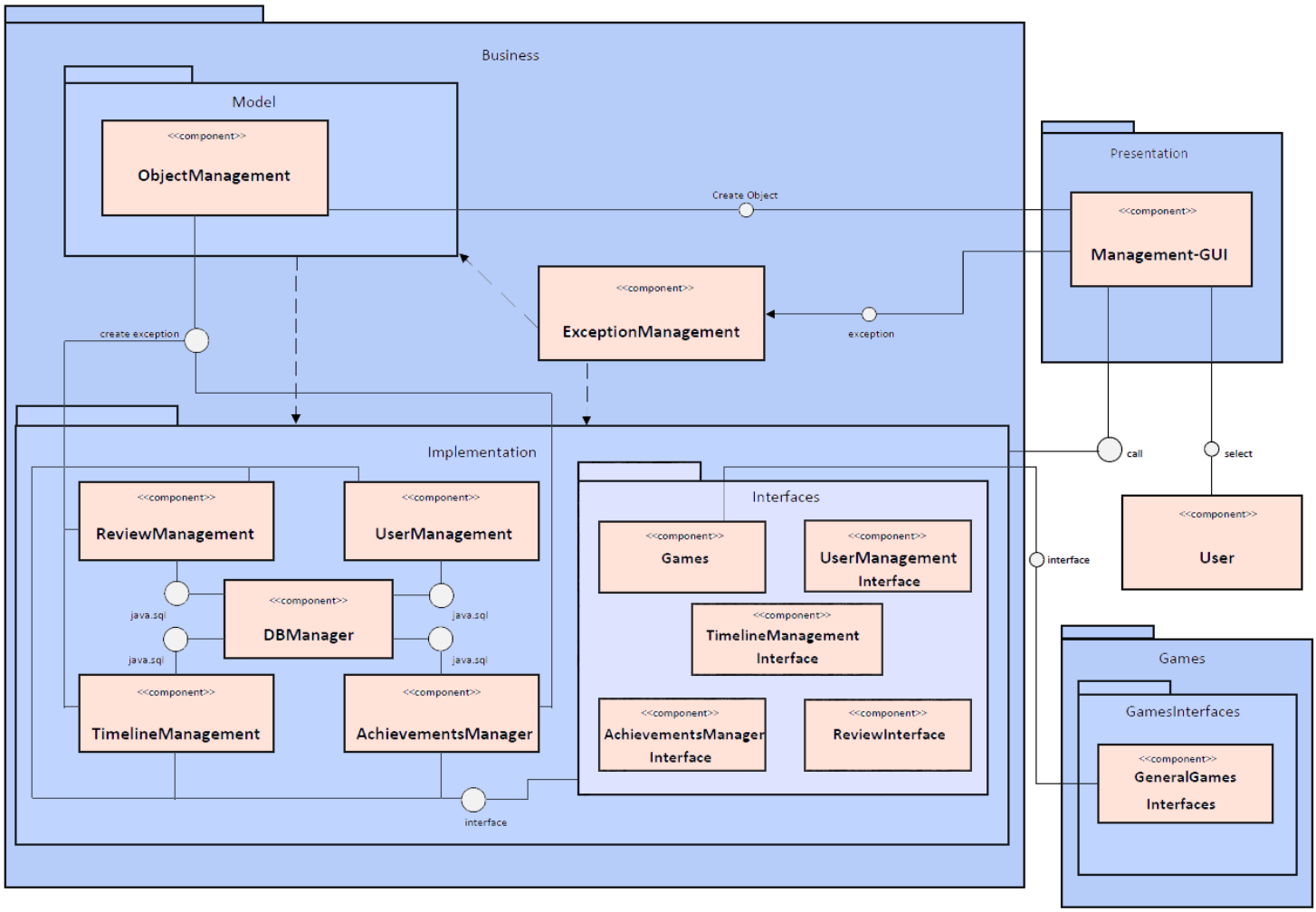
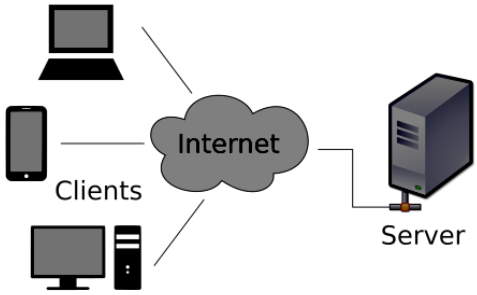
DESCRIZIONE DELL'ARCHITETTURA SCELTA
(Client-Server)



Chi sono i Client e cosa fanno: Il software client "Presentation" in questo caso si limita ad interfacciarsi con il server. I client accedono ai servizi o alle risorse di un'altra componente, detta server, attraverso un'interfaccia grafica (GUI) con la quale effettuano le varie operazioni a disposizione.

