Relazione progetto OOP

SCOUT APP

A cura di:
Lorenzo Valgimigli
Riccardo Soro
Giovanni Martelli

SOMMARIO

Questo documento è una relazione del progetto svolto dagli studenti Lorenzo Valgimigli, Riccardo Soro e Giovanni Martelli per il corso di “Programmazione Ad Oggetti” dal professore Mirko Viroli. Lo scopo di questo documento è quello di evidenziare il lavoro svolto dal team, sottolineando le metodiche adottate, i problemi riscontrati durante lo sviluppo e le soluzioni scelte.

INDICE:

Capitolo 1

ANALISI:

Introduzione:

Il progetto vede come dominio il mondo scout. E' per tanto necessaria una descrizione di quest'ultimo poiché non tutti hanno una conoscenza abbastanza approfondita per poter capire a pieno il lavoro svolto.

Dominio:

Il mondo scout è un mondo complesso che si basa su sistemi gerarchici, regole scritte e a volte non scritte. Specificatamente il dominio scelto coinvolge il reparto ovvero il gruppo di ragazzi di età compresa tra i 12 e i 17 anni. Il reparto non si può definire un semplice gruppo poiché presenta delle divisioni interne che analizzeremo in seguito, regole complesse sia per i capi sia per i ragazzi da dover rispettare. In più esistono due ingressi nel reparto quello formale con il pagamento delle tasse annuali e quello informale tramite dei simboli puramente scoutistici. Inoltre il reparto a differenza di quello che può essere un gruppo pomeridiano o un centro estivo si pone come obbiettivo l'educazione secondo principi scout dei ragazzi. Questa educazione avviene tramite strumenti ben precisi quali attività mirate, dialogo ma soprattutto tramite quello che

si definisce “Sentiero”. Il “Sentiero” è un percorso di crescita tra ragazzo e capo attraverso degli obbiettivi e il raggiungimento di tali.

Verrà ora analizzato il reparto nel dettaglio

Il reparto deve avere una staff ovvero un organo composto da soli capi che si occupa di tutelare e gestire i ragazzi presenti nel gruppo. La staff è così composta:

1. Capo formato maschio
2. Capo formato femmina
3. Capi aiuto (da 1 a N)
4. Rover (Non sono ancora capi)

L’analisi dei capi e delle tipologie verrà ignorata poiché non ha nessun valore al fine della relazione e del progetto stesso.

Il reparto è composto da ragazzi i quali vengono ricollocati in squadriglie cioè sottogruppi che vivono in maniera semi autonoma nel reparto. Ogni squadriglia ha ragazzi di tutte le età infatti i ragazzi della stessa annata vengono divisi equamente tra le squadriglie. Questa divisione spetta alla staff. Ogni squadriglia ha:

- Un capo che solitamente è il più grande della squadriglia il cui compito è gestire la squadriglia in modo da renderla il più funzionale possibile.
- Un vice che aiuta il capo e lo sostituisce qualora mancasse
- Un trice che ha lo stesso ruolo del vice e ne prende le veci qualora o capo o vice mancassero.
- Una cassa dove viene tenuta la cancelleria.
- Un portafoglio che contiene i soldi della squadriglia prodotti dagli auto finanziamenti.
- Una batteria ovvero un insieme di pentole e pentoloni per cucinare
- Un nome che rappresenta un animale (Aquile, Volpi, Albatros …)

Un ragazzo nel reparto oltre ad appartenere alla squadriglia appartiene al reparto come singolo per tanto ogni ragazzo:

- A un certo punto del primo anno fa la promessa. Il simbolo che garantisce il suo impegno all'interno degli scout e fuori.
- A un certo punto fa il battesimo ovvero un rituale di ingresso all'interno del reparto e del mondo scout stesso in cui gli viene dato un nome detto Totem.
- Segue un percorso di crescita a più livelli detto sentiero. Per ogni livello insieme ad un capo si concordano 4 obbiettivi. Uno per ciascuno dei seguenti campi: fede, famiglia, scuola, relazioni. Quando sono stati raggiunti si sale di livello. Il livelli sono: Scoperta, Competenza, Responsabilità.
- E' possibile prendere delle specialità. Una specialità si prende dimostrando delle competenze in quel determinato campo esempio: "Amico degli animali" si prende conoscendo gli animali delle nostre colline, come rispettarli etc. etc. Le specialità sono molteplici.

Ogni reparto svolge delle attività in particolare delle uscite. Ma uscita è forse troppo generico. Verranno di seguito spiegate nel dettaglio:

- USCITA: per uscita si intende un escursione di 2 giorni con tutto il reparto.
- USCITA DI SQUADRIGLIA: si intende un escursione di 1 o più giorni con tutta la squadriglia.
- CAMPO: si intende un escursione di 3 o più giorni per tutto il reparto.
- GEMELLAGGIO: si intende un escursione di 1 o più giorni di tutto il reparto insieme ad uno o più altri reparti.
- EVENTO DI ZONA: si intende un escursione di uno o più giorni con tutti i reparti della Zona (Es Zona Forlì)

Requisiti applicativi:

L'applicazione si pone come obbiettivo quello di gestire un reparto nel migliore dei modi rappresentando un supporto utile e funzionale per i capi. Specificatamente i requisiti base sono:

- Creazione nuovo reparto
- Creazione Squadriglia
- Aggiunta membro in reparto e quindi in Squadriglia
- Gestione del singolo
- Gestione delle squadriglie
- Gestione del gruppo

Per "Gestione del singolo" si intende:

- Sentiero
- Specialità e competenze
- Ruolo nella squadriglia
- Promessa
- Totem
- Tasse

Per "Gestione della Squadriglia" si intende:

- Aggiunta e rimozione membro
- Gestione gerarchia (Capi, vice, trice)
- Gestione cassa, batteria, soldi e cancelleria

Per "Gestione gruppo" si intende:

- Tasse
- Capi
- Escursioni

Note favorevoli:

Buona conoscenza del mondo scout poichè Lorenzo Valgimigli partecipa agli scout da oltre 10 anni. In più il mondo scout presenta serie di regole standard precise valide per ogni gruppo in Italia. Questo aiuta a definire con precisione certe funzionalità dell'applicazione. In più si possono considerare varie espansioni per il futuro.

