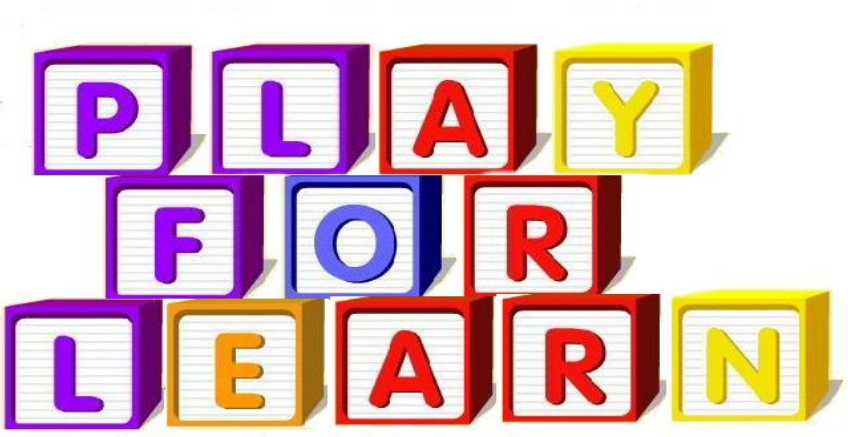
UNIVERSITÀ DEGLI STUDI DI SALERNO

Ingegneria del Software

Object Design Document

ANNO ACCADEMICO 2016/2017

Versione 1.2



Partecipanti:

NOME	MATRICOLA
Della Porta Raffaele	0512102538

Revision History:

DATA	VERSIONE	DESCRIZIONE	AUTORE
14/02/2017	1.1	Inserimento dei package e interfaccia delle classi	Raffaele Della Porta
19/02/2018	1.2	Revisione delle interfaccia delle classi con inserimento di	Raffaele Della Porta

		pre e post-conditions	

Indice

1. Introduzione

Questo documento, usato come supporto dell'implementazione, ha lo scopo di produrre un modello capace di interagire in modo coerente e preciso tutti i servizi individuati nelle fasi precedenti. In particolare, definisce l'interfaccia delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design. Nei paragrafi successivi sono specificati i trade-off, le linee guida e i design pattern per l'implementazione.

1.1 Compromessi dell'Object Design

- **Comprensibilità vs Tempo:** La comprensibilità del codice è un aspetto molto importante, soprattutto per la fase di testing. Ogni classe e metodo deve essere facilmente interpretabile anche da chi non ha collaborato nel progetto. Nel codice vengono usati i commenti standard per facilitare la comprensione del codice sorgente. Ovviamente questa

caratteristica aggiunge un incremento di tempo allo sviluppo del nostro progetto.

- **Prestazione vs Costi:** Non avendo a disposizione alcun budget, utilizzeremo materiale open source per la realizzazione del sistema, con lo scopo di renderlo sia performante che efficiente.
- **Interfaccia vs Easy-Use:** L'interfaccia grafica del sistema grazie all'utilizzo di layout accuratamente organizzati ad-hoc si presenta semplice ed intuitiva, permettendo una facile gestione anche del database(Easy-Use).
- **Sicurezza vs Efficienza:** Nel nostro sistema la sicurezza non è un aspetto fondamentale, in quanto non gestiamo dati sensibili. A causa dei i tempi piuttosto limitati, ci limiteremo ad implementare un sistema di sicurezza basato sul login dell'utente (username e password).
- **Tempo di risposta vs Spazio di memoria:** La scelta di utilizzare un Database relazionale è scaturita da diversi vantaggi, tra cui:
    - Gestione consistente dei dati.
    - Accesso veloce e concorrente ai dati.
    - Tempo di risposta basso rispetto all'utilizzo del file system.

  Ci sono ovviamente anche degli svantaggi in termini di spazio, un BD richiede più spazio in memoria.

1.3 Definizioni, acronimi, abbreviazioni

In questa sezione si specificano gli acronimi e le abbreviazioni utilizzate nel seguito. Essi, infatti, pur essendo di uso comune, potrebbero indurre a interpretazioni personali, quindi potenzialmente diverse da quelle sottintese in questa trattazione.

- PFL: PlayForLearn, nome del sistema che verrà sviluppato.
- Alunno: Utente del sistema.
- Insegnante: Amministratore del sistema.
- UI: user interface.
- SDD: System Design Document.
- HW/SW: Hardware/Software.
- ODD: Object Design Document.
- DBMS: Database Management System.

