**Università di Salerno**

**Corso di Ingegneria del Software**

**ClickEat**

**Object Design Document**

**Versione 1.0**



|  |  |
| --- | --- |
| Progetto: ClickEat | Versione: 1.0 |
| Documento: Object design document | Data: 16/01/2019 |

**Partecipanti:**

|  |  |
| --- | --- |
| Nome | Matricola |
| Cupito Andrea [CA] | 0512104538 |
| Amoriello Luca [AL] | 0512104658 |
| Pasquariello Giovanni [PG] | 0512105020 |
| Russo Vincenzo [RV] | 0512104130 |

|  |  |
| --- | --- |
| Scritto da: | Cupito Andrea |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versione | Descrizione | Autore |
| 16/01/2019 | 1.0 | Stesura del Object design document | Membri del gruppo |

Sommario

1. **Introduzione**
   1. Object design trade off
   2. Linee guida per la Documentazione delle interfacce
   3. Definizioni, acronimi e abbreviazioni
   4. Riferimenti
2. **Packages**

2.1 Package Model

2.2 Package Manager

2.3 Package Control

2.4 Package Connection

2.5 Package Utils

2.6 Package Test

**3. Interfaccia delle classi**

3.1 ManagerUtente

3.2 ManagerPiatto

3.3 ManagerOrdinazione

3.4 ManagerTavolo

3.5 ManagerIngrediente

1. **Introduzione**
   1. **Object Design Trade-Off**

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto generalmente il nostro sistema software, tralasciando molti aspetti implementativi. Il seguente documento ha lo scopo di produrre un modello capace di integrare, in modo coerente e preciso, tutto ciò che è stato individuato nelle fasi precedenti. In particolare, definisce le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design. Inoltre, definisce i trade-off e le linee guide per la codifica del sistema.

**Comprensibilità VS Tempo**

Il codice deve essere quanto più comprensibile possibile per facilitarne la manutenzione ed eventuali modifiche future. Il codice sarà quindi accompagnato da commenti che esplicitano le singole caratteristiche del frammento. È quindi ovvio che questa caratteristica provochi un aumento dei tempi di sviluppo.

**Prestazioni VS Costi**

Essendo il nostro progetto sprovvisto di budget, al fine di mantenere prestazioni elevate, verranno utilizzati dei template open source esterni (in particolare Bootstrap).

**Interfaccia VS Usabilità**

L’interfaccia grafica è stata progettato per essere semplice e concisa, in maniera tale da rendere l’interazione con l’utente il meno complessa possibile. Essa fa uso di form e pulsanti disposti in maniera da rendere semplice l’utilizzo del sistema da parte dell’utente finale.

**Sicurezza VS Efficienza**

La sicurezza, come già descritto nel RAD, rappresenta uno degli aspetti fondamentali del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su Username e Password degli utenti.

**1.2 Linee guida per la Documentazione delle interfacce**

Gli sviluppatori dovranno seguire alcune linee guida per la stesura del codice.

**Naming Convention:**

* È buona norma utilizzare nomi che siano:
  + Descrittivi
  + Pronunciabili
  + Di uso comune
  + Non troppo lunghi
  + Non abbreviati
  + Utilizzare solo caratteri consentiti (a-z, A-Z, 0-9)

**Variabili:**

* I nomi delle variabili devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola (per indentificare l’inizio di una nuova parola). Quest’ultime devono essere dichiarate ad inizio blocco, solamente una per riga e devono essere tutte allineate.

**Esempio:** numeroIntero

* È inoltre possibile, in alcuni casi, utilizzare la notazione **underscore** “\_”, quando utilizziamo variabili costanti oppure quando vengono utilizzate proprietà statiche.

**Esempio:** CREATE\_PIATTO

**Metodi:**

* I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Il nome del metodo tipicamente consiste di un verbo che identifica l’azione da compiere, seguito dal nome di un oggetto. I nomi dei metodi per l’accesso e la modifica delle variabili devono essere del tipo: getNomeVariabile() e setNomeVariabile().