Challenge

Il mondo scout è molto vasto perciò la prima sfida rappresenta saper scegliere cosa implementare e cosa no senza far perdere il significato iniziale del progetto.

Extra

Sarebbe utile che l'applicativo oltre alle funzioni base fornisca i seguenti servizi:

1. Invio mail automatizzato per segnalare la creazione di un evento
2. Checking dal sito 'Buona Caccia' degli eventi in programma

Capitolo 2

Design

In questo capitolo verrà spigato il design progettuale e i pattern scelti. Verranno inoltre elencati vantaggi e svantaggi del lavoro in team e problemi riscontrati. Verranno spiegate inoltre le parti più complesse del progetto.

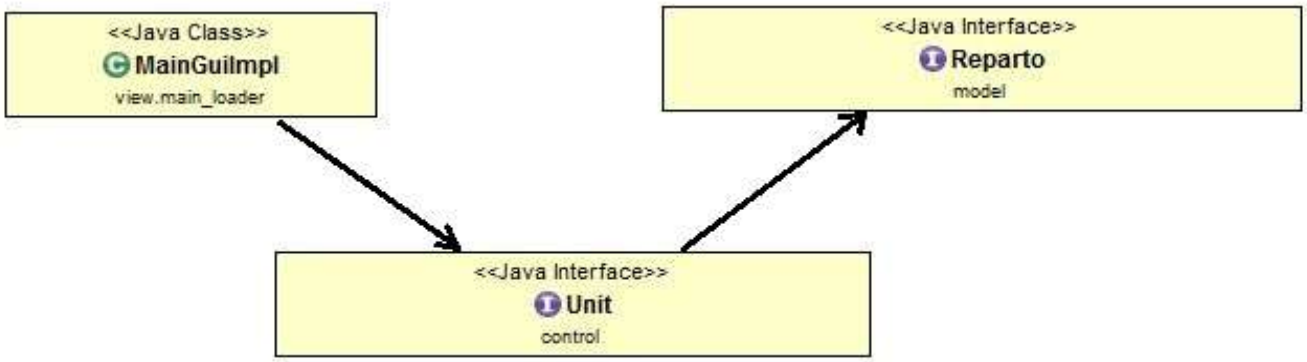
Architettura

Il team (ricordo composto da Lorenzo Valgimigli, Riccardo Soro, Giovanni Martelli) ha lavorato seguendo il pattern progettuale MCV ovvero Model Control View. Il model ha avuto il compito di descrivere gli elementi in causa valutando anche quali comportamenti avrebbe dovuto implementare e quali invece avrebbe dovuto lasciar implementare al controller. Gli elementi sono numerosi e vari. Il controller si occupa delle interazioni tra gli elementi e della gestione di essi. Mentre la view fornisce un interfaccia grafica chiara e semplice da utilizzare. Ogni parte ha lavorato nella maniera più indipendente possibile. Nella nostra organizzazione il controller ha fatto da ponte tra Model e Grafica rendendo i due elementi completamente autonomi tra loro.

I compiti sono stati così ripartiti:

- Model: Riccardo Soro
- Controller: Lorenzo Valgimigli
- View: Giovanni Martelli

Di seguito Lo schema UML che mostra come i tre ruoli hanno lavorato tra di loro:



Si noti come la classe “unit” si ponga da intermediario tra la Entry della View e la entry del Model.

MODEL (Riccardo Soro)

Il model si è posto come obiettivo la realizzazione delle classi base e dei relativi metodi di interazione da mettere a disposizione per il Controller e per la View implementando alcuni controlli sulla correttezza delle operazioni effettuate e dei valori inseriti.

Obiettivi Model:

