Laurea Triennale in Informatica - Università di Salerno
Corso di Ingegneria del Software - Prof. C.Gravino

ODD
Object Design Document
SmartGym

Riferimento	thISTeam_ODD
Versione	1.0
Data	23/01/2023
Destinatario	Docente di Ingegneria del Software 2022/23 (Carmine Gravino)
Presentato da	thISTeam
Approvato da	

Laurea Triennale in Informatica - Università di Salerno
Corso di Ingegneria del Software - Prof. C.Gravino

Revision History

Data	Versione	Descrizione	Autori
23/01/2023	0.1	Creazione del documento	thISTeam
25/01/2023	0.2	Stesura dell'introduzione	thISTeam
26/01/2023	0.3	Stesura del capitolo 2	thISTeam
27/01/2023	0.4	Stesura del capitolo 4-5	thISTeam
13/01/2023	0.5	Definite le class interfaces	thISTeam
17/02/2023	1.0	Revisione	thISTeam

Laurea Triennale in Informatica - Università di Salerno
Corso di Ingegneria del Software - Prof. C.Gravino

Team Members

Nome	Ruolo nel progetto	Acronimo	Informazioni di contatto
Daniele Fabiano	Team Member	DF	d.fabiano3@studenti.unisa.it
Mariantonietta Maselli	Team Member	MM	m.maselli3@studenti.unisa.it
Michele Spinelli	Team Member	MS	m.spinelli26@studenti.unisa.it

Laurea Triennale in Informatica - Università di Salerno
Corso di Ingegneria del Software - Prof. C.Gravino

Indice

1. Introduzione

SmartGym propone di semplificare la distribuzione e la fruizione della scheda esercizi degli atleti iscritti alla palestra, al fine di ridurre l'uso del cartaceo creando uno strumento di digitalizzazione dei processi che portano alla realizzazione di una scheda esercizi.

In questa prima sezione del documento sono riportate le linee guida per la scrittura del codice utilizzate poi nella fase di implementazione. Sono anche elencate definizioni, acronimi e abbreviazioni di alcuni termini tecnici, oltre che riferimenti e link utili ai documenti passati.

1.1 Linee Guida per la scrittura del codice

Risulta necessario seguire alcune linee guida e convenzioni per realizzare delle interfacce corrette. Lo sviluppo android nativo fa uso del linguaggio Java, per cui sono da prendere in considerazione tutte le code conventions per Java. Inoltre Android fa uso del linguaggio XML per la definizione dei layout e della configurazione dell'app. Sono linkati di seguito varie informazioni su linee guida e regole da rispettare per la corretta scrittura di codice Java in Android e degli specifici tag xml da utilizzare

- Code Conventions for the Java Programming Language: Contents
- App resources overview | Android Developers
- App Manifest Overview | Android Developers
- Stile di codice Java AOSP per i contributori

1.2 Definizione, acronimi e abbreviazioni

In questa sezione vanno inserite tutte le definizione e abbreviazioni, acronimi

- **Package:** una raccolta di classi in un unico gruppo per organizzare al meglio la struttura del progetto
- **Gradle:** sistema di build per applicazioni Java
- **JavaDoc:** strumento fornito dal linguaggio Java, per generare la documentazione del codice a partire dai commenti inseriti nelle varie classi

1.3 Riferimenti e Link Utili

Di seguito una lista di riferimenti ad altri documenti utili durante la lettura.
Link ai precedenti documenti:

- SOW
- RAD
- SDD

Link al documento di tracciabilità dei vincoli:

- Tracciabilità vincoli

Link alla documentazione Android dove viene spiegato nel dettaglio la struttura del progetto:

- Projects overview | Android Developers

2. Packages

Il sistema è stato organizzato in package, così come definito dalla suddivisione dei sottosistemi nel documento di System Design. La struttura delle directory è definita secondo le scelte architetturali prese e si rifà alla struttura standard di directory stabilita dal progetto Android e dal sistema di build Gradle. Il progetto è visualizzabile sia attraverso la struttura concettuale definita da Android che dalla struttura fisica definita da Gradle.

