Università degli Studi di Salerno

Corso di Ingegneria del Software

CheckUp –
OBJECT DESIGN DOCUMENT
V1.0

Data 08/02/2021

Partecipanti:

Cognome e Nome	*Matricola*
Tamburello Martina	0512105452

Revision History:

Data	*Versione*	*Cambiamenti*	*Autori*
08/01/2021	1.0	Stesura documento	Tamburello Martina
12/01/2021	1.1	Correzione package model	Tamburello Martina

Sommario

1. Introduzione

1.1 Object Design Trade-Off

Durante la fase di analisi e di progettazione del sistema sono stati individuati diversi compromessi per lo sviluppo del sistema. Anche durante la fase di Object Design sorgono diversi compromessi che saranno analizzati in questo paragrafo:

Memoria / Estensibilità: Il sistema garantisce un'alta modularità come specificato dai requisiti non funzionali (CheckUp_RAD_V1.5) e dai trade off individuati nella fase di System Design (CheckUp_SDD_V1.3). Il sistema deve quindi permettere l'estensibilità a discapito della memoria utilizzata.

Affidabilità / Tempo di risposta: Il sistema deve sempre garantire una risposta affidabile e consistente, a discapito del tempo di risposta.

Criteri di manutenzione / Criteri di performance: Il sistema sarà implementato preferendo la manutenibilità alla performance in modo da facilitare gli sviluppatori nel processo di aggiornamento del software.

Costi / Sicurezza: Per evitare una situazione di over-budget, il sistema garantisce il minimo livello di sicurezza sufficiente.

Nella seguente tabella, il Design Goal in **grassetto** indica il design goal prioritario.

Trade-Off

Memoria	Estensibilità
Affidabilità	Tempo di risposta
Criteri di manutenzione	Criteri di performance
Costi	Sicurezza

1.2 Componenti Off-the-Shelf

Saranno utilizzate componenti Off-the-Shelf nell'implementazione del sistema:

Per quanto riguarda il Front End, verrà utilizzato il framework Bootstrap (https://getbootstrap.com/docs/4.4/getting-started/introduction/) per una gestione dello stile più semplice. Questo framework è open source e integra sistemi per gestire in maniera semplificata il codice HTML e CSS.

Dal punto di vista funzionale invece, verrà usato JQuery, un framework di JavaScript che permette di agire sul DOM del documento HTML, di effettuare chiamate AJAX e di gestire le animazioni della pagina web in maniera molto più semplice.

Per quanto riguarda il lato Back End, sarà utilizzato il WebServer Tomcat.

1.3 Design Pattern

Il sistema utilizzerà 2 Design Pattern:

- **Singleton**
  Il singleton è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.
  La classe DBConnection, che si occupa di creare e mantenere una connessione al database, è una classe singleton, così che possa essere acceduta in maniera atomica, senza creare molteplici connessioni.

- **DAO (Data Access Object)**

Il modello DAO viene utilizzato per separare l'API o le operazioni di accesso ai dati di basso livello dai servizi aziendali di alto livello. Le componenti del Data Access Object Pattern sono:

- **Data Access Object Interface:** interfaccia che definisce le operazioni standard da eseguire su uno o più oggetti del modello.
- **Data Access Object concrete class**: classe implementa l'interfaccia precedente. Essa ottiene i dati dal meccanismo di archiviazione utilizzato, nel caso specifico, un database.

1.4 Linee guida per la documentazione delle interfacce

In questo paragrafo verranno definite le linee guida a cui un programmatore deve attenersi nell'implementazione del sistema.

In ciascun sotto-paragrafo, quando si parla di indentazione ci si riferisce ad una tabulazione.

1.4.1 Classi e Interfacce Java

Lo standard nella definizione delle classi e delle interfacce Java è quello definito da Google (http://google.github.io/styleguide/javaguide.html).
Ciascuna classe e ciascun metodo deve essere documentato seguendo lo stile di documentazione JavaDoc.

1.4.2 Java Servlet Pages (JSP)

Le JSP costruiscono pagine HTML in maniera dinamica che devono rispettare il formato definito nel sotto paragrafo sottostante. Devono anche attenersi alle linee guida per le classi Java definite nel sotto-paragrafo soprastante. Inoltre, le JSP devono attenersi alle seguenti puntualizzazioni:

1. Il tag di apertura (<%) è seguito immediatamente dalla fine della riga;
2. Il tag di chiusura (%>) si trova all'inizio di una riga che non contiene nient'altro;
3. Istruzioni Java che consistono di una sola riga possono contenere il tag di apertura e chiusura sulla stessa riga;

1.4.3 Pagine HTML

Le pagine HTML devono essere conformi allo standard HTML5. Inoltre, il codice HTML deve utilizzare l'indentazione per facilitare la lettura e di conseguenza la manutenzione. Le regole di indentazione sono le seguenti:

1. Ogni tag deve avere un'indentazione maggiore del tag che lo contiene;
2. Ogni tag di chiusura deve avere lo stesso livello di indentazione di quello di apertura;

1.4.4 File JavaScript

Gli script JavaScript seguono anche essi le linee guida definite dallo standard Google.
Sia i file che i metodi JavaScript devono essere documentati seguendo lo stile di JavaDoc.

1.4.5 Fogli di stile CSS

Ogni foglio di stile deve essere inizializzato da un commento analogo a quello presente nei file Java.

Ogni regola CSS deve essere formattata nel seguente modo:

1. I selettori della regola si trovano a livello 0 di indentazione, uno per riga;
2. L'ultimo selettore della regola è seguito da parentesi graffa aperta ({);
3. Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai selettori;
4. La regola è terminata da una parentesi graffa chiusa (}), collocata da sola su una riga;

1.4.6 Script SQL

Le istruzioni e le clausole SQL devono essere costituite da sole lettere maiuscole.

I nomi delle tabelle sono costituiti da sostantivi singolari, scritti in minuscolo.

I nomi degli attributi devono essere costituiti da sole lettere minuscole e possono contenere più parole separate da un underscore (_).

1.5 Definizioni, acronimi, abbreviazioni

Definizioni

DOM = Modella la struttura di pagine web.

Acronimi

RAD= Requirements Analysis Document

SDD = System Design Document

ODD = Object Design Document

GUI = Graphical User Interface (Interfaccia grafica utente)

HTML = HyperText Markup Language

CSS = Cascading Style Sheet

DOM = Document Object Model

AJAX = Asynchronous JavaScript And XML

1.6 Riferimenti

Libro:

-- Object-Oriented Software Engineering (Using UML, Pattern, and Java) Third Edition

Autori:

-- Bernd Bruegge

-- Allen H. Dutoit

Documenti:

-- CheckUp_RAD_V1.5.pdf – Requirements Analysis Document

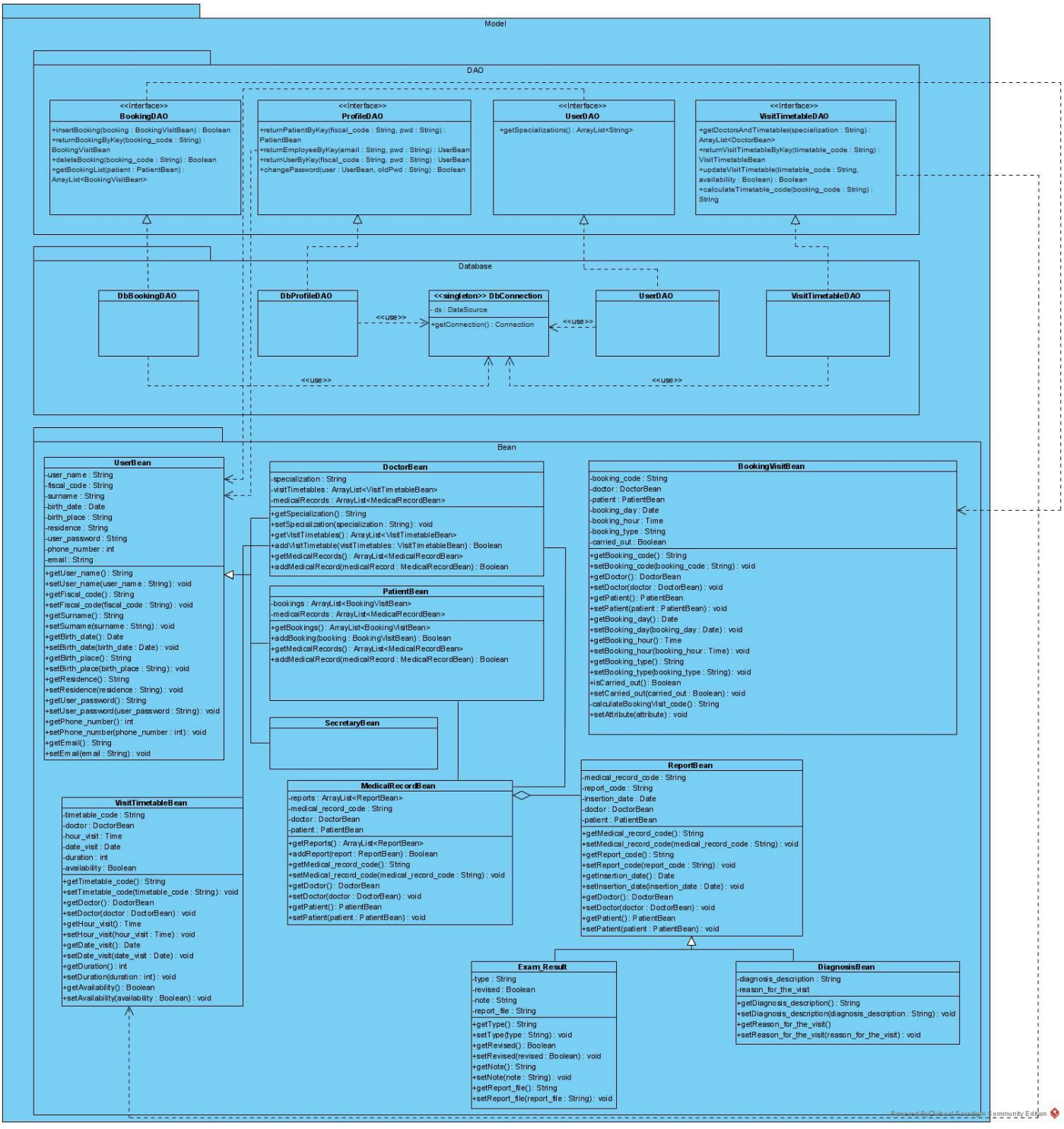
-- CheckUp_SDD_V1.3.pdf – System Design Document

2. Packages

2.1 Model

Il package Model contiene le classi che si occupano di modellare la logica di business del sistema. È suddiviso in 3 sottopackage:

- **Bean**
  Contiene le classi che modellano gli oggetti del dominio applicativo. Ciascuna classe definisce dei metodi get e set.
- **Dao**
  Contiene le interfacce dei Data Access Object (DAO), ovvero oggetti che si relazionano con il gestore della persistenza per effettuare operazioni di tipo CRUD (Create, Read, Update, Delete)
- **Database**
  Contiene le implementazioni delle interfacce definite nel package Dao e contiene una classe DbConnection attraverso la quale avviene la connessione al database.

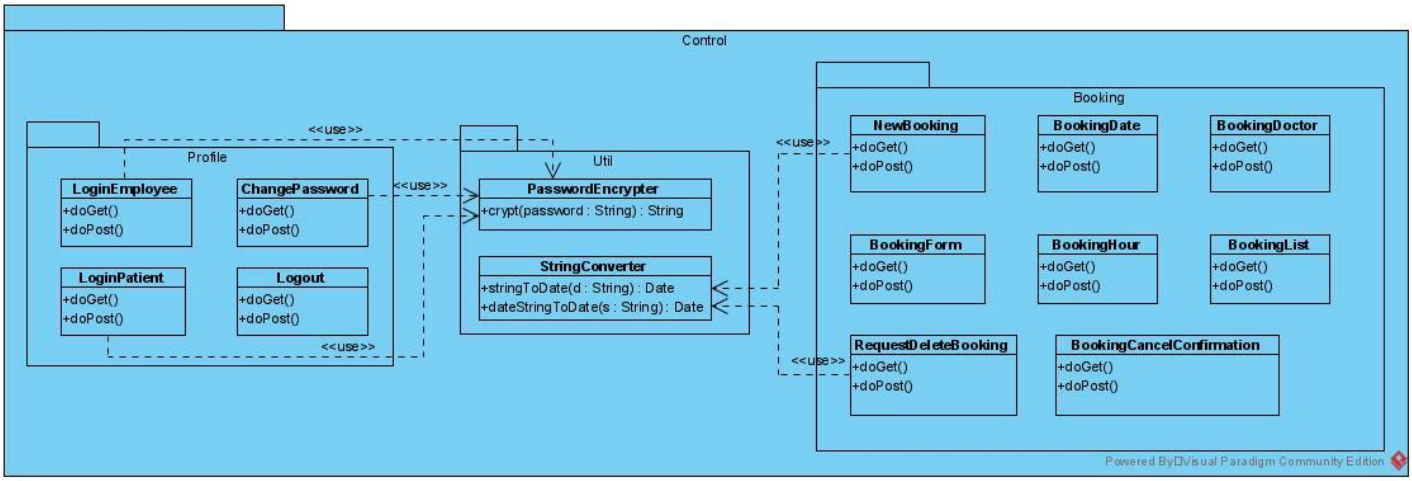


2.2 Control

Il package Control contiene le classi che gestiscono le richieste del client. Tutti i sottopackage contengono classi che estendono la classe **HttpServlet**, ad eccezione del package **Util** che invece contiene classi per effettuare operazioni ricorrenti.