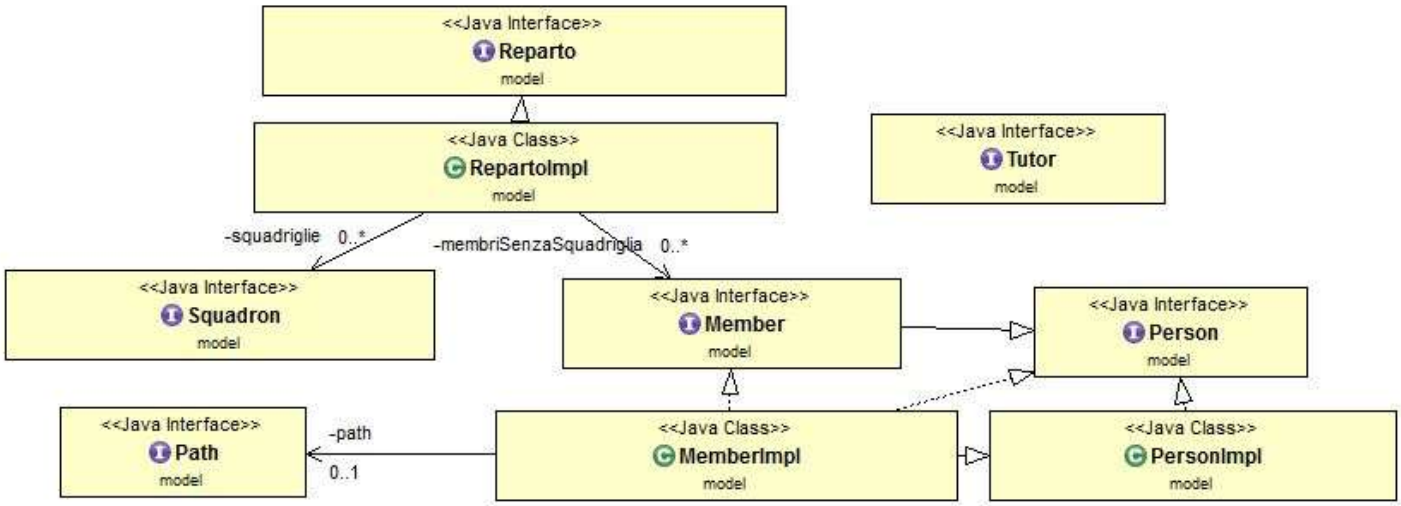
- descrizione del dominio analizzato
- gestione di operazioni o dati non validi
- corrette relazioni tra le classi

Sviluppo:

L' insieme delle classi è stato suddiviso in due blocchi, il primo riguardante i reparti e il secondo le escursioni.

Reparto:

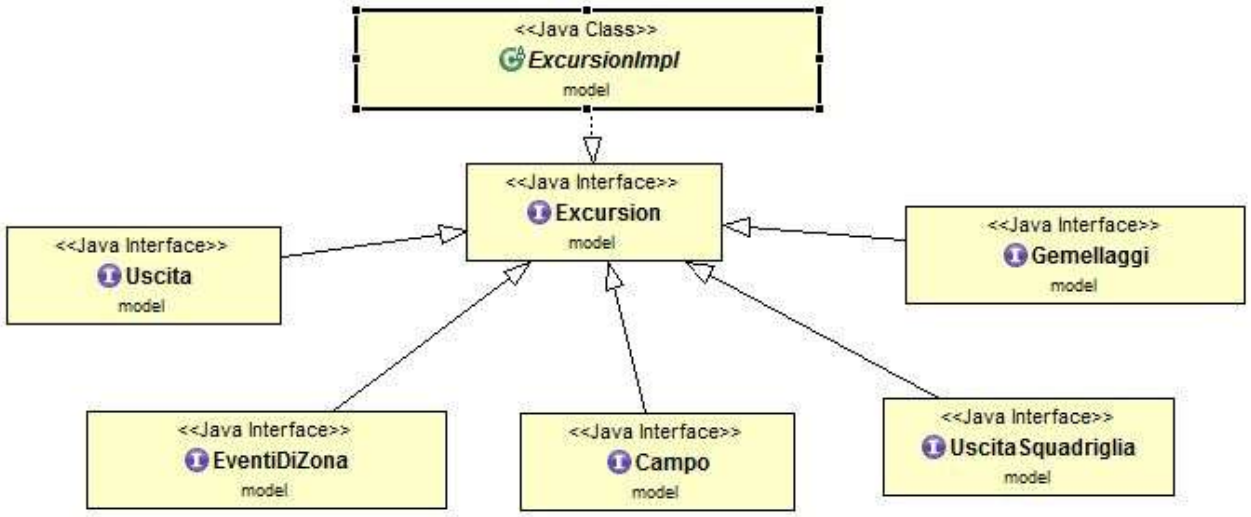
Il reparto per definizione è un insieme di squadriglie che a loro volta sono formate da membri, con relativi tutor.



Escursioni:

Le escursioni hanno molti punti in comune tra di loro (es:tutte hanno il prezzo, una data di inizio, una data di fine….) ma variano per durata e componenti .

E' stata pertanto modellata una classe astratta generica dalla quale si estendono tutte le altre.



Problemi Riscontrati:

- Controllo delle date di inizio e fine nelle varie escursioni.

Soluzioni:

Si è scelto di implementare il metodo astratto di controllo nella classe astratta ed effettuare un override in ogni escursione per implementare le giuste regole di controllo ispirandomi al Template Method, un pattern progettuale della Gof.

CONTROLLER (Lorenzo Valgimigli)

Nel team il controller ha avuto come obbiettivo quello di fornire tutto ciò che la grafica necessitava, gestire i rapporti tra le varie entità, relazionare l'applicazione con il Sistema Operativo, creare gli oggetti e soprattutto rendere completamente autonomi tra loro view e model.

OBBIETTIVI CONTROLLER:

- Indipendenza tra View e Model
- Coordinamento
- Relazione Applicazione e Sistemo Operativo in particolare in fase di salvataggio

SVILUPPO

L'entry point del Controller si chiama "Unit" che include in sé l'entry point del Model "Reparto" e fornisce tutte le funzioni per la View. Il cuore della "Unit" è il container ovvero una classe che racchiude tutto l'organico del reparto su chi la View può fare ricerche e visualizzazioni, senza toccare il reparto effettivo. Unit fornisce poi anche il nome per il salvataggio. Grazie alla "Unit" Model e View hanno potuto lavorare in parallelo indipendenti. Per salvataggio è stato scelto di usare la Serializzazione e la scrittura dell'intero oggetto. Al file creato viene aggiunta l'estensione ".sct" . L'applicazione crea in autonomia una directory nella cartella home dell'utente chiamata "ScoutApp". All'interno troviamo un file "ImpScout.txt" che contiene le impostazioni. Al momento contiene solo la directory di salvataggio.

