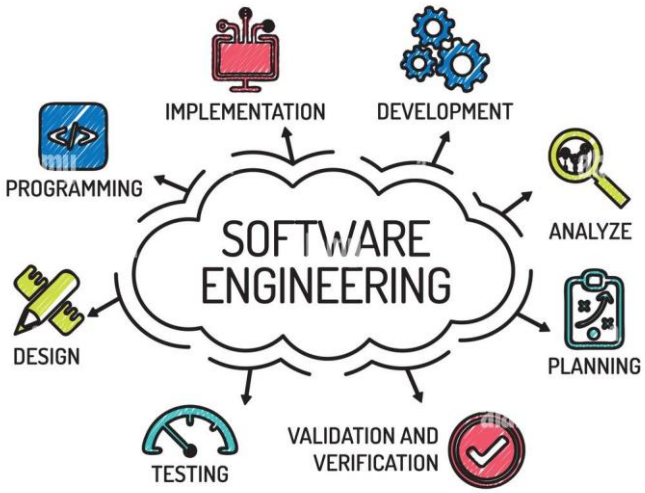
Università degli Studi di Palermo

Ingegneria del Software (9 CFU)

Object Design Document (ODD)

Progettazione di un software per il supporto ad un'azienda di servizi al cittadino



Un progetto realizzato da:

Bova Gabriele
Carini Riccardo
Cimino Giuseppe
La Rosa Mazza Nicolò

CdL: Ingegneria Informatica LT
Anno Accademico: 2022/2023

Sommario

1. Introduction 3
1.1 Object design trade-offs 3
1.2 Interface documentation guidelines 3
1.3 Definitions, acronyms, and abbreviations 3
1.4 References 4
2. Packages 4
2.1 it.unipa.ingegneriasoftware 5
2.2 java 10
2.3 javax 12
2.4 com.itextpdf.text 12
2.5 com.toedter 12
3. Class Interfaces 12
4. Glossary 20

1. Introduction

L'Object Design Document (ODD) è un documento che descrive come tutte le scelte fatte in fase di analisi e in fase di progettazione del sistema si concretizzano, