Laurea Triennale in Informatica-Università di Salerno Corso di Ingegneria del Software- Prof C. Gravino

Object Design Document

MediCare

Riferimento	
Versione	0.5
Data	02/02/2024
Destinatario	C. Gravino
Presentato da	Andrea Gisolfi, Giacomo Favale, Giovanni Nigro, Antonio Merola
Approvato da	

Team Composition

Ruolo	Nome	Acronimo	Contatti
Team Member	Andrea Gisolfi	AG	a.gisolfi4@studenti.unisa.it
Team Member	Giacomo Favale	GF	g.favale1@studenti.unisa.it
Team Member	Giovanni Nigro	GN	g.nigro32@studenti.unisa.it
Team Member	Antonio Merola	AM	a.merola29@studenti.unisa.it

Sommario

Revision History

Data	Versione	Descrizione	Autori
29/12/2023	0.1	Prima stesura	AG, GF, GN, AM
29/12/2023	0.2	Stesura paragrafo 2	AG
20/01/2024	0.3	Implementazione interfacce delle classi	AG
27/01/2024	0.4	Stesura paragrafo 5	AG, GN, GM, AM
28/01/2024	0.5	Revisione Finale	AG, GN, GM, AM

1. Introduzione

Lo scopo del sistema MediCare è quello di fornire una gestione adeguata delle prenotazioni, ma non solo; l'obiettivo è infatti quello di fornire un sistema efficiente ed intuitivo che sia in grado di fornire utilità sia ai più giovani che ai più anziani.
In questa sezione del presente documento vengono presentate le linee guida per la scrittura del codice, elencate le definizioni, gli acronimi e le varie abbreviazioni utilizzate in tutto il documento, ed infine i riferimenti e link utili.

1.1 Linee Guida per la Scrittura del Codice

In questa sezione vengono mostrate le regole da rispettare durante la progettazione e l'implementazione del sistema. Vi è riportata di seguito una lista delle documentazioni e convenzioni ufficiali usate durante la stesura del codice, con i relativi link ufficiali:

- **javadoc:**https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html
- **JavaFX:** https://openjfx.io/openjfx-docs/
- **MySQL:** https://dev.mysql.com/doc/refman/8.0/en/

1.2 Definizioni, acronimi e abbreviazioni

Sono riportate di seguito alcune definizioni presenti nel documento:

- **Package**: raggruppamento di classi, interfacce o file correlati
- **Design Pattern**: template di soluzioni a problemi "ricorrenti", usati per ottenere un alto grado di riuso ed usabilità
- **Application Layer** – (pattern Three Layer): sezione che si occupa della logica di business dell'applicazione
- **Presentation Layer** – (pattern Three Layer): sezione che si occupa della logica di visualizzazione dell'applicazione
- **Storage Layer** – (pattern Three Layer): sezione che si occupa dell'interazione con il database

- **Interfaccia**: insieme di signature delle operazioni offerte dalla classe
- **javadoc**: sistema di documentazione offerto da Java, generato sottoforma di interfaccia, in modo da rendere la documentazione accessibile e facilmente leggibile

1.3 Riferimenti utili

Qui di seguito sono riportati riferimenti ad altri documenti utili durante la lettura di questo documento:

- Requirement Analysis Document
- Statement of Work
- System Design Document
- Test Plan
- Test Case Specification
- Matrice di tracciabilità

2. Packages

In questa sezione del documento viene mostrata la suddivisione del nostro sistema in package, partendo da quanto definito all'interno dell'SDD (System Design Document).
Questa suddivisione è nata da alcune scelte architetturali e riprende la struttura standard definita da **JavaFX**.