Sottopackage:

- **Profile:**
  Contiene le servlet per effettuare le operazioni di login, logout, modifica password;
- **Booking:**
  Contiene le servlet per gestire le richieste riguardanti le prenotazioni;
- **Util:**
  Contiene classi che fungono d'ausilio alle precedenti.

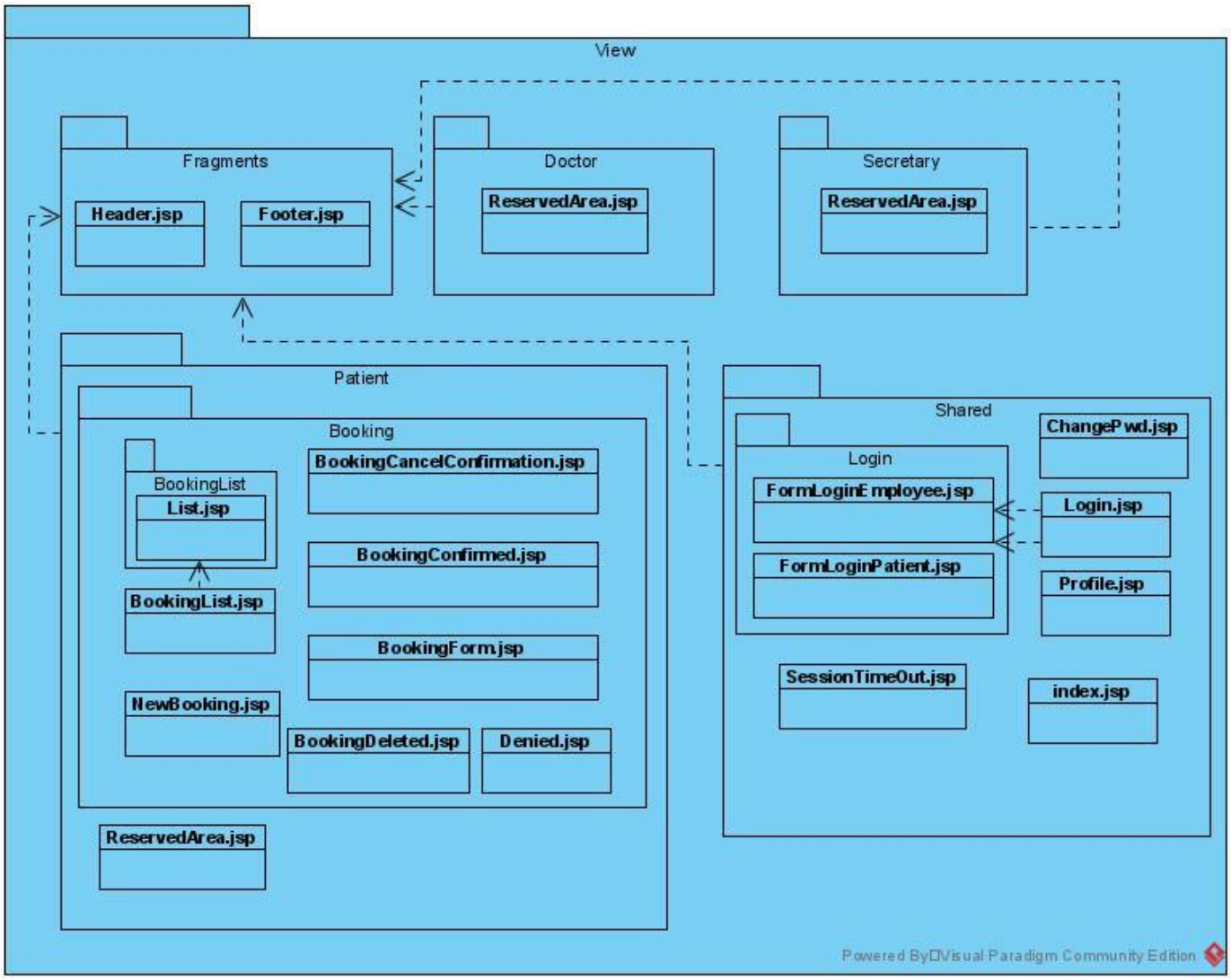


2.3 View

Il package view contiene tutte le pagine JSP tramite le quali gli utenti interagiscono con il sistema.

