UNIVERSITÀ DEGLI STUDI DI SALERNO

Ingegneria del Software

Docente:

Andrea De Lucia

Studenti:

Raffaele Vitiello

Alessio Cuccurullo

Francesco Giuliano

OBJECT DESIGN DOCUMENT

REVISION HISTORY

Data	Versione	Descrizione	Autore
14/01/19	1.0	Completamento stesura documento	Raffaele Vitiello Alessio Cuccurullo
18/01/19	1.1	Prima revisione del documento con correzioni	Raffaele Vitiello Alessio Cuccurullo
24/01/19	1.2	Revisione finale con correzioni	Raffaele Vitiello

1. INTRODUZIONE

1.1. Object design trade-offs

Dopo aver realizzato i documenti RAD ed SDD, dove sono stati omessi gli aspetti che riguardano l'implementazione del sistema, si va a stilare il documento di Object Design che ha come obbiettivo la creazione di un modello che interagisca con tutte le componenti che sono state analizzate nei documenti citati. In questo documento vengono definite le interfacce delle classi, le operazioni, i tipi, gli argomenti e le signature dei sottosistemi definiti nel SDD. Si specificano quinidi i trade-off e le linee guida.

Funzionalità vs Tempo

Non saranno implementate le funzionalità secondarie per presentare il prima possibile il progetto, come ad esempio registrazione, modifica informazioni personali.

Per questo motivo porremo maggiore attenzione alle funzionalità chiave di Sine Charta.

Leggibilità del codice vs Velocità di sviluppo

Il codice dovrà essere comprensibile per semplificare manutenzione e modifiche future. Utilizzeremo quanto più possibile dei commenti per indicare lo scopo delle classi e metodi, rallentando lo sviluppo del sistema.

Scalabilità vs Manutenibilità

Sine Requie è un gioco in continua evoluzione. Il suo regolamento, infatti, è soggetto a piccoli cambiamenti nel corso degli anni. Sine Charta deve essere progettato in maniera tale da poter tenere il passo con lo sviluppo del gioco reale.

Usabilità vs Sicurezza

Sine Charta non è e non vuole essere un software per memorizzare dati sensibili. Pertanto ci occuperemo soprattutto di migliorare l'esperienza di gioco dell'utente finale attraverso un sistema usabile, rispetto all'implementazione di sistemi di sicurezza dei dati ad hoc.

1.2. Linee Guida per la Documentazione delle interfacce

Per lo scrittura del codice si seguiranno le seguenti linee guida:

Naming Convention

Utilizzeremo le seguenti convenzioni per i nomi:

- Descrittivi
- Pronunciabili
- Di uso comune
- Di lunghezza media
- Non abbreviati
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Varibili

- I nomi delle variabili devono sempre iniziare con una lettera minuscola e le parole successive con una maiuscola. Le dichiarazioni delle variabili devono essere fatte all'inizio del blocco di codice, le variabili dello stesso tipo sono dichiarate sulla stessa riga.
- In alcuni casi viene utilizzato il carattere underscore "_", per le variabili costanti oppure quando vengono utilizzate delle proprietà statiche.

Metodi

- I nomi dei metodi devono iniziare con una lettera minuscola, e le parole successive con lettera maiuscola. Solitamente il nome di un metodo consiste in un verbo che identifica l'azione da svolgere, seguite dal nome di un oggetto.
- I nomi dei metodi per l'accesso alle variabili devono essere del tipo "getNomeVaribile()", mentre i metodi per la modifica delle variabili devono essere del tipo "setNomeVariabile()".
- Se viene utilizzata una variabile all'interno di un metodo, questa deve essere sempre dichiarata prima del suo utilizzo e deve essere utilizzata per un solo scopo, per facilitare la leggibilità del codice.
- Ad ogni metodo viene aggiunta una descrizione per specificare la loro funzione, come i valori riguardanti gli argomenti, i valori di ritorno e le eccezioni. Questa descrizione deve essere posizionata prima della dichiarazione del metodo.

Classi e pagine

- I nomi delle classi devono iniziare con una lettera maiuscola e anche le parole successive, mentre i nomi delle pagine possono iniziare sia con minuscole che con maiuscole.

