|  |
| --- |
| **Università degli Studi di Salerno Corso di Ingegneria del Software** |

**Hotel Campus  
Object Design Document  
Versione 1.3**

Immagine che contiene schermata, Elementi grafici, design

Descrizione generata automaticamente

Data: 03/01/2025

**Coordinatore del progetto:**

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
|  |  |
|  |  |

**Partecipanti:**

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
| Luca Del Bue | 0512116173 |
| Salvatore Di Martino | 0512116932 |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Scritto da:** | Team members |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autore** |
| 05/12/2024 | 0.1 | Prima stesura | Team members |
| 09/12/2024 | 0.2 | Specifica delle interfacce con contratti OCL dei metodi: GestioneServiziService, GestioneUtentiService e Prenotazione | Luca Del Bue |
| 10/12/2024 | 0.3 | Specifica delle interfacce con contratti OCL dei metodi: GestioneCamereService, GestionePrenotazioniService e Cliente | Salvatore Di Martino |
| 16/12/2024 | 1.0 | Modifica alla specifica dell’interfaccia GestionePrenotazioniService | Luca Del Bue |
| 26/12/2024 | 1.1 | Aggiunta setRuolo alla specifica dell’interfaccia GestioneUtentiService | Salvatore Di Martino |
| 01/01/2025 | 1.2 | Revisione contenuto documento – modifica Packages | Salvatore Di Martino |
| 03/01/2025 | 1.3 | Modifica ottimizzazione del modello a oggetti | Salvatore Di Martino |

Indice

1. Introduzione 4

1.1. Scopo del Sistema 4

1.2. Obiettivi di progettazione 4

1.3. Linee guida per la documentazione dell’interfaccia 4

1.4. Ottimizzazione del modello a oggetti 4

2. Packages 6

2.1. Struttura del progetto 6

3. Interfaccia delle classi 9

4. Design Pattern 17

1. Introduzione

## Scopo del Sistema

Lo scopo del sistema è permettere la gestione dell’attività di una struttura alberghiera. Si intende

gestire le camere, i servizi offerti e le prenotazioni effettuate dai clienti. Queste operazioni

verranno eseguite rispettivamente dal gestore delle camere e servizi, ossia il direttore, e dal

gestore delle prenotazioni. Inoltre, i clienti avranno la possibilità di registrarsi, accedere,

finalizzare una prenotazione e visualizzarle.

## Obiettivi di progettazione

**Usabilità**

Il sistema deve garantire la validità dei dati di input attraverso vincoli e notifiche d’errore che segnalano eventuali incongruenze, assicurando consistenza e correttezza. L'interfaccia utente deve essere semplice, intuitiva, responsive e adattabile a diversi dispositivi, con una barra di navigazione che garantisca accessibilità e navigabilità uniforme.

**Riusabilità**

Il sistema deve garantire risuabilità, attraverso il concetto di ereditarietà del paradigma object oriented e l’utilizzo di diversi design pattern.

**Affidabilità**

Il sistema realizzato deve essere robusto, effettuando delle verifiche sui dati inseriti in modo tale da gestire gli input non validi, aggiungendo un ulteriore livello di controllo oltre alla prima verifica effettuata dopo l’inserimento degli input da parte dell’utente. Questo consente di affrontare situazioni non previste attraverso la gestione delle eccezioni.

## Linee guida per la documentazione dell’interfaccia

Le convenzioni usate nell’implementazione del sistema fanno riferimento alla specifica:

* Java Sun: [*https://checkstyle.sourceforge.io/sun\_style.html*](https://checkstyle.sourceforge.io/sun_style.html)

## Ottimizzazione del modello a oggetti

**Immagine che contiene testo, diagramma, linea, Piano

Descrizione generata automaticamente**

Il modello a oggetti, definito in fase di analisi, è stato ristrutturato aggiungendo l'attributo booleano isDeleted nelle classi Servizio e Camera per implementare la cancellazione logica degli oggetti nel database. In questo modo è possibile mantenere sempre un riferimento alla camera o al servizio prenotato, nonché alle relative informazioni persistenti, anche se l'oggetto viene "eliminato", ossia reso non disponibile.

Inoltre, la classe di associazione Servizio Prenotato è stata resa una classe concreta per gestirne la persistenza.