Il package si divide in diversi sottopackage:

- **Patient**
  Contiene le pagine JSP attraverso le quali il paziente comunica con il sistema;
- **Doctor**
  Contiene le pagine JSP attraverso le quali il medico comunica con il sistema;
- **Secretary**
  Contiene le pagine JSP attraverso le quali il segretario comunica con il sistema;
- **Shared**
  Contiene pagine JSP che vengono utilizzati da più attori del sistema per effettuare le operazioni più comuni.
- **Fragments**
  Contiene frammenti di pagine JSP che sono ricorrenti in tutte le altre pagine.



3. Interfacce delle Classi

Il seguente capitolo fornirà le interfacce delle classi del package "**View**".

Nome classe	BookingDAO
Descrizione	Classe che definisce i metodi dell'entità BookingVisitBean (Prenotazione).
Attributi e metodi	+insertBooking (booking : BookingVisitBean) : Boolean +returnBookingByKey(booking_code : String) : BookingVisitBean +deleteBooking(booking_code : String) : Boolean +getBookingList(patient : PatientBean) : ArrayList<BookingVisitBean>
Pre-condizione	**Context** BookingDAO:: insertBooking (booking : BookingVisitBean): Boolean **Pre** booking != null **Context** BookingDAO:: returnBookingByKey(booking_code : String) : BookingVisitBean **Pre** booking_code != null **Context** BookingDAO:: deleteBooking(booking_code : String) : Boolean **Pre** booking_code != null **Context** BookingDAO:: getBookingList(patient : PatientBean) : ArrayList<BookingVisitBean> **Pre** patient != null
Post-condizione	
Invariante	

Nome classe	UserDAO
Descrizione	Classe che definisce i metodi dell'entità UserBean e di quelle che da essa ereditano.
Attributi e metodi	+getSpecializations() : ArrayList<String>
Pre-condizione	**Context** UserDAO:: getSpecializations() : ArrayList<String>
Post-condizione	
Invariante	

Nome classe	ProfileDAO
Descrizione	Classe che definisce i metodi dell'entità UserBean e di quelle che da essa ereditano.
Attributi e metodi	+returnPatientByKey(fiscal_code : String, pwd : String) : PatientBean +returnEmployeeByKey(email : String, pwd : String) : UserBean +returnUserByKey(fiscal_code : String, pwd : String) : UserBean +changePassword(user : UserBean, oldPwd : String)
Pre-condizione	**Context** ProfileDAO:: returnPatientByKey(fiscal_code : String, pwd : String) : PatientBean **Pre** fiscal_code != null && pwd != null **Context** ProfileDAO:: returnEmployeeByKey(email : String, pwd : String) : UserBean **Pre** email != null && pwd != null **Context** ProfileDAO:: returnUserByKey(fiscal_code : String, pwd : String) : UserBean **Pre** fiscal_code != null && pwd != null **Context** ProfileDAO:: changePassword(user : UserBean, oldPwd : String) **Pre** user != null && oldPwd != null
Post-condizione	
Invariante	

Nome classe	VisitTimetableDAO
Descrizione	Classe che definisce i metodi dell'entità VisitTimetableBean.
Attributi e metodi	+getDoctorsAndTimetables(specialization : String) : ArrayList<DoctorBean> +returnVisitTimetableByKey(timetable_code : String) : VisitTimetableBean

	+updateVisitTimetable(timetable_code : String, availability : Boolean) : Boolean +calculateTimetable_code(booking_code : String) : String
Pre-condizione	**Context** VisitTimetableDAO:: getDoctorsAndTimetables(specialization : String) : ArrayList<DoctorBean> **Pre** specialization != null **Context** returnVisitTimetableByKey(timetable_code : String) : VisitTimetableBean **Pre** timetable_code != null **Context** VisitTimetableDAO:: updateVisitTimetable(timetable_code : String, availability : Boolean) : Boolean **Pre** timetable_code != null **Context** VisitTimetableDAO:: calculateTimetable_code(booking_code : String) : String **Pre** booking_code != null
Post-condizione	
Invariante	