Oltre al file troviamo la directory di salvataggio di default chiamata “SaveProject”.

PROBLEMI RISCONTRATI:

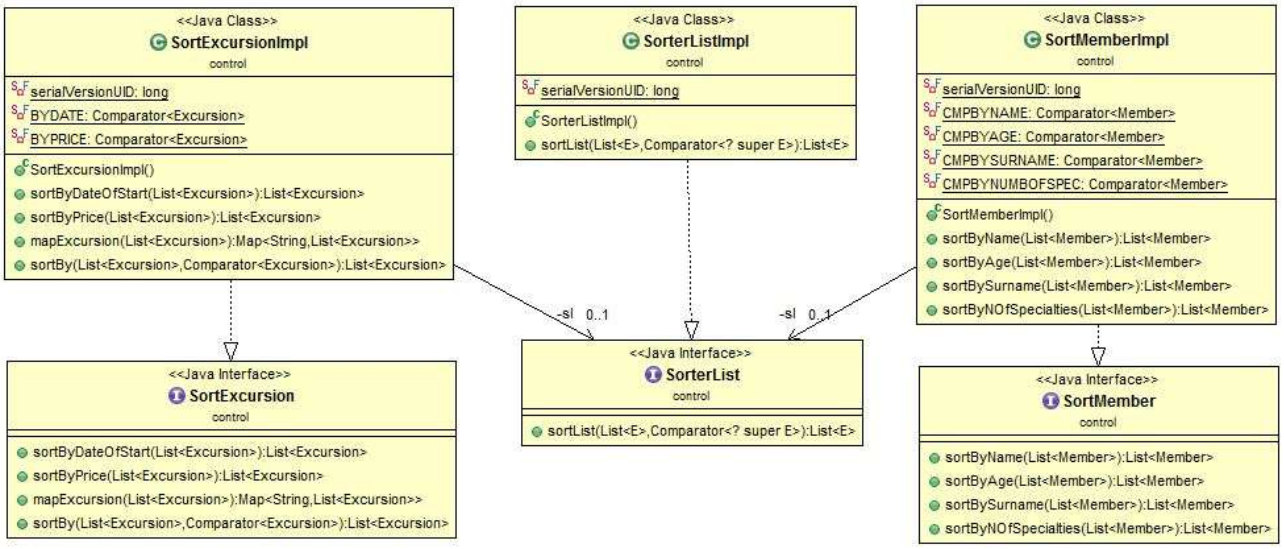
- Creazione oggetti
- Classe di Sorting

SOLUZIONI

Per la creazione degli oggetti è stato scelto un metodo “Factory” Statico ovvero una classe che presentasse per ogni oggetto istanziabile un metodo statico per la creazione di esso. Sono presenti più metodi, uno per ogni costruttore ma può capitare di avere più metodi per descrivere meglio il comportamento di alcuni oggetti. Ad esempio l’oggetto capo ha un solo costruttore e accetta una variabile che setta il sesso. Nella “Factory” ci sono due metodi “getLeaderM” e “getLeaderF” che producono un capo maschio e uno femmina settando il sesso in automatico.

Per quanto riguarda il Sorting si è presentato il problema di una classe che dovesse fornire una lista di oggetti ordinata in varie possibilità che l’utente deve poter scegliere in runtime. E’ stata adottato il Pattern “Strategy” ovvero sono stati scritti svariati comparatori tramite Lambda e un unico metodo di Sort che accettasse un comparatore e una lista. Due classi poi decorano la classe di Sort generica. Ogni classe include in sé l’oggetto generico e mette a disposizione dei metodi più specifici e gli opportuni comparatori. Questo pattern che per definizione “svincola l’Algoritmo dalla classe” permette di avere una classe efficiente e semplice da modificare per implementazioni future.

UML SORT


<<Java Class>>
SortExcursionImpl
control
serialVersionUID: long
BYDATE: Comparator<Excursion>
BYPRICE: Comparator<Excursion>
SortExcursionImpl()
sortByDateOfStart(List<Excursion>):List<Excursion>
sortByPrice(List<Excursion>):List<Excursion>
mapExcursion(List<Excursion>):Map<String,List<Excursion>>
sortBy(List<Excursion>,Comparator<Excursion>):List<Excursion>
<<Java Class>>
SorterListImpl
control
serialVersionUID: long
SorterListImpl()
sortList(List<E>,Comparator<? super E>):List<E>
<<Java Class>>
SortMemberImpl
control
serialVersionUID: long
CMPBYNAME: Comparator<Member>
CMPBYAGE: Comparator<Member>
CMPBYSURNAME: Comparator<Member>
CMPBYNUMBOFSPEC: Comparator<Member>
SortMemberImpl()
sortByName(List<Member>):List<Member>
sortByAge(List<Member>):List<Member>
sortBySurname(List<Member>):List<Member>
sortByNOfSpecialties(List<Member>):List<Member>
-sl 0..1
-sl 0..1
<<Java Interface>>
SortExcursion
control
sortByDateOfStart(List<Excursion>):List<Excursion>
sortByPrice(List<Excursion>):List<Excursion>
mapExcursion(List<Excursion>):Map<String,List<Excursion>>
sortBy(List<Excursion>,Comparator<Excursion>):List<Excursion>
<<Java Interface>>
SorterList
control
sortList(List<E>,Comparator<? super E>):List<E>
<<Java Interface>>
SortMember
control
sortByName(List<Member>):List<Member>
sortByAge(List<Member>):List<Member>
sortBySurname(List<Member>):List<Member>
sortByNOfSpecialties(List<Member>):List<Member>