La classe Cliente è stata trasformata in un’entità persistente, mantenendo una relazione con l’entità Utente, che gestisce i dati dell’account (ad esempio email, password e informazioni personali). Questa scelta progettuale permette di tracciare in modo efficace le prenotazioni associate a ciascun cliente, garantendo al contempo una chiara distinzione tra i diversi tipi di utenti e le loro responsabilità all'interno del sistema.

1. Packages

## Struttura del progetto

In questa sezione viene presentata la struttura organizzativa dei file all’interno del progetto. È la tipica organizzazione delle directory di un progetto SpringBoot, con Maven come gestore delle dipendenze.

* **.mvn**, contiene i file di configurazione Maven
* **src**, contiene i file sorgente del progetto
  + **main**
    - **java**, contiene i package e i file Java
    - **resources**, contiene le risorse relative all’interfaccia utente
      * **static**, contiene le risorse statiche (CSS, JS)
      * **templates**, contiene i file HTML dinamici realizzati con Thymeleaf
  + **test**
    - **java**, contiene le classi di testing
* **target**, contiene i file build di Maven

Il sistema è costituito da un package generale chiamato *it.unisa.hotelcampus*, e all’interno di esso è presente un package per ogni sottosistema individuato. Inoltre, all’interno del package generale, vi è un package *model*, contenente le classi entity e i DAO per gestire la persistenza, ed un package *utils*, dove vengono inserite le classi di utilità per funzionalità comuni all’intero sistema.

Immagine che contiene testo, Rettangolo, schermata, diagramma

Descrizione generata automaticamente

**Package Gestione Utenti**

Immagine che contiene testo, schermata, Rettangolo, diagramma

Descrizione generata automaticamente

**Package Gestione Camere**

**Immagine che contiene testo, schermata, diagramma, Rettangolo

Descrizione generata automaticamente**

**Package Gestione Servizi**

**Immagine che contiene testo, schermata, Rettangolo, diagramma

Descrizione generata automaticamente**

**Package Gestione Prenotazioni**

Immagine che contiene testo, schermata, diagramma, Rettangolo

Descrizione generata automaticamente

1. Interfaccia delle classi

**JavaDoc**

La documentazione del codice delle classi verrà realizzata utilizzando JavaDoc.

Link JavaDoc HotelCampus: //da aggiungere

Di seguito vengono descritte le interfacce pubbliche di ciascun sottosistema. Viene specificata una descrizione dell’interfaccia, le dipendenze con altre classi o packages, attributi, operazioni ed eccezioni che possono essere sollevate.

**Package Gestione Utenti**

|  |  |
| --- | --- |
| Interfaccia | GestioneUtentiService |
| Descrizione | Gestione Utenti fornisce il servizio relativo all’autenticazione, creazione ed eliminazione degli account |
| Metodi | + autentica(String email, String password) : Utente  + creaUtente(String nome, String cognome, Date dataDiNascita, String nazionalità, String email, String password) : Utente  + elimina(Utente account) : boolean  + getUtente(String email) : Utente  + getUtenti() : Collection<Utente>  + getClienti() : Collection<Cliente>  + setRuolo(Utente account, String ruolo) : void |
| Invariante di classe | // |