Vista del progetto concettuale (Android View)

- **manifests:** contiene il file di manifesto *AndroidManifest.xml* necessario per la configurazione dell'applicazione e della dichiarazione di servizi e activity aggiuntive
- **java:** contiene il codice sorgente dell'applicazione, in questo caso scritto nel linguaggio Java
- **res:** contiene tutte le risorse che non sono codice sorgente, come i file XML dei layout, e risorse multimediali, ognuna nella corrispondente sottocartella.
- **Gradle Scripts:** raccoglie tutti i file di build e della configurazione di build del progetto

Vista del progetto fisica (Project View)

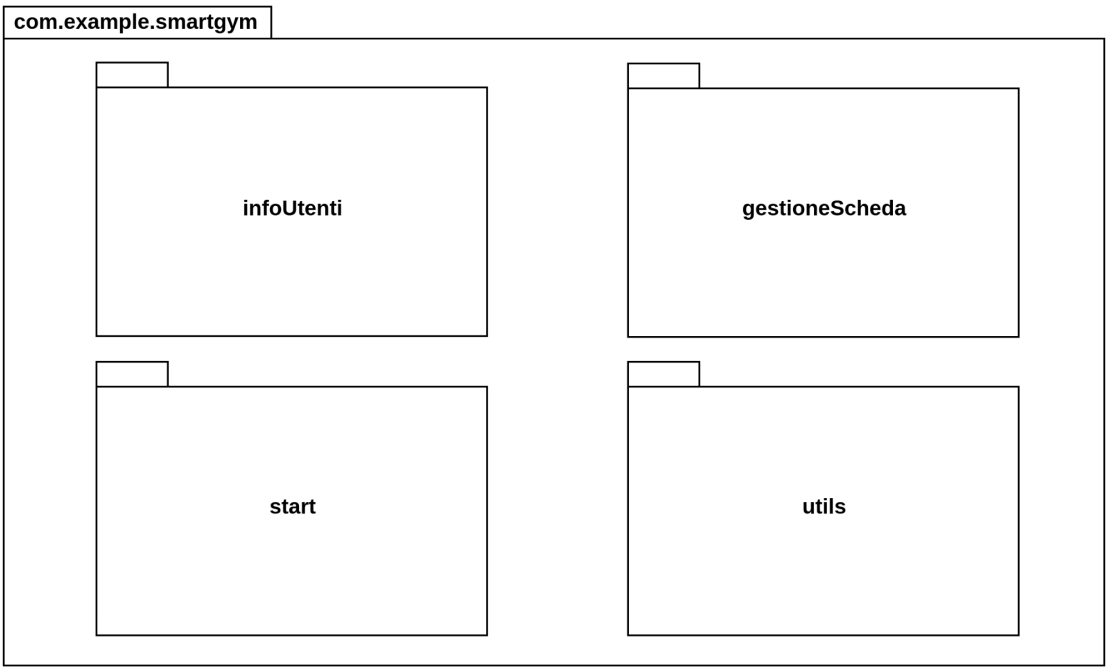
- **module-name/:** cartella root del progetto dove sono contenute tutte le altre sottocartelle e i file di configurazione della build per importare moduli e librerie
- **build/:** contiene tutti i file di output della build
- **libs/:** se presente, contiene delle librerie private aggiunte manualmente
- **src/:** contiene tutte le risorse relative a codice sorgente e codice di test nella sottocartella **java**, layout e contenuti multimediali nella sottocartella **res**

Package smartgym
Il package principale del sistema è stato realizzato seguendo l'organizzazione per sottosistemi e dell'architettura Three-Tier:

