

# Università degli Studi di Salerno

Corso di Ingegneria del Software

## Football League Manager

Object Design Document

Versione 1.6



Data: 01/01/2017

## Partecipanti

Nome	Matricola
Nocerino Christian Giuseppe	0512103250
Manzo Gerardo	0512102974
Sarubbi Raffaele	0512102934
Capaldo Giovanni	0512102980
Iannuzzi Serena	0512103216

Scritto da	Raffaele Sarubbi
------------	------------------

## Revision History

Data	Versione	Descrizione	Autore
01/01/2017	1.0	Aggiunta sezione: Introduction (1.1, 1.2, 1.3)	1. Raffaele Sarubbi 2. Gerardo Manzo
05/01/2017	1.1	Aggiunta sezione: Packages	Raffaele Sarubbi
07/01/2017	1.2	Aggiornata sezione: Introduction (1.1)	1. Gerardo Manzo 2. Raffaele Sarubbi
12/01/2017	1.3	Aggiunta sezione: Class interfaces	1. Raffaele Sarubbi 2. Serena Iannuzzi
15/01/2017	1.4	Aggiunta sezione: Glossary	Raffaele Sarubbi
20/01/2017	1.5	Correzioni	1. Gerardo Manzo 2. Serena Iannuzzi 3. Raffaele Sarubbi
27/01/2017	1.6	Aggiornata sezione: Class interfaces	1. Serena Iannuzzi 2. Raffaele Sarubbi

# Indice

<b>1. Introduction</b>	<b>3</b>
1.1 Object design trade-offs	3
1.2 Interface documentation guidelines	
1.2.1 Packages	
1.2.2 Classes	
1.2.3 Methods	
1.2.4 Fields, Variables and Constants	
1.2.5 Comments	
1.3 Definitions, acronyms, and abbreviations	3
1.3.1 Definitions	
1.3.2 Abbreviations	3
1.4 References	3
<b>2. Packages</b>	<b>3</b>
<b>3. Class interfaces</b>	<b>3</b>
<b>4. Glossary</b>	<b>3</b>

# 1. Introduction

## 1.1 Object design trade-offs

### Memory space vs. Response time

Le informazioni delle partite vengono replicate sulle istanze delle classi Squadra, (calcolo della classifica). Utilizzando questo approccio si utilizzerà un maggior quantitativo di memoria a discapito del tempo di risposta.

### Buy vs. Build

Il software è scritto da zero tranne per l'utilizzo di componenti off-the shelf tra cui i driver JDBC, mentre Bootstrap e FontAwesome per la parte di presentazione e estetica.

### Modularity vs. Efficiency

Si è deciso di favorire la modularità, piuttosto che l'efficienza, in quanto essa facilita molto l'implementazione e la manutenzione del software e la sua comprensione sia da parte dei programmatori intenzionati a modificarlo, sia da parte degli utenti.

## 1.2 Interface documentation guidelines

Sono state definite varie convenzioni che saranno rispettate dagli sviluppatori del sistema durante la fase di implementazione del software FLM. Tali convenzioni riguardano, in particolare, i nomi di pacchetti, classi, metodi, campi e variabili che faranno parte del codice, l'organizzazione dei pacchetti ed i commenti al codice del software.

### 1.2.1 Packages

- Tutte le classi che fanno parte del sito devono appartenere al package flm.
- Ogni package deve rappresentare un sottosistema e quindi che tutte le classi appartenenti ad esso sono collegate.

### 1.2.2 Classes

- Ogni classe deve avere un nome che sia composto da uno o più sostantivi singolari/plurali della lingua italiana o della lingua inglese, ognuno dei quali deve avere la prima lettera di ogni parola in maiuscolo.

- Ogni classe deve avere un nome che sia significativo per il suo scopo relativamente al software di cui fa parte.

### **1.2.3 Methods**

- Ogni metodo deve avere un nome composto da uno o più termini singolari della lingua italiana o della lingua inglese.
- Il primo termine del nome di ogni metodo deve essere totalmente in minuscolo, mentre gli altri termini del nome devono avere la lettera iniziale in maiuscolo e le rimanenti lettere in minuscolo.
- Ogni nome di metodo deve essere significativo relativamente al comportamento e allo scopo dello stesso metodo all'interno della sua classe e dell'intera applicazione.

### **1.2.4 Fields, Variables and Constants**

- Ogni campo ogni variabile deve avere un nome composto da uno o più termini che siano sostantivi della lingua italiana o inglese, il primo dei quali deve essere al singolare.
- Ogni costante deve avere un nome che abbia tutte le sue lettere in maiuscolo.

### **1.2.5 Comments**

- I commenti del codice devono essere commenti Javadoc o commenti del linguaggio Java.
- Ogni commento deve spiegare una parte di codice in maniera chiara e facile da comprendere da lettori estranei all'implementazione del progetto.
- Ogni commento deve essere breve, in modo da favorire la sua lettura.
- Ogni controllo fatto nel codice per garantire la corretta esecuzione di una o più istruzioni deve essere commentato brevemente e semplicemente.

## **1.3 Definitions, acronyms, and abbreviations**

### **1.3.1 Definitions**

- Utente registrato (Allenatori, arbitri e amministratore): Colui che utilizza il sistema

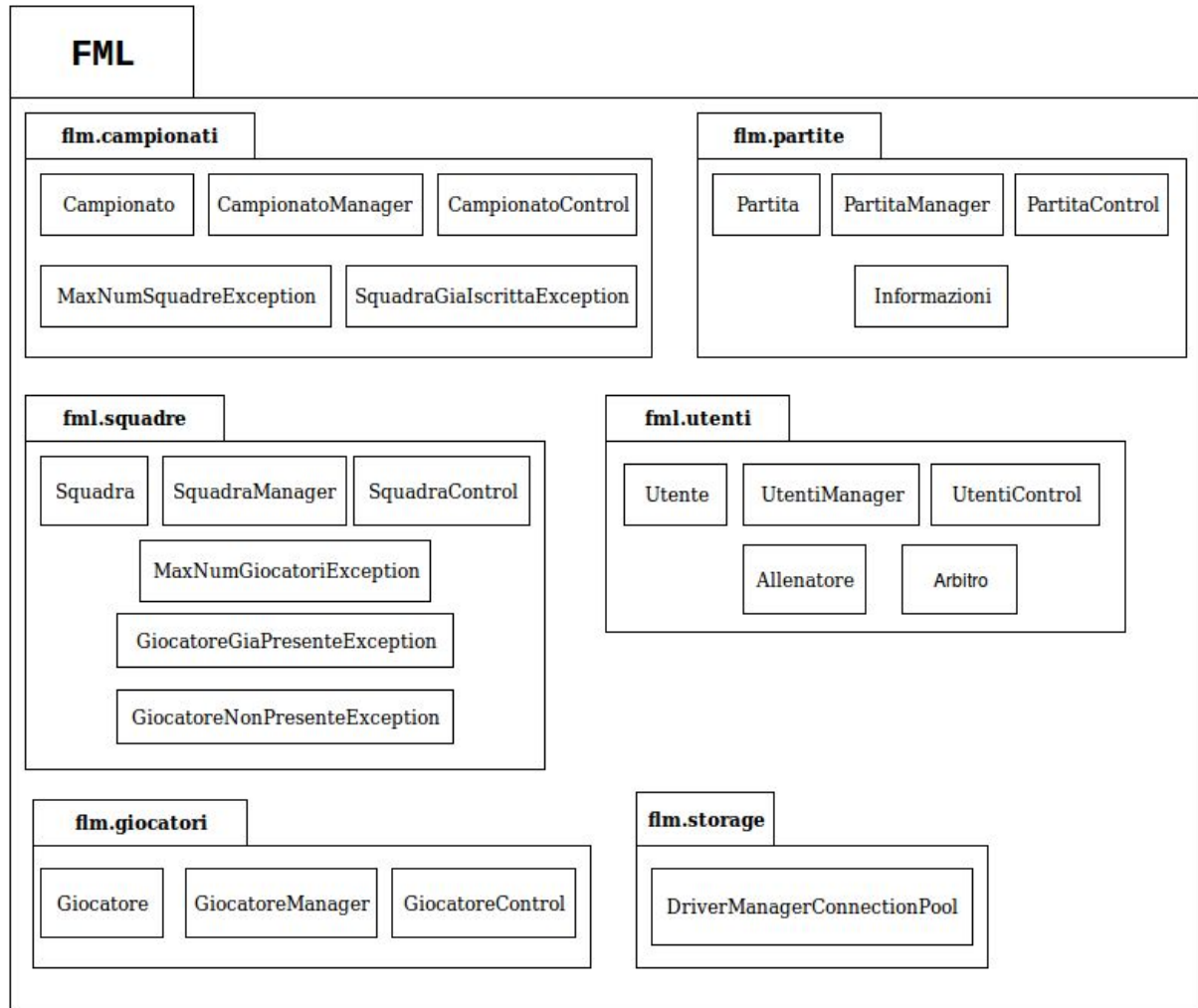
### 1.3.2 Abbreviations

- FLM: Football League Manager

## 1.4 References

- B. Bruegge, A.H. Dutoit, *Object Oriented Software Engineering – Using UML, Patterns and Java*, Prentice Hall, 3rd edition, 2009.

## 2. Packages





### 3. Class interfaces

Campionato
<ul style="list-style-type: none"><li>- id : int</li><li>- nomeCampionato : String</li><li>- numSquadre : int</li><li>- squadre</li><li>- calendario</li></ul>
<ul style="list-style-type: none"><li>+ getId () : int</li><li>+ setId (int id) : void</li><li>+ getNomeCampionato () : String</li><li>+ setNomeCampionato (String nomeCampionato) : void</li><li>+ getNumSquadre () : int</li><li>+ setNumSquadre (int numSquadre) : void</li><li>+ iscriviSquadra (Squadra squadra) : void</li><li>+ getSquadre() : Collection&lt;Squadra&gt;</li><li>+ aggiungiPartita () : void</li><li>+ getCalendario() : Collection&lt;Partita&gt;</li></ul>