1.3 Riferimenti

Per la realizzazione del sistema sono stati utilizzati i seguenti materiali di riferimento:

- Android guida: https://developer.android.com/index.html
- Android Programming: The Big Nerd Ranch Guide: http://www.bignerdranch.com/we-write/android-programming.html
- B.Bruegge, A.H. Dutoit, Object Oriented Software Engineering - Using UML, Patterns and Java, Prentice Hall, 3rd edition, 2009.

2. Linee Guida per l'implementazione

2.1. Nomi dei file

Il software PFL è stato sviluppato in Android, quindi utilizza i seguenti suffissi per i file:

- Per i file di layout il suffisso per indicare l'estensione è .xml
- Per i sorgenti Java il suffisso per indicare l'estensione è .java

2.2. Organizzazione dei file

Un file consiste in sezioni che dovrebbero essere separate da linee vuote e da un commento che identifica ogni sezione. File più grandi di 700/800 righe sono poco leggibili e devono essere evitati.

2.3. File sorgenti e di layout

Ogni file sorgente o di layout conterrà una singola classe pubblica o un'interfaccia. I file sorgenti Java sono composti secondo questa struttura:

```
/**
 * Nome della classe
 *
 * Descrizione
 *
 * Informazioni di versione
 */
package it.unisa.di.mp.threadno;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;
```

Commenti di inizio. Tutti i file sorgenti devono iniziare con un commento in stile Java, che elenca il nome della classe, descrizione e informazioni sulla versione.

Istruzioni di package e import. La prima istruzione sarà quella del package, seguita da una serie di import.

Dichiarazione di classi. L'ordine in cui una classe deve apparire è il seguente:

- Commento di documentazione della classe (/** */)
- Istruzioni della classe java.

- Commento di implementazione della classe se necessario. Questo commento deve contenere informazioni generali sulla classe e di come viene implementata.
- Variabili di istanza.
- Costruttore e infine i metodi**.**

2.4. Database

Il database è organizzato in tabelle, i nomi delle tabelle devono seguire le seguenti regole:

- Sono costituiti da sole lettere minuscole
- Se il nome è costituito da più parole, queste sono separate da un underscore (_);
- Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.
- il nome delle tabelle che rappresentano relazioni devono essere verbi all'infinito.

I nomi dei campi devono seguire le seguenti regole:

- Sono costituiti da sole lettere minuscole;
- Se il nome è costituito da più parole, queste sono separate da un underscore (_);
- Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

3. Indentazione

3.1. Lunghezza delle linee

Evitare linee di lunghezza superiore a 80 caratteri, poiché non sono di facile lettura; lo stesso vale anche per la documentazione, bisogna evitare frasi molto lunghe e articolate.

3.2. Spostamento di linee

Quando un'espressione supera la lunghezza della linea, occorre spezzarla secondo i seguenti principi generali:

- Interrompere la linea dopo la virgola;
- Interrompere la linea prima di un operatore;
- Allineare la nuova linea con l'inizio dell'espressione della linea precedente

4. Dichiarazioni

Per una maggiore pulizia e leggibilità del codice è consigliabile dichiarare una variabile per riga, o anche più variabili dello stesso tipo sulla stessa linea, ma mai di tipi diversi. Così facendo è possibile inserire un commento al fianco dell'elemento che ci interessa, nel caso in cui ce ne fosse la necessità.

4.1. Dichiarazioni di classi

Quando si codificano le classi, si dovrebbero rispettare le seguenti regole di formattazione:

- Non mettere spazi tra il nome del metodo e le parentesi "(" che apre la lista dei parametri;
- La parentesi graffa aperta "{" si trova alla fine della stessa linea dell' istruzione di dichiarazione;
- La parentesi graffa chiusa "}" va inserita dopo l'ultima linea di codice della classe, all'inizio della linea successiva, mantenendo la tabulazione della linea precedente.

