Corso di programmazione ad oggetti – Anno 2013/2014
Corso di laurea Ingegneria e Scienze Informatiche - Università di Bologna, sede di Cesena.

Realizzazione di jrefsystem: un'applicazione per la gestione delle attività di un arbitro.

Componenti: Mattia Borrillo.

• Analisi del problema

E' stata pensata la realizzazione di un'applicazione per la gestione elettronica delle attività di un arbitro di calcio.
A tal proposito si individuano molte funzionalità; guardando verso il lato gestionale delle attività agonistiche effettuate da tale figura è stato deciso di gestire tre categorie di "eventi": le partite, gli allenamenti e i test atletici a cui mensilmente un arbitro si sottopone per poter avere l'abilitazione ad effettuare la propria professione.
Delineando queste tre categorie che verranno gestite, questa è una lista generale delle funzionalità che si è deciso di implementare:

• Aggiunta delle partite a un archivio storico.
• Vista dell'archivio storico delle partite filtrate per opzioni selettive.
• Visualizzazione delle statistiche per ogni squadra e categoria.
• Gestione dei rimborsi spese delle partite.
• Gestione degli allenamenti: aggiunta e vista di archivio con le descrizioni.
• Gestione delle prove di test atletico jo-jo : aggiunta e visualizzazione grafica dei risultati.

Il tutto è fornito con un servizio di registrazione al sistema mediante una username e una password che daranno contemporaneamente una funzione di gestione multiutente e una protezione dei dati personali di chi usufruisce del servizio.

I dati saranno caricati e salvati in modo automatico all'apertura e alla chiusura dell'applicazione ma sarà possibile specificare dove e come salvare e/o caricare i dati.

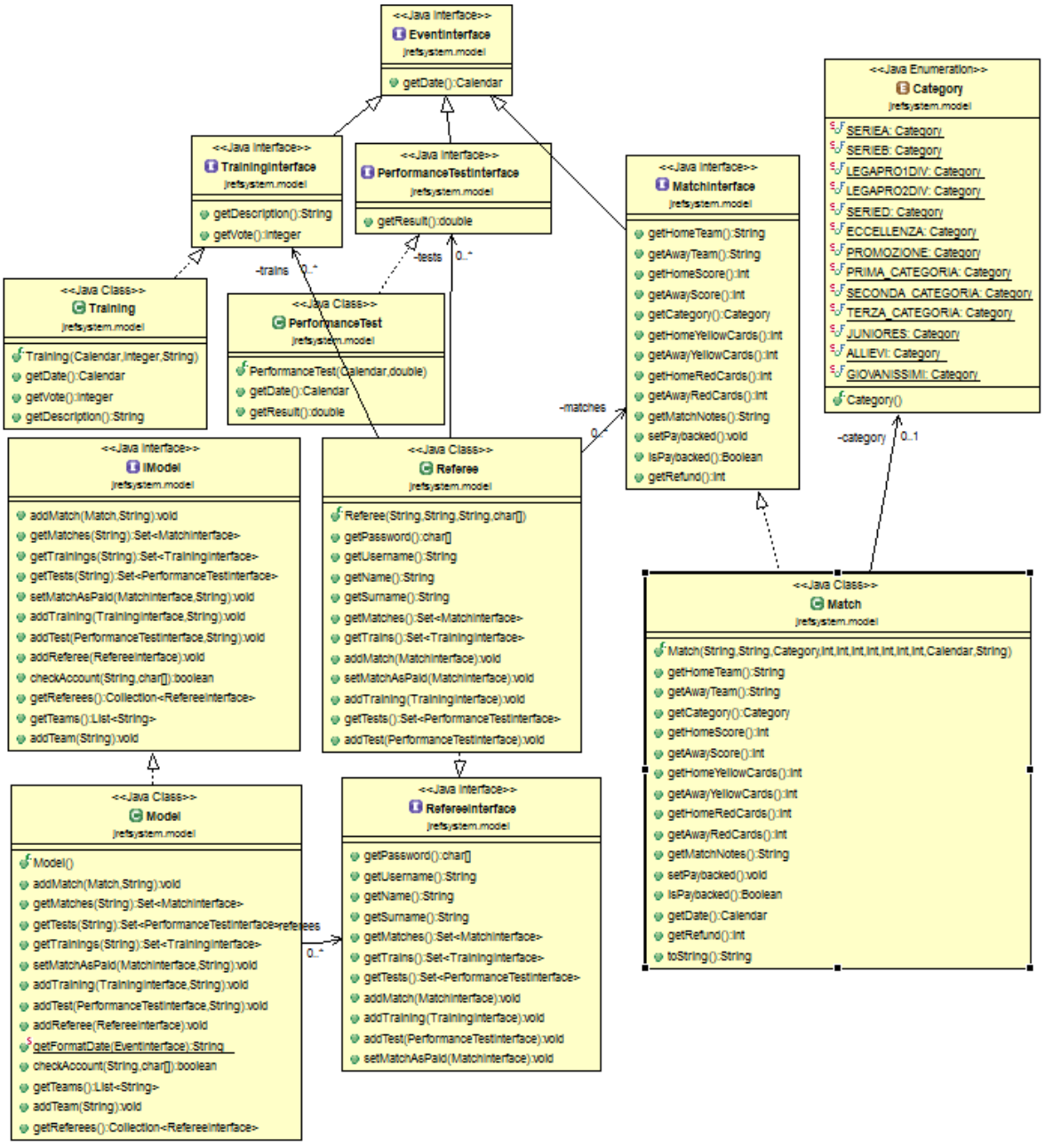
Per la realizzazione delle funzionalità sopra citate è stato necessario implementare un'applicazione GUI based che si presenterà con un pannello di login il quale darà la possibilità di registrarsi in caso sia la prima volta che si usufruisca del servizio oppure di inserire le proprie credenziali.
Una volta verificate queste ultime, all'utente si aprirà una vista che darà la possibilità di gestire tutti i servizi elencati.