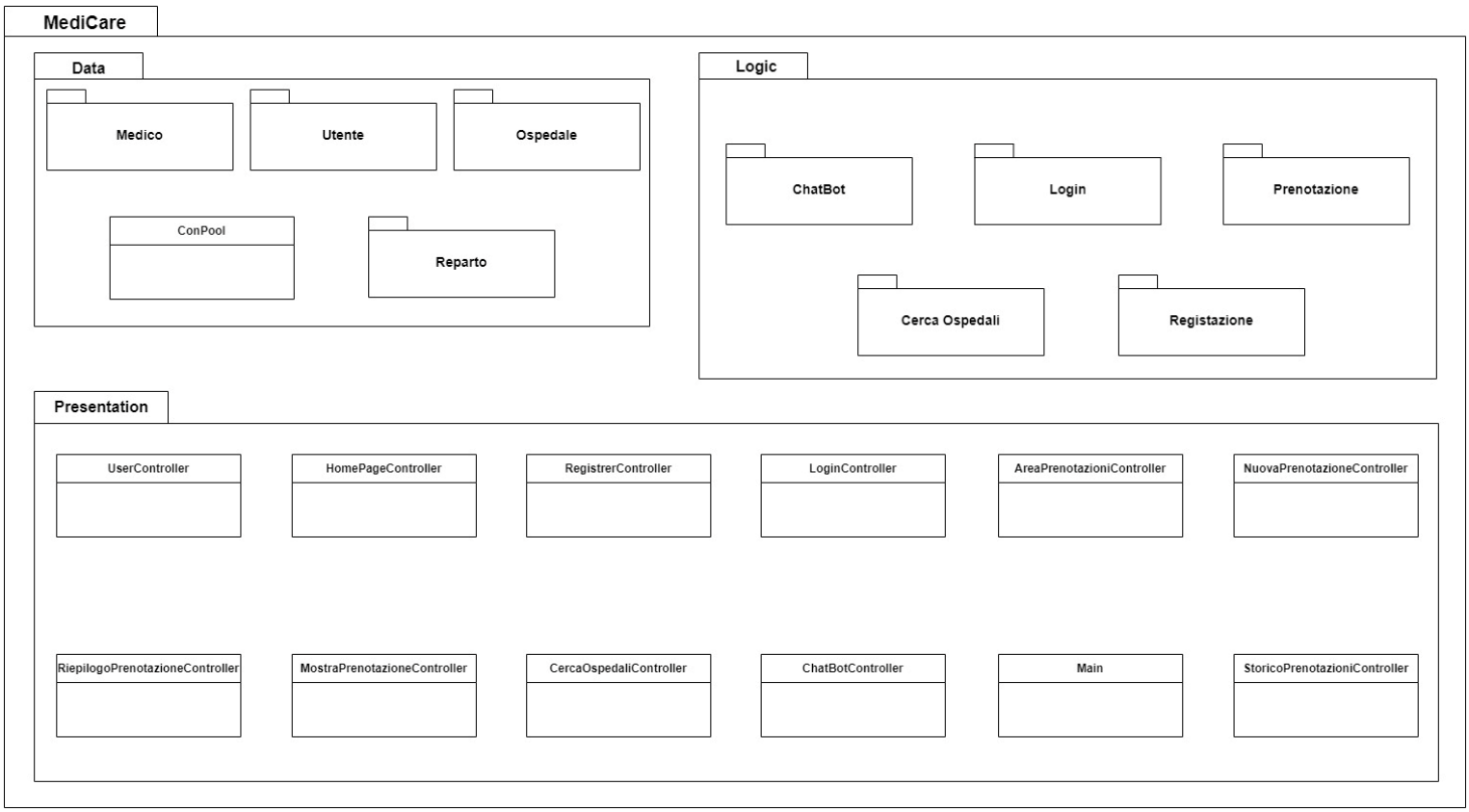
- **.idea**
- **.mvn**: contiene tutti i file di configurazione per Maven
- **src**, contiene i file sorgenti
    - **main**
        - **java**, contiene le classi Java relative alle componenti Logic, Data e Presentation
            - **test**, contiene le classi Java necessarie per l'implementazione del testing

        - **resources**, contiene i file necessari per l'utilizzo di JavaFX
    - **target**, contiene tutti i file prodotti dal sistema di build di Maven