4.2. Commenti

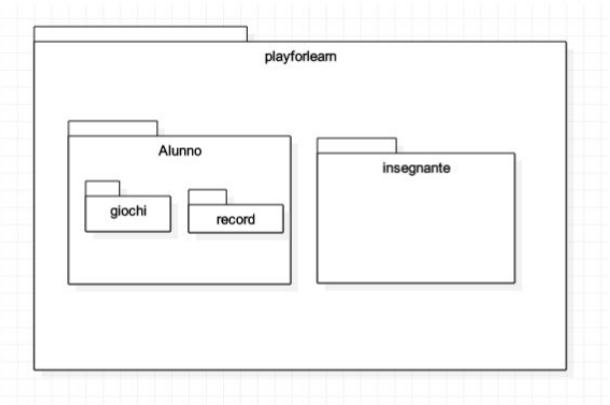
Nella scrittura del codice verranno utilizzati commenti di implementazione, esplicativi rispetto la logica utilizzata e commenti di documentazione, esplicativi rispetto le funzioni del codice. I commenti di documentazione sono in formato javadoc, così da facilitare una eventuale rilettura in fase di ristrutturazione. Mentre i commenti nei file .xml sono delimitati da (<!-- -->). I commenti per i file di layout vengono utilizzati per una descrizione dei vari layout che si vogliono presentare.I programmi possono avere due tipi di commenti: commenti a linea singola e commenti multilinea. I commenti a linea singola vengono indicati con "//", sono brevi commenti che possono apparire su singola linea di codice ed indentati a livello del codice che seguono. Mentre i commenti multilinea sono delimitati da "/*" e da "*/", il testo tra i due token è un commento.

3. Design Pattern

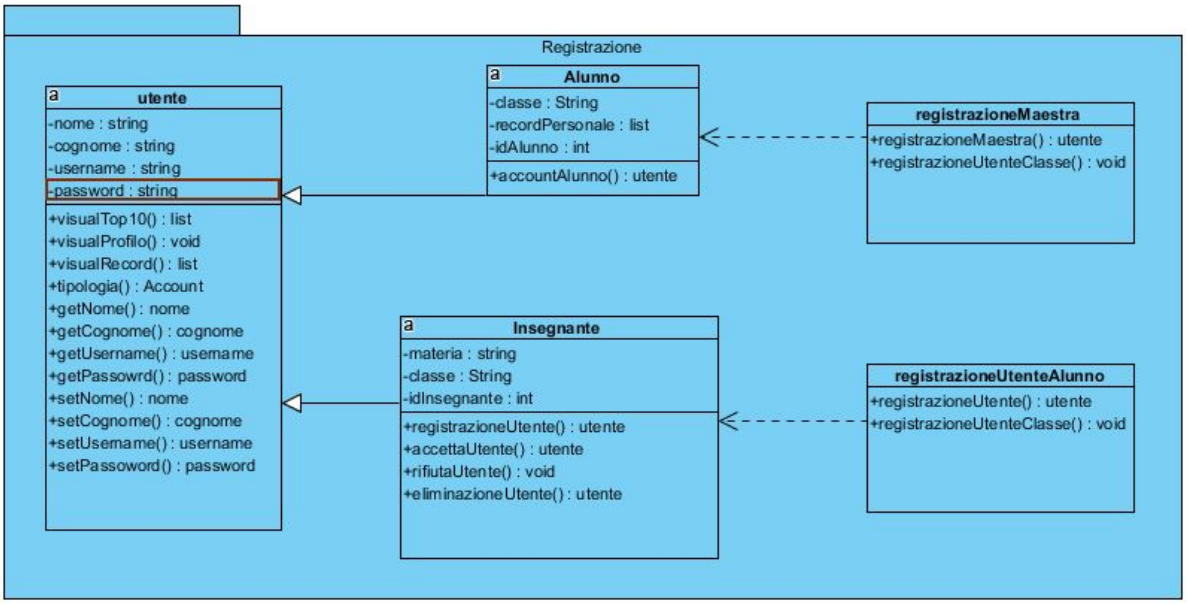
Non vi è la necessità di usare pattern di programmazione.