- # Progettazione architetturale

In questa fase si sono prese le decisioni per come costruire effettivamente l'applicazione a livello strutturale: si è deciso per l'uso dello schema MVC.
Lo schema di progettazione MVC riesce a fornire una netta separazione fra le parti di model, view e controller regolando allo stesso tempo le interazioni fra le parti stesse; ora analizzeremo volta per volta ognuna delle tre parti per poi passare ad analizzare le loro interazioni con un esempio che riguarderà l'aggiunta di una partita al sistema.

- ***Model.***

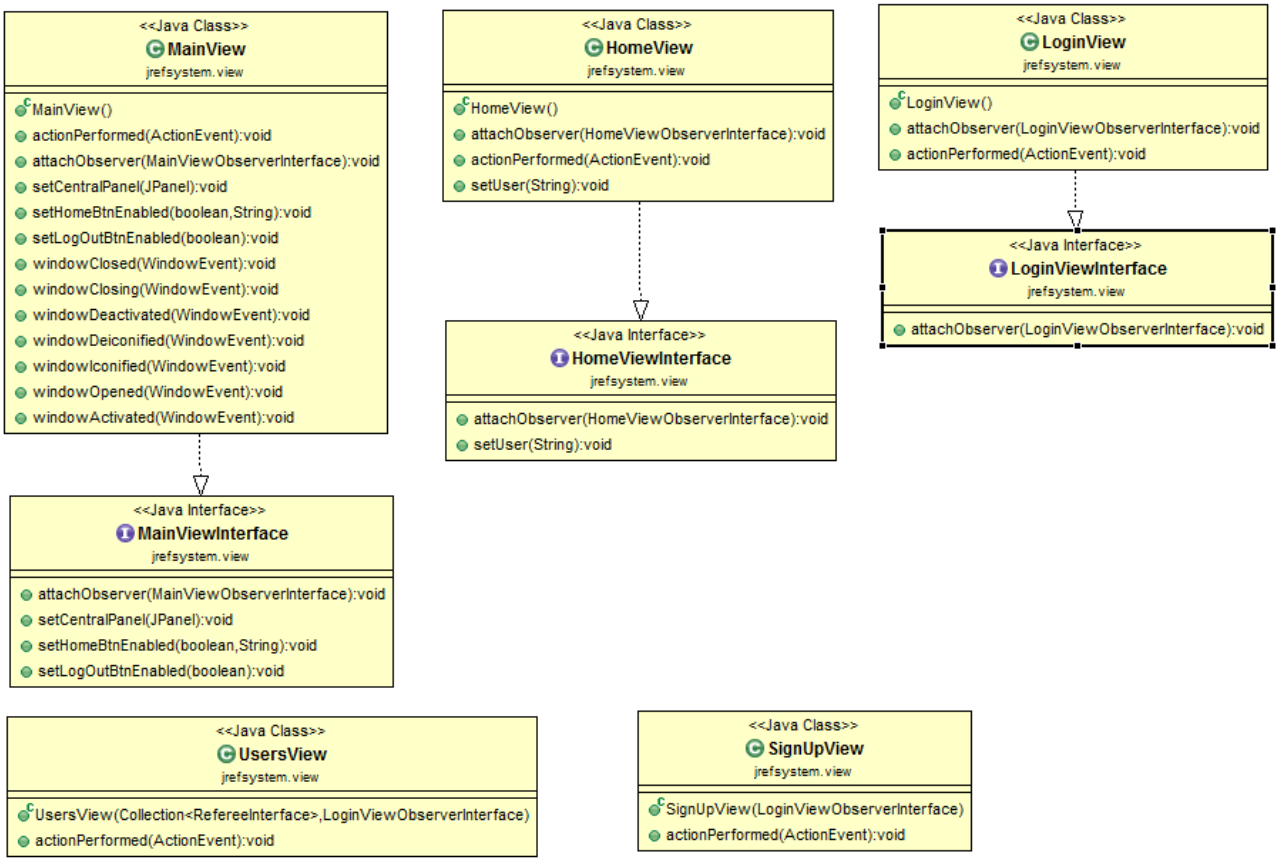
E' fornito lo schema UML della parte di model dell'applicazione.