#### Campi

- private id AS int → Id univoco del campionato
- private nomeCampionato AS String → Nome del campionato
- private numSquadre AS int → Numero delle squadre iscritte al campionato
- private Squadre → Nomi delle squadre aggiunte a un campionato
- private calendario → Date delle partite del campionato

#### Metodi

- public getId () : int → Restituisce un intero contenente l'id univoco del campionato
- public getNomeCampionato () : String → Restituisce una stringa contenente il nome del campionato

- `public getNumSquadre () : int →` Restituisce un intero contenente il numero delle squadre che partecipano al campionato
- `public setId (int id) : void →` Imposta come id del Campionato l'intero *id*
- `public setNomeCampionato (String nomeCampionato) : void →` Imposta come nome del Campionato la stringa *nomeCampionato*
- `setNumSquadre (int numSquadre) : void →` Imposta come numero delle squadre del campionato l'intero *numSquadre*
- `iscriviSquadra (Squadra squadra) : void →` Inserisce una nuova squadra al campionato
- `public getSquadre() : Collection<Squadra> →` Restituisce le Squadre che partecipano al Campionato
- `aggiungiPartita (Partita partita) : void →` Aggiunge una nuova partita al campionato
- `public getCalendario() : Collection<Partita> →` Restituisce il calendario delle partite

<b>Giocatore</b>
<ul style="list-style-type: none"> <li>- id : int</li> <li>- nome : String</li> <li>- cognome : String</li> </ul>
<ul style="list-style-type: none"> <li>+ getId () : int</li> <li>+ setId (int id) : void</li> <li>+ getNome () : String</li> <li>+ setNome (String Nome) : void</li> <li>+ getCognome () : String</li> <li>+ setCognome (String Cognome) : void</li> </ul>

### Campi

- `private id AS int →` Id univoco del giocatore
- `private nome AS String →` Nome del giocatore
- `private cognome AS String →` Cognome del giocatore