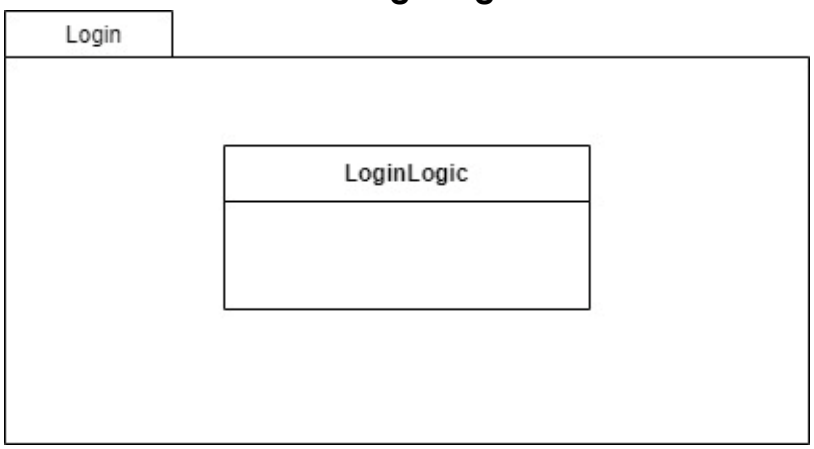
Package di MediCare

La seguente sezione riporta la struttura dei package di MediCare. La struttura prevede 3 package:

1. **package Logic**: contiene i sottosistemi per gestire l'accesso, la registrazione, la gestione delle prenotazioni, la ricerca degli ospedali, il chatbot.
2. **package Data**: contiene le classi Entity e i DAO per l'accesso al Database
3. **package Presentation**: contiene le classi utili per l'implementazione e il controllo dell'interfaccia dell'applicazione

