|  |
| --- |
| http://www.fondazioneantoniodellamonica.com/images/logo_unisa.png  **Ingegneria del Software** |

****



**Docente:**

*Andrea De Lucia*

**Studenti:**

Raffaele Vitiello

Alessio Cuccurullo

Francesco Giuliano

**OBJECT DESIGN DOCUMENT**

[1. INTRODUZIONE 4](#_Toc534300428)

[**1.1** **Object design trade-offs** 4](#_Toc534300429)

[**1.2** **Linee Guida per la Documentazione delle interfacce** 5](#_Toc534300430)

[**1.3** **Definizioni, acronimi e abbreviazioni** 6](#_Toc534300431)

[**1.4** **Riferimenti** 6](#_Toc534300432)

[2 DESIGN PATTERN 7](#_Toc534300433)

[**2.1** **Design pattern globali** 7](#_Toc534300434)

[3 PACKAGE COMPONENTS 8](#_Toc534300435)

[4 CLASS INTERFACES 9](#_Toc534300436)

[5 GLOSSARIO 10](#_Toc534300437)

1. **INTRODUZIONE**
   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, risspetto all'implementazione di sistemi di sicurezza dei dati ad hoc.

* 1. **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. **Definizioni, acronimi e abbreviazioni**

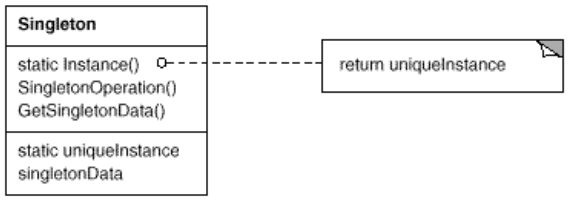
**Acronimi:**

**Abbreviazioni:**

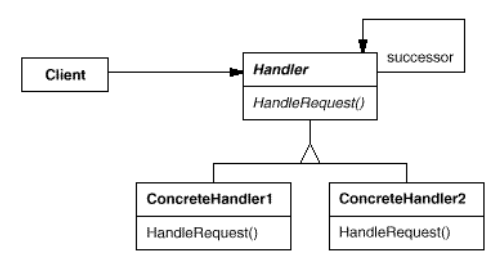
* 1. **Riferimenti**
* Documento RAD Sine Charta.
* Documento SDD Sine Charta.
* B. Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java, Prentice Hall, 3rd edition, 2009 .
* Design Patterns: Elements of Reusable Object-Oriented Software, Gang of four, 1994.

1. **DESIGN PATTERN**
   1. **Design pattern globali**

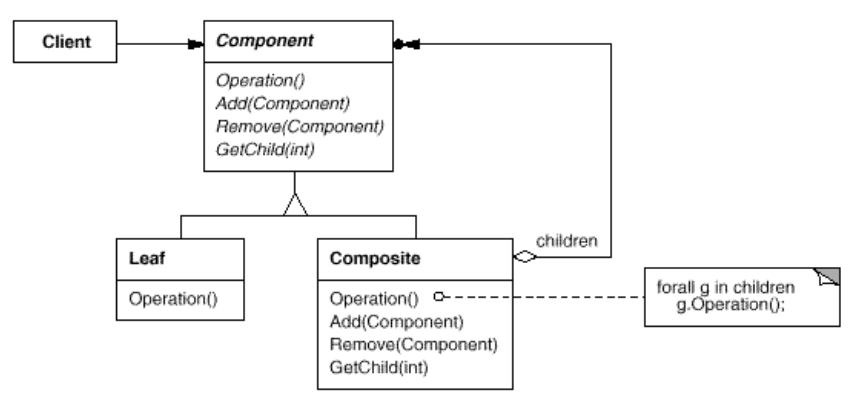
Sine Charta utilizza il Singleton pattern per consentire di istanziare un Manager una e una sola volta.



Sine Charta utilizza il Chain of Responsibility pattern durante la creazione del personaggio. Riducendo l'accoppiamento tra le classi che si occupano dell'oggetto Personaggio, sarà possibile facilitare le operazioni di manutenzione e scrittura del codice.



Sine Charta utilizza il Composite Pattern nella rappresentazione dei tarocchi e delle carte da poker. Di fatto, i due mazzi di carte hanno caratteristiche differenti, ma vengono utilizzate allo stesso modo. Con il Composite Pattern potremo separare le due astrazioni delle carte, mantenendo un comportamento di default unico.