## Metodi

- `public getId () : int →` Restituisce un intero contenente l'id univoco del giocatore
- `public getNome () : String →` Restituisce una stringa contenente il nome del giocatore
- `public getCognome () : int →` Restituisce una stringa contenente il cognome del giocatore
- `public setId (int id) : void →` Imposta come id del Giocatore l'intero *id*
- `public setNome (String nome) : void →` Imposta come nome del giocatore la stringa *nome*
- `public setCognome (String cognome) : void →` Imposta come cognome del giocatore la stringa *cognome*

Informazioni
<ul style="list-style-type: none"><li>- <code>id : int</code></li><li>- <code>goal : int</code></li><li>- <code>assist : int</code></li><li>- <code>cartellino : int</code></li><li>- <code>squalifica : int</code></li><li>- <code>motivazione : String</code></li><li>- <code>partita</code></li><li>- <code>giocatore</code></li></ul>
<ul style="list-style-type: none"><li>+ <code>getId () : int</code></li><li>+ <code>setId (int id) : void</code></li><li>+ <code>getGoal () : int</code></li><li>+ <code>setGoal (int goal) : void</code></li><li>+ <code>getAssist () : int</code></li><li>+ <code>setAssist (int assist) : void</code></li><li>+ <code>getCartellino () : int</code></li><li>+ <code>setCartellino (int cartellino) : void</code></li><li>+ <code>getSqualifica () : int</code></li></ul>

- + setSqualifica (int squalifica) : void
- + getMotivazione () : String
- + setMotivazione (String motivazione) : void
- + getGiocatore ()
- + setGiocatore () : void
- + getPartita ()
- + setPartita () : void

### Campi

- private id AS int → Id univoco dell' informazione
- private goal AS int → Numero di goal segnati durante una partita
- private assist AS int → Numero di assist all'interno di una partita
- private cartellino AS int → Numero di cartellini all'interno di una partita
- private squalifica AS int → Numero di squalifiche all'interno di una partita
- private motivazione AS String → Motivazione per la squalifica di un giocatore
- private partita → Informazioni relative alla partita
- private giocatore → Informazioni relative al giocatore

### Metodi

- public getId () : int → Restituisce un intero contenente l'id univoco del Informazione
- public getGoal () : int → Restituisce un intero contenente il numero di Goal avvenuti in una partita
- public getAssist () : int → Restituisce un intero contenente il numero di assist in una partita
- public getCartellino () : int → Restituisce un intero contenente il numero di cartellini in una partita
- public getSqualifica () : int → Restituisce un intero contenente il numero delle squalifiche in una partita
- public getMotivazione () : String → Restituisce una stringa contenente la motivazione della squalifica di un giocatore
- public getGiocatore () → Restituisce le informazioni del giocatore

- `public getPartita ()` → Restituisce le informazioni della partita
- `public setId (int id) : void` → Imposta come id dell'Informazione l'intero
- `public setGoal (int goal) : void` → Imposta come numero di goal di una partita l'intero *goal*
- `public setAssist (int assist) : void` → Imposta come numero di assist di una partita l'intero *assist*
- `public setCartellino (int cartellino) : void` → Imposta come numero di cartellini di una partita l'intero *cartellino*
- `public setSqualifica (int squalifica) : void` → Imposta come numero di squalifiche di una partita l'intero *squalifica*
- `public setMotivazione (String motivazione) : void` → Imposta come motivazione di squalifica di un giocatore la stringa *motivazione*
- `public setGiocatore ()` → Imposta le informazioni del giocatore
- `public setPartita ()` → Imposta le informazioni della partita

Partita
<ul style="list-style-type: none"> <li>- id : int</li> <li>- giornata : int</li> <li>- goalCasa : int</li> <li>- goalOspite : int</li> <li>- data</li> <li>- campionato</li> <li>- squadraCasa</li> <li>- squadraOspite</li> <li>- arbitro</li> </ul>
<ul style="list-style-type: none"> <li>+ getId () : int</li> <li>+ setId (int id) : void</li> <li>+ getGiornata () : int</li> <li>+ setGiornata (int giornata) : void</li> <li>+ getGoalCasa () : int</li> </ul>

```
+ setGoalCasa (int goalCasa) : void
+ getGoalOspite () : int
+ setGoalOspite (int goalOspite) : void
+ getData ()
+ setData () : void
+ getCampionato ()
+ setCampionato () : void
+ getSquadraCasa ()
+ getSquadraOspite ()
+ getArbitro ()
+ setArbitro (): void
+ setCasa (): void
+ setOspite (): void
```