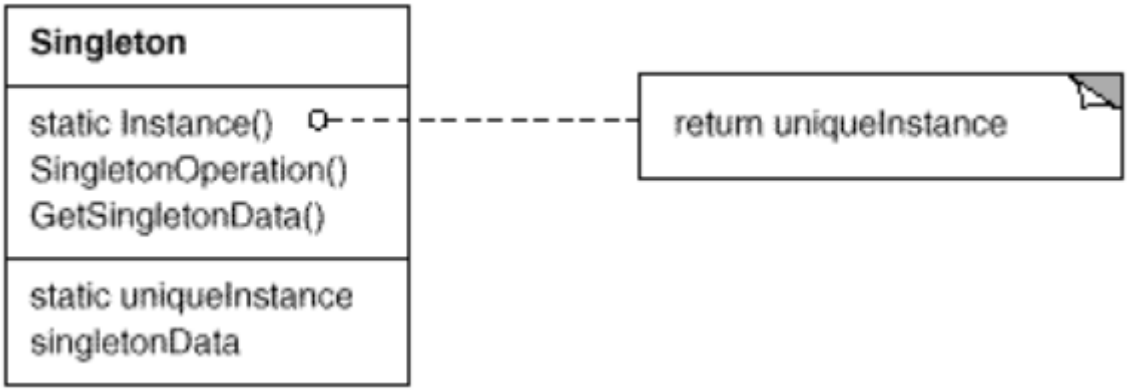
1.3. Riferimenti

- Documento RAD Sine Charta.
- Documento SDD Sine Charta.
- B. Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java, Prentice Hall, 3rd edition, 2009 .
- Design Patterns: Elements of Reusable Object-Oriented Software, Gang of four, 1994.

2. DESIGN PATTERN

2.1. Design pattern globali

Sine Charta utilizza il Singleton pattern per consentire di istanziare un Manager una e una sola volta.



3. PACKAGE COMPONENTS

Package bean

UtenteRegistratoBean	È l'utente registrato al sistema
PersonaggioBean	È l'alter-ego del giocatore
OggettoBean	È parte dell'equipaggiamento del personaggio
AbilitaBean	È una particolare abilità del personaggio
StoriaBean	È il contesto immaginario nel quale agiscono i personaggi
SessioneBean	È l'insieme di incontri ed eventi descritti dal moderator
KeywordBean	È un evento, una persona o simili che ricorre in una storia
NemicoBean	È un nemico presente nel manuale base di Sine Requie

Package Autenticazione

Login	È il modulo che gestisce l'accesso e controlla lo stato della sessione
Logout	È il modulo che permette ad un utente di effettuare il logout
Visualizza_profilo	È il modulo che consente di visualizzare le informazioni del profilo utente
Modifica_dati_personali	È il modulo che si occupa della modifica dei dati personali
Recupera_password	È il modulo che permette di recuperare la propria password
Recupera_username	È il modulo che permette di recuperare il proprio username

Package Registrazione

Registrazione	È il modulo che si occupa della registrazione a Sine Charta

Package Storia

Editor_storia	È il modulo che consente ad un moderatore di scrivere una nuova storia o di modificarne una già esistente
Invia_inviti	È il modulo che si occupa di invitare dei giocatori alla propria storia
Accetta_inviti	È il modulo che permette ad un giocatore di accettare un invito ad una storia
Crea_pg	È il modulo che consente ad un giocatore di creare il proprio personaggio
Gioca	È il modulo che permette ad un giocatore di iniziare a giocare a Sine Requie

Package Sessione

Editor_sessione	È il modulo che consente ad un moderatore di modificare una sessione legata ad una storia già esistente ed aggiungere keyword
Gestione_sessione	È il modulo che permette ad un moderatore di avviare una sessione, caricandone il contenuto, keyword, lo scontro e il suo ordine di chiamata
Gestione_mazzo	È il modulo che consente di estrarre una carta da uno dei due mazzi, oppure di mischiarlo
Gestione_pg	È il modulo che permette ad un giocatore di visualizzare la scheda, aggiungere ferite, curare ferite e gestire l'equipaggiamento del proprio personaggio
Gestione_ferite	È il modulo che permette di aggiungere o curare le ferite a pg e npc