Nello schema si può notare un'interfaccia chiamata "EventInterface" che fornisce il punto di raccordo fra le tre entità individuate in fase di analisi: allenamenti, test e partite.
Per ognuna di queste è gestita un'opportuna interfaccia implementata dalla rispettiva classe (Es. la classe Match implementa l'interfaccia MatchInterface) , esiste poi la classe "centrale" (e dico così perché ha riferimenti a ognuna delle 3 classi che implementano le entità partita, test e allenamento) Referee che implementa l'entità arbitro e la rispettiva interfaccia RefereeInterface.
Questa classe , come detto, gestisce collezioni di partite, allenamenti e test; e la classe Model che implementa l'interfaccia IModel si compone di una collezione di Referee.
Quest'ultima classe, il Model, è la classe di riferimento la cui interfaccia pubblica è composta da metodi che permettono di interagire con i dati tramite operazioni come per esempio aggiungere una partita o un altro evento a un arbitro specifico.
Bisogna fare una puntualizzazione sulle entità Category e su una, che poi si è deciso di non implementare, Team : per la prima si è optato per un enum in quanto fornisce una certa normalizzazione a un'informazione che si mantiene costante nel tempo (le categorie del mondo calcistico a meno di riforme radicali rimangono quelle) , per la seconda si è deciso di gestire una collezione all'interno della classe Model stessa in quanto permette una maggiore flessibilità a quella che è una gestione delle squadre che , soprattutto, nelle categorie minori tendono a cambiare, fallire , essere ridenominate e quant'altro.