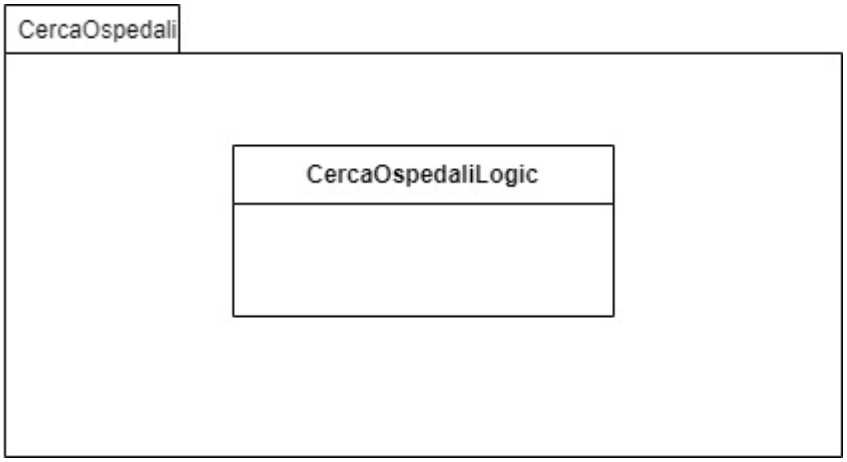


Package Login

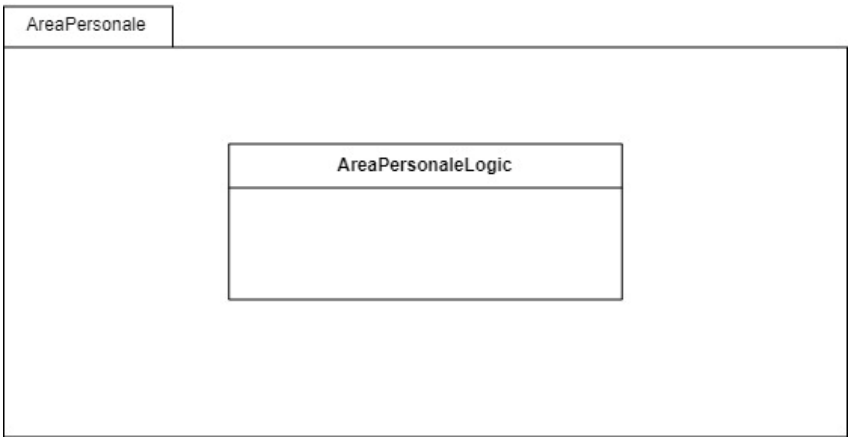


Package Registrazione

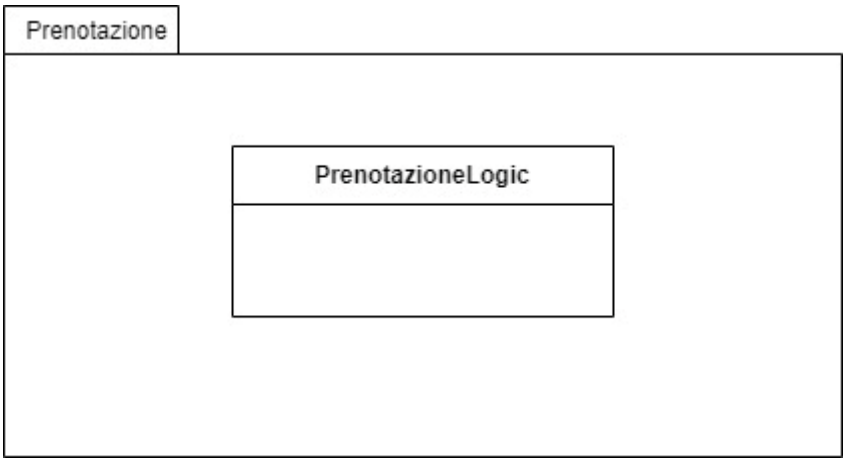
Package CercaOspedali



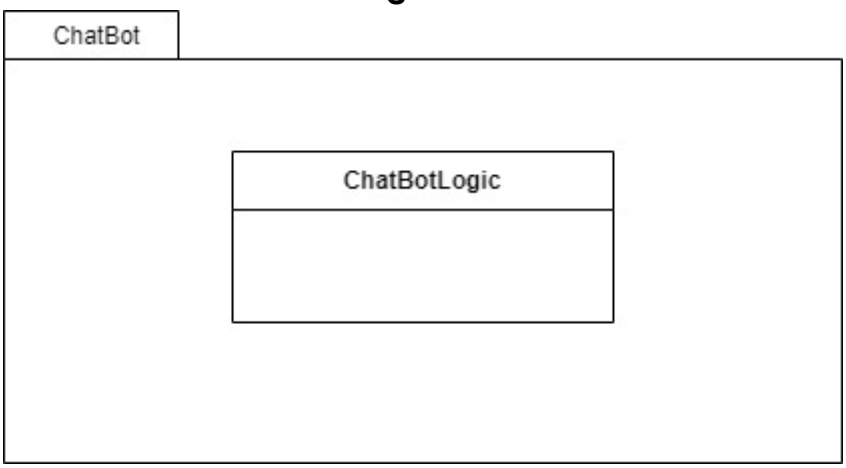
Package AreaPersonale



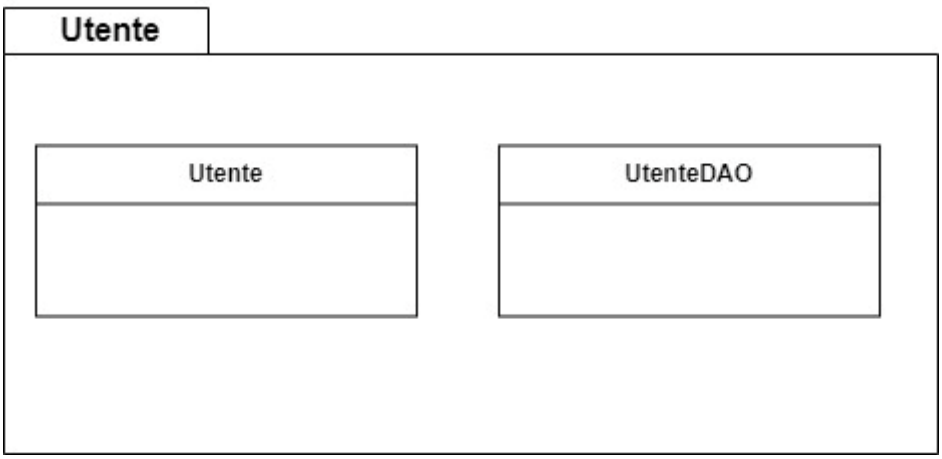
Package Prenotazione



Package ChatBot

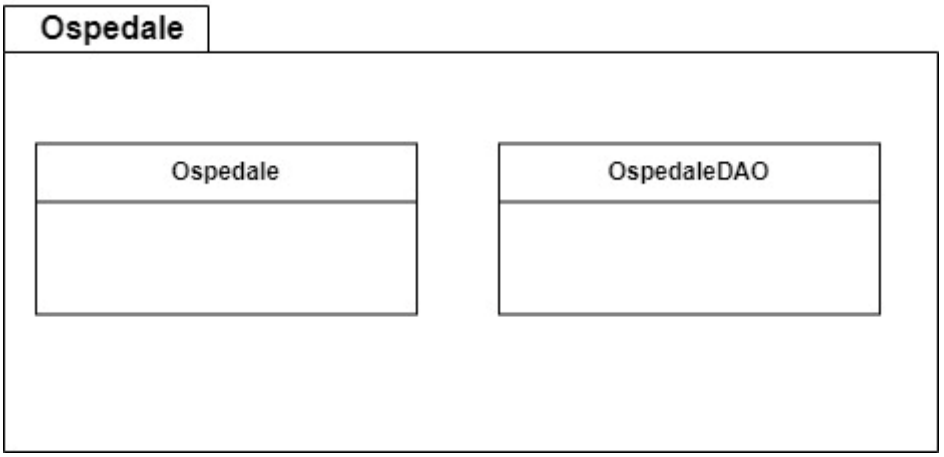


Package Utente

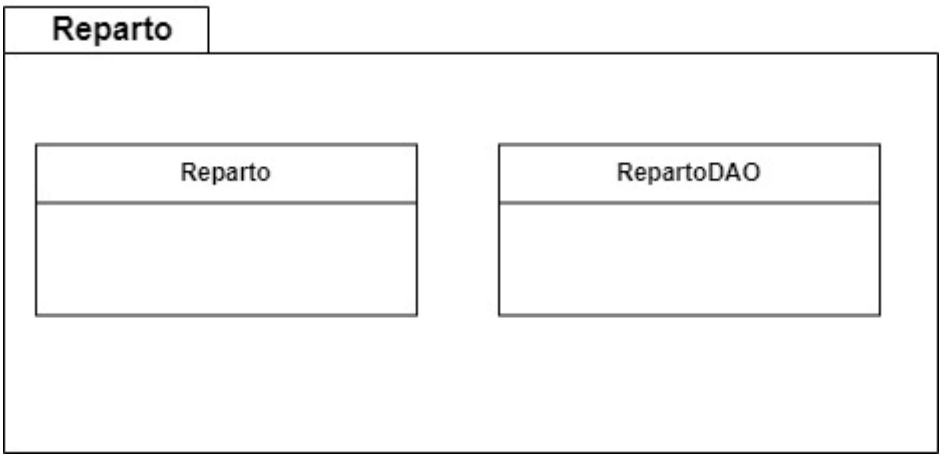


Package Medico

Package Ospedale



Package Reparto



3. Class Interfaces

JavaDoc

Di seguito viene riportato il link ad una pagina GitHub pages, la quale contiene la JavaDoc di MediCare.
Il link in questione è il seguente: https://gisolfi02.github.io/

Vengono riportate di seguito alcune interfacce del sistema, in particolare quelle relative al package **Logic**.

1. **Package Login**

LogicTier.MediCare.Login

Nome Classe	**LoginLogic**
Descrizione	La seguente classe si occupa della logica del login da parte di un utente.
Metodi	+ doLogin(String email, String password): Utente
Invariante di classe	/

Nome Metodo	**+ doLogin(String email, String password)**
Descrizione	Il seguente metodo consente di effettuare il login da parte dell'utente.
Pre-condizione	email != null && password != null
Post-condizione	Utente != null

2. **Package Registrazione**

LogicTier.MediCare.Registrazione

Nome Classe	**RegistazioneLogic**
Descrizione	La seguente classe si occupa della logica della registrazione da parte di un utente.
Metodi	+ doRegistrazione(String nome, String cognome, String email, String password, String telefono, LocalDate dataDiNascita): int

Invariante di classe	/

Nome Metodo	+ doRegistrazione(String nome, String cognome, String email, String password, String telefono, LocalDate dataDiNascita)
Descrizione	Il seguente metodo si occupa di effettuare la registrazione dell'utente.
Pre-condizione	nome.matches("([A-Z][a-z]+)") && cognome.matches("([A-Z][a-z']+)") && email.matches("^[A-Za-z0-9+_.-]+@(.+)$") && password.matches("^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-.]).{8,}$") && dataDiNascita == *null* && dataDiNascita >= 18 anni && telefono.matches("[0-9]{10}")
Post-condizione	/