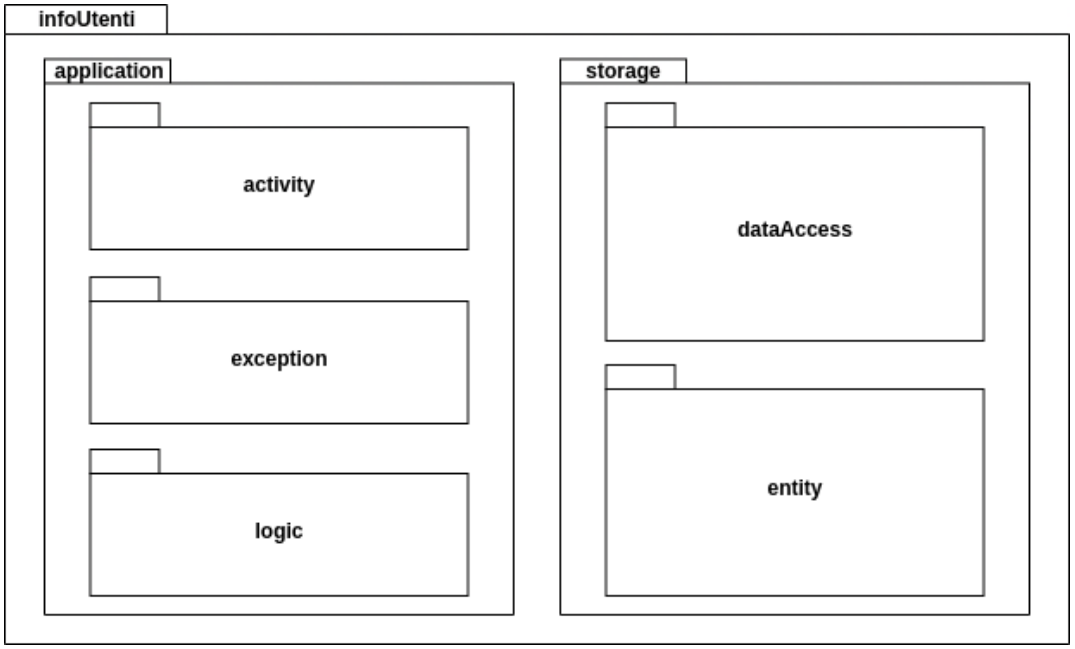
- I package di sottosistema sono organizzati come segue:
  - **application:** al suo interno sono presenti tutte le classi relative alla logica applicativa,di controllo e di quella dei layout, opportunamente raccolte in:
    - **activity:** contiene le classi che estendono AppCompatActivity o Fragment e si occupano della logica relativa ai layout
    - **exception:** contiene le classi che realizzano eccezioni personalizzate
    - **logic:** contiene le classi che si occupano realmente della logica applicativa
  - **storage:** al suo interno sono presenti tutte le classi entity e quelle relative alla logica di accesso alla persistenza dei dati, opportunatamente raccolte in:
    - **dataAccess:** contiene tutte le classi che si occupano di interrogare il database e restituire i risultati delle query
    - **entity:** contiene tutte le classi che modellano le entità del sottosistema

- **start:** contiene le classi che si occupano dell'avvio dell'app e della homepage