Chi sono i Server e cosa fanno: Il software server "Business", oltre alla gestione logica del sistema, deve implementare tutte le tecniche di gestione degli accessi, allocazione e rilascio delle risorse, condivisione e sicurezza dei dati o delle risorse "ReviewManagement" "UserManagement". Nel nostro caso attribuiamo a esso la gestione del Database. Inoltre si occupa anche della gestione degli eventi scaturiti dall'utente grazie alla componente Listener-Eventi.

Architettura CLIENT-SERVER



L'architettura *client-server* è formata da due tipi di moduli: il client e il server che, in genere, sono eseguiti su macchine diverse collegate in rete.

Il Server svolge le operazioni necessarie per la realizzazione di un servizio, ad esempio la gestione di un database con i relativi aggiornamenti dei dati e la loro integrità;

Il Client può effettuare diverse operazioni richiedendo quindi un terminale con capacità elaborative (come ad esempio un PC o uno Smartphone). Tipicamente il client gestisce la porzione di interfaccia utente dell'applicazione e provvede ad inviare al server e richieste formulate dall'utente. In pratica il client è quella parte dell'applicazione che l'utente vede e con la quale interagisce. La richiesta formulata dal client viene formattata secondo un determinato protocollo (ad es: *Hyper-Text Transfer Protocol [http]* sul quale si basa il *World Wide Web [www]*) in modo che il server possa capirla e, viceversa, formatta la risposta proveniente dal server in modo che sia a sua volta comprensibile all'utente.

Questa architettura quindi ha diversi risvolti positivi, i più importanti:

- ***Risorse Centralizzate:*** è solo il server a poter gestire le risorse comuni a tutti gli utenti evitando così dati ripetuti o che si contraddicono.
- ***Sicurezza dei Dati:*** essendo i dati visibili a tutti generalmente in sola lettura, ossia non da tutti modificabili, c'è una maggiore sicurezza della permanenza e della correttezza dei dati.

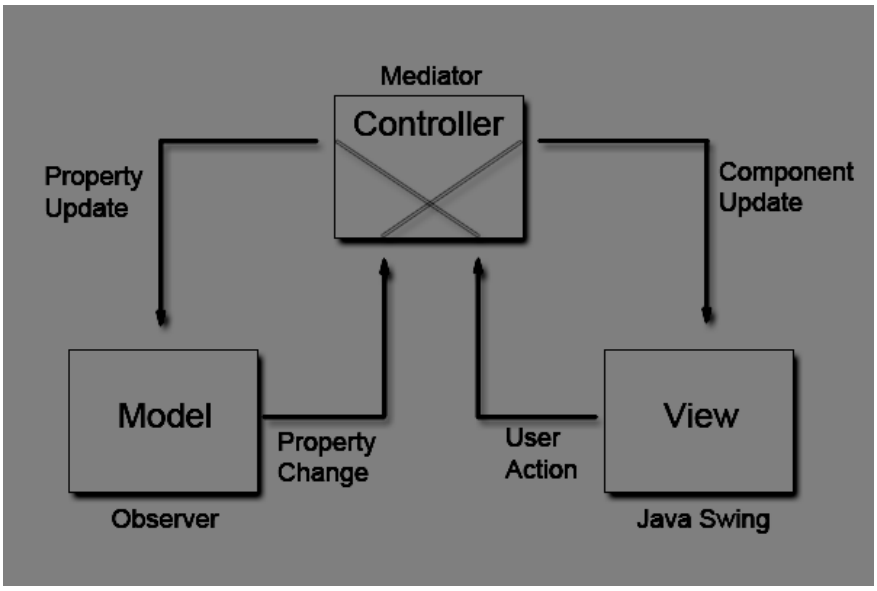
SOFTWARE ARCHITECTURE MODEL

Il modello architetturale scelto per la realizzazione del diagramma è il MVC (Model-View-Controller). Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

Model: rappresentazione del modello di dominio, e quindi i dati utili per accedere all'applicazione.

View: Interfaccia utente, visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti.

Controller: Controllo dell'interazione uomo-macchina, riceve i comandi dell'utente (in genere attraverso la view) e li attua modificando lo stato degli altri due componenti.