1. **PACKAGE COMPONENTS**

**Package bean**

|  |  |
| --- | --- |
| **UtenteRegistratoBean** | È l'utente registrato al sistema |
| **MazzoBean** | È un insieme di carte dello stesso tipo |
| **CartaPokerBean** | È una singola carta da gioco che compone il mazzo da poker |
| **TaroccoBean** | È un singolo tarocco che compone il mazzo dei tarocchi |
| **PersonaggioBean** | È l'alter-ego del giocatore |
| **OggettoBean** | È parte dell'equipaggiamento del personaggio |
| **ArmaBean** | È un particolare oggetto 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 |
| **MazzoManager** | È la classe che permette di instanziare oggetti di tipo Mazzo 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 |
| **CartaManager** | È la classe che permette di instanziare oggetti di tipo Carta 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 |

1. **Class Interfaces**

Package: Gestione utente

Invarianti:

Per la registrazione di un nuovo utente, è necessario che lo username e l'indirizzo e-mail non siano già presenti nel database.

@inv getUtenteById(username)!=null || getUtenteByMail(email)!=null

Il login può essere effettuato solo se l'utente ha inserito la password corrispondente allo username immesso, ammesso che esista.

@inv !getUtenteByLogin(username, password)

Il logout può essere effettuato solo se l'utente è loggato.

@inv !utente

Il recupero delle credenziali può essere effettuato solo se esiste un account associato all'indirizzo email inserito.

@inv !getUtenteByMail(email)

La modifica delle informazioni personali può essere effettuata se il parametro inserito è diverso da quello già presente.

@inv newParam==param

Nome: registraUtente(username, password, email, nome, cognome)

@pre !getUtenteById(username)

@pre !getUtenteByMail(email)

@post getUtenteById(username)

Nome: login(username, password)

@pre getUtenteByLogin(username, password)!=null

@post utente= getUtenteById(username)

Nome: logout()

@pre utente

@post !utente

Nome: recuperaCredenziali(email)

@pre uTemp= getUtenteByMail(email)

@post uTemp.user

@post uTemp.password

Nome: modificaInfo(param, newParam)

@pre utente.param

@post utente.param= newParam

Package: gestioneStoria

Invarianti:

Per la creazione di una storia, è necessario che siano stati inseriti un nome, una descrizione e un'ambientazione. Inoltre il nome non può essere un duplicato.

@inv getStoriabyNome(nome) != null

@inv not exists getStoriaByNome(nome)

@inv getStoriabyDescrizione(descrizione) != null

@inv ambientazione != null

Per la creazione di una keyword, bisogna che la chiave sia abbastanza lunga e che sia inserita una descrizione.

@inv getKeywordById(chiave) != null

@inv chiave.length >2

Per l'invito ad una storia, bisogna che la storia e lo username esistano e che l'utente non sia già stato invitato.

@inv getUtenteById(utenteGiocatore.username)!= null

@inv getStoriaByNome(storia.titolo) != null

@inv utente.isInvited(storia.titolo)== true

Nome: creaStoria(nome, descrizione, ambientazione)

@pre !getStoria(nome)

@post getStoriaByNome(nome)

Nome: creaKeyword(chiave, descrizione)

@pre !getKeyword(chiave)

@post getKeyword(chiave)

Nome: invita(storia.titolo, utenteGiocatore.username)

@pre !getStoriaByNome(storia.titolo)

@pre !getUtenteById(utenteGiocatore.username)

@pre utente.isInvited(storia.titolo)== false

@post utente.isInvited(storia.titolo)== true

Nome: accettaInvito(storia.titolo, utenteGiocatore.username)

@pre !getStoriaByNome(storia.titolo)

@pre !getUtenteById(utenteGiocatore.username)

@pre utente.isInvited(storia.titolo)== true

@post getPgInStoria(storia.titolo, utenteGiocatore.username) == pgVuoto

Nome: creaPG(generalità, tDominante, caratteristiche, abilità)

@pre getPgInStoria(storia.titolo, utenteGiocatore.username) == pgVuoto

@post storia.utenteGiocatore.pg != pgVuoto && storia.utenteGiocatore.pg != null

Nome: gioca(personaggio)

@pre storia.utenteGiocatore.pg != pgVuoto && storia.utenteGiocatore.pg != null

Package: Sessione

Nome: creaSessione(storia.titolo)

@pre getStoriaByNome(nome)!= null

@post storia.numeroSessioni+= 1

Nome: avviaSessione(sessione)

@pre getSessione(storia.titolo, numSessione)!= null

Nome: modificaSessione(sessione)

@pre getSessione(storia.titolo, numSessione)!= null

Nome: caricaKeyword(chiave)

@pre getKeyword(storia.titolo, chiave)!= null

@post getDescrizione(chiave)

Nome: estrai()

@pre mazzo.rimaste> 0

@post mazzo.rimaste -= 1

@post getCarta(idCarta)

Nome: mischia()

@pre mazzo.rimaste< mazzo.totale

@post mazzo.rimaste== mazzo.totale

Nome: chiamaProssimo(attuale)

@pre isSelected(attuale)== true

@post isSelected(attuale+=1)== true || isSelected(1)== true

Nome: aggiungiFerita(danno, parteColpita)

@pre pg.isAlive()== true

@post getStatoParte(parteColpita) || pg.isAlive()== false

Nome: modificaCaratteristiche()

@pre pg.isAlive()== true