**Esempio:** getId(), setId()

* I commenti dei metodi devono essere raggruppati in base alla loro funzionalità, la descrizione dei metodi deve apparire prima di ogni dichiarazione di metodo, e deve descriverne le funzionalità. Deve includere, quando richiesto, le Precondizioni e Post-condizioni.

**Classi e pagine:**

* I nomi delle classi e delle pagine devono cominciare con la lettera maiuscola, e anche le parole seguenti al suo interno cominceranno con una maiuscola. I nomi di quest’ultime devono fornire informazioni sul loro scopo, per rendere comprensibile la loro funzionalità anche a sviluppatori che non hanno lavorato direttamente al codice.

**Esempio:** ServletLogin.java

* Ogni classe java contiene una breve introduzione alla classe; l’introduzione indica:
  + Nome della classe;
  + Autore/i;
  + Versione.

Esempio:

/\*

\* Classe: ServletLogin

\* Autore: Andrea Cupito

\* Versione: 1.0

\*/

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

**Acronimi:**

* + **RAD:** Requirements Analysis Document
  + **SDD:** System Design Document
  + **ODD:** Object Design Document

**Abbreviazioni:**

* + **DB:** Database
  1. **Riferimenti**
  + Bernd Bruegge & Allen H. Dutoit, Object-Oriented Software Engineering - Using UML, Patterns, and Java™, Third Edition, 2009
  + Documento RAD ClickEat
  + Documento SDD ClickEat

1. **Packages**

La gestione del nostro sistema è suddivisa in tre livelli (**three-tier**):

* Interface Layer
* Application Logic Layer
* Storage Layer

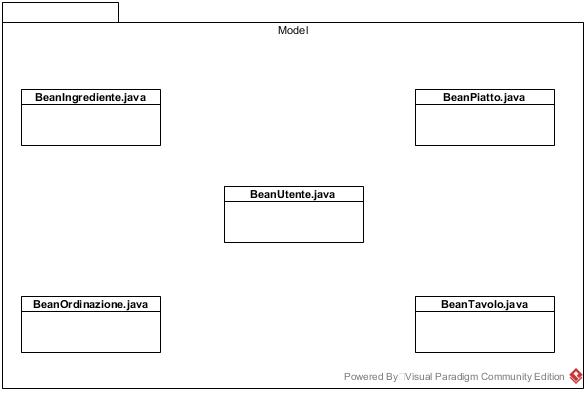
Il sistema è suddiviso in appositi packages, ognuno dei quali si occupa di una specifica parte del sistema. Le classi contenute nei package si occupano di gestire le varie richieste da soddisfare.

|  |  |
| --- | --- |
| Interface Layer | Rappresenta l’interfaccia del sistema, offre all’utente la possibilità di interfacciarsi con quest’ultimo. In tal modo può sia inviare che ricevere dati. |

|  |  |
| --- | --- |
| Application Logic Layer | Ha il compito di elaborare i dati da inviare al Client; spesso ha necessità di accedere ai dati persistenti, tramite lo Storage Layer.  Si occupa delle seguenti gestioni:   * **Gestione Utente** * **Gestione Ristorante** * **Gestione Sala** * **Gestione Ordinazione** |

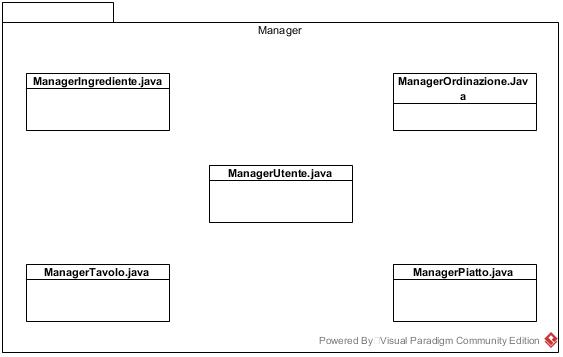
|  |  |
| --- | --- |
| Storage Layer | Ha il compito di memorizzare i dati persistenti del sistema, mediante un DBMS. Inoltre, è responsabile della comunicazione con l’Application Logic Layer, inoltrando le richieste di quest’ultimo al DBMS e fornendo i risultati in risposta. |