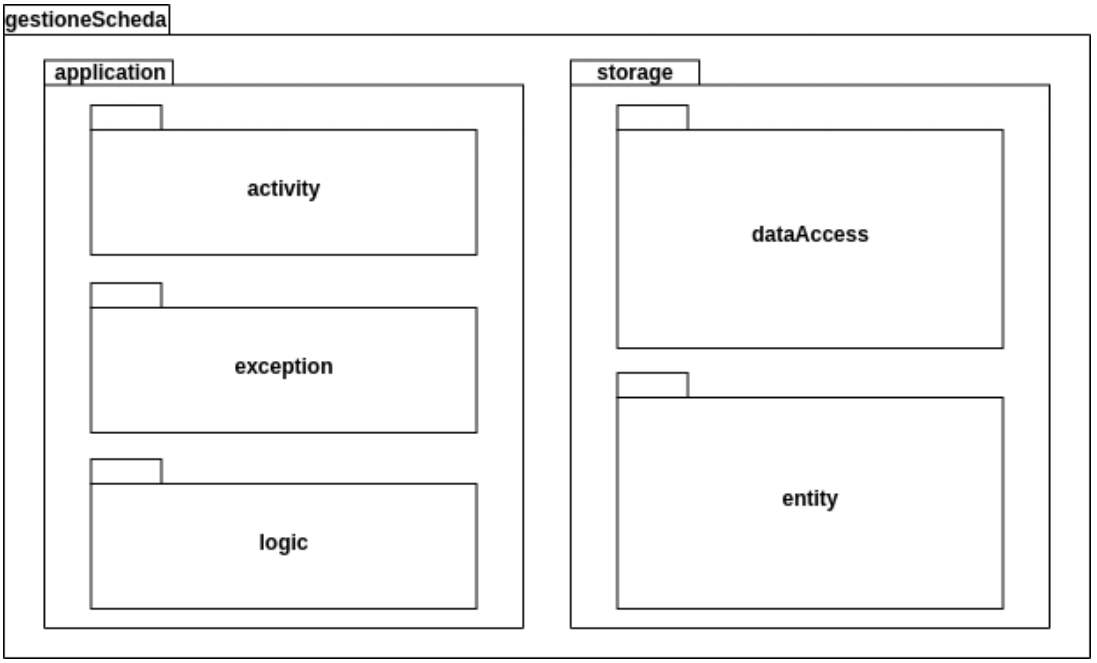
- **utils:** contiene le classi che possono essere utili su diversi sottosistemi



Package infoUtenti



Package gestioneScheda



3. Class Interfaces

In questa sezione saranno presentate le interfacce dei due package che identificano i sottosistemi implementati.

[Javadoc di SmartGym]

package infoUtenti

Nome classe	InfoUtentiService
Descrizione	Questa classe permette di gestire le operazioni relative al sottosistema InfoUtenti (login, logout, registrazione modifica e cancellazione del profilo)
Metodi	+login(String email, String password): Task<AuthResult> +getUserLogged():FirebaseUser +createUser(String email, String password): Task<AuthResult> +saveAthlete(Atleta atleta, String id):Task<Void> +logout():void +getAthletebyId(Atleta atleta, String id):Task<DocumentSnapshot> +editAthleteInfo(Atleta atleta, String id):Task<Void> +insertAthleteFeatures(Atleta atleta, String id):Task<Void> +editAthleteFeatures(Atleta atleta, String id):Task<Void> +deleteUser():Task<Void> +deleteAthlete():Task<Void>
Invariante di classe	/

Nome metodo	createUser(String email, String password)
Descrizione	Questo metodo permette di creare il record di autenticazione dell'utente
Pre-condizione	/
Post-Condizione	**context:** InfoUtentiService::createUser(String email, String password) **post:** LoginRegistration.createUser(email, password).isSuccessful() == true

Nome metodo	**saveAthlete(Atleta atleta, String id)**
Descrizione	Questo metodo permette di salvare le informazioni anagrafiche relative all'utente atleta sul documento con relativo id
Pre-condizione	**context:** InfoUtentiService::saveAthlete(Atleta atleta, String id) **pre:** createUser(email, password)
Post-condizione	**context:** InfoUtentiService::saveAthlete(Atleta atleta, String id) **post:** LoginRegistration.saveAthlete(Atleta atleta, String id).isSuccessful() == true
Nome metodo	**login(String email, String password)**
Descrizione	Questo metodo permette di effettuare il login nel sistema
Pre-condizione	/
Post-condizione	**context:** InfoUtentiService::login(String email, String password) **post:** InfoUtentiService.getUserLogged() != null
Nome metodo	**getUserLogged()**
Descrizione	Questo metodo restituisce le informazioni di autenticazione dell'utente attualmente loggato nel sistema
Pre-condizione	**context:** InfoUtentiService::getUserLogged() **pre:** InfoUtentiService.createUser(String email, String password)
Post-condizione	**context:** InfoUtentiService::getUserLogged() **post:** userLogged.getEmail() != null
Nome metodo	**logout()**
Descrizione	Questo metodo permette di effettuare il logout dal sistema
Pre-condizione	**context:** InfoUtentiService::logout() **pre:** getUserLogged() != null
Post-condizione	**/**
Nome metodo	**getAthletebyId(String id)**
Descrizione	Questo metodo permette di recuperare le informazioni dell'utente atleta attraverso il suo id univoco di registrazione
Pre-condizione	**context:** InfoUtentiService::getAthletebyId(String id) **pre:** saveAthlete(Atleta atleta, String id)