4. Packages

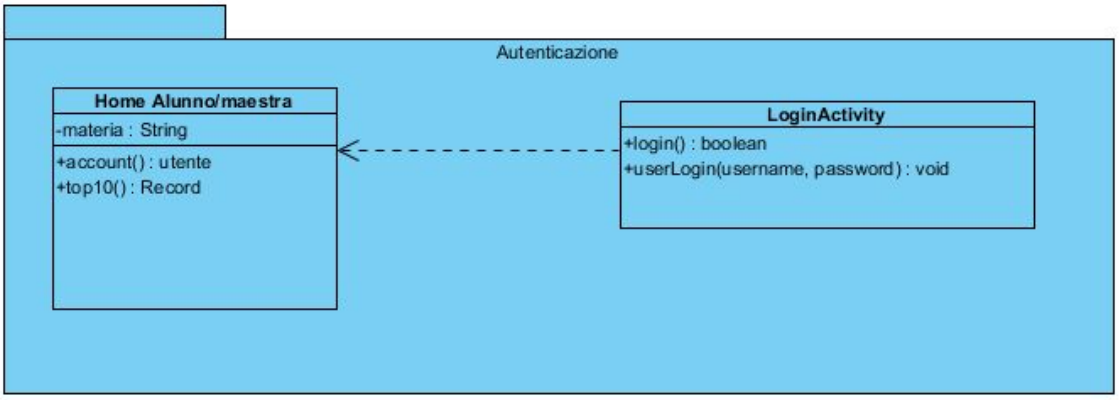
La IDE di programmazione di Android, Android Studio organizza il codice in package, organizzando in modo analogo anche i file relativi all'installazione e all'avvio del sistema.



Questo package riguarda la gestione dell'account e all'interno sono definite le attività per aggiungere account.

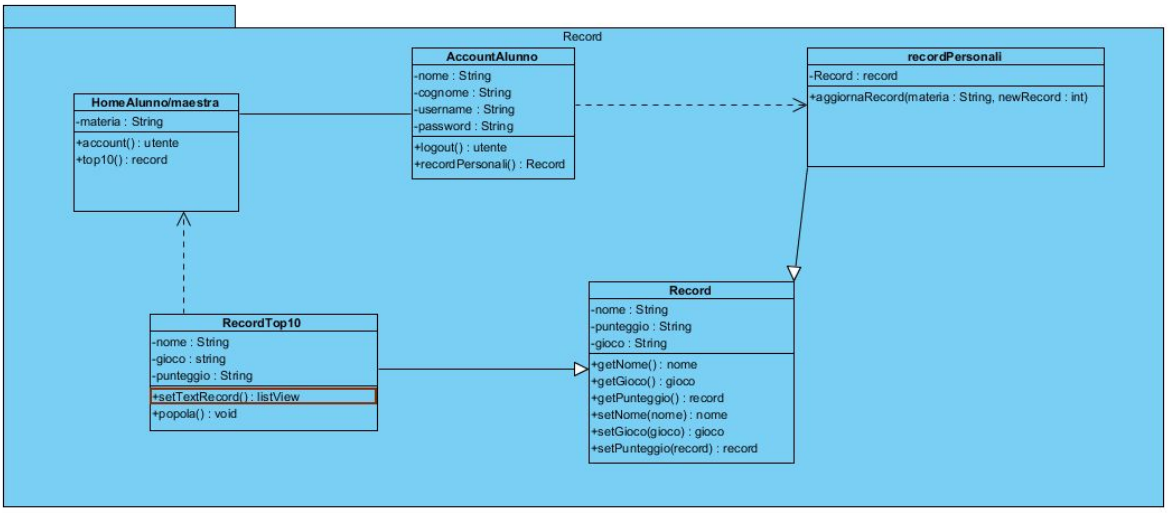


Questo package riguarda la gestione dell'autenticazione e riguarda le attività per effettuare login e logout.