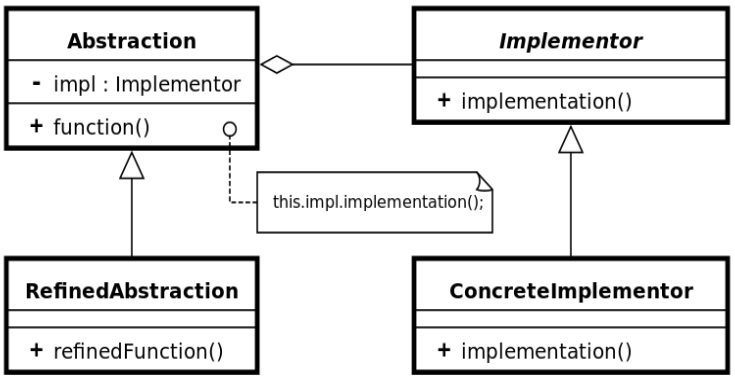


Nella nostra piattaforma questi ruoli sono ben distinti grazie all'utilizzo dei package.
Nell'ordine:

- Il package ***Business*** si occupa della parte ***Model***, implementando quindi la logica di business che riguarda l'accesso, la modifica e il recupero dei dati dell'applicazione.
- Il package ***Presentation*** svolge il ruolo di ***View***, quindi è delegato alla gestione dell'interfaccia utente.
- Il package ***Listener*** è il nostro ***Controller***, il quale riceve i comandi dell'utente tramite le View (Presentation) ed effettua le operazioni richieste. Insieme al package ***Model*** si occupa quindi di implementare tutta la logica di business del sistema.

Altri pattern utilizzati:

Bridge Pattern



Il bridge pattern è un design pattern (modello di progettazione) della programmazione ad oggetti che permette di separare l'interfaccia di una classe (che cosa si può fare con la classe) dalla sua implementazione (come lo fa). In tal modo si può usare l'ereditarietà per fare evolvere l'interfaccia o l'implementazione in modo separato.

Abbiamo fatto uso del bridge durante l'implementazione di un gioco generico. L'interfaccia games, contiene i metodi *getXP() e setXP()*. Questa, verrà poi implementata da un gioco specifico (es: snake). In questo modo, abbiamo utilizzato un modello 'astratto' valido per qualsiasi implementazione.

DAO (DATA ACCESS OBJECT)

Il ***DAO*** (*Data Access Object*) è un pattern architetturale per la gestione della persistenza: si tratta fondamentalmente di una classe con relativi metodi che rappresenta un'entità tabellare di un RDBMS, usata principalmente per stratificare e isolare l'accesso ad una tabella tramite query (poste all'interno dei metodi della classe) ovvero al *data layer* da parte della business logic creando un maggiore livello di astrazione ed una più facile manutenibilità. I metodi del DAO con le rispettive query dentro verranno così richiamati dalle classi della business logic.

SINGLETON

Abbiamo deciso di utilizzare il design pattern creazionale Singleton per la gestione della connessione al database nella nostra applicazione. Esso è uno dei pattern fondamentali della programmazione ad oggetti. Lo scopo di questo pattern è di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire unpunto di accesso globale a tale istanza.

JAVA SWING

Per lavorare l'interfaccia grafica useremo SWING. Il package Swing, è composto da componenti realizzati completamente in Java, ricorrendo unicamente alle primitive di disegno più semplici, tipo "traccia una linea" o "disegna un cerchio", accessibili attraverso i metodi dell'oggetto Graphics, un oggetto AWT utilizzato dai componenti Swing per interfacciarsi con la piattaforma ospite. Le primitive di disegno sono le stesse su tutti i sistemi grafici, e il loro utilizzo non presenta sorprese: il codice java che disegna un pulsante Swing sullo schermo di un PC produrrà lo stesso identico risultato su un Mac o su un sistema Linux. Questa architettura risolve alla radice i problemi di uniformità visuale, visto che la stessa identica libreria viene ora utilizzata, senza alcuna modifica, su qualunque JVM. Liberi dal vincolo del "minimo comune denominatore", i progettisti di Swing hanno scelto di percorrere la via opposta, creando un package ricco di componenti e funzionalità spesso non presenti nella piattaforma ospite.

MySQL

MySQL è un RDBMS **open source** e **libero**, e rappresenta una delle tecnologie più note e diffuse nel mondo dell'IT. MySQL nacque nel 1996 per opera dell'azienda svedese Tcx, basato su un DBMS relazionale preesistente, chiamato mSQL. Il progetto venne distribuito in modalità open source per favorirne la crescita.

Dal 1996 ad oggi, MySQL si è affermato molto velocemente prestando le sue capacità a moltissimi software e siti Internet. I motivi di tale successo risiedono nella sua capacità di mantenere fede agli impegni presi sin dall'inizio:

- alta efficienza nonostante le moli di dati affidate;

- integrazione di tutte le funzionalità che offrono i migliori DBMS: *indici*, *trigger* e *stored procedure* ne sono un esempio, e saranno approfonditi nel corso della guida;

- altissima capacità di integrazione con i principali linguaggi di programmazione, ambienti di sviluppo e suite di programmi da ufficio.