# Esercitazione 2 - ACMEWex

0000217387 - Marco Santarelli

24 giugno 2005

## Indice



**Sommario**

Relazione inerente all'esercitazione 2.a: ACMEWex, un'applicazione per la gestione degli appelli d'esame fornita di controllo del log-in. ACMEWex ha la funzione di permettere agli studenti di registrarsi agli appelli in via informatizzata, previa apertura di questi da parte dei docenti. L'applicazione deve fornire funzionalitá relative alla registrazione agli appelli,alla visualizzazione delle statistiche di ogni appello e di ogni studente in base ai risultati ottenuti, la ricerca di appelli, l'apertura di questi, la modifica di tutti i dati immessi e un controllo degli accessi relativamente sicuro in modo da impedire "sabotaggi" del sistema, il calcolo della media dei risultati ottenuti. I dati devono essere memorizzati in maniera permanente in modo che alla chiusura dell'applicazione nulla vada perduto e che all'apertura venga restaurato lo stato dell'ultima esecuzione.

Codice implementato da Alai Giovanni, DeSanti Matteo, Santarelli Marco.

LaTeX

# 1 Analisi del problema

L'applicazione da costruire in questa seconda esercitazione si rivela da subito parecchio piú complessa di quella con la quale abbiamo avuto a che fare al nostro esordio, poiché va sviluppata in modo che sia possibile interagire con essa tramite interfaccia grafica. Non é solo questo aspetto peró che ne complica l'architettura: ci sono numerosi elementi da modellare come oggetti e non risulterá facile gestire una tale mole di elementi. Come al solito ci saranno due tipi di richieste di cui tener conto, quelle che il programma é esplicitamente tenuto a seguire e che derivano da direttive della consegna, e quelle riguardanti nostre particolari scelte di implementazione; passiamo a esaminarle entrambe.

## 1.1 Requisiti del progetto

Il nostro progetto dovrá tassativamente soddisfare 2 categorie di servizi:

*Per gli studenti*:

- Iscrizione ad un appello
- Visualizzazione elenco degli appelli disponibili, entro due date
- Visualizzazione risultato di un appello, se disponibile
- Visualizzazione elenco degli appelli sostenuti e media dei risultati (di quelli in cui é stato promosso)
- Statistiche: visualizzazione dell'andamento dei risultati conseguiti nel tempo

*Per i docenti*:

- Apertura (creazione) nuovo appello d'esame
- Rimozione / variazione appello d'esame esistente
- Visualizzazione elenco studenti iscritti ad un appello, con eventuale salvataggio su file
- Inserimento risultati relativi ad un appello
- Statistiche: visualizzazione andamento nel tempo dei risultati relativi agli appelli di una certa materia

distinguiamo quindi ció che dovremo tassativamente realizzare secondo queste direttive e ció che potremo realizzare come meglio crediamo in modo da esser certi di soddisfare tutte le richieste.

### 1.1.1 Richieste della consegna

Per poter soddisfare le funzionalitá richieste dovremo porre particolare attenzione alla modellazione delle nostre entitá sottoforma di oggetti: avremo bisogno di tre principali strutture:

- Una struttura che ci permetta di instanziare un appello, che tenga traccia di tutti i dati a esso appartenenti
- Una struttura che consenta di tener traccia di un numero a priori indefinito di appelli e che ci permetta di accedere a ciascuno di essi in maniera rapida e diretta
- Un insieme di strutture che ci permettano di gestire questi elementi tramite interfaccia grafica

Piú in particolare, per soddisfare le richieste, ci sono ulteriori accorgimenti da seguire:
L'appello, oltre al suo nome, tipo e luogo, dovrá tener traccia di data d'inizio, data di chiusura iscrizioni, numero massimo degli iscritti, elenco degli iscritti e loro voti, media di questi e vari metodi per poter impostare e recuperare questi dati, come per esempio qualche modo di ottenere il risultato che ha conseguito un particolare studente. Ad un esame piú attento ci siamo accorti che sarebbe stato necessario anche aggiungere un riferimento al docente che ha aperto l'appello, poiché é possibile che esistano due esami, della stessa materia, uguali in tutto e per tutto tranne che per appartenere a due docenti diversi, una possibile eventualitá. E' evidente peró che non si potrá fornire alcune funzionalitá come appartenenti al singolo appello, quali le statistiche generali.
L'elenco degli appelli deve fornire funzionalitá che permettano di recuperare un appello in modo rapido ed univoco una volta conosciuto il nome o la posizione nell'elenco, in modo che si possa quindi applicare tutti i metodi forniti sull'appello in questione. A questo livello é possibile implementare metodi come per esempio la costruzione delle statistiche