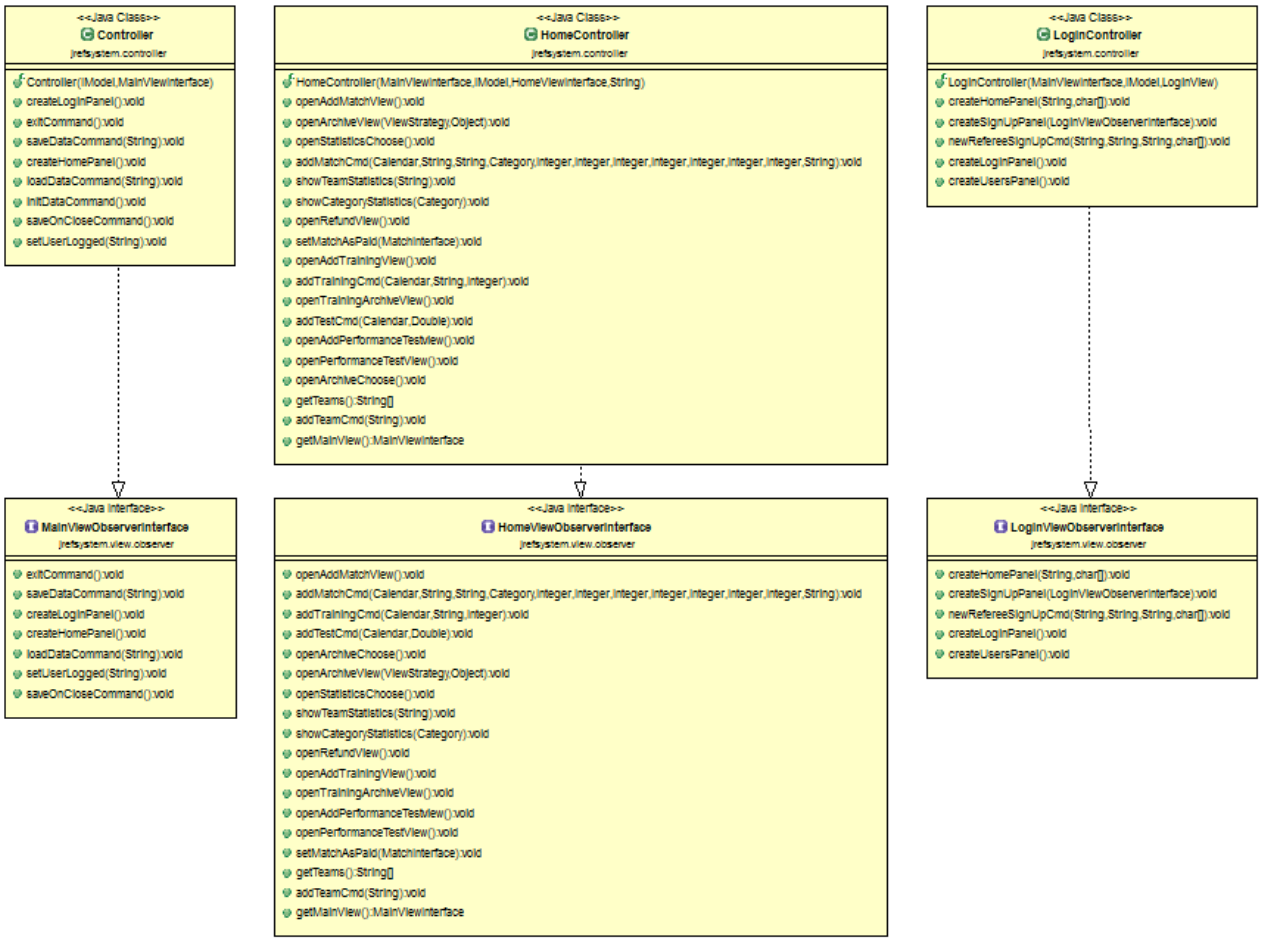
- ***View.***

E' fornito lo schema della parte (principale) di view dell'applicazione.



Le classi MainView, HomeView, LoginView, realizzate implementando le corrispondenti interfacce, sono le classi principali della parte di view dell'applicazione.
La MainView rappresenta la vista principale ed è creata all'apertura dell'applicazione per rimanere attiva fino alla sua chiusura, questa è l'unica classe che estende JFrame mentre le altre classi estendono JPanel e sono dei parametri passati al metodo setCentralPanel(JPanel) della MainView nel momento in cui debbano essere effettivamente visualizzate a video.
La classe LoginView è la vista che permette l'interazione con una finestra di login dalla quale l'utente può usare le proprie credenziali per entrare nel sistema oppure può registrarsi al servizio, funzione, quest'ultima, della quale si occupa la SignUpView.
La UsersView dà, invece, una vista degli utenti registrati.
La HomeView è la vista che "appartiene" a un preciso utente loggato nel sistema e che offre la possibilità, richiamando poi le rispettive view che saranno elencate in modo dettagliato in seguito, di scegliere quale operazione compiere.