* 1. **Package Model**

****

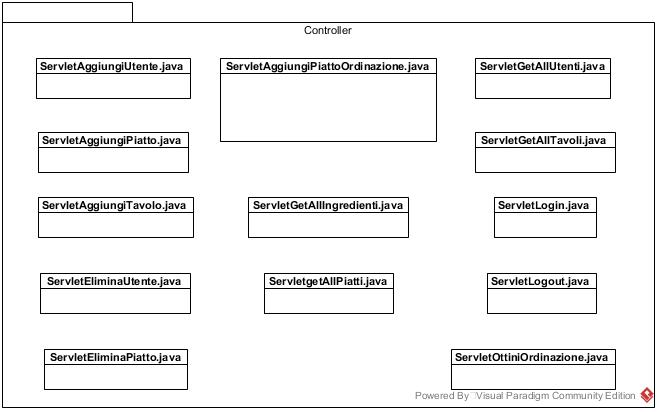
|  |  |
| --- | --- |
| **Classe:** | **Descrizione:** |
| BeanIngrediente.java | Descrive un Ingrediente inserito nel sistema |
| BeanUtente.java | Descrive un Utente inserito nel sistema |
| BeanPiatto.java | Descrive un Piatto presente nel sistema |
| BeanTavolo.java | Descrive un Tavolo presente nel sistema |
| BeanOrdinazione.java | Descrive un’Ordinazione presente nel sistema |

* 1. **Package Manager**

****

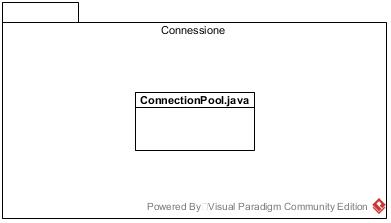
|  |  |
| --- | --- |
| **Classe:** | **Descrizione:** |
| ManagerIngrediente.java | Permette la gestione degli ingredienti presenti nel sistema |
| ManagerUtente.java | Permette di manipolare le informazioni relative all’entità Utente |
| ManagerPiatto.java | Permette la gestione dell’entità Piatto |
| ManagerTavolo.java | Permette la gestione dell’entità Tavolo |
| ManagerOrdinazione.java | Permette di gestire l’entità Ordinazione |

* 1. **Package Control**

****

|  |  |
| --- | --- |
| **Classe:** | **Descrizione:** |
| ServletAggiungiUtente.java | Permette di aggiungere un nuovo Utente nel sistema |
| ServletAggiungiPiatto.java | Permette di aggiungere un nuovo Piatto nel sistema, con le informazioni che si desiderano |
| ServletAggiungiTavolo.java | Permette l’inserimento di un nuovo Tavolo nel sistema |
| ServletEliminaUtente.java | Permette di eliminare un Utente dal sistema |
| ServletEliminaPiatto.java | Permette di eliminare un Piatto presente nel sistema |
| ServletAggiungiPiattoOrdinazione.java | Permette di aggiungere un Piatto ad un’ordinazione, ritorna l’ordinazione così ottenuta |
| ServletgetAllIngredienti.java | Ritorna una lista contenente tutti gli Ingredienti attualmente presenti nel sistema |
| ServletGetAllPiatto.java | Ritorna una lista contenente tutti i Piatti attualmente presenti nel sistema |
| ServletGetAllUtenti.java | Ritorna una lista contenente tutti gli Utenti attualmente registrati nel sistema |
| ServletGetAllTavoli.java | Ritorna una lista contenente tutti i Tavoli attualmente presenti nel sistema |
| ServletLogin.java | Responsabile del Login dell’Utente |
| ServletLogout.java | Responsabile del Logout dell’Utente |
| ServletOttieniOrdinazione.java | Ritorna una lista contenente tutte le Ordinazioni attualmente presenti nel sistema |

* 1. **Package Connection**

****

|  |  |
| --- | --- |
| **Classe:** | **Descrizione:** |
| ConnectionPool.java | Responsabile della connessione con il DB |

1. **Interfaccia della classi**

Passiamo alla specifica dei singoli metodi per la gestione delle entità sopracitate.

