il control è formato principalmente da uno switch, che una volta ottenuto il comando da eseguire, andrà a istanziare la classe richiesta quindi generando la View o andando ad aprire lo sfoglia o passando delle informazioni al Model.

Quindi le View sono tutte quante generate dal Control e poi si autodistruggono una volta che hanno terminato il loro compito. Queste View in ogni caso vanno ad utilizzare anche dei metodi del Control per fare dei controlli sull'inserimento delle informazioni dell'utente.

Il Control essendo un cuore del processo avrà anche l'istanza delle classi Model e Merc_View e sarà il mediatore passando le informazioni da una classe all'altra per ottenere al meglio e nei tempi più brevi il risultato voluto.

Il Control è stata la parte più complessa da programmare perché essendo stato sviluppato da 2 persone contemporaneamente è stato difficile cercare di trovare un comune accordo su come andare a definire il codice per ottenere il risultato voluto da entrambi.

Model

Il Model come già detto si occupa di inviare al prompt i comandi mercurial da eseguire .

In tutte le classi del package troviamo un Process p messo in esecuzione da Runtime.getRuntime.exec(command) dove command corrisponde alla stringa necessaria per far eseguire il prompt .

Prima cosa ho dovuto capire come funzionassero le classi Process e Runtime e come riuscire a indicare al prompt in che directory dovesse trovarsi di volta in volta a seconda dell'esigenza.

Questo è stato risolto tramite PrintStream che prendeva in ingresso p.getOutpuStream che mi ha permesso di passare al processo i comandi che volevo inviargli.

Tramite thrmod riuscivo di volta in volta a vedere quali informazioni stavano passando attraverso lo stream.

Si è posto poi il problema di capire come stampare queste informazioni.

Per questo scopo è stata creata una inner class ReturnString che a seconda di quale metodo viene chiamato salva un messaggio di errore (dato da getErrorStream()) o una stringa contenente le informazioni che sono passate nell'InputStream del processo.

I metodi diversi di ReturnString vengono richiamati da thrmod a seconda del valore di una variabile intera impostata uguale a zero o maggiore di zero a seconda di quale informazione voglio venga visualizzata.

Con un'informazione contente un errore viene poi visualizzata una JOptionPane con un messaggio di errore altrimenti una JOPtionPane contenente un messaggio di buona riuscita.

La classe principale del Package Model è ModelImpl che implementa l'interfaccia Model. Questa classe ha il compito di eseguire l'operazione principale ovvero quella di verificare la presenza o meno dell'installazione di mercurial tramite un metodo booleano che riporta true o false

(check_Existence) al Control.

In questa classe sono inoltre presenti vari metodi (doLog(), doCommit(), docopy()…) che richiamano un metodo della classe CommandModelProject per fargli eseguire il comando da loro passato.

La classe CommandLog invece si occupa di salvare e passare al Control una lista di Stringhe contenente il log.

Quest'ultima viene richiamata da doLog, metodo presente anche esso in ModelImpl.

Conclusioni

La maggior parte delle ore del progetto sono state impiegate nella fase di progettazione. Infatti abbiamo impiegato molto tempo a capire quali classi dovessero essere usate per comunicare con il prompt.

Tramite forum vari di stackoverflow siamo riuscire a capire che doveva essere usata la libreria Process con Runtime e che il comando da eseguire per il prompt di windows dovesse essere “cmd”.

Ci sarebbe piaciuto renderlo eseguibile nei vari sistemi operativi ma purtroppo non siamo riusciti a individuare le informazioni adeguate per adempiere al compito.