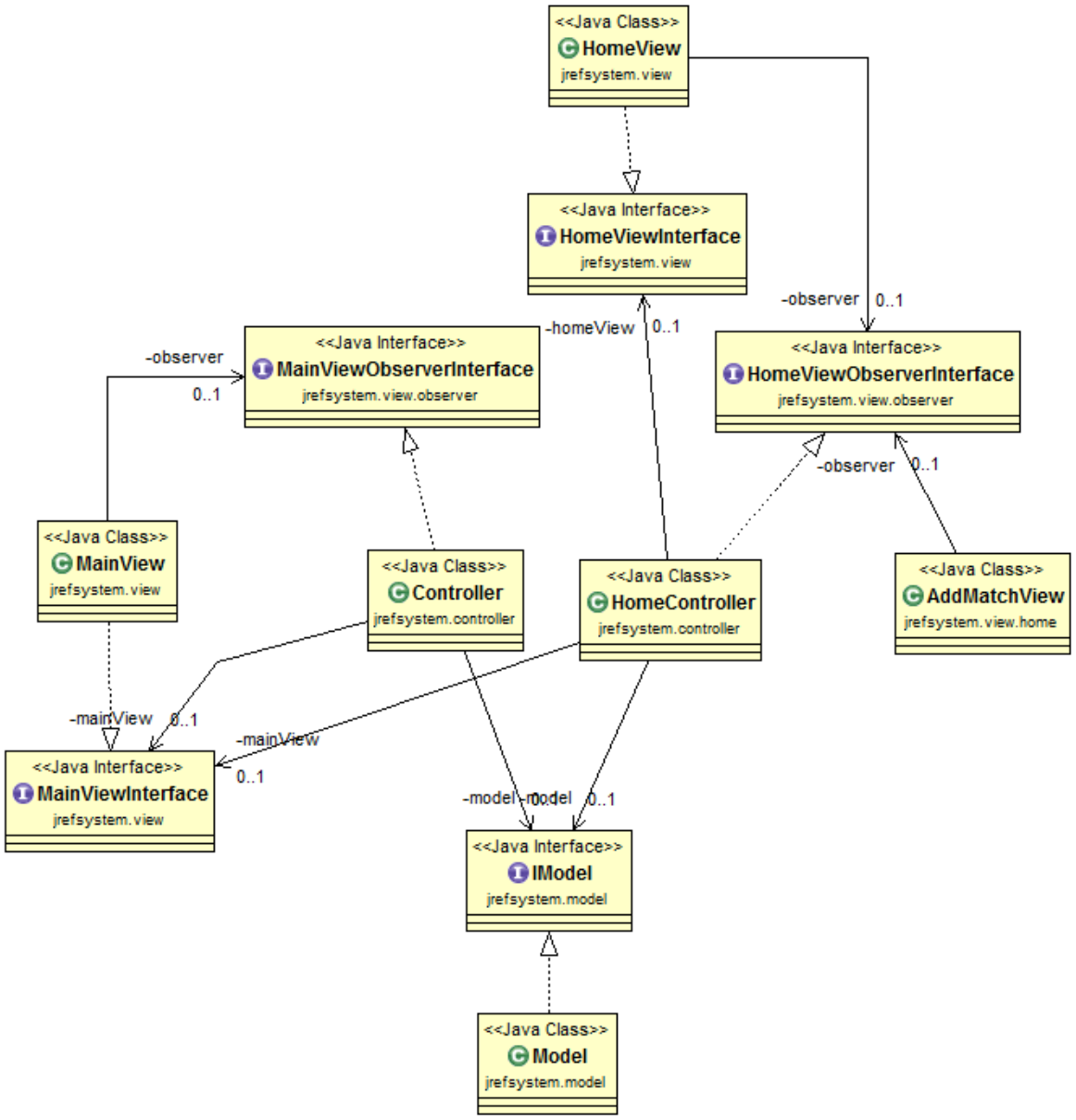
- ***Controller.***



Per quanto riguarda la realizzazione del controller si è deciso di utilizzare il pattern observer, quindi ognuno dei tre controller presenti all'interno dell'applicazione implementa un'interfaccia che ogni vista principale corrispondente aggancerà come osservatore nel momento in cui sarà creata l'istanza della classe.
Quindi per esempio, ma vale per tutte e tre le classi controller, la classe "Controller" implementa l'interfaccia "MainViewObserverInterface" che è a una sua volta parametro del costruttore della classe "MainView" ; sarà quindi , il controller, l'osservatore degli eventi generati dall'istanza della classe "MainView".
Sono stati individuati tre tipi di controller: il primo è il controller principale associato alla vista principale che viene creata all'avvio dell'applicazione mentre gli altri due sono i controller associati alle viste "HomeView" e "LoginView" e saranno generati nel momento in cui le corrispondenti viste saranno chiamate dagli eventi generati dall'utente durante l'utilizzo dell'applicazione per raccogliere a loro volta i comandi generati dalle view e poter, in accordo con la struttura MVC, manipolare la view e i dati contenuti nel model dell'applicazione.
L'"HomeController" fornisce supporto alle operazioni di un utente registrato che ha effettuato il login come per esempio aggiungere una partita o aprire una specifica vista di archivio.
Il "LoginController" fornisce supporto alle operazioni che hanno a che fare con il tentativo di login di un utente o la registrazione di un nuovo utente e genera, dopo un login a andato a buon fine, una istanza della classe "HomeController" passando a quest'ultima il controllo delle operazioni.
Il "Controller" è la classe principale che fornisce supporto alle operazioni che hanno a che fare con apertura chiusura dell'applicazione e con il salvataggio dei dati.

- ***Visione d'insieme di una piccola parte di MVC.***

E' presentata di seguito una sezione UML dell'interazione fra le tre componenti dell'MVC. (Sono state usate le sole intestazioni delle classi e delle interfacce per questione di spazio.)



In particolare questa figura riguarda lo schema che si ha quando si esegue l'operazione di aggiunta di un match; abbiamo i due controller (Controller e HomeController) che implementano le interfacce MainViewObserverInterface e HomeViewObserverInterface referenziate a loro volta dalle classi appartenenti alla parte view (MainView, HomeView e AddMatchView) che in complesso realizzano il pattern observer.
A loro volta i controller referenziano il Model dell'applicazione.
In sostanza i controller sono in grado di modificare le view e il model in base al tipo di comando ricevuto dalle view stesse.
Questi comandi sono forniti dall'insieme dei metodi pubblici delle interfacce MainViewObserverInterface e HomeViewObserverInterface.

Per quanto riguarda la funzione di aggiunta di un match la cronologia delle operazioni è la seguente: partendo dal presupposto che l'istanza della Classe Controller abbia generato, in seguito al login di un utente, le istanze delle classi HomeView e HomeController, quando un utente chiederà di poter aggiungere un match al Model l'opportuno comando eseguito dall'HomeController creerà la AddMatchView mostrandola a video.
A sua volta, dopo aver raccolto i dati del match, questa genererà attraverso il proprio osservatore implementato dall'interfaccia HomeViewObserverInterface e quindi dall'HomeController stesso un comando che sarà gestito da quest'ultimo e che modellerà i dati per poter aggiungerli al Model e comunicherà a video l'avvenuta operazione.
In questa semplice operazione si può quindi notare il funzionamento del modello MVC e la centralità che il controller ha nel ricevere comandi, eseguirli e aggiornare le altre due componenti.