Post-condizione	**context:** InfoUtentiService::getAthletebyId(String id) **post:** id.equals(documentSnapshot.getId()) == true
Nome metodo	**editAthleteInfo(Atleta newAtleta String id)**
Descrizione	Questo metodo permette di modificare le informazioni relative alle caratteristiche all'atleta
Pre-condizione	**context:** InfoUtentiService::editAthleteInfo(Atleta newAtleta, String id) **pre:** saveAthlete(Atleta atleta, String id)
Post-condizione	**context:** InfoUtentiService::editAthleteInfo(Atleta newAtleta, String id) **post:** newAtleta.compareTo(oldAtleta) != 0
Nome metodo	**insertAthleteFeatures(Atleta atleta, String id)**
Descrizione	Questo metodo permette di inserire le caratteristiche relative all'atleta nel relativo documento con id
Pre-condizione	**context:** InfoUtentiService::insertAthleteFeatures(Atleta atleta, String id) **pre:** getUserLogged() != null
Post-condizione	**context:** InfoUtentiService::insertAthleteFeatures(Atleta atleta, String id) **post:** atleta.areFeaturesEmpty() == false
Nome metodo	**editAthleteFeatures(Atleta newAtleta, String id)**
Descrizione	Questo metodo permette di aggiornare le caratteristiche relative all'atleta nel relativo documento con id
Pre-condizione	**context:** InfoUtentiService::editAthelteFeatures(Atleta atleta, String id) **pre:** getUserLogged() != null
Post-condizione	**context:** InfoUtentiService::editAthleteFeatures(Atleta atleta, String id) **post:** newAtleta.compareTo(oldAtleta) != 0
Nome metodo	**deleteUser()**
Descrizione	Questo metodo permette di cancellare il record di autenticazione dell'utente

Pre-condizone	**context:** InfoUtentiService::deleteUser() **pre:** getUserLogged() != null
Post-condizione	**context:** InfoUtentiService::deleteUser() **post:** getUserLogged() == null
Nome metodo	**deleteAthelte()**
Descrizione	Questo metodo permette di cancellare le informazioni e le caratteristiche relative all'utente atleta
Pre-condizione	**context:** InfoUtentiService::deleteUser() **pre:** getAthletebyId != null
Post-condizione	**context:** InfoUtentiService::deleteUser() **post:** getAthletebyId == null

package gestioneScheda

Nome classe	**GestioneSchedaService**
Descrizione	Questa classe permette di gestire le operazioni relative al sottosistema GestioneScheda
Metodi	+saveScheda(Map<String, Object> scheda):Task<Void> +updateScheda(Map<String, Object> scheda, String schedaId):Task<Void> +getSchedaById(String schedaId):Task<DocumentSnapshot> +getAllUserSchede(String userId):Task<DocumentSnapshot> +getAllEsercizi():Task<QuerySnapshot> +getSchedaInUso(String userId):Task<QuerySnapshot>
Invariante di classe	/