View (Giovanni M. Martelli)

Nel team la View si è posta come obbiettivo il fornire all'utente finale una GUI il più possibile piacevole da vedere e semplice da utilizzare, considerando sia gli aspetti prettamente stilistici,sia quelli funzionali(quante "sezioni" presentare, in quale sezione mettere una certa funzione, etc…). La GUI è stata implementata cercando di rispettare a pieno il pattern MVC, richiamando il Controller per le varie operazioni.

OBBIETTIVI VIEW:

- Produrre una GUI piacevole alla vista
- Produrre una GUI completa e semplice da utilizzare

SVILUPPO

L'entry point della View è la classe LoaderImpl che permette all'utente di gestire i reparti(caricando/eliminando i salvataggi precedenti o creandone di nuovi). LoaderImpl si occupa anche di richiamare la classe MainGUI; questa classe è il cuore della GUI e permette all'utente di spostarsi tra le varie sezioni(suddivise in quattro package: excursion_manager, fee_manager, unit_manager, online). Le prime tre sezioni(Gestione Reparto, Gestione Tasse, Gestione Eventi) sono state implementate rispettando una struttura comune: la prima classe che viene chiamata è [nome sezione]Main e dal suo interno è possibile navigare nelle sottosezioni presenti tramite un JTree(ad esempio: dalla classe FeeManagerMain, tramite il JTree , è possibile visualizzare le Tasse per l'intero reparto, quelle di una singola squadriglia, etc…). Per implementare le tre suddette sezioni si è scelto che le tre classi

[nome sezione]Main estendessero la classe MyJPanelWithJTreeImpl che setta la disposizione degli elementi comuni a tutte le sezioni e permette alle classi [nome sezione]Main di aggiungere a runtime elementi al JTree e di settare il TreeSelectionListener; inoltre in queste tre classi è definita una Inner class utilizzata per settare i ToolTip dei nodi del JTree. Tutte le sotto-sezioni sono implementate nelle altre classi presenti nel package; queste classi sono composte da una outer class che ha come unico scopo l'essere il nodo del JTree(facendo un override sul metodo toString), mentre l'inner class è il JPanel vero e proprio. Si è scelto questo sistema per far si che ad ogni cambio di selezione nel JTree il pannello centrale venisse aggiornato agli ultimi dati presenti in memoria.

PROBLEMI RISCONTRATI:

- Utilizzare un unico JFrame per le parti principali dell'applicazione(ossia le varie sezioni e sottosezioni), evitando però che l'utente rimanesse bloccato su un unico contesto.
- JScrollPane presenti in tutte le sezioni, con elementi mostrati in base al contesto, senza dover ogni volta riscrivere il codice comune
- JDialog per la ricerca di elementi, in base al contesto, senza dover ogni volta riscrivere il codice comune.