### Campi

- private id AS int → Id univoco della partita
- private giornata AS int → Ciascuno dei giorni prefissati in cui le squadre giocano le partite
- private goalCasa AS int → Numero di goal fatti in casa
- private goalOspite AS int → Numero di goal fatti dalla squadra ospite
- private data → Data in cui si svolge la partita
- private campionato → Campionato in cui si svolge la partita
- private squadraCasa → Squadra che gioca nel proprio stadio
- private squadraOspite → Squadra che non gioca nel proprio stadio
- private arbitro → Persona che arbitra la partita

### Metodi

- public getId () : int → Restituisce un intero contenente l'id univoco della partita
- public getGiornata () : int → Restituisce un intero contenente il numero della giornata

- `public getGoalCasa () : int →` Restituisce un intero contenente il numero di goal effettuati dalla squadra che gioca in casa
- `public getGoalOspite () : int →` Restituisce un intero contenente il numero di goal effettuati dalla squadra che non gioca in casa
- `public getData () →` Restituisce la data in cui si è svolta la partita
- `public getCampionato () →` Restituisce il campionato il cui si svolge la partita
- `public getSquadraCasa () →` Restituisce il nome della squadra che gioca in casa
- `public getSquadraOspite () →` Restituisce il nome della squadra che gioca fuori casa
- `public getArbitro () →` Restituisce l'arbitro della partita
- `public setId (int id) : void →` Imposta come id della partita l'intero *id*
- `public setGiornata (int giornata) : void →` Imposta come numero di giornata l'intero *giornata*
- `public setGoalCasa (int goalCasa) : void →` Imposta come numero di goal fatti dalla squadra di casa l'intero *goalCasa*
- `public setGoalOspite (int goalOspite) : void →` Imposta come numero di goal fatti dalla squadra ospite l'intero *goalOspite*
- `public setData () : void →` Imposta come data in cui avviene la partita *data*
- `public setCampionato () : void →` Imposta come Campionato nel quale si fa la partita *campionato*
- `public setArbitro () : void →` Imposta come colui che arbitra la partita *arbitro*
- `public setCasa () : void →` Imposta come squadra di casa *casa*
- `public setOspite () : void →` Imposta come squadra ospite *ospite*

#### Squadra

- `id : int`
- `nomeSquadra : String`
- `vittorie : int`
- `pareggi : int`
- `sconfitte : int`
- `goalFatti : int`
- `goalSubiti : int`

- statolscrizione : int
- allenatore
- rosa
- campionato

- + getId () : int
- + setId (int id) : void
- + getNomeSquadra () : String
- + setNomeSquadra (String NomeSquadra) : void
- + getVittorie () : int
- + setVittorie (int Vittorie) : void
- + getPareggi () : int
- + setPareggi (int Pareggi) : void
- + getSconfitte () : int
- + setSconfitte (int Sconfitte) : void
- + getGoalFatti () : int
- + setGoalFatti (int GoalFatti): void
- + getGoalSubiti () : int
- + setGoalSubiti (int GoalSubiti) : void
- + getStatolscrizione () : int
- + iscriviSquadra () : void
- + confermaSquadra () : void
- + getAllenatore ()
- + setAllenatore (): void
- + getRosa ()
- + aggiungiGiocatore () : void
- + rimuoviGiocarore () : void
- + getCampionato ()
- + setCampionato () : void