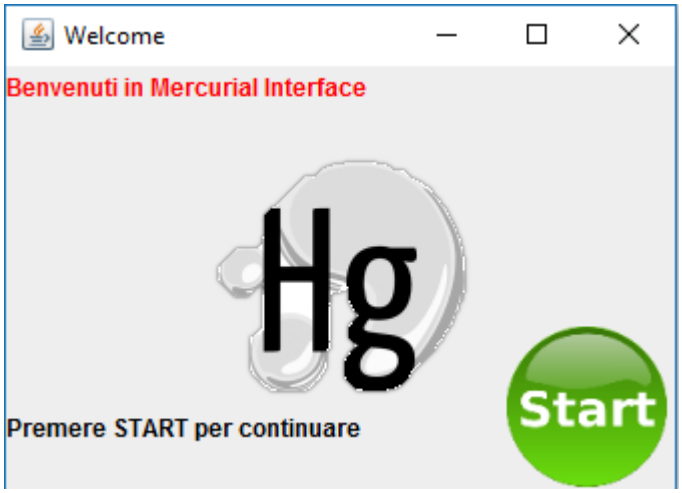
In ogni caso è stato scritto nel progetto nella parte model un metodo in grado di verificare in che sistema operativo si trova, in modo da poterlo migliorare in futuro.

Mentre per la parte visuale come già detto se ci fosse stato più tempo sarebbero stati aggiunti più dettagli, mentre è stato impiegato più tempo per l’efficienza e l’ordine del codice.

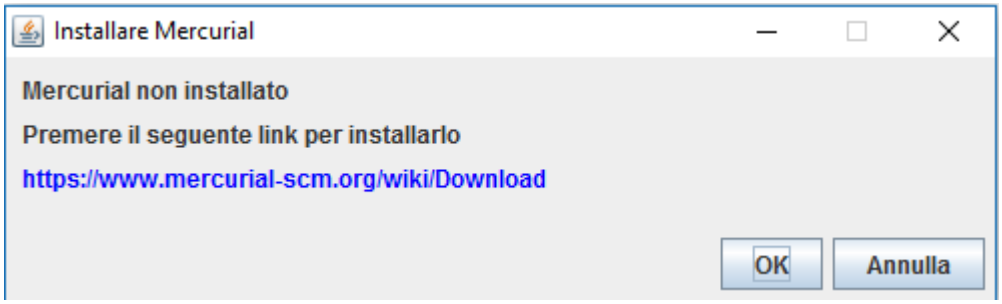
In ogni caso sono stati approfonditi molti concetti prima meno chiari e quindi è stato un accrescimento personale per entrambi.

Guida all’utilizzo

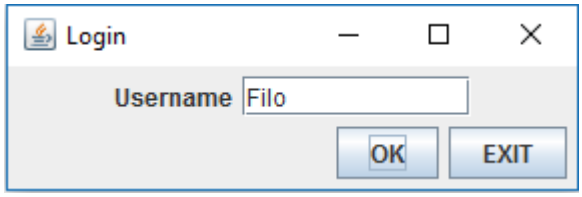
Questa è la GUI di Benvenuto.



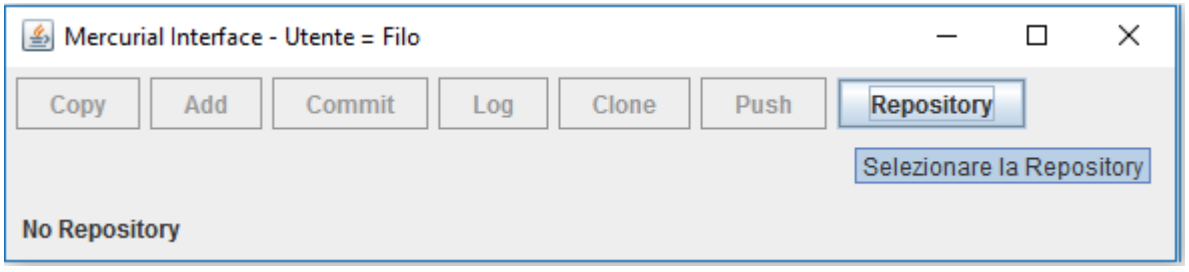
La seguente View si può visualizzare solo se non si ha Mercurial Installato.