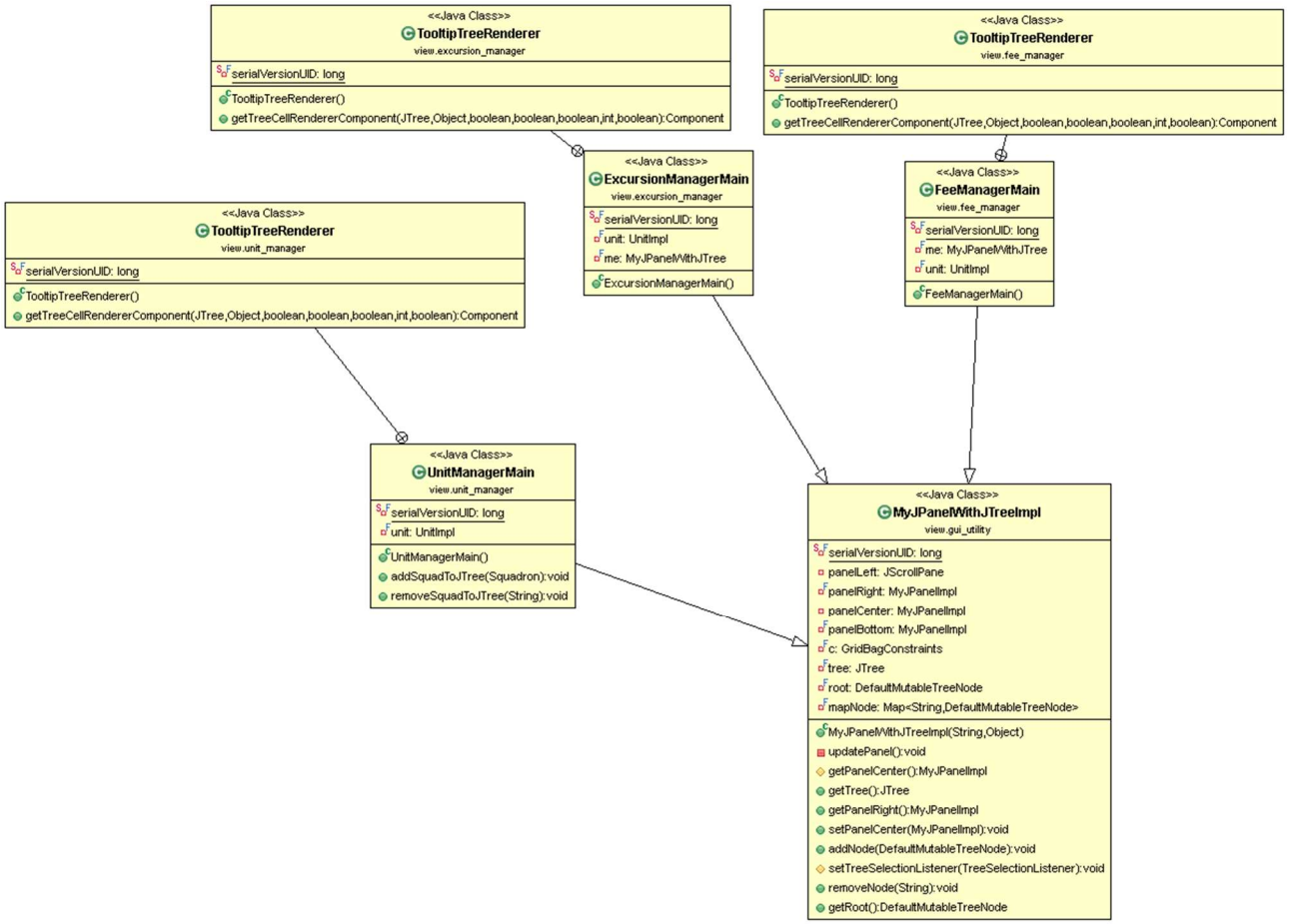
SOLUZIONI

Per quanto riguarda il JFrame si è scelto di utilizzare un unico JFrame per le parti principali del programma e delle JDialog per tutte le parti di inserimento dei dati. Per attuare l'unicità del JFrame si è deciso di seguire il pattern Singleton, richiamando sempre il metodo getInstance(), della classe MyJFrameSingletonImpl oppurtunamente implemetata, per usufruire del JFrame. E' inoltre

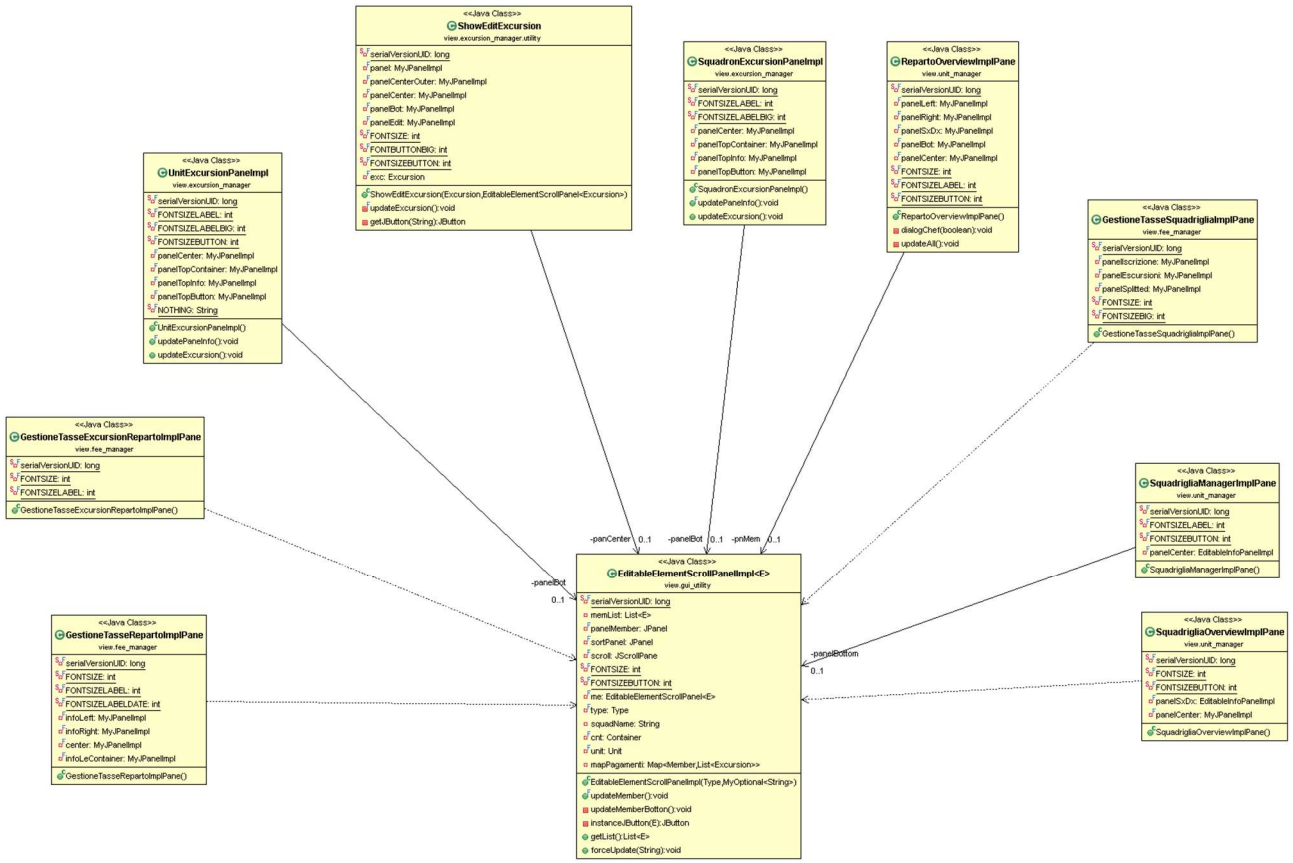
presente un metodo getInstance(Unit u) da chiamare la prima volta che si vuole utilizzare il JFrame, questo perché in MyJFrameSingletonImpl è presente un riferimento alla Unit(controller) attualmente in uso, permettendo così a tutte le altre classi di accedere direttamente alla Unit caricata(tremite il metodo getUnit())