### Campi

- private id AS int → Id univoco della squadra



- private nomeSquadra AS String → Nome della squadra
- private vittorie AS int → Numero di vittorie della squadra
- private pareggi AS int → Numero di pareggi fatti dalla squadra
- private sconfitte AS int → Numero di sconfitte subite
- private goalFatti AS int → Numero di goal fatti dalla squadra
- private goalSubiti AS int → Numero di goal subiti
- private statoScrizione AS int → Numero dello stato di Iscrizione
- private allenatore → Nome dell' allenatore della squadra
- private rosa → elenco dei giocatori iscritti nella squadra
- private campionato → Nome del campionato in cui è iscritta la squadra

## Metodi

- public getId () : int → Restituisce un intero contenente l'id univoco della Squadra
- public getNomeSquadra () : string → Restituisce una stringa contenente il nome della Squadra
- public getVittorie () : int → Restituisce un intero contenente il numero di vittorie della squadra
- public getPareggi () : int → Restituisce un intero contenente il numero di pareggi della squadra
- public getSconfitte () → Restituisce un intero contenente il numero di sconfitte della squadra
- public getGoalFatti () → Restituisce un intero contenente il numero di goal fatti dalla squadra
- public getGoalSubiti () → Restituisce il numero di goal subiti dalla squadra
- public getStatoIscrizione () → Restituisce lo stato di Iscrizione della squadra
- public iscriviSquadra () → Cambia lo stato della squadra (statoIscrizione = ATTESA\_CONFERMA)
- public confermaSquadra () → Cambia lo stato della squadra (statoIscrizione = SQUADRA\_ISCRITTA)
- public getAllenatore () → Restituisce l'allenatore della squadra
- public getRosa () → Restituisce tutti i giocatori della squadra
- public aggiungiGiocatore () : void → Aggiunge un giocatore

- `public rimuoviGiocatore () : void` → Rimuove un giocatore
- `public getCampionato ()` → Restituisce il campionato
- `public setId (int id) : void` → Imposta come id della squadra l'intero *id*
- `public setNomeSquadra (String nomeSquadra) : void` → Imposta come nome della squadra la stringa *nomeSquadra*
- `public setVittorie (int vittorie) : void` → Imposta come numero di vittorie della squadra l'intero *vittorie*
- `public setPareggi (int pareggi) : void` → Imposta come numero di pareggi della squadra l'intero *pareggi*
- `public setSconfitte (int sconfitte) : void` → Imposta come numero di sconfitte della squadra l'intero *sconfitte*
- `public setGoalFatti (int goalFatti) : void` → Imposta come numero di goal fatti dalla squadra l'intero *goalFatti*
- `public setGoalSubiti (int goalSubiti) : void` → Imposta come numero di goal subiti dalla squadra l'intero *goalSubiti*
- `public setAllenatore () : void` → Imposta come allenatore della squadra *allenatore*
- `public setCampionato () : void` → Imposta come Campionato in cui partecipa la Squadra *campionato*

Utente
<ul style="list-style-type: none"> <li>- Id : int</li> <li>- Nome : String</li> <li>- Cognome : String</li> <li>- Email : String</li> <li>- Password : String</li> </ul>
<ul style="list-style-type: none"> <li>+ getId () : int</li> <li>+ setId (int id) : void</li> <li>+ getNome () : String</li> <li>+ setNome (String Nome) : void</li> <li>+ getCognome () : String</li> </ul>

- + setCognome (String Cognome) : void
  - + getEmail () : String
  - + setEmail (String Email) : void
  - + getPassword (): String
  - + setPassword (String Password) : void

### Campi

- private id AS int → Id univoco della squadra
- private nome AS String → nome dell'utente
- private cognome AS String → cognome dell'utente
- private email AS String → email dell'utente
- private password AS String → password dell'utente

### Metodi

- public getId () : int → Restituisce un intero contenente l'id univoco della Squadra
- public getNome () : String → Restituisce una stringa contenente il nome dell'utente
- public getCognome () : String → Restituisce una stringa contenente il cognome dell'utente
- public getEmail () : String → Restituisce una stringa contenente l'e-mail dell'utente
- public getPassword () : String → Restituisce una stringa contenente la password dell'utente
- public setId (int id) : int → Imposta come id dell'utente l'intero *id*
- public setNome (String nome) : String → Imposta come nome dell'utente la stringa *nome*
- public setCognome (String cognome) : String → Imposta come cognome dell'utente la stringa *cognome*
- public setEmail (String email) : String → Imposta come email dell'utente la stringa *email*
- public setPassword (String password) : String → Imposta come password dell'utente la stringa *password*



## 4. Glossary

### A

- Attore: è un'entità fuori dal sistema che deve essere modellata e che interagisce con il sistema.