Package Manager

UsersManager	**È la classe che permette di instanziare oggetti di tipo utenteRegistrato interfacciandosi con il database**
StoryManager	È la classe che permette di instanziare oggetti di tipo Storia interfacciandosi con il database
PersonaggioManager	È la classe che permette di instanziare oggetti di tipo Personaggio interfacciandosi con il database
NPCManager	È la classe che permette di instanziare oggetti di tipo Nemico interfacciandosi con il database
EquipManager	È la classe che permette di instanziare oggetti di tipo Equip interfacciandosi con il database
AbilitaManager	È la classe che permette di instanziare oggetti di tipo Abilita interfacciandosi con il database
KeywordManager	È la classe che permette di instanziare oggetti di tipo Keyword interfacciandosi con il database
SessioneManager	È la classe che permette di instanziare oggetti di tipo Sessione interfacciandosi con il database

4. CLASS INTERFACES

Nome classe	**UsersManager**
Descrizione	Manager che gestisce gli utenti.
Pre-condizioni	**Context:** UsersManager:: User doretrieveByKey(string user) **pre:** user != null **Context:** UsersManager:: Set<Personaggio> aggiungiListaPGUser(User user) **Pre:** user != null **Context** UsersManager:: Set<Storia> aggiungiStorieUser(User user) **pre:** user != null **Context** UsersManager:: Collection<User> doRetrieveAll(String order) **Pre:** order == "nomeTabella" **Context** UsersManager:: void doSave(User user) **Pre:** user != null **Context** UsersManager:: boolean eliminaUtente(String username) **Pre:** username != null **Context** UsersManager :: boolean checkUser(String username) **Pre:** username != null
Post-Condizioni	
Invarianti	

Nome Classe	**StoryManager**
Descrizione	Manager che gestisce le storie.
Pre-condizioni	**Context** StoryManager:: Storia getStoria(int idStoria, String username) **Pre:** **Context** StoryManager:: Collection<Storia> listaStorie(String username) **Context** StoryManager:: void aggiungiStoria(Storia storia , String username) **Pre:** storia != null && username != null **Context** StoryManager:: boolean eliminaStoria(int idStoria, String username) **Pre:** idStoria != null && username != null **Context** StoryManager:: int selectLastId() **Context** StoryManager:: Collection<Storia> getStoriaByFlag(User user, int flag) **Pre:** user != null && flag != null **Context** StoryManager:: Personaggio getPersonaggioForStory(User utente, int idStory) **Pre:** utente != null && idStory != null **Context** StoryManager:: Storia getSimpleStory(int idStory) **Context** StoryManager:: Set<SessioneDiGioco> aggiungiSessioniById(int idStory) **Pre:** idStory != null **Context** StoryManager:: Set<SessioneDiGioco> aggiungiSessioniAllaStoria(Storia storia, User utenteMod) **Pre:** storia != null && utenteMod != null **Context** StoryManager:: Storia getStoriaDelPG(Personaggio pg) **Context** StoryManager:: void aggiungiStoria(Storia storia) **Pre:** storia != null **Context** StoryManager:: void aggiungiATable(User utente, int flag) **Pre:** utente != null && flag != null **Context** StoryManager:: boolean eliminaStoria(int idStoria) **Pre:** idStoria != null **Context** StoryManager:: boolean eliminaRiferimentoHaTable(String username, int idStory) **Pre:** username != null && idStory != null
Post-Condizioni	
Invarianti	

Nome Classe	**SessioneManager**
Descrizione	Manager che gestisce le sessioni delle storie.
Pre-condizioni	**Context** SessioneManager:: SessioneDiGioco recuperoSessioneStoria(Storia storia, User utente, int numSessione) **pre:** storia != null && utente != null && numSessione != null **Context** SessioneManager:: Collection<SessioneDiGioco> recuperoTutteLeSessioni(Storia storia, User utente) **pre:** storia != null && utente != null **Context** SessioneManager:: Storia getAnStory(int idStoria) **pre:** idStoria != null **Context** SessioneManager:: Set<Keyword> aggiungiListaKeyword(SessioneDiGioco sessione) **pre:** sessione != null **Context** SessioneManager:: void salvareSessioni(SessioneDiGioco sessioneDiGioco) **pre:** sessioneDiGioco != null **Context** SessioneManager:: void aggiornareSessioni(SessioneDiGioco sessioneDiGioco, String nuovoContenuto) **pre:** sessioneDiGioco != null && nuovoContenuto != null **Context** SessioneManager:: boolean eliminaSingolaSessione(int numero, int IdStory, String username) **pre:** numero != null && idStory != null && usernema != null **Context** SessioneManager:: boolean eliminaSessioni(int IdStory, String username) **pre:** idStory != null && username != null **Context** SessioneManager:: Collection<SessioneDiGioco> recuperoSessioni(int idStory) **pre:** idStory != null **Context** SessioneManager:: int getNumeroSessioniStoria(Storia storia) **pre:** storia != null
Post-Condizioni	
Invarianti	