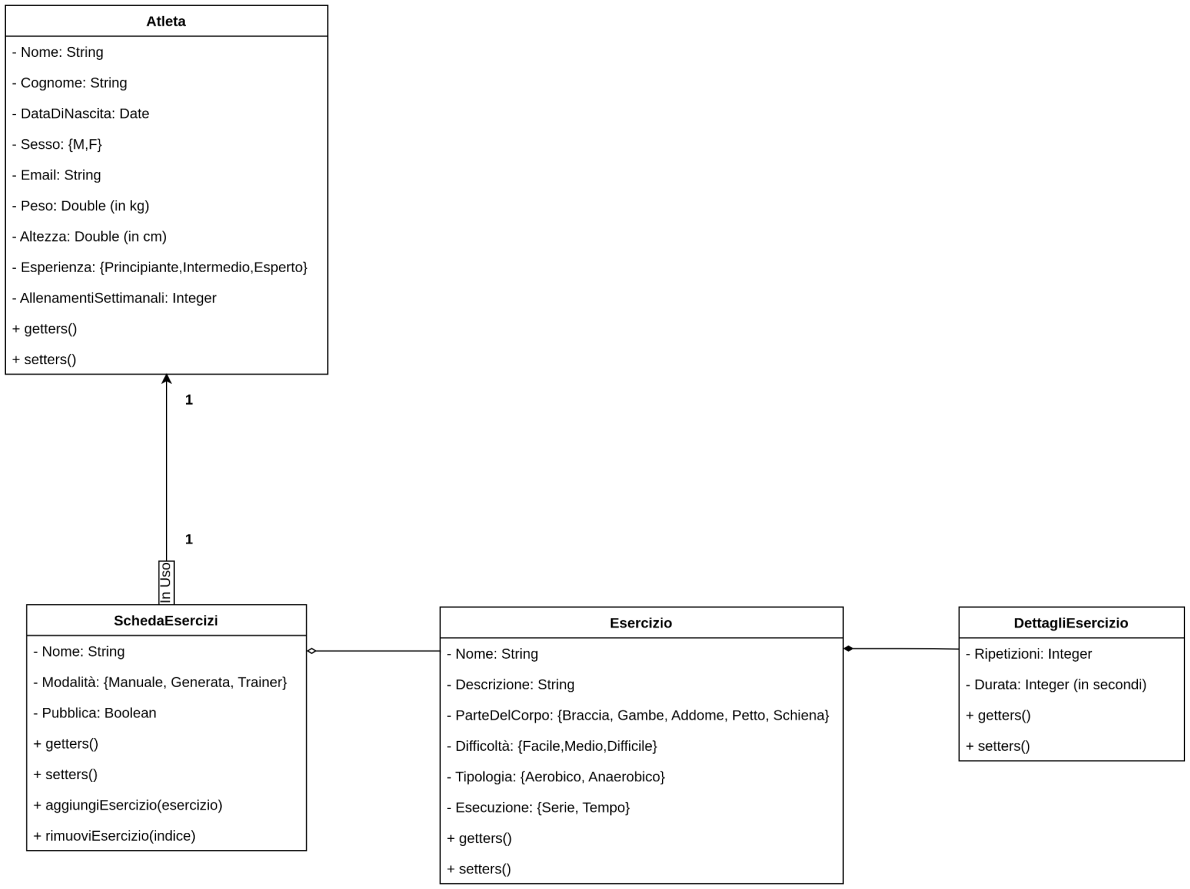
Nome metodo	**saveScheda(Map<String, Object> scheda)**
Descrizione	Questo metodo permette di salvare una scheda esercizi nel database
Pre-condizione	**context:** GestioneSchedaService::saveScheda(Map<String, Object> scheda) **pre:** 3 <= scheda.size() <= 10 AND (3 <= esercizio.getRipetizioni() <= 30 OR 20 <= esercizio.getDurata()