* 1. **Manager Utente**

Trattasi del Manager che gestisce il Model Utente.

NOTA: collezioneUtente rappresenta la lista di Utenti attualmente registrati nel sistema; utile per rappresentare vincoli su collezioni di Utenti.

|  |  |
| --- | --- |
| **Classe:** | ManagerUtente |
| **Descrizione:** | Questa classe permette l’inserimento, l’eliminazione e la gestione dell’entità Utente. |
| **Pre-condizioni:** | **context:** ManagerUtente  creaUtente(String nomeUtente, String passwordUtente, String cognomeUtente, String ruoloUtente, String idLogin)  **pre:** nomeUtente <> null && passwordUtente <> null && cognomeUtente <> null && ruoloUtente <> null && idLogin <> null.  3 <= nomeUtente->size() <= 35 && 8 <= passwordUtente->size() <= 35 && 3 <= cognomeUtente->size() <= 35 && 8 <= ruoloUtente->size() <= 35 &&  4 <= idLogin->size () <= 35.  collezioneUtenti -> not exists(idLogin)  **context:** ManagerUtente  eliminaUtente(Integer idUtente)  **pre:** idUtente <> null  collezioneUtente -> exists(idUtente)  **context:** ManagerUtente  ottieniUtenti()  **context**: ManagerUtente  valoriLogin (String idLogin, String passwordUtente)  **pre:** idLogin <> null && passwordUtente <> null  collezioneUtenti -> exists(idLogin) && collezioneUtenti -> exists (passwordUtente)  **context:** ManagerUtente  eliminaUtenteViaIdLogin (String idLogin)  **pre:** idLogin <> null  collezioneUtenti -> exists (idLogin) |
| **Post-condizioni:** | **context:** ManagerUtente  creaUtente(String nomeUtente, String passwordUtente, String cognomeUtente, String ruoloUtente, String idLogin)  **post:** BeanUtente.nomeUtente = nomeUtente && BeanUtente.cognomeUtente = cognomeUtente && BeanUtente.passwordUtente = passwordUtente && BeanUtente.ruoloUtente = ruoloUtente && BeanUtente.idLogin = idLogin  **context:** ManagerUtente  eliminaUtente(Integer idUtente)  **post:** collezioneUtenti -> not exists (utente.idUtente)  **context:** ManagerUtente  ottieniUtenti()  **post:** listaUtenti <> null  **context**: ManagerUtente  valoriLogin (String idLogin, String passwordUtente)  **post:** BeanUtente <> null  **context:** ManagerUtente  eliminaUtenteViaIdLogin (String idLogin)  **post:** collezioneUtenti -> not exists (Utente.idLogin) |

* 1. **Manager Piatto**

Trattasi del Manager che gestisce il Model Piatto.

NOTA: collezionePiatti rappresenta la lista di Piatti attualmente registrati nel sistema; utile per rappresentare vincoli su collezioni di Piatti; collezionePiattiCompleti rappresenta la lista di Piatti comprendente la lista di ingredienti allegati ad essi.