3. Package Prenotazione

LogicTier.MediCare.Prenotazione

Nome Classe	PrenotazioneLogic
Descrizione	La seguente classe si occupa della logica di tutti il sottosistema di prenotazioni.
Metodi	+ getMedico(*int* id): Medico + getMedici(String reparto, String ospedale): ArrayList<String> + getReparti(String nomeOspedale): ArrayList<String> + getOspedale(*int* id): Ospedale + getOspedali(String comune): ArrayList<String> + getComuni():ArrayList<String>

	+ getOre(LocalDate data, String medico): ArrayList<String> + doPrenotazione(String nome, String cognome, String cf, LocalDate data, String ora, String emailUtente, String medico, String ospedale): *int* + *salvaPrenotazione(Prenotazione prenotazione): boolean* + *getPrenotazione():Prenotazione* + *getPrenotazioni(Utente u): ArrayList<Prenotazione>* + *eliminaPrenotazione(int codice): void* + *salvaModifiche(int codice, LocalDate data, String ora): int*
Invariante di classe	/

Nome Metodo	**+ getMedico(*int* id)**
Descrizione	*Il seguente metodo restituisce un medico in base al suo id.*
Pre-condizione	/
Post-condizione	/
Nome Metodo	**+ getMedici(String reparto, String ospedale)**
Descrizione	*Il seguente metodo restituisce la lista dei nomi dei medici che appartengono a un deteterminato reparto di un determinato ospedale.*
Pre-condizione	/
Post-condizione	/
Nome Metodo	**+ getReparti(String nomeOspedale)**
Descrizione	*Il seguente metodo restituisce la lista dei reparti di un determinato ospedale.*

Pre-condizione	/
Post-condizione	/
Nome Metodo	**+ getOspedale(*int* id)**
Descrizione	*Il seguente metodo ritorna un ospedale in base al suo id.*
Pre-condizione	/
Post-condizione	/
Nome Metodo	**+ getOspedali(String comune)**
Descrizione	*Il seguente metodo restituisce la lista dei nomi degli ospedali in un determinato comune.*
Pre-condizione	/
Post-condizione	/
Nome Metodo	**+ getComuni()**
Descrizione	*Il seguente metodo restituisce la lsita dei comuni presenti nel database.*
Pre-condizione	/
Post-condizione	/
Nome Metodo	**+ getOre(LocalDate data, String medico)**
Descrizione	*Il seguente metodo restituisce le ore disponibili per la prenotazione per un determinato medico in un determinato giorno.*
Pre-condizione	/
Post-condizione	/
Nome Metodo	**+ doPrenotazione(String nome, String cognome, String cf, LocalDate data, String ora, String emailUtente, String medico, String ospedale)**

Descrizione	*Il seguente metodo si occupa di effettuare la prenotazione di un paziente per un determinato medico.*
Pre-condizione	nome.matches("([A-Z][a-z]+)") && cognome.matches("([A-Z][a-z']+)") && cf.matches("[A-Z]{6}\\d{2}[A-Z]\\d{2}[A-Z]\\d{3}[A-Z]") && data == *null* && data > data odierna && ora == *null*
Post-condizione	/
Nome Metodo	***+ salvaPrenotazione(Prenotazione prenotazione)***
Descrizione	*Il seguente metodo si occupa di memorizzare la prenotazione.*
Pre-condizione	/
Post-condizione	prenotazioneDAO.doSave(prenotazione)!= null
Nome Metodo	***+ getPrenotazione()***
Descrizione	*Il seguente metodo restituisce la prenotazione appena effettuata.*
Pre-condizione	/
Post-condizione	/
Nome Metodo	***+ getPrenotazioni(Utente u)***
Descrizione	*Il seguente metodo restituisce la lista delle prenotazioni effettuate da un utente specifico.*
Pre-condizione	/
Post-condizione	/
Nome Metodo	***+ eliminaPrenotazione(int codice)***

Descrizione	*Il seguente metodo effettua l'eliminazione di una prenotazione.*
Pre-condizione	/
Post-condizione	/
Nome Metodo	***+ salvaModifiche(int codice, LocalDate data, String ora)***
Descrizione	*Il seguente metodo effettua la modifica della prenotazione.*
Pre-condizione	data != null && data > data odierna && ora != null && ora != empty
Post-condizione	/