	<= 60)
Post-Condizione	**context:** GestioneSchedaService::saveScheda(Map<String, Object> scheda) **post:** getSchedaById(idScheda) != null
Nome metodo	**updateScheda(Map<String, Object> newScheda, String schedaId)**
Descrizione	Questo metodo permette di aggiornare una scheda esercizi precedentemente salvata
Pre-condizone	**context:** GestioneSchedaService::update(Map<String, Object> newScheda, String schedaId) **pre:** saveScheda(Map<String, Object> scheda)
Post-condizione	**context:** GestioneSchedaService::update(Map<String, Object> Newscheda, String schedaId) **post:** newScheda.compareTo(oldScheda) != 0
Nome metodo	**getSchedaById(String schedaId)**
Descrizione	Questo metodo permette di recuperare una scheda esercizi in base al suo id
Pre-condizione	**context:** GestioneSchedaService::getSchedaById(String schedaId) **pre:** saveScheda(Map<String, Object> scheda)
Post-condizione	**context:** GestioneSchedaService::getSchedaById(String schedaId) **post:** id.equals(documentSnapshot.getId()) == true
Nome metodo	**getAllUserScheda(String userId)**
Descrizione	Questo metodo permette di recuperare tutte le schede esercizi dell'utente
Pre-condizione	**context:** GestioneSchedaService::getAllUserScheda(String userId) **pre:** saveScheda(Map<String, Object> scheda)
Post-Condizione	**context:** GestioneSchedaService::getAllUserScheda(String userId) **post:** scheda.getAtleta().getId() == userId
Nome metodo	**getAllEsercizi();**

Descrizione	Questo metodo permette di recuperare tutti gli esercizi
Pre-condizione	**context:** GestioneSchedaService::getAllEsercizi() **pre:**
Post-Condizione	**context:** GestioneSchedaService::getAllEsercizi() **post:**
Nome metodo	**getSchedaInUso(String userId)**
Descrizione	Questo metodo permette di salvare una scheda esercizi nel database
Pre-condizione	**context:** GestioneSchedaService::getSchedaInUso(String userID) **pre:** SchedaLogic.setSchedaInUso(idNuovaSchedaInuso, String inUso, boolean b)
Post-Condizione	**context:** GestioneSchedaService::getSchedaInUso(String userID) **post:** scheda.getAtleta().getId() == userId

4. Class Diagram Ristrutturato

In questa sezione è presentato il class diagram ristrutturato. La versione mostrata prende in considerazione solamente le entità che saranno parte dell'implementazione iniziale del sistema.



Sono state effettuate le seguenti operazioni:

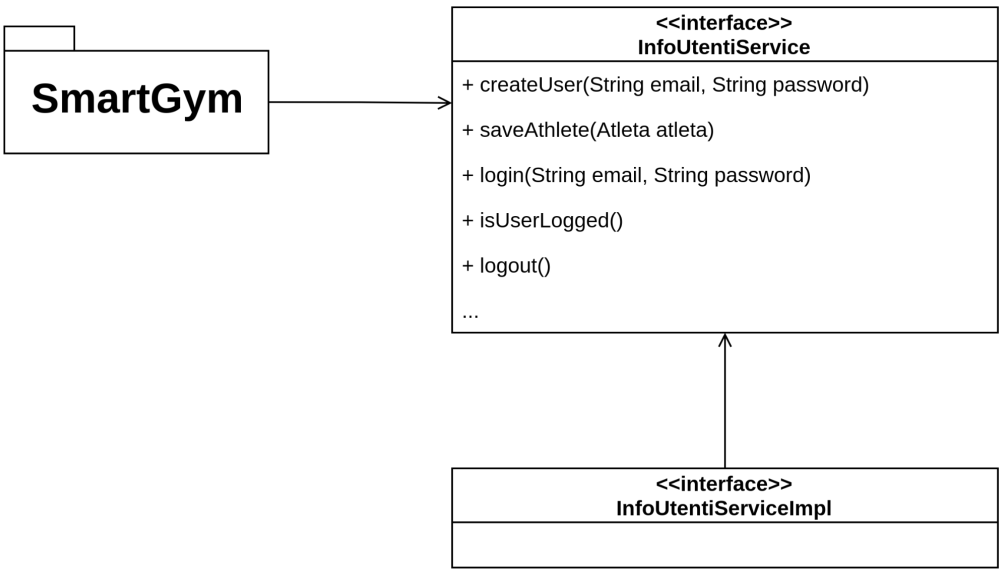
- Specificata la visibilità per ogni attributo di classe
- Specificate la signature e la visibilità per le operazioni di ogni classe