Per quanto riguarda invece il JScrollPane e la JDialog di ricerca si è deciso di implementare delle classi che fossero autonomamente capaci, in base ai parametri passati al costruttore, di decidere quali elementi mostrare, permettessero l'aggiornamento di tali elementi dopo eventuali cambiamenti(solo JScrollPane) e permettessero all'utente di eseguire operazioni in base al contesto. Per permettere l'eventuale espansione di tali classi si è deciso di creare(in ognuna delle due classi) un enumerazione che definisse il contesto, un metodo che eseguisse la selezione degli elementi(nel caso di EditableElementJScrollPane una chiamata a questo metodo esegue anche l'aggiornamento dopo eventuali cambiamenti) e uno che definisse, sempre in base all'elemento dell'enumerazione passato al costruttore, quale operazione eseguire. Con questo sistema è talvolta possibile, per contesti diversi, utilizzare un unico metodo di selezione ed eseguire differenti operazioni, e viceversa. Impostando la classe in questo modo per eventuali espansioni basterebbe aggiungere una entry all'enumerazione, un "caso specifico" nel metodo di selezione e un "caso specifico" nel metodo che esegue l'operazione, lasciando il compito di creare i JButton(aggiungendoli e aggiornandoli a runtime) al codice della classe stessa. Queste due classi sono inoltre state create usando i Java Generics, permettendo così di essere utili in ambiti diversi(al momento nell'applicazione sono utilizzate sia per i Member che per le Excursion). Da notare di seguito che queste due classi sono utilizzate praticamente ovunque nell'applicazione.