- **Divisione in package**.

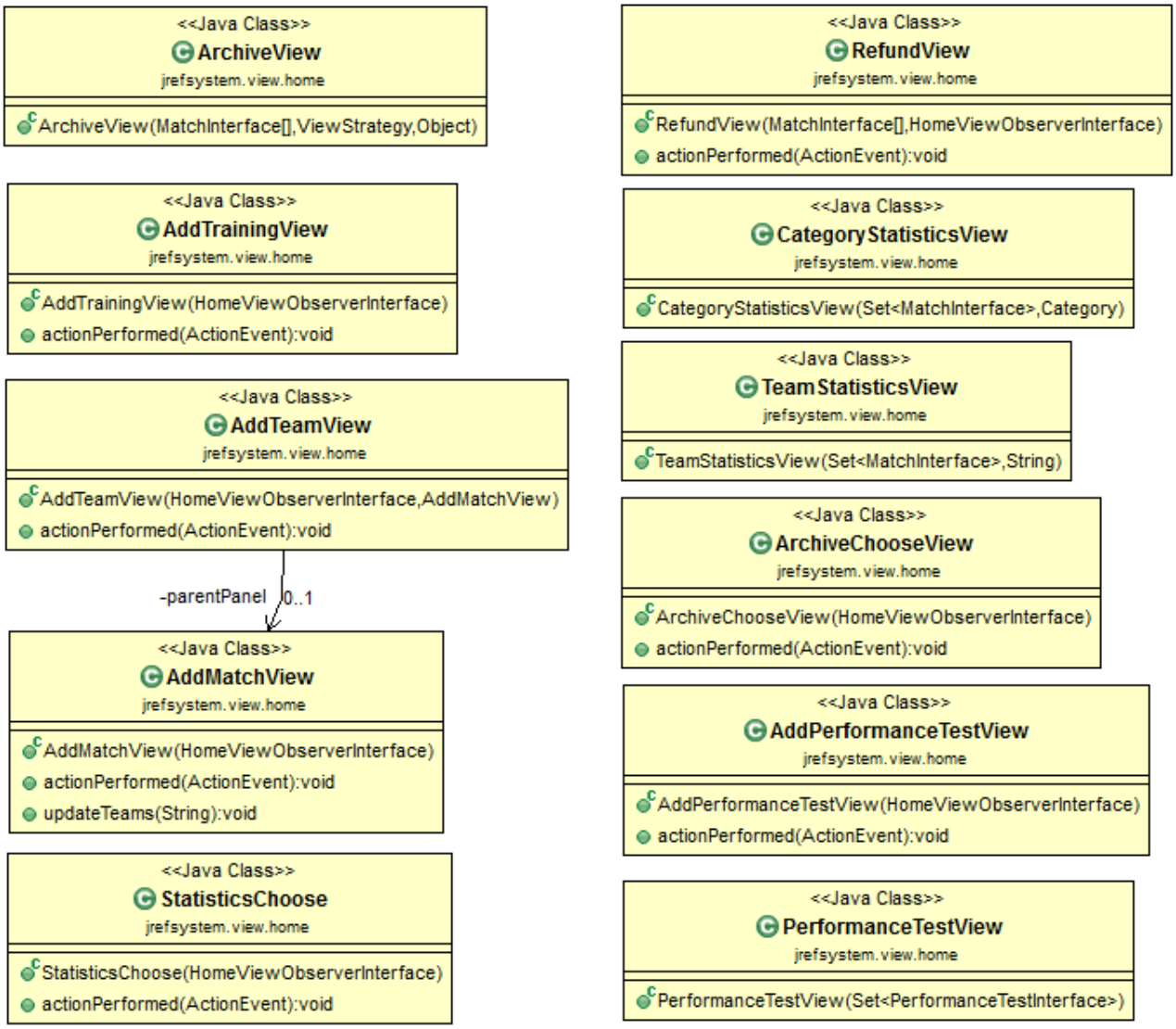
  - jrefsystem.controller: Contiene i sorgenti che implementano i controller dell'applicazione.
    - jrefsystem.controller.Controller : controller associato alla MainView.
    - jrefsystem.controller.HomeViewController : controller associato alla schermata home dell'utente.
    - jrefsystem.controller.LoginViewController : controller associato alla schermata di login dell'applicazione.
  - jrefsystem.exceptions: Contiene i sorgenti che incapsulano le eccezioni.
    - jrefsystem.exceptions.SameDataException : viene generata quando si tenta di inserire dati con un valore della data già presente.
    - jrefsystem.exceptions.SameTeamException : viene generata quando una partita ha squadra di casa uguale alla squadra in trasferta.
    - jrefsystem.exceptions.SameUserException : viene generata quando un utente tenta di registrarsi con un nome già presente nel sistema.
  - jrefsystem.main : Contiene la classe Main dell'applicazione.
  - jrefsystem.model : Contiene i sorgenti che implementano la parte di model dell'applicazione.
    - jrefsystem.model.Category : enum che contiene i valori delle categorie dei campionati calcistici.
    - jrefsystem.model.EventInterface : interfaccia implementata da tutte le classi che rappresentano un evento (es. la classe Match).
    - jrefsystem.model.IModel : interfaccia implementata dalla classe Model.
    - jrefsystem.model.Match : classe che incapsula l'entità partita.
    - jrefsystem.model.MatchInterface : interfaccia implementata dalla classe Match.
    - jrefsystem.model.Model : classe che incapsula il Model dell'applicazione e le collezioni usate per memorizzare i dati.
    - jrefsystem.model.PerformanceTest : classe che incapsula l'entità test atletico.
    - jrefsystem.model.PerformaceTestInterface: interfaccia implementata dalla classe PerformanceTest.
    - jrefsystem.model.Referee : classe che incapsula l'entità arbitro.
    - jrefsystem.model.RefereeInterface : interfaccia implementata dalla classe Referee.
    - jrefsystem.model.Training : classe che incapsula l'entità allenamento.
    - jrefsystem.model.TrainingInterface : interfaccia implementata dalla classe Training.