da visualizzare per il docente o la media dei risultati dello studente, poiché abbiamo una visibilitá completa di tutti gli appelli presenti nel nostro database. Potremo quindi applicare a questo livello il controllo richiesto esplicitamente che non sia presente piú di un appello con la data dell'esame fissata in un dato giorno.
La struttura del programma deve soddisfare il modello Model-View-Controller, come richiesto da consegna. Sará necessario quindi dare alla struttura che deve gestire le entitá di cui sopra (che fanno parte del Model) una netta distinzione tra ció che riguarda l'interfaccia in sé (View) e ció che é richiesto per farla funzionare correttamente e fornire quelle funzionalitá piú generali non applicabili come metodi delle singole entitá(ció che viene detta struttura Controller). Dobbiamo inoltre osservare la richiesta della persisenza dei dati, quindi a questo livello si renderá necessario implementare metodi che permettano di salvare in maniera totale i dati accumulati durante l'esecuzione del programma e che ci permettano poi di recuperarli alla riapertura successiva dell'applicazione.

### 1.1.2 Decisioni autonome sulla progettazione

La complessitá dell'applicazione tira in ballo molto piú spesso che in precedenza la nostra inventiva per trovare e scegliere il metodo migliore di realizzare ció che ci viene richiesto, compito peraltro non sempre facile senza accettare dei compromessi in questo caso. Prendiamo in esame innanzitutto la natura dell'applicazione: deve fornire due tipi diversi di servizi in base all'utente che ne fa uso: ci risulta quindi naturale sviluppare come prima cosa un modulo di riconoscimento utente con username e password che a seconda del tipo di utente loggato fornisce servizi differenti e un metodo che permetta la registrazione autonoma di uno studente: nonostante non sia richiesto esplicitamente aggiungeremo inoltre anche una classe che rappresenta un Amministratore del sistema: in questo modo potremo registrare nuovi docenti e nuovi amministratori solo tramite un tipo account privilegiato, impedendo che chiunque possa registrarne uno ed effettuare operazioni pericolose per l'integrá dei dati del programma.[1]
Si renderá quindi necessario costruire astrazioni di Studenti,Docenti e Amministratori e una struttura che possa contenere i dati di tutti, una sorta di database degli utenti. Per precauzione poi, si dovrá trovare il modo di poter controllare la password senza che questa sia di pubblico dominio, anche se la sicurezza dei dati non é il nostro obbiettivo.
Nella progettazione dell'interfaccia grafica, inoltre, sará bene cercare di eliminare il piú possibile, se non definitivamente, la possibilitá che l'utente cerchi di effettuare operazioni non consentite, come l'inserimento di una data sbagliata[2] scegliendo in maniera oculata il metodo migliore di raccolta delle informazioni sotto questo punto di vista: nel caso della data, per esempio, sarebbe opportuno far scegliere all'utente il giorno tra una lista prefissata che cambia in base al mese e all'anno prescelti(anche questi da una lista se possibile)in modo da non essere costretti ad effettuare dei controlli su ogni dato immesso, ma solo quelli strettamente necessari, che sono giá di per sé una mole consistente, mentre cercare di applicare il piú possibile il principio della scelta tra elementi decisi a priori, in modo da poter prevedere, pur sempre fino a un certo limite, l'evolversi dell'interazione con l'utente ed evitare gli intoppi che ne possono derivare.