|  |  |
| --- | --- |
| Nome Metodo | + autentica(String email, String password) : Utente |
| Descrizione | Il metodo autentica consente di verificare se un utente esiste nel sistema ed è autorizzato ad accedere. |
| Pre-condizione | context GestioneUtentiService::autentica(String email, String password): Utente  pre: email <> null and password <> null |
| Post-condizione | context GestioneUtentiService::autentica(String email, String password): Utente  post: result <> null and result.email = email |
| Nome Metodo | + creaUtente(String nome, String cognome, Date dataDiNascita, String nazionalità, String email, String password) : Utente |
| Descrizione | Il metodo creaUtente consente di creare un nuovo utente nel sistema. Restituisce l’oggetto Utente creato se i parametri forniti sono validi e non esistono utenti con la stessa email. |
| Pre-condizione | context GestioneUtentiService::creaUtente(String nome, String cognome, Date dataDiNascita, String nazionalità, String email, String password) : Utente  pre: nome <> null and nome.size() > 0  pre: cognome <> null and cognome.size() > 0  pre: dataDiNascita <> null and dataDiNascita <= Date::now()  pre: nazionalità <> null and nazionalità.size() > 0  pre: email <> null and email.size() > 0  pre: password <> null and password.size() >= 8  pre: self.getUtente(email) = null |
| Post-condizione | context GestioneUtentiService::creaUtente(String nome, String cognome, Date dataDiNascita, String nazionalità, String email, String password) : Utente  post: result <> null and self.getUtente(email) = result |
| Nome Metodo | + elimina(Utente account) : boolean |
| Descrizione | Il metodo elimina consente di eliminare l’account di un utente dal sistema. |
| Pre-condizione | Context GestioneUtentiService::elimina(Utente account) : boolean  pre: self.getUtente(account.email) = account |
| Post-condizione | Context GestioneUtentiService::elimina(Utente account) : boolean  post: result = true and self.getUtente(account.email) = null |
| Nome Metodo | + getUtente(String email) : Utente |
| Descrizione | Il metodo getUtente consente di estrarre un utente dal sistema mediante la sua email. |
| Pre-condizione | Context GestioneUtentiService::getUtente(String email) : Utente  pre: email <> null and email.size() > 0 |
| Post-condizione | Context GestioneUtentiService::getUtente(String email) : Utente  post: result <> null and self.getUtenti().exist(u | u.email = email)) |
| Nome Metodo | + getUtenti() : Collection<Utente> |
| Descrizione | Il metodo getUtenti consente di estrarre tutti gli utenti dal sistema |
| Pre-condizione | // |
| Post-condizione | // |
| Nome Metodo | + getClienti() : Collection<Cliente> |
| Descrizione | Il metodo getClienti consente di estrarre tutti gli utenti di tipo cliente dal sistema. |
| Pre-condizione | // |
| Post-condizione | // |
| Nome Metodo | + setRuolo(Utente account, String ruolo) : void |
| Descrizione | Il metodo setRuolo consente di modificare il ruolo di un utente. |
| Pre-condizione | Context GestioneUtentiService::setRuolo(Utente account, String ruolo) : void  pre: ruolo <> null and ruolo.size()>0  pre: account != null |
| Post-condizione | Context GestioneUtentiService::setRuolo(Utente account, String ruolo) : void  post: account.ruolo = ruolo |

**Package Gestione Camere**

|  |  |
| --- | --- |
| Interfaccia | GestioneCamereService |
| Descrizione | Gestione Camere permette di inserire, rimuovere le camere e verificarne la disponibilità |
| Metodi | + getCamere() : Collection<Camera>  + creaCamera(int numero, TipoCamera tipo, int numeroMaxOspiti, int quadratura, int costo, String immagine) : Camera  + rimuoviCamera(Camera camera) : boolean  + getCamereDisponibili(Date checkIn, Date checkOut, int numeroOspiti) : Collection<Camera>  + verificaDisponibilita(Camera camera, Date checkIn, Date checkOut): boolean |
| Invariante di classe | // |