5. Design Pattern

In questa sezione sono illustrati tutti i design pattern individuati che saranno utilizzati all'interno del nostro sistema. Viene inoltre illustrato lo scenario dove il design pattern è utilizzato, oltre che una breve descrizione del pattern e un diagramma delle classi che ne fanno uso

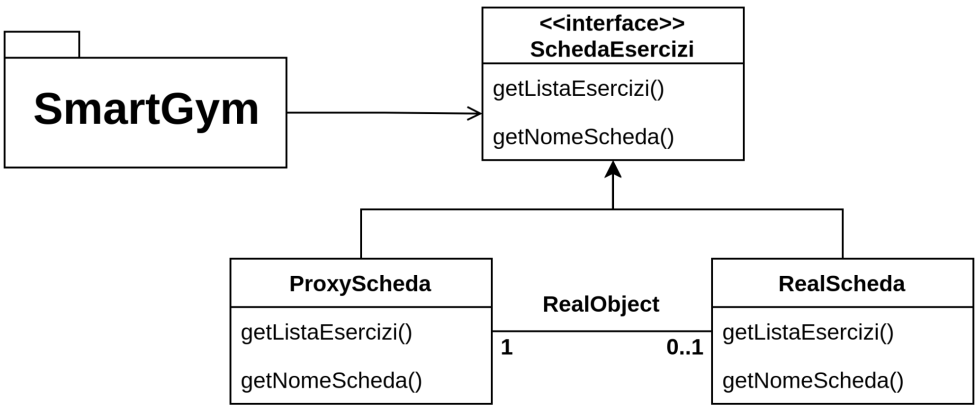
Facade

Il design pattern Facade è un particolare design pattern strutturale che ci permette di esporre un'interfaccia unificata del sottosistema per concedere agli altri sottosistemi un numero limitato di operazioni. Questa soluzione è solitamente usata nei sistemi con architetture chiusa. Nel caso del nostro sistema, abbiamo di utilizzare il Facade per ogni package di sottosistema individuato.



Proxy

Il design pattern Proxy è un particolare design pattern strutturale che permette di ritardare la creazione e l'inizializzazione di oggetti all'interno del sistema. In questo modo la creazione e l'inizializzazione saranno effettuate al tempo necessario. Nel nostro caso, facciamo uso del Proxy nel seguente scenario: L'utente accede alla sezione relativa alle schede esercizi dove è presente il listato delle schede da lui salvate. In questo punto del sistema, non è necessario caricare l'intero oggetto ma basterà caricare un oggetto proxy che conterrà il nome della scheda e una lista vuota degli esercizi. Quando l'utente seleziona una voce del listato, la relativa scheda sarà del tutto caricata attraverso l'oggetto reale che prevederà una lista inizializzata di esercizi insieme a tutti gli altri parametri opportunamente inizializzati. Grazie a questo design pattern ritardiamo il caricamento degli esercizi nella lista, che sarà effettuato solamente se necessario.



6. Glossario

Nella presente sezione sono raccolte le sigle o i termini del documento che necessitano di una definizione.

Sigla/Termine	Definizione
Activity	La componente principale di android. Rappresenta una vista grafica di una sezione del sistema, realizzata attraverso XML e con una logica di presentazione gestita da una classe Java
Fragment	Una sottoparte di Activity. Utilizzata per fissare alcune componenti della UI su più viste di varie sezioni del sistema
Gradle	Sistema di build utilizzato dall'ambiente di sviluppo integrato per Android, Android Studio