- jrefsystem.util : Contiene la classe ViewStrategy che è l'interfaccia che permette la realizzazione del pattern strategy quando si chiede la visualizzazione dei dati nelle view di archivio.
- jrefsystem.view : Contiene le view principali dell'applicazione , quelle che si interfacciano direttamente con i controller.
    - jrefsystem.view.HomeView : classe che incapsula la view della home di un utente loggato all'interno del sistema.
    - jrefsystem.view.HomeViewInterface : interfaccia implementata dalla classe HomeView.
    - jrefsystem.view.LoginView : classe che incapsula la view di login dell'applicazione.
    - jrefsystem.view.LoginViewInterface : interfaccia implementata dalla classe LoginView.
    - jrefsystem.view.MainView : classe che incapsula la view principale dell'applicazione (ogni altra view viene gestita da questa classe come pannello posizionato al centro del BorderLayout).
    - jrefsystem.view.MainViewInterface : interfaccia implementata dalla classe MainView.
    - jrefsystem.view.SignUpView : classe che incapsula la view che permette di registrarsi come nuovo utente.
    - jrefsystem.viewUsersView : classe che permette la visualizzazione della lista degli utenti.
- jrefsystem.view.home : Contiene i sorgenti delle classi che incapsulano le funzioni che un utente loggato può richiedere al sistema mediante la view incapsulata dalla classe HomeView.
    - jrefsystem.view.view.AddMatchView : view dalla quale è possible aggiungere un Match al Model.
    - jrefsystem.view.view.AddPerformanceTestView : view dalla quale è possibile aggiungere un PerformanceTest al Model.
    - jrefsystem.view.view.AddTeamView : view dalla quale è possibile aggiungere una squadra al Model.
    - jrefsystem.view.view.AddTrainingView : view dalla quale è possibile aggiungere un Training al Model.
    - jrefsystem.view.view.ArchiveChooseView : view dalla quale è possibile scegliere come estrarre le partite da visualizzare tramite la classe ArchiveView.
    - jrefsystem.view.view.ArchiveView : view che , una volta passata come parametro la strategia mediante un'implementazione dell'interfaccia VewStrategy, visualizza le partite desiderate in dettaglio.
    - jrefsystem.view.view.CategoryStatisticsView : view che visualizza le statistiche della categoria desiderata.
    - jrefsystem.view.view.PerformanceTestView : view che visualizza il grafico dei test atletici.
    - jrefsystem.view.view.RefundView : view dalla quale si possono gestire i rimborsi spese.
    - jrefsystem.view.view.StatisticsChoose : view che permette si scegliere quali statistiche si vogliono visualizzare.
    - jrefsystem.view.view.TeamStatisticsView : view che permette di visualizzare le statistiche del team desiderato.
    - jrefsystem.view.view.TrainingArchiveView : view che visualizza in dettaglio gli allenamenti inseriti dall'utente.
- jrefsystem.view.observer : Contiene i sorgenti delle interfacce che permettono di realizzare il pattern observer fra le classi controller e le rispettive view.

    - jrefsystem.view.observer.HomeViewObserverInterface :interfaccia che viene implementata dal controller observer della HomeView.