### B

### C

- Class Diagram: Rappresenta le classi del sistema con i loro attributi e operazioni. Inoltre mostra le relazioni tra le classi (associazioni, relazioni e gerarchie di specializzazione/generalizzazione); può essere utilizzato a diversi livelli di dettaglio (nell'analisi e nel design del sistema).
- Entry condition: E' un vincolo che deve essere soddisfatto quando viene invocata per la prima volta un'operazione.
- Exit condition: E' un vincolo che deve essere soddisfatto quando termina l'operazione.

### D

- Diagram: Rappresentazione grafica di una collezione di elementi del modello. UML supporta i seguenti diagrammi: dei casi d'uso, di sequenza, di collaborazione, di attività, di stato, delle classi, degli oggetti, dei comportamenti e di dispiegamento.

### E

- Extend: Relazione tra un caso d'uso estendente e uno base, che specifica come il comportamento definito dallo use case estendente è incorporato nel comportamento del caso d'uso base. Extends viene usato anche per definire dei casi eccezionali di un caso d'uso.

### F

- FLM: Football League Manager, sistema che permette la creazione e la gestione di un campionato.

G

H

I

- Include: E' un tipo di relazione che serve a fattorizzare una parte del comportamento di un caso d'uso e metterlo in un altro caso d'uso.

L

- Login: La login è un'informazione che serve per accedere ad un sistema, è necessaria ma non sufficiente; ad essa è sempre associata ad una password che è rappresentata da una stringa alfanumerica.
- Logout: Rappresenta l'uscita dal sistema corrente che impone come prossima operazione un nuovo accesso al sistema con un account differente.

M

- Modello: E' un concetto astratto che descrive un sottoinsieme del sistema.
- Mock-ups: Produzione completa dell'interfaccia utente.

N

O

- Object Model: Modello di tipo UML che chiarisce le relazioni tra vari oggetti identificati per lo sviluppo del sistema software. Comprende anche le molteplicità, i versi e gli attributi delle relazioni tra gli oggetti identificati in fase di progettazione.

P

Q

## R

- Requisiti non funzionali: Requisito che specifica proprietà richieste al sistema, come vincoli ambientali e di sviluppo, prestazioni, dipendenze dalla piattaforma, di manutenibilità, estensibilità, sicurezza e affidabilità. Requisito che sancisce vincoli di carattere fisico relativi ai requisiti funzionali.
- Requirements analysis: Insieme di dati utili a chiarire al cliente tutto ciò che concerne le decisioni iniziali, i requisiti, gli obiettivi e i modelli relativi alla progettazione del sistema software da sviluppare.

## S

- Scenario: Un'istanza di un caso d'uso.
- Sequence Diagram: È utilizzato per definire la logica di uno scenario (specifica sequenza di eventi) di un caso d'uso. È uno dei principali input per l'implementazione dello scenario; mostra gli oggetti coinvolti specificando la sequenza temporale dei messaggi che gli oggetti si scambiano.
- Statechart Diagram: È normalmente utilizzato per modellare il ciclo di vita degli oggetti di una singola classe; mostra gli eventi che causano la transizione da uno stato all'altro, le azioni eseguite a fronte di un determinato evento.

## T

## U

- UML: È uno standard usato per modellare software orientato agli oggetti.
- Use Case: Rappresenta un possibile modo di utilizzo del sistema e descrive le interazioni tra esso e gli attori.
- Use Case Diagram: Diagramma che serve a descrivere il comportamento funzionale del sistema da come è percepito dall'utente.
- Utente non registrato: Colui che interagisce con il sistema per visualizzare le partite e le classifiche di un campionato.
- Utente Registrato: Colui che interagisce col sistema per usufruire dei servizi che il sistema offre.

V

Z