|  |  |
| --- | --- |
| Nome Metodo | + getCamere() : Collection<Camera> |
| Descrizione | Restituisce tutte le camere registrate e non eliminate logicamente nel sistema. |
| Pre-condizione | // |
| Post-condizione | Context GestioneCamereService::getCamere() : Collection<Camera>  post: result 🡪 forAll(c | c.isDeleted = false) |
| Nome Metodo | **+** creaCamera(int numero, TipoCamera tipo, int numeroMaxOspiti, int quadratura, int costo, String immagine) : Camera |
| Descrizione | Permette di creare una nuova camera con i parametri specificati. Se il numero di camera fornito corrisponde ad una camera attiva nel sistema, questa viene cancellata logicamente e viene creata la nuova camera attiva. |
| Pre-condizione | context GestioneCamereService::creaCamera(int numero, TipoCamera tipo, int numeroMaxOspiti, int quadratura, int costo, String immagine) : Camera  pre: numero > 0  pre: tipo <> null  pre: numeroMaxOspiti > 0  pre: quadratura > 0  pre: costo > 0  pre: immagine <> null and immagine.size() > 0 |
| Post-condizione | context GestioneCamereService::creaCamera(int numero, TipoCamera tipo, int numeroMaxOspiti, int quadratura, int costo, String immagine) : Camera  post: result <> null and self.getCamere() 🡪 include(result) |
| Nome Metodo | + rimuoviCamera(Camera camera) : boolean |
| Descrizione | Cancella logicamente una camera attiva nel sistema. |
| Pre-condizione | context GestioneCamereService::rimuoviCamera(Camera camera) : boolean  pre: camera <> null  pre: self.getCamere() 🡪 include(camera) |
| Post-condizione | context GestioneCamereService::rimuoviCamera(Camera camera) : boolean  post: result = true and (not self.getCamere() 🡪 include(camera)) |
| Nome Metodo | + getCamereDisponibili(Date checkIn, Date checkOut, int numeroOspiti) : Collection<Camera> |
| Descrizione | Restituisce tutte le camere registrate e non eliminate logicamente nel sistema che soddisfano determinati parametri. |
| Pre-condizione | Context GestioneCamereService::getCamereDisponibili(Date checkIn, Date checkOut, int numeroOspiti) : Collection<Camera>  pre: checkIn <> null and checkOut <> null  pre: checkIn < checkOut  pre: numeroOspiti > 0 |
| Post-condizione | Context GestioneCamereService::getCamereDisponibili(Date checkIn, Date checkOut, int numeroOspiti) : Collection<Camera>  post: (result <> null and result 🡪 forAll(c | c.numeroMaxOspiti >= numeroOspiti and self.verificaDisponibilita(c, checkIn, checkOut))) |
| Nome Metodo | + verificaDisponibilita(Camera camera, Date checkIn, Date checkOut) : boolean |
| Descrizione | Verifica se una camera specifica è disponibile per il periodo desiderato |
| Pre-condizione | Context GestioneCamereService::verificaDisponibilita(Camera camera, Date checkIn, Date checkOut) : boolean  pre: camera <> null  pre: checkIn <> null and checkOut <> null  pre: checkIn < checkOut  pre: self.getCamere() 🡪 include(camera) |
| Post-condizione | Context GestioneCamereService::verificaDisponibilita(Camera camera, Date checkIn, Date checkOut) : boolean  post: result = true and not camera.prenotazioni 🡪 exists(p | (p.dataCheckIn <= checkIn and p.dataCheckOut >= checkOut) or (checkIn < p.dataCheckIn < checkOut) or (checkIn < p.dataCheckOut < checkOut)) |

**Package Gestione Servizi**

|  |  |
| --- | --- |
| Interfaccia | GestioneServiziService |
| Descrizione | Gestione Servizi permette di inserire, rimuovere le camere e verificarne la disponibilità |
| Metodi | + getServizi() : Collection<Servizio>  + creaServizio(String nome, String descrizione, int costo, String immagine) : Servizio  + rimuoviServizio(Servizio servizio) : boolean |
| Invariante di classe | // |

|  |  |
| --- | --- |
| Nome Metodo | + getServizi() : Collection<Servizio> |
| Descrizione | Restituisce tutti i servizi registrati e non eliminati logicamente nel sistema |
| Pre-condizione | // |
| Post-condizione | context GestioneServiziService::getServizi() : Collection<Servizio>  post: result 🡪 forAll(s | s.isDeleted = false) |
| Nome Metodo | + creaServizio(String nome, String descrizione, int costo, String immagine) : Servizio |
| Descrizione | Permette di creare un nuovo servizio con i parametri specificati. |
| Pre-condizione | context GestioneServiziService::creaServizio(String nome, String descrizione, int costo, String immagine) : Servizio  pre: nome <> null and nome.size() > 0  pre: descrizione <> null and descrizione.size() > 0  pre: costo > 0  pre: immagine <> null and immagine.size() > 0 |
| Post-condizione | context GestioneServiziService::creaServizio(String nome, String descrizione, int costo, String immagine) : Servizio  post: result <> null and self.getServizi() 🡪 include(result) |
| Nome Metodo | + rimuoviServizio(Servizio servizio) : boolean |
| Descrizione | Cancella logicamente un servizio attivo nel sistema. |
| Pre-condizione | context GestioneServiziService::rimuoviServizio(Servizio servizio) : boolean  pre: servizio <> null  pre: self.getServizi() 🡪 include(servizio) |
| Post-condizione | context GestioneServiziService::rimuoviServizio(Servizio servizio) : boolean  post: result = true and (not self.getServizi() 🡪 include(servizio)) |