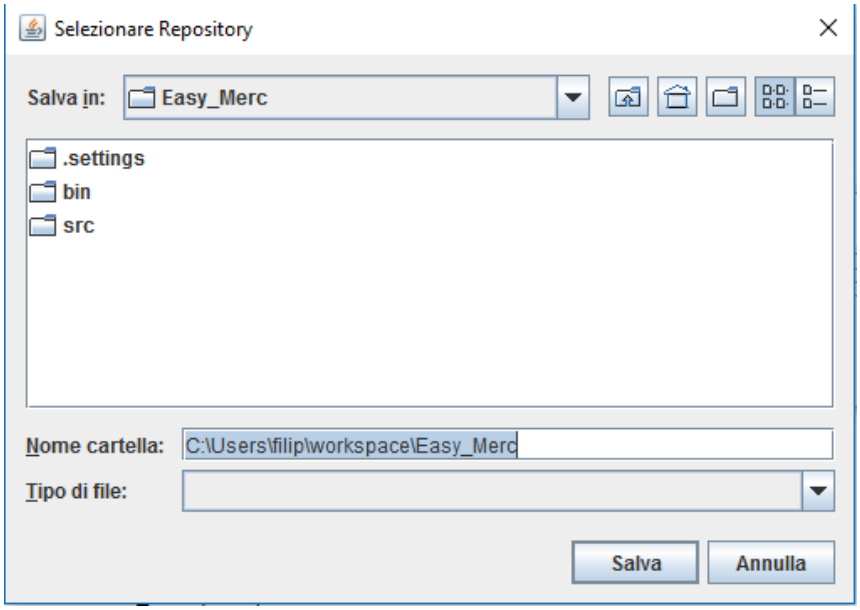
L’Interfaccia di Login dove si potrà inserire il proprio utente.



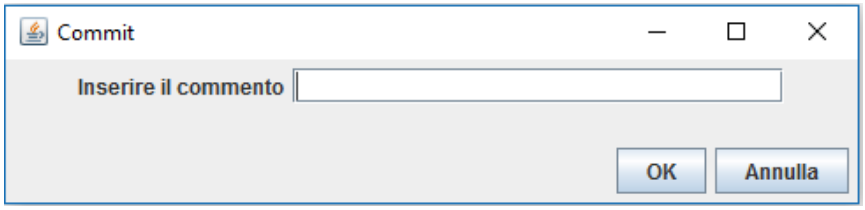
La Merc_View è la View principale con i bottoni ed i ToolTip.



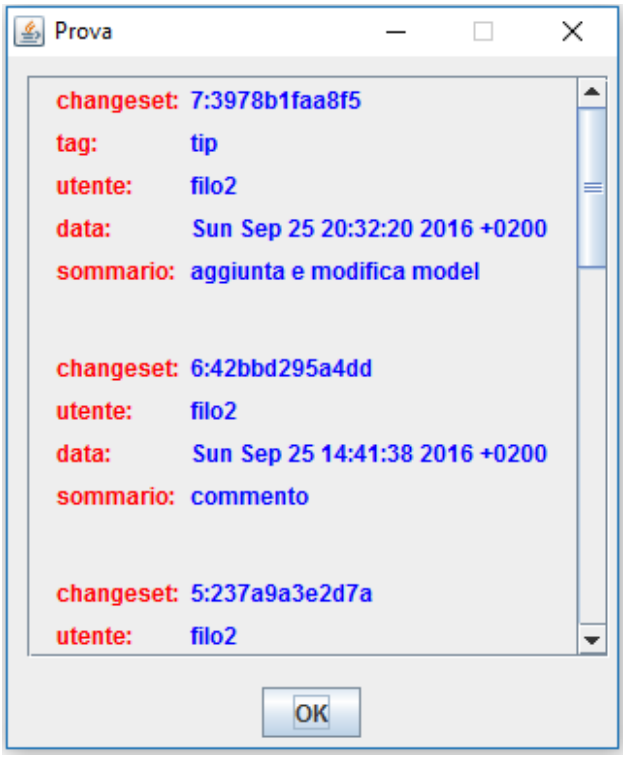
Questa GUI corrisponde allo sfoglia.