Nome Classe	**SessioneManager**
Descrizione	Manager che gestisce i personaggi.
Pre-condizioni	**Context** PersonaggioManager:: Personaggio getSimplePGByStory(User utente, int idStory) **pre:** utente != null && idStory != null **Context** PersonaggioManager:: Set<Abilita> aggiungiListaAbilitaPG(Personaggio pg) **pre:** pg != null **Context** PersonaggioManager:: Storia setStoriaPersonaggio(Personaggio pg) **pre:** pg != null **Context** PersonaggioManager:: Set<Oggetto> aggiungiListaOggettiPG(Personaggio pg) **pre:** pg != null **Context** PersonaggioManager:: int getStoriaPersonaggioById(Personaggio pg) **pre:** pg != null **Context** PersonaggioManager:: Collection<Personaggio> getAllPgByStory(Storia storia) **pre:** **Context** PersonaggioManager:: User getUserByPG(Personaggio pg) **pre:** pg != null **Context** PersonaggioManager:: Collection<Personaggio> listaPG(User user) **pre:** **Context** PersonaggioManager:: void creaPersonaggio(Personaggio pg, int idStoria) **pre:** pg != null && idStoria != null **Context** PersonaggioManager:: void updateFeritePg(int idPG, String areaFerita, int danno) **pre:** idPG != null && areaFerita != null && danno != null **Context** PersonaggioManager:: boolean eliminaPG(Personaggio pg) **pre:** pg != null
Post-Condizioni	
Invarianti	

Nome Classe	**EquipManager**
Descrizione	Manager che gestisce l'equipaggiamento del personaggio.
Pre-condizioni	**Context** EquipManager:: Oggetto getOggettoPersonaggioById(Personaggio pg, int idStory) **pre :** pg != null && idStory != null **Context** EquipManager:: Collection<Oggetto> getListaOggettiPG(Personaggio pg) **pre :** **Context** EquipManager:: void inserisciOggetto(Oggetto oggetto, Personaggio pg) **pre :** oggetto != null && pg != null **Context** EquipManager:: boolean rimuoviOggetto(Oggetto oggetto, Personaggio pg) **pre :** oggetto != null && pg != null **Context** EquipManager:: void updateQuantitaOggetto(Oggetto oggetto, Personaggio pg, int newQuantity) **pre :** oggetto != null && pg != null != newQuantity != null **pre:** newQuantity > 0
Post-Condizioni	
Invarianti	

Nome Classe	**AbilitaManager**
Descrizione	Manager che gestisce le abilità del personaggio.
Pre-condizioni	**Context** AbilitaManager:: Abilita getAbilitaByName(Personaggio pg, String nome) **pre :** pg != null && nome != null **Context** AbilitaManager:: Collection<Abilita> getListaAbilitaByPG(Personaggio pg) **pre :** **Context** AbilitaManager:: void aggiungiAbilita(Abilita ability, Personaggio pg) **pre :** ability != null && pg != null **Context** AbilitaManager:: boolean eliminaAbilita(Personaggio pg) **pre :** pg != null
Post-Condizioni	
Invarianti	

Nome Classe	**KeywordManager**
Descrizione	Manager che gestisce le keyword delle sessioni di gioco.
Pre-condizioni	**Context** KeywordManager:: Keyword prendereKeyword(SessioneDiGioco sessione, String chiave) **Pre:** sessione != null && chiave != null **Context** KeywordManager:: Collection<Keyword> listaKeyword(SessioneDiGioco session) **Pre:** **Context** KeywordManager:: boolean eliminaKeyword(int idKeyWord, int numero) **Pre:** idKeyWord != null && numero != null **Context** KeywordManager:: aggiungiKeyword(Keyword nuovaKey, SessioneDiGioco sessione) **Pre:** nuovaKeyWord != null && sessione != null
Post-Condizioni	
Invarianti	