**Package Gestione Prenotazioni**

|  |  |
| --- | --- |
| Interfaccia | GestionePrenotazioniService |
| Descrizione | Gestione Prenotazioni ha il compito di aggiungere ed eliminare le prenotazioni effettuate dai clienti |
| Metodi | + getPrenotazioni() : Collection<Prenotazioni>  + creaPrenotazione(Date dataCheckIn, Date dataCheckOut, int numeroOspiti, Camera camera, List<ServizioPrenotato> servizi, Cliente cliente) : Prenotazione  + eliminaPrenotazione(Prenotazione prenotazione) : boolean  + cercaPrenotazioni(String email, Date checkIn, Date checkOut) : Collection<Prenotazioni> |
| Invariante di classe | // |

|  |  |
| --- | --- |
| Nome Metodo | + getPrenotazioni() : Collection<Prenotazioni> |
| Descrizione | Restituisce tutte le prenotazioni effettuate registrate nel sistema |
| Pre-condizione | // |
| Post-condizione | // |
| Nome Metodo | + creaPrenotazione(Date dataCheckIn, Date dataCheckOut, int numeroOspiti, Camera camera, List<ServizioPrenotato> servizi, Cliente cliente) : Prenotazione |
| Descrizione | Permette di creare una nuova prenotazione nel sistema specificando i parametri della prenotazione, la camera ed i servizi prenotati e il cliente che la effettua |
| Pre-condizione | context GestionePrenotazioniService::creaPrenotazione(Date dataCheckIn, Date dataCheckOut, int numeroOspiti, Camera camera, List<ServizioPrenotato> servizi, Cliente cliente) : Prenotazione  pre: dataCheckIn <> null and dataCheckIn >= Date::now()  pre: dataCheckOut <> null and dataCheckOut > Date::now()  pre: numeroOspiti > 0  pre: camera <> null  pre: servizi <> null  pre: cliente <> null  pre: dataCheckIn < dataCheckOut  pre: numeroOspiti <= camera.numeroMaxOspiti  pre: GestioneCamereService.verificaDisponibilita(camera, dataCheckIn, dataCheckOut) = true  pre: servizi 🡪 forAll(s | s.numeroServizi <= numeroOspiti) |
| Post-condizione | context GestionePrenotazioniService::creaPrenotazione(Date dataCheckIn, Date dataCheckOut, int numeroOspiti, Camera camera, List<ServizioPrenotato> servizi, Cliente cliente) : Prenotazione  post: self.getPrenotazioni() 🡪 include(result) and result <> null  post: cliente.getPrenotazioni() 🡪 include(result) |
| Nome Metodo | + eliminaPrenotazione(Prenotazione prenotazione) : boolean |
| Descrizione | Permette di eliminare una prenotazione che non è ancora stata consumata |
| Pre-condizione | context GestionePrenotazioniService::eliminaPrenotazione(Prenotazione prenotazione) : boolean  pre: prenotazione <> null  pre: prenotazione.dataCheckIn > Date::now() |
| Post-condizione | context GestionePrenotazioniService::eliminaPrenotazione(Prenotazione prenotazione) : boolean  post: result = true and (not self.getPrenotazioni 🡪 include(result)) |
| Nome Metodo | + cercaPrenotazioni(String email, Date checkIn, Date checkOut) : Collection<Prenotazioni> |
| Descrizione | Restituisce le prenotazioni presenti nel sistema che rispettano i parametri specificati |
| Pre-condizione | context GestionePrenotazioniService::cercaPrenotazioni(String email, Date checkIn, Date checkOut) : Collection<Prenotazioni>  pre: (checkIn < checkOut and checkIn <> null and checkOut <> null) or (checkIn <> null and checkOut = null) or (checkOut <> null and checkIn = null) or (checkIn = null and checkOut = null) |
| Post-condizione | context GestionePrenotazioniService::cercaPrenotazioni(String email, Date checkIn, Date checkOut) : Collection<Prenotazioni>  post: result <> null and result 🡪 forAll(p | self.getPrenotazioni() 🡪 include(p)) |