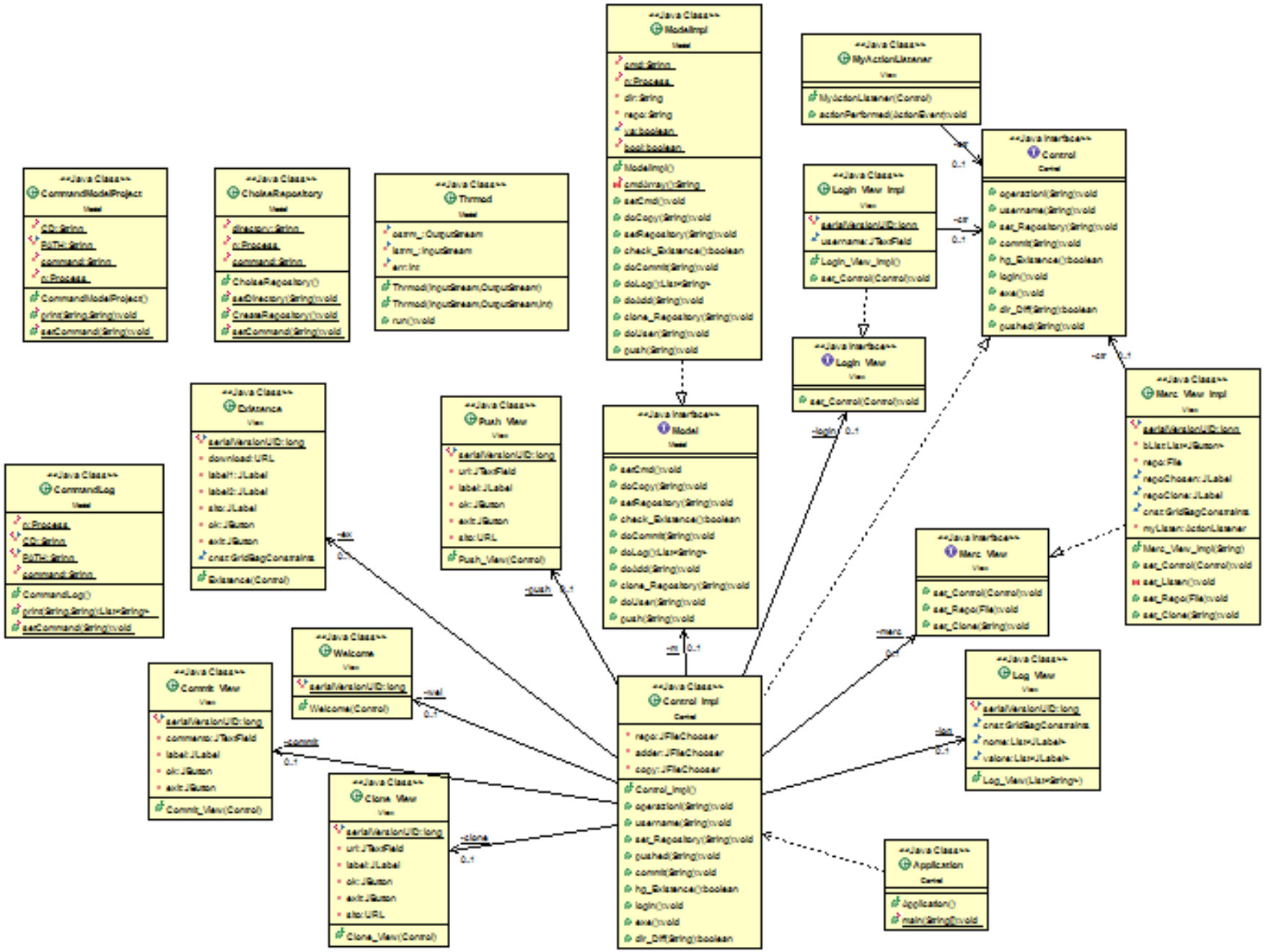


Commit_View, Clone_View e Push_View sono tutte visivamente simili.



La Log_View dove si può vede la cronologia.