|  |  |
| --- | --- |
| **Classe:** | ManagerPiatto |
| **Descrizione:** | Questa classe permette l’inserimento, l’eliminazione e la gestione dell’entità Piatto. |
| **Pre-condizioni:** | **context:** ManagerPiatto  ottieniPiatto (int idPiatto)  **pre:** idPiatto <> null && collezionePiatti -> exists (Piatto.idPiatto)  **context:** ManagerPiatto  ottieniListaPiatti()  **context:** ManagerPiatto  IngredientiNelPiatto (int idPiatto)  **pre:** idPiatto <> null && collezionePiatti -> exists (Piatto.idPiatto)  **context:** ManagerPiatto  creaPiatto (String nomePiatto, Float prezzoPiatto, String categoriaPiatto)  **pre:** nomePiatto <> null && prezzoPiatto <> null && categoriaPiatto <> null  4 <= nomePiatto -> size () <= 35 && prezzoPiatto > 0.00  collezionePiatti -> not exists ( nomePiatto )  **context:** ManagerPiatto  InserisciIngredientiNelPiatto (BeanPiatto piatto)  **pre:** piatto <> null && collezionePiatti -> exists ( piatto )  **context:** ManagerPiatto  settaIdPiatto (BeanPiatto piatto)  **pre:** piatto<> null && collezionePiatti -> exists ( piatto )  **context:** ManagerPiatto  eliminaPiatto (int idPiatto)  **pre:** idPiatto<> null && collezionePiatti -> exists ( piatto.idPiatto )  **context:** ManagerPiatto  eliminaPiattoiFromIngredientiPiatto (int idPiatto)  **pre:** idPiatto<> null && collezionePiatti -> exists ( piatto.idPiatto ) |
| **Post-condizioni:** | **context:** ManagerPiatto  ottieniPiatto (int idPiatto)  **post:** BeanPiatto <> null  **context:** ManagerPiatto  ottieniListaPiatti()  **post:** ArrayList <BeanPiatto> <> null  **context:** ManagerPiatto  IngredientiNelPiatto (int idPiatto)  **Post:** ArrayList <BeanIngrediente> <> null  **context:** ManagerPiatto  creaPiatto (String nomePiatto, Float prezzoPiatto, String categoriaPiatto)  **post:** BeanPiatto piatto; piatto.nomePiatto = nomePiatto && piatto.prezzoPiatto = prezzoPiatto && piatto.categoriaPiatto = categoriaPiatto  **context:** ManagerPiatto  InserisciIngredientiNelPiatto (BeanPiatto piatto)  **Post:** piatto.listaIngredienti != null  **context:** ManagerPiatto  settaIdPiatto (BeanPiatto piatto)  **post:** piatto.idPiatto = idPiatto  **context:** ManagerPiatto  eliminaPiatto (int idPiatto)  **post:** collezionePiatti -> not exists ( piatto.idPiatto )  **context:** ManagerPiatto  eliminaPiattoiFromIngredientiPiatto (int idPiatto)  **post:** collezionePiattiCompleti -> not exists ( piatto.idPiatto ) |

* 1. **Manager Ordinazione**

Trattasi del Manager che gestisce il Model Piatto.

|  |  |
| --- | --- |
| **Classe:** | ManagerOrdinazione |
| **Descrizione:** | Questa classe permette la gestione dell’entità Ordinazione. |
| **Pre-condizioni:** | **context:** ManagerOrdinazione  creaOrdinazione()  **context:** ManagerOrdinazione  impostaIdEDataOrdine ( BeanOrdinazione ordinazione )  **pre:** collezioneOrdinazioni -> exists ( ordinazione )  **context:** ManagerOrdinazione  ottieniOrdinazione (int numeroTavolo)  **pre:** numeroTavolo <> null |
| **Post-condizioni:** | **context:** ManagerOrdinazione  creaOrdinazione()  **post:** new BeanOrdinazione  **context:** ManagerOrdinazione  impostaIdEDataOrdine ( BeanOrdinazione ordinazione )  **post:** ordinazione.numeroOrdinazione <> null && ordinazione.dataOrdinazione <> null  **context:** ManagerOrdinazione  ottieniOrdinazione (int numeroTavolo)  **post:** BeanOrdinazione <> null |

* 1. **Manager Tavolo**

Trattasi del Manager che gestisce il Model Piatto.