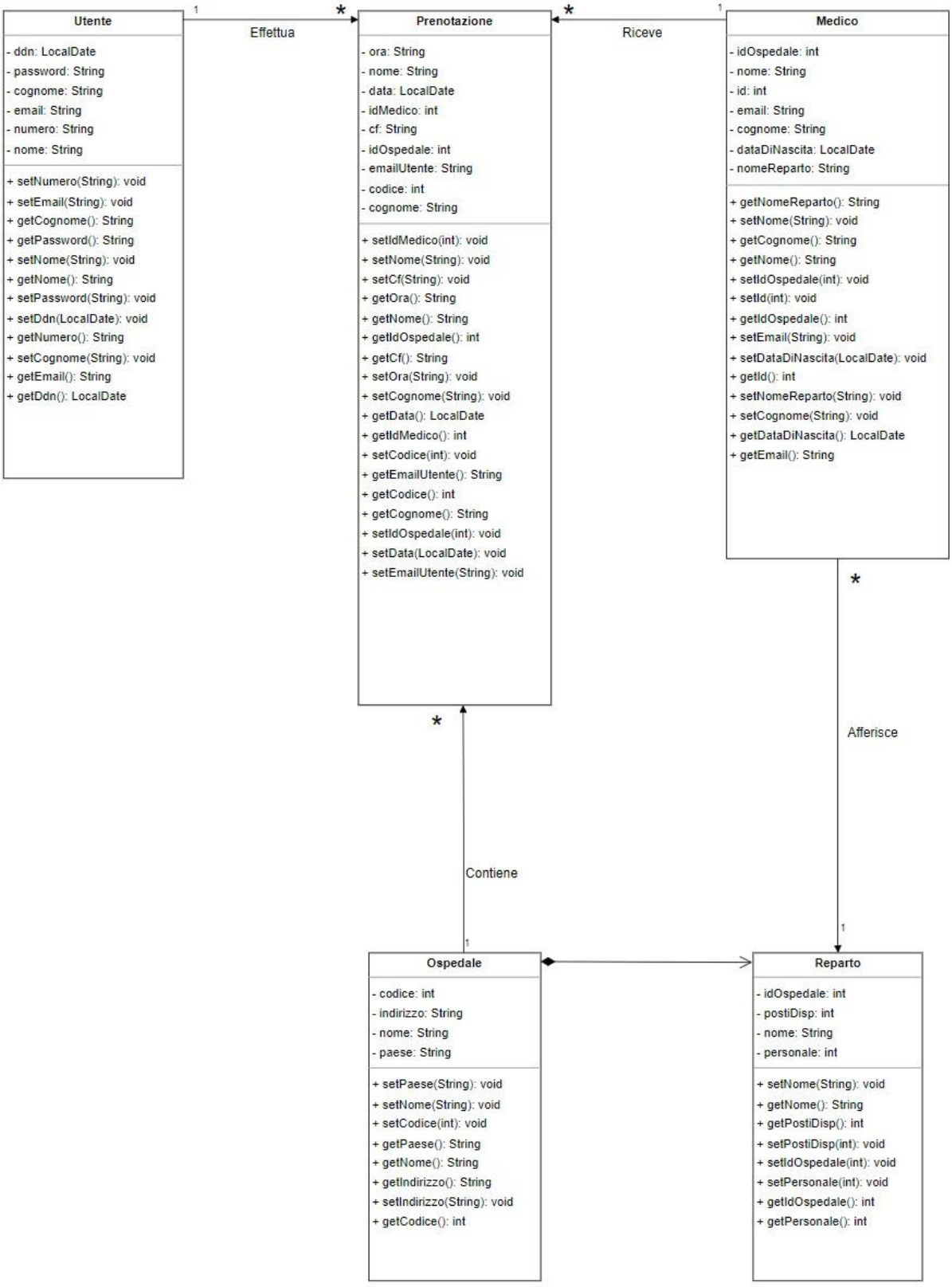
4. Package CercaOspedali

LogicTier.MediCare.CercaOspedali

Nome Classe	**CercaOspedaliLogic**
Descrizione	*La seguente classe si occupa della logica di ricerca degli ospedali.*
Metodi	+ cercaOspedali(String ricerca): ArrayList<String>
Invariante di classe	/