«Java Class»
CommandModelProject
Model
CD: String
PATH: String
command: String
n: Process
CommandModelProject()
print(String,String)(void
setCommand(String)(void
«Java Class»
ChoiseRepository
Model
directory: String
n: Process
command: String
ChoiseRepository()
setDirectory(String)(void
CreateRepository()(void
setCommand(String)(void
«Java Class»
Thmod
Model
osnn_: OutputStream
isnn_: InputStream
err: Int
Thmod(InputStream,OutputStream)
Thmod(InputStream,OutputStream,int)
run()(void
«Java Class»
ModelImpl
Model
cmd: String
n: Process
dir: String
repo: String
ok: boolean
esci: boolean
ModelImpl()
cmdArray()(String
setCmd()(void
doCopy(String)(void
setRepository(String)(void
check_Esistance()(boolean
doCommit(String)(void
doLog()(List<String>
doAdd(String)(void
clone_Repository(String)(void
doUsan(String)(void
push(String)(void
«Java Class»
MyActionListener
View
MyActionListener(Control)
actionPerformed(ActionEvent)(void
«Java Interface»
Control
Control
operazioni(String)(void
username(String)(void
set_Repository(String)(void
commit(String)(void
hg_Esistance()(boolean
login()(void
exe()(void
dir_Diff(String)(boolean
pushed(String)(void
«Java Class»
Login_View_Impl
View
serialVersionUID: long
username: JTextField
Login_View_Impl()
set_Control(Control)(void
«Java Interface»
Login_View
View
set_Control(Control)(void
«Java Class»
Marc_View_Impl
View
serialVersionUID: long
bList: List<JButton>
repo: File
repoChosen: JLabel
repoClone: JLabel
cnst: GridBagConstraints
myUsan: ActionListener
Marc_View_Impl(String)
set_Control(Control)(void
set_Usan()(void
set_Repo(File)(void
set_Clone(String)(void
«Java Class»
Esistance
View
serialVersionUID: long
download: URL
label1: JLabel
label2: JLabel
sto: JLabel
ok: JButton
exit: JButton
cnst: GridBagConstraints
Esistance(Control)
«Java Class»
Push_View
View
serialVersionUID: long
url: JTextField
label: JLabel
ok: JButton
exit: JButton
sto: URL
Push_View(Control)
«Java Interface»
Model
Model
setCmd()(void
doCopy(String)(void
setRepository(String)(void
check_Esistance()(boolean
doCommit(String)(void
doLog()(List<String>
doAdd(String)(void
clone_Repository(String)(void
doUsan(String)(void
push(String)(void
«Java Class»
CommandLog
Model
n: Process
CD: String
PATH: String
command: String
CommandLog()
print(String,String)(List<String>
setCommand(String)(void
«Java Class»
Welcome
View
serialVersionUID: long
Welcome(Control)
«Java Class»
Commit_View
View
serialVersionUID: long
commento: JTextField
label: JLabel
ok: JButton
exit: JButton
Commit_View(Control)
«Java Class»
Clone_View
View
serialVersionUID: long
url: JTextField
label: JLabel
ok: JButton
exit: JButton
sto: URL
Clone_View(Control)
«Java Interface»
Marc_View
View
set_Control(Control)(void
set_Repo(File)(void
set_Clone(String)(void
«Java Class»
Log_View
View
serialVersionUID: long
cnst: GridBagConstraints
nome: List<JLabel>
valore: List<JLabel>
Log_View(List<String>)
«Java Class»
Control_Impl
Control
repo: JFileChooser
adder: JFileChooser
copy: JFileChooser
Control_Impl()
operazioni(String)(void
username(String)(void
set_Repository(String)(void
pushed(String)(void
commit(String)(void
hg_Esistance()(boolean
login()(void
exe()(void
dir_Diff(String)(boolean
«Java Class»
Application
Control
Application()
main(String[])(void
-ctr
0..1
-ctr
0..1
-ctr
0..1
-login
0..1
-ok
0..1
-push
0..1
-m
0..1
-marc
0..1
-log
0..1
-wel
0..1
-commit
0..1
-clone
0..1