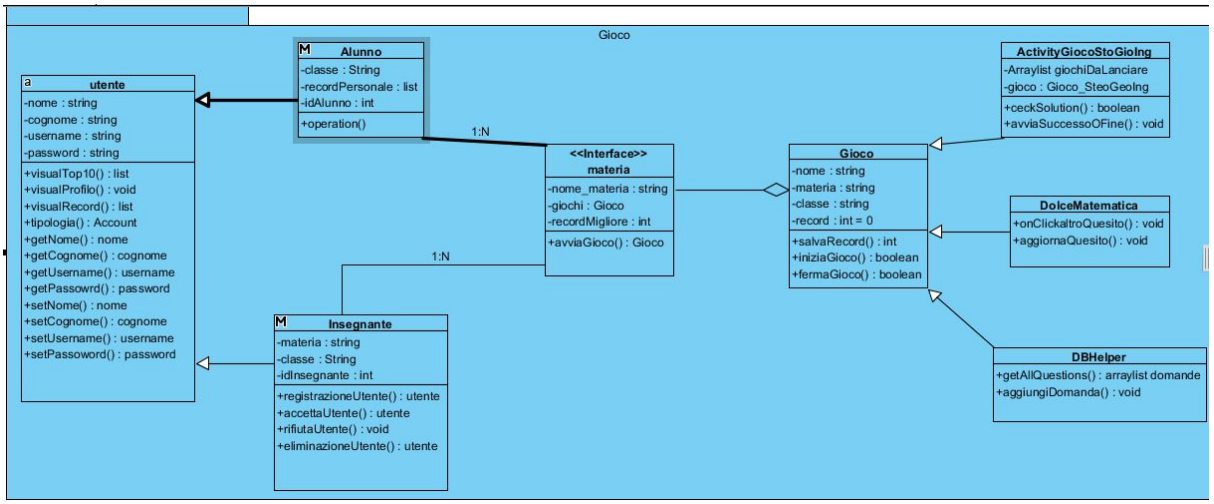


Controller	Metodo	Parametri	Descrizione
login()	POST	$us,$pw	Metodo che verifica username e password per effettuare il login.
logout()	POST	nessun parametro	Metodo che effettua il logout.
userLogin() : Request	POST	$a:Utente	Metodo che verifica username e password per l'autenticazione al sistema.

Questo package riguarda la gestione dei record e la relativa top10 degli alunni dell'istituto.



Questo package riguarda la gestione dei giochi con i relativi inerenti giochi.



5. Interfaccia delle classi

Di seguito saranno elencate le classi che si occupano delle interazioni tra gli oggetti.

Gestione Login.

Specifica contracts con OCL

context Login:: userLogin(u) pre: not isUtenteLogin(u) u:Utente.

context Login:: userLogin(u) post: isUtenteLogin(u)

context Login:: logout(u) pre: isUtenteLogin(u)

context Login:: logout(u) post: not isUtenteLogin(u)

Gestione Account.

Context Utente inv: nome.lenght<=20

Context Utente inv : cognome.lenght<=20

Context Utente inv: username.lenght<=20

Context Utente inv: password.lenght <=10

Context Utente:: visualProfilo(u) pre : not isUtenteLogin(u)

Context Utente:: visualProfilo(u) post: isUtenteLogin(u)

Context Utente:: registrazioneUtente(u) pre: not isUtenteRegister(u)

Context Utente:: registrazioneUtente(u) post: isUtenteRegister(u)

Context Utente:: eliminazioneUtente(u) pre: isUtenteRegister(u)

Context Utente:: eliminazioneUtente(u) post: not isUtenteRegister(u)

Gestione Record.

Context Record inv: getPunteggio()>=0

Context Record:: visualRecordPersonale(r) pre: isStartedGame(g)

Context Record:: visualRecordPersonale(r) post: not isStartedGame(g)

Context Record:: visualTop10(r) pre: isStartedGame(g)

Context Record:: visualTop10(r) post: not isStartedGame(g)

Gestione giochi.

Context Gioco inv: nome.lenght<=15

Context Gioco inv: descrizione.lenght<=100

Context Gioco:: iniziaGioco(g) pre: isUtenteLogin(u)

Context Gioco:: iniziaGioco(g) post: isStartedGame(g)

Context Gioco:: fermaGioco(g) pre: isStartedGame(g)

Context Gioco:: fermaGioco(g) post: not isStartedGame(g)

Context Gioco:: salvaRecord(r) pre: isStartedGame(g)

ContextGioco:: salvaRecord(r) post: not isStartedGame(g)

ContextGioco:: salvaRecord(r) post: visualRecordPersonale(r)

6. Glossario

Termine	Descrizione
Attributo	Rappresenta una proprietà di un oggetto; è definito da un nome, un tipo e può avere un valore di default. Gli attributi definiscono lo stato dell'oggetto, e non sono condivisi con altri oggetti.
Accoppiamento	Il grado di dipendenza tra due elementi.
Database relazionali	È una raccolta di informazioni di vario tipo, strutturare in modo da essere facilmente reperibili in base ad una chiave di ricerca primaria determinata. Le tabelle contengono dati logicamente correlati e sono messe in relazione tra loro.
Override	Operazione di ridefinizione di attributi e/o metodi in sottoclassi.
JavaDoc	Strumento che estrae dai commenti di un programma, la documentazione

	dettagliata del codice.
Classe	Astrazione che specifica una categoria di oggetti, individuati da comportamenti e caratteristiche simili.
Interfaccia	L’insieme di tutte le signature definite per le operazioni di un oggetto. L’interfaccia definisce l’insieme delle richieste alle quali l’oggetto può rispondere.