- jrefsystem.view.observer.LoginViewObserverInterface : interfaccia che viene implementata dal controller observer della LoginView.
- jrefsystem.view.observer.MainViewObserverInterface : interfaccia che viene implementata dal controller observer della MainView.

- **Diagrammi UML delle classi del package jrefsystem.view.home.**



- **Librerie esterne.**

Nella realizzazione del progetto sono state usate due librerie esterne : la libreria jCalendar i cui sorgenti sono stati scaricati dal seguente sito http://toedter.com/jcalendar/ ; e la libreria jFreeChart i cui sorgenti sono stati scaricati dal seguente sito http://www.jfree.org/jfreechart/.

- # Testing.

Per effettuare il testing dell'applicazione non si è utilizzato nessun tool specifico ma si è proceduto al test passo-passo delle varie funzionalità implementate per cercare di scoprire i che bug che si potessero manifestare.
Una volta conclusa la realizzazione globale dell'elaborato si è testato un utilizzo muti utente e praticamente del tutto simile a quello che un normale utilizzatore effettua nel corso di un periodo di attività agonistica.
In riassunto l'attività effettuata è stata questa: si sono registrati vari utenti tentando di registrarne anche con username uguali, si è effettuato il login con credenziali giuste o sbagliate e si è testata la funzione che permette di visualizzare l'insieme degli utenti iscritti.
Essendo presente una funzione di auto salvataggio e caricamento dei dati in fase di apertura e di chiusura dell'applicazione si sono tentate le stesse operazioni aprendo e chiudendo più finestre, si è testata nel contempo anche la funzione di salvataggio e caricamento dei dati in modo manuale attraverso il menù a tendina.
Successivamente, inserendo delle credenziali esatte nel sistema si sono iniziate a testare le funzioni offerte a un utente partendo dall'aggiunta di un match con tutte le possibili situazioni delicate come per esempio un match con due squadre uguali; sono poi state aggiunte svariate partite.
Si è testata la funzione di archivio con tutte le possibili di modalità di estrazione dei dati e confrontando i risultati ottenuti con quelli attesi; effettuando il logout di un utente e il login di un altro si è voluto provare l'effettiva funzionalità del multi-user aggiungendo e mostrando altre partite.
Passando di pari passo per tutte le altre funzionalità ho notato la presenza di qualche piccolo bug come per esempio la visualizzazione delle statistiche per categoria che mostrava dati non sempre esatti e sono stati corretti fino a raggiungere una versione dell'applicazione che per tutte le combinazioni d'uso provate non ha mostrato malfunzionamenti.

- # Note finali.

Il processo di sviluppo dell'applicazione ha seguito uno sviluppo abbastanza lineare considerando il fatto che è stato tenuto da una persona sola che oltretutto conosce bene l'ambito di interesse a cui il lavoro è rivolto.
Nella presentazione dell'idea di progetto era stata valutata la possibile implementazione di un'integrazione con il servizio di Dropbox che poi non si è realizzata a causa della mancanza di conoscenza verso alcuni aspetti necessari e il cui apprendimento avrebbe ampliato la mole di lavoro eccedendo dai limiti; si è quindi deciso di virare verso l'implementazione del multiutente.
L'implementazione di questo ultimo aspetto ha determinato una riorganizzazione del codice e la progettazione di tutte le classi necessarie che sono poi state aggiunte alla struttura generale del progetto, questo ha successivamente generato alcuni fastidiosi malfunzionamenti che sono stati eliminati con un'ulteriore riorganizzazione della struttura del Model e in particolare con l'utilizzo di una collezione dati di tipo diverso.

- ## Valutazione elaborato.

Considerata la proposta di progetto fatta in prima istanza e il risultato finale ottenuto credo che tutte le funzionalità proposte siano state implementate in maniera soddisfacente eccezion fatta per il cambio in corso d'opera che ha riguardato la scelta di abbandonare l'integrazione di dropbox e favore della gestione multi-utente.
La veste grafica è un po' semplice , anche troppo, e questo è dovuto a una carenza mia del senso estetico; ho cominciato a livello personale ad utilizzarlo per la gestione della mia attività sperimentalmente e ho proposto ad altri colleghi di potermi aiutare per darmi feedback in senso di reale utilità del progetto in questa sua versione molto base, se dovesse in qualche modo piacere ho intenzione di portare avanti l'esperienza aggiungendo funzioni che magari mi saranno poi suggerite da chi lo utilizzerà.
Mi posso comunque ritenere soddisfatto di come ho svolto il lavoro, ritengo sia stato molto utile sotto l'aspetto didattico per fissare tutti i concetti trattati nel corso e sotto l'aspetto dell'esperienza personale per cominciare ad interfacciarsi con quello che è il percorso di sviluppo del software.