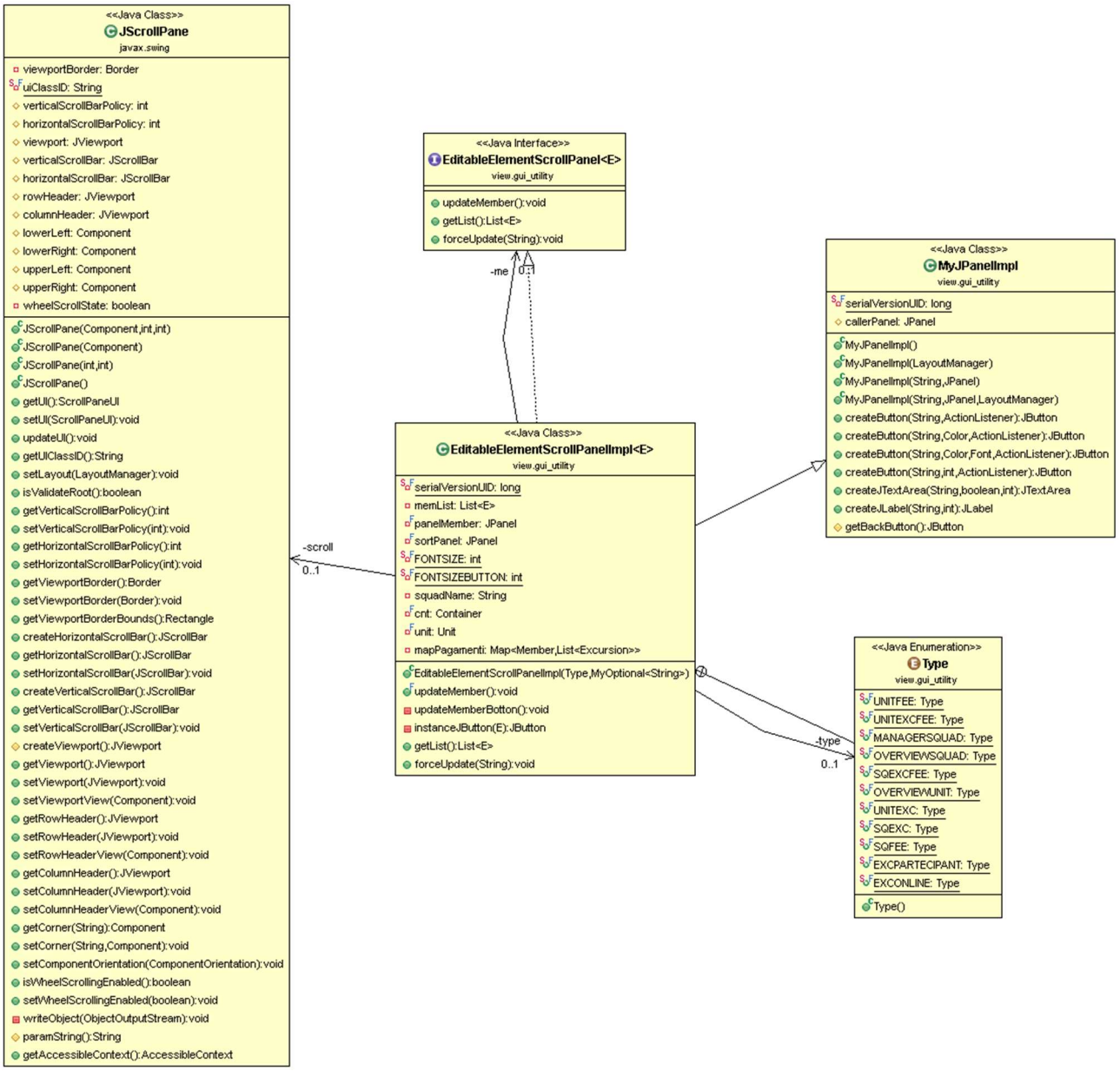
UML [sezione]Main



UML presenza EditableElementJScrollPane nell'applicazione



UML EditableElementJScrollPane



CAPITOLO 3
Extra

In questa sezione verranno mostrate brevemente gli extra realizzati. Viene premesso che per quanto riguarda gli Extra il Pattern Architetturale MVC non è stato rispettato poiché gli extra sono stati implementati una volta finito il proprio lavoro per i requisiti minimi. Avendo terminato in tempi molto diversi non è stato possibile lavorare di gruppo.

INVIO MAIL

L'obbiettivo era avvisare per ogni evento creato automaticamente il genitore o chi ne faceva le veci di ogni ragazzo tramite una mail automatica. I problemi riscontrati erano:

- Trovare una libreria che supportasse l'invio delle mail
- Gestire l'invio in ordine di: una mail al giorno al massimo e non ad ogni accensione, invio mail per i compleanni, controllo pagamenti

SVILUPPO

E' stata utilizzata la libreria "javax.mail" della oracle version "1-5-6". Per gestire l'invio mail che non ne inviasse più di una al giorno, per lo stesso motivo e alla stessa persona è stato implementato un controllo sulle date che salva la data dell'ultimo invio.

ACQUISIZIONE EVENTI DA BUONA CACCIA

Questo extra è stato realizzato scaricando il codice HTML del sito di "Buona Caccia" (http://buonacaccia.net/Events.aspx?RID=&CID=20) e manipolando le stringhe in modo da ricavare tutti i dati necessari.

PROBLEMI

l'AGESCI (Organo a capo del sito) non mette a disposizione API, quindi si è dovuta implementare una piccola classe che si interfacciasse con il sito, che ha richiesto molto tempo. Per tanto non è stato possibile trovare soluzioni robuste a futuri cambiamenti del codice html della pagina.

Questo Extra implementa una funzione che apre il browser di default del sistema per raggiungere la descrizione del singolo evento sul sito di buona caccia. Questa funzione non è garantita su tutti i sistemi operativi. La compatibilità è garantita con : Windows (tutti),

Per arginare il problema è stato inserito il link nell'interfaccia in modo da permettere all'utente anche su sistemi operativi non supportati di collegarsi comunque al sito dell'evento.

CAPITOLO 4
Testing

In questo capitolo verrà discussa brevemente la fase di testing.

TESTING AUTOMATICO

Per quanto riguarda le classi del Model e del Control è stato preferito un testi automatizzato tramite “junit”. Sono state testate le funzioni più importanti. Per quanto riguarda la View i test sono stati eseguiti “a mano” dal responsabilie della view in fase di sviluppo e da tutti i componenti del gruppo fase di testing finale.

TESTING MANUALE

La parte più importante della fase di testing. Una volta terminato il programma è stato lanciato e testato in varie condizioni, testando gli inserimenti, cercando i bug o semplici errori grafici.

CAPITOLO 4
Svolgimento del lavoro

Package

Ognuno ha scelto autonomamente come chiamare i package.

Control: Il control ha i seguenti package:

- Control dove ci sono le classi
- Control.exception dove ci sono le eccezioni
- Control.myUtil dove ci sono le classi di utiliti

Model: Il model ha i seguenti Package:

- model.exception dove sono contenute le eccezioni
- model.escursioni dove sono contenute le classi riguardati le escursioni
- model.reparto dove sono contenute le classi inerenti al reparto

View: La view ha i seguenti Package:

- excursion_manager: classi sezione Gestione Eventi
  - excursion_manager.utility: classi di utility sezione Gestione Eventi
- fee_manager: classi sezione Gestione Tasse
  - fee_manager.utility: classi di utility sezione Gestione Tasse
- unit_manager: classi sezione Gestione Reparto
  - unit_manager.utility:classi di utility sezione Gestione Rep
- online:classi per sezione Online(eventi buonacaccia.net)
- gui_utility: classi di utility utilizzate in tutta la GUI

- main_loader: classi per la GUI iniziale(caricamento reparto)

SVILUPPO

Inizialmente si sono decise le linee guida da seguire per ognuno in modo che gli studenti potessero lavorare in massima autonomia. Data l'inesperienza del team e il poco dettaglio di descrizione iniziale, ci sono state alcune modifiche in corso d'opera ridefinendo dei dettagli allungando così i tempi. E' stato subito evidente che una volta organizzate le parti secondo il pattern architetturale MVC il team ha potuto lavorare velocemente senza troppi intoppi. La parte più lunga è stata senza dubbio quella di "Testing" e "Bug resolving".

NOTE PERSONALI DEL TEAM

A seguito delle conoscenze apprese e del risultato ottenuto ci riteniamo soddisfatti del progetto. Molto utile è stata l'esperienza del team, e del pattern MVC. Il programma funziona bene e si accosta molto all'idea iniziale. Verrà proposto nel prossimo anno, da Lorenzo Valgimigli, alla sua staff di reparto sperando possa aiutare i capi.