**Package Entity**

|  |  |
| --- | --- |
| Interfaccia | Prenotazione |
| Descrizione | Modella una prenotazione interna al sistema |
| Metodi | // |
| Invariante di classe | context Prenotazione  inv: self.dataCheckIn < self.dataCheckOut  inv: self.numeroOspiti <= self.camera.numeroMaxOspiti |

|  |  |
| --- | --- |
| Interfaccia | Cliente |
| Descrizione | Modella un cliente della struttura |
| Metodi | + creaPrenotazione(Prenotazione prenotazione) : boolean  + eliminaPrenotazione(Prenotazione prenotazione) : boolean  + cercaPrenotazioni(Date checkIn, Date checkOut) : Collection<Prenotazione> |
| Invariante di classe | // |

|  |  |
| --- | --- |
| Nome Metodo | + creaPrenotazione(Prenotazione prenotazione) : boolean |
| Descrizione | Associa una prenotazione al cliente che l’ha effettuata |
| Pre-condizione | context Cliente::creaPrenotazione(Prenotazione prenotazione) : boolean  pre: prenotazione <> null |
| Post-condizione | context Cliente::creaPrenotazione(Prenotazione prenotazione) : boolean  post: result = true and self.prenotazioni 🡪 include(prenotazione) |
| Nome Metodo | + eliminaPrenotazione(Prenotazione prenotazione) : boolean |
| Descrizione | Rimuove una prenotazione futura associata ad un cliente |
| Pre-condizione | context Cliente::eliminaPrenotazione(Prenotazione prenotazione) : boolean  pre: prenotazione <> null  pre: prenotazione.dataCheckIn > Date::now() |
| Post-condizione | context Cliente::eliminaPrenotazione(Prenotazione prenotazione) : boolean  post: result = true and not self.prenotazioni 🡪 include(prenotazione) |
| Nome Metodo | + cercaPrenotazioni(Date checkIn, Date checkOut) : Collection<Prenotazione> |
| Descrizione | Restituisce tutte le prenotazioni di un cliente che rispettano i parametri forniti |
| Pre-condizione | context Cliente::cercaPrenotazioni(Date checkIn, Date checkOut) : Collection<Prenotazione>  pre: checkIn <> null  pre: checkOut <> null  pre: checkIn < checkOut |
| Post-condizione | context Cliente::cercaPrenotazioni(Date checkIn, Date checkOut) : Collection<Prenotazione>  post: result <> null  post: result 🡪 forAll(p | self.prenotazioni 🡪 include(p)) |

1. Design Pattern

**Facade**

Il Facade Pattern è un design pattern strutturale che fornisce un'interfaccia semplificata per un insieme complesso di sottosistemi. Questo pattern nasconde la complessità del sistema sottostante fornendo un'interfaccia unificata di più alto livello, rendendo il sistema più facile da utilizzare. È particolarmente utile quando si ha bisogno di fornire un'interfaccia semplice per un sottosistema complesso. Si garantisce così un alto disaccoppiamento e si rende la piattaforma più manutenibile e più aggiornabile, poiché basterà cambiare l’implementazione dei metodi dell’interfaccia per implementare le modifiche.

Nel nostro caso ogni sottosistema avrà la propria interfaccia:

* GestioneServiziService
* GestioneUtentiService
* GestioneCamereService
* GestionePrenotazioniService

**DAO**

Un Data Access Object (DAO) è un pattern progettuale che fornisce un'interfaccia per interagire con un database in modo astratto. Grazie a questa struttura, è possibile eseguire operazioni sui dati senza preoccuparsi dei dettagli specifici del database. Il DAO consente di mappare le operazioni dell'applicazione sullo stato persistente dei dati, offrendo un modo standardizzato per gestire la persistenza.

In un'applicazione Spring Boot, il pattern DAO viene implementato creando una classe che rappresenta l'entità del database e un'interfaccia repository che estende una delle interfacce fornite da Spring Data JPA, come JpaRepository. Questo approccio consente di accedere e gestire i dati tramite metodi già pronti, come il salvataggio, la ricerca e la cancellazione, senza dover scrivere query SQL manuali.