Possiamo elencare peró, per sottolinearne l'importanza, quei controlli ai quali non si potrá rinunciare e che sará bene inserire in fase di progettazione in tutti i luoghi opportuni

- In tutti gli inserimenti di dati come quelli relativi alla registrazione o all'apertura di un nuovo appello controllare che si sia effettivamente inserito del testo per evitare la creazione di utenti o appelli senza nome e di campi nulli per errore.
- Nella registrazione all'appello controllare che la data odierna non sia dopo la chiusura delle iscrizioni
- Nella registrazione all'appello controllare che il numero massimo degli iscritti non venga superato
- Nella registrazione all'appello controllare che lo studente non sia giá iscritto per evitare di iscriverlo 2 volte o di sovrascrivere il voto
- Nell'assegnamento dei voti controllare che l'esame sia giá stato sostenuto
- Nel calcolo della media considerare solo gli utenti che hanno sostenuto l'esame ed ignorare coloro che si sono ritirati
- Nell'apertura di un'appello controllare che gli iscritti massimi non siano pari a zero

[1] Al momento in cui scrivo non era ancora disponibile il testo della 3a esercitazione
[2] Per esempio il 30/02/2004

- Nell'apertura di un'appello controllare che le date di chiusura iscrizioni e di inizio esame siano nel giusto ordine cronologico e che siano dopo la data odierna

Il resto dei controlli sulla validitá dei dati risulterá piú comodo effettuarlo in maniera implicita nella costruzione dell'interfaccia grafica: per esempio non sará assolutamente necessario controllare se l'utente che tenta di registrarsi esista nel database, poiché, per via dell'architettura di autenticazione degli utenti, sará impossibile che un utente non presente nel database tenti di registrarsi a un appello.

# 2 Progetto della soluzione

La parte piú difficile della progettazione, una volta definita la struttura complessiva, sará quello di trovare il modo di gestire in maniera efficiente tutti gli elenchi di cui dovremo disporre, poiché non risulterá sempre immediata la costruzione di un metodo semplice e allo stesso tempo rapido che permetta di recuperare i vari elementi dall'insieme. La struttura dati di cui faremo largo uso a questo scopo é l'HashMap della classe Java.util, poiché permette l'inserimento di elementi di tipo non eterogeneo in un elenco tramite l'associazione a chiavi definibili anche dinamicamente a run-time: la sua flessibilitá ci tornerá molto utile per gestire i dati della nostra applicazione. Ci tornerá utile inoltre costruire una classe che rappresenterá le date all'interno del nostro programma, in modo da costruirci metodi e costruttori su misura e personalizzati che ci consentano di evitare l'uso di quelli complessi e articolati della classe *Calendar* e di quelli ideali per il nostro scopo ma deprecati della classe *Date* giá esistenti in Java. Sará opportuno poi inserire in una classe a parte metodi di utilitá che per loro natura non si possono definire come appartenenti a nessuna delle entitá prima descritte, ma che vengono utilizzati in caso di necessitá all'interno di numerose classi.

## 2.1 Model-View-Controller

Lo schema UML delle classi e delle loro interdipendenze é contenuto nel file *UML.png* all'interno della cartella doc. Non é stato incluso direttamente nella relazione per motivi di comoditá a causa delle sue elevate dimensioni dovute all'elevato numero di entitá da rappresentare. Nello schema ho omesso per chiarezza alcune classi riguardanti la grafica e le informazioni dettagliate per alcune classi poiché non arricchivano lo schema in alcun modo e lo rendevano, anzi, meno immediatamente comprensibile. Come é possibile notare, vi é una netta separazione tra le parti: le varie classi si possono dividere nelle tre categorie Model, View e Controller, come richiesto, nel seguente modo:

- Nella categoria Controller rientrano le classi *Controller* e *Listener* che descriveremo piú avanti in dettaglio,che comunque svolgono funzioni generali quali il controllo del login e la gestione degli appelli la prima, la gestione di tutti gli eventi generati dal programma la seconda.
- Nella categoria View si annovera la classe omonima, che si occupa del disegnare a video i dati e la finestra principale, e le due piccole classi *StudenteStats* e *DocenteStats*, che, come si puó evincere dal nome, ci torneranno utili per disegnare le statistiche riguardanti studenti e docenti.
- Nella categoria Model si possono inserire quasi tutte le rimanenti classi.