|  |  |
| --- | --- |
| **Classe:** | ManagerTavolo |
| **Descrizione:** | Questa classe permette la gestione dell’entità Tavolo |
| **Pre-condizioni:** | **context:** ManagerTavolo  ottieniTavoli()  **context:** ManagerTavolo  creaTavolo ( int numeroTavolo )  **pre:** numeroTavolo <> null && collezioneTavoli -> exists ( numeroTavolo )  **context:** ManagerTavolo  eliminaTavolo ( int numeroTavolo )  **pre:** numeroTavolo <> null && collezioneTavoli -> exists ( numeroTavolo )  **context:** ManagerTavolo  freeTavolo ( BeanTavolo tavolo)  **pre:** tavolo <> null  **context:** ManagerTavolo  isOccupato ( int numeroTavolo)  **pre:** numeroTavolo <> null && collezioneTavoli -> exists ( numeroTavolo )  **context:** ManagerTavolo  isContoPresente ( int numeroTavolo)  **pre:** numeroTavolo <> null && collezioneTavoli -> exists ( numeroTavolo )  **context:** ManagerTavolo  setOccupato ( int numeroTavolo)  **pre:** numeroTavolo <> null && collezioneTavoli -> exists ( numeroTavolo )  **context:** ManagerTavolo  getTavolo ( int numeroTavolo)  **pre:** numeroTavolo <> null && collezioneTavoli -> exists ( numeroTavolo )  **context:** ManagerTavolo  getOrdinazioneDiTavolo ( int numeroTavolo)  **pre:** numeroTavolo <> null && collezioneTavoli -> exists ( numeroTavolo ) |
| **Post-condizioni:** | **context:** ManagerTavolo  ottieniTavoli()  **post:** ArrayList <BeanTavolo> <> null  **context:** ManagerTavolo  creaTavolo ( int numeroTavolo )  **post:** BeanTavolo tavolo <> null && collezioneTavoli -> exists ( tavolo )  **context:** ManagerTavolo  eliminaTavolo ( int numeroTavolo )  **post:** collezioneTavoli -> not exists ( numeroTavolo )  **context:** ManagerTavolo  freeTavolo ( BeanTavolo tavolo)  **post:** tavolo.flagConto = false && tavolo.flagOccupato = false && tavolo.numeroOrdinazione = null  **context:** ManagerTavolo  isOccupato ( int numeroTavolo)  **context:** ManagerTavolo  isContoPresente ( int numeroTavolo)  **context:** ManagerTavolo  setOccupato ( int numeroTavolo)  **post:** tavolo.flagOccupato = true  **context:** ManagerTavolo  getTavolo ( int numeroTavolo)  **post:** tavolo <> null  **context:** ManagerTavolo  getOrdinazioneDiTavolo ( int numeroTavolo)  **post:** tavolo.numerOrdinazione <> null IF collezioneTavoli -> exists (numerOrdinazione) |

* 1. **Manager Ingrediente**

Trattasi del Manager che gestisce il Model Piatto.

|  |  |
| --- | --- |
| **Classe:** | ManagerIngrediente |
| **Descrizione:** | Questa classe permette la gestione dell’entità Ingrediente |
| **Pre-condizioni:** | **context:** ManagerIngrediente  creaIngrediente ( String nomeIngrediente )  **pre:** nomeIngrediente <> null && collezioneIngredienti -> not exists ( nomeIngrediente )  **context:** ManagerIngrediente  eliminaIngrediente ( int idIngrediente )  **pre:** idIngrediente <> null  **context:** ManagerIngrediente  ottieniListaIngredienti ()  **context:** ManagerIngrediente  ricercaPerId (int idIngrediente)  **pre:** idIngrediente <> null  **context:** ManagerIngrediente  ricercaPerNome (String nomeIngrediente)  **pre:** nomeIngrediente <> null |
| **Post-condizioni:** | **context:** ManagerIngrediente  creaIngrediente ( String nomeIngrediente )  **post:** BeanIngrediente <> null && BeanIngrediente.nomeIngrediente 0 nomeIngrediente  **context:** ManagerIngrediente  eliminaIngrediente ( int idIngrediente )  **post:** collezioneIngredienti -> not exists ( idIngrediente )  **context:** ManagerIngrediente  ottieniListaIngredienti ()  **post:** ArrayList <BeanIngrediente> <> null  **context:** ManagerIngrediente  ricercaPerId (int idIngrediente)  **post:** BeanIngrediente <> null  **context:** ManagerIngrediente  ricercaPerNome (String nomeIngrediente)  **post:** BeanIngrediente <> null |