Nome Metodo	**+ cercaOspedali(String ricerca)**
Descrizione	*Il seguente metodo che di cercare gli ospedali presenti nella località inserita dall'utente.*
Pre-condizione	ricerca != empty
Post-condizione	/

4. Class Diagram Ristrutturato



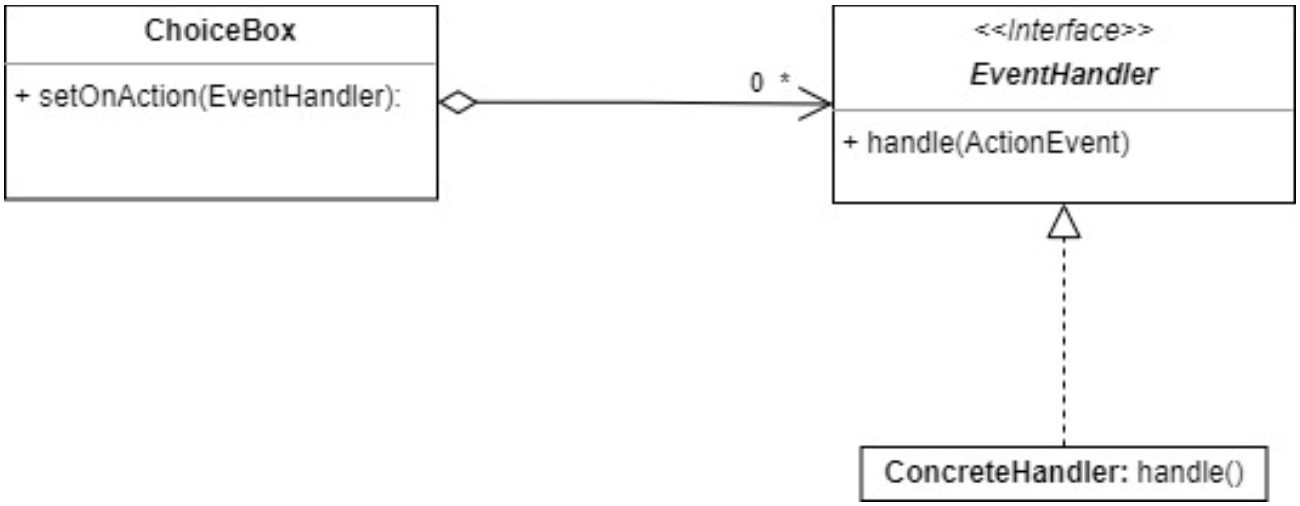
5. Elementi di Riuso

5.1 Design Pattern usati

Nella seguente sezione riportiamo il design pattern utilizzato per la realizzazione del progetto.

Observer

Durante lo sviluppo abbiamo riscontrato la necessità di effettuare aggiornamenti di porzioni grafiche a schermo dinamicamente, per tale motivo abbiamo ritenuto opportuno l'utilizzo del Design Pattern Observer.
In particolare, nella schermata in cui viene effettuata la prenotazione, gli stati dei ChoiceBox vengono aggiornati sequenzialmente quando viene selezionato un elemento all'interno di uno di essi. Questo avviene tramite l'interfaccia EventHandler con il metod handle(), opportunamente implementato tramite @Override.



5.2 Componenti COTS

Per la realizzazione dell'interfaccia del sistema abbiamo utilizzato l'applicazione SceneBuilder. SceneBuilder è un visual layout tool che permette di sviluppare rapidamente l'interfaccia di un applicazione JavaFX, senza codificare.

6. Glossario

Sigla/termine	Definizione
Package	Raggruppamento di classi ed interfacce.
DAO	Data Access Object. Pattern architetturale mirato alla gestione della persistenza dei dati.
Three-layer	Modello di organizzazione del codice applicativo basato sulla separazione delle funzionalità logiche del sistema
Observer	L'Observer pattern è un design pattern utilizzato come base architetturale di molti sistemi di gestione di eventi.
Componenti COTS	Con il termine componenti COTS ci si riferisce a componenti hardware e software disponibili sul mercato per l'acquisto da parte di aziende di sviluppo interessate a utilizzarli nei loro progetti (https://www.wikiwand.com/it/COTS).