## 2.2 Progettazione interna delle singole entitá

### 2.2.1 Modellazione utenti e interdipendenze

Per la gestione del log-in abbiamo avuto la necessitá di creare classi apposite per distinguere tra i vari utenti. Dato peró che gli unici metodi che differivano tra utente e utente erano quelli per calcolare lo username in base ai dati immessi e, solo per lo studente, un campo addizionale per la matricola, abbiamo ritenuto opportuno estendere una classe base chiamata User e specializzarla in base alle nostre esigenze per Studenti, Docenti e Amministratori. Abbiamo trovato che la specializzazione era una soluzione migliore rispetto all'uso di un interfaccia(che comunque é stata costruita per specificare il contratto che la classe user doveva soddisfare) implementata dalle 3 classi, poiché in tal modo avremmo dovuto riscrivere il codice anche per i metodi comuni. In tal modo possiamo distinguere i vari tipi di utente oltre che al tipo di username, anche dalle alle classi con le quali sono descritti. I metodi sono basilari, la modifica dei dati personali e la restituzione di essi é banale, l'unica caratteristica interessante é il cambiamento e la verifica della password, che per sua natura deve rimanere nascosta una volta definita: da notare infatti come i metodi *setPassword* e *getPassword*, i complementi logici di tutti gli altri metodi della classe, siano assenti o diversi:infatti al posto di *getPassword* compare *parsePassword*, un metodo che restituisce un booleano che vale *true* solo se la stringa passata come argomento é uguale alla password memorizzata. La particolare architettura di *setPassword* si differenzia dagli altri metodi *"get"* per il

fatto che richiede come argomento la vecchia password per poterla cambiare. Per i dettagli implementativi si rimanda alla sezione apposita.

### 2.2.2 Database degli utenti

I vari utenti, all'atto della registrazione, sono immagazzinati in una struttura definita dalla classe usersDb, la quale utilizza un HashMap per tener traccia di tutti gli utenti registrati, immettendoli in questa struttura dati, che per sua natura non ha un ordine ben preciso, associando a ciascun utente come chiave il proprio id. La classe deve altresí implementare i metodi per il salvataggio e il caricamento del database e metodi per l'inserimento, la rimozione e la restituzione di un utente, in modo che una volta inserito in archivio un utente sia ancora possibile effettuare tutte le operazioni di modifica dei dati personali su di esso e il controllo della password.

### 2.2.3 Oggetto appello

Anche la classe appello é soprattutto una raccolta di campi da memorizzare: sono importanti quindi i metodi di salvataggio e caricamento, costruiti in modo da creare un nuovo file per ogni appello, il cui nome viene modificato[3] in caso di cambiamento dei dati. Come preannunciato in precedenza, anche qui si fará utilizzo di HashMap per tenere traccia delle iscrizioni e dei voti, in particolare si cercherá di ridurre al minimo la ridondanza utilizzandone una unica in cui all'atto di iscrizione di un utente si aggiungerá nella chiave corrispondente al suo id un intero pari a -1, che sta a indicare che non ha ancora sostenuto l'esame, che eventualmente verrá mutato con i metodi opportuni per settare i voti, in un intero dallo 0 al 31 che rappresentano i voti da 0 a 30 e lode.

### 2.2.4 View,StudenteStats,DocenteStats

Poiché la nostra intenzione per l'aspetto grafico dello UI era di avere un'unica finestra ridisegnata di volta in volta, piuttosto che estendere le classi JPanel e JFrame abbiamo preferito un'unica classe a Frame fisso il cui pannello, sempre lo stesso, viene ridisegnato ogni volta che si invoca una delle procedure della classe, che ci permettono di mantenere unicamente in questa classe il codice che effettua modifiche all'ambiente grafico e di inserire nella classe che fungerá da listener solo il richiamo dei metodi. I numerosi campi pubblici della classe devono essere tali per permetterci di recuperare i dati dai vari componenti delle finestre dall'esterno della classe, poiché la classe View é e deve essere solo deputata alla gestione grafica. Nell'implementazione finale di questa classe si deve cercare di sostituire ove possibile gli inserimenti di dati tramite *TextField* con delle scelte multiple tramite *ComboBox*, in modo da seguire le direttive anticipate nell'analisi ed evitare di dover gestire dei controlli sui dati immessi che é possibile aggirare tramite un accurato studio dello UI.

### 2.2.5 Listener,Controller

La classe Listener, come si puó arguire dal poco fantasioso nome, non é altro che un unico listener che implementa numerose interfacce per la gestione di alcuni tipi di eventi, e si occupa della gestione di questi richiamando le operazioni necessarie dal Controller. Quest'ultimo non é altro che la classe principale che si occupa di istanziare gli oggetti e che contiene i metodi di uso generale. Si é preferito inserire qui la struttura deputata al contenimento degli appelli e i metodi ad essa relativi, per non far levitare troppo il numero delle classi da costruire.

## 2.3 Eventi

La dinamica di funzionamento di tutto il sistema é relativamente semplice: viene innanzitutto eseguito il controller, che per prima cosa carica da disco i vecchi dati salvati in precedenza e ne ricostituisce lo stato, in seguito apre la finestra dell'interfaccia grafica e vi registra il listener. Quindi a questo punto il programma rimane in attesa di azioni dell'utente, il clic del mouse su un tasto, la chiusura della finestra principale, e cosí via...non appena al listener viene notificato di uno di questi eventi questi si comporta di conseguenza a seconda di quale tasto é stato cliccato, in quale *ComboBox* é cambiata la scelta e via dicendo. La catena dell'interazione si conclude solo quando viene scelto il tasto "Uscita" o viene chiusa la finestra principale: sia nel primo che nel secondo caso il listener reagisce con il salvataggio dei dati aggiornati all'ultima modifica effettuata prima della chiusura e il programma si chiude, lasciando il sistema pronto per una nuova esecuzione.

[3] In realtá per praticitá il vecchio file col vecchio nome viene cancellato e al suo posto viene salvato un nuovo file con il nome opportuno, si rimanda ai dettagli implementativi anche in questo caso

# 3 Sviluppo delle classi in Java

Per l'illustrazione in dettaglio dei sorgenti delle classi rimando alle classi stesse, che sono state commentate accuratamente, e alla documentazione della classe in formato *javadoc*. Il codice é troppo lungo e complesso per essere commentato in dettaglio sulla relazione, risulterebbe assai scomodo saltare da una classe all'altra a ogni rimando o esecuzione di procedure, per questo motivo si é commentato minuziosamente il codice a un livello che sarebbe stato impossibile raggiungere descrivendolo passo passo in queste pagine, ottenendo un risultato migliore e piú comodo da consultare, magari con il diagramma UML a portata di mano.

# 4 Test e casi d'uso - Database iniziale

Il programma viene fornito con un database giá ricco di dati immessi a priori, spesso senza utilizzare l'interfaccia ma per altre vie per poter tentare di forzare aspetti non osservabili tramite la normale interazione. Tramite questo database é stato possibile evidenziare alcuni difetti, che sono stati corretti, ma é ancora a disposizione per poter effettuare prove in maniera rapida, senza doversi soffermare a crearne uno. Alleghiamo nel file Log-in.txt alcune coppie di nome utente e password per consentire a chi di dovere di verificare l'efficienza del programma[4] e alcune classi di test eseguibili che portano a termine alcune operazioni stampandone in console il risultato.

# 5 Complessitá

Con il crescere della complessitá delle applicazioni da realizzare ci siamo resi conto di quanto siano importanti al fine di compiere un buon lavoro le fasi di analisi e progettazione. É fondamentale decidere a priori almeno la struttura portante dell'applicazione per non dover poi incappare in problemi insormontabili che potrebbero portare a una riscrittura quasi totale del codice e a un inutile dispendio di tempo prezioso. La complessitá porta anche all'utilizzo sempre piú spinto e frequente dei principi di ereditarietá e subclassing quando sono applicabili, poiché possono risparmiare un sacco di tempo a chi deve programmare il sistema e ne diminuiscono spesso la complessitá e la mole. Abbiamo poi fatto i primi esperimenti con la grafica, un ambiente che permette innumerevoli e illimitati modi di lavoro, e abbiamo imparato a muovervi i primi passi. L'ultimo concetto nuovo ed importante che é entrato in gioco in questa esercitazione é l'architettura ad eventi: l'accorgersi che il flusso delle operazioni non é piú ordinato da un corso logico ininterrotto e ben definito ma per cosí dire é a tratti "sospeso" per riprendere il suo corso non appena l'utente compie la sua scelta consente di comprendere pienamente i meccanismi che dominano i sistemi piú complessi.

[4] Leggere attentamente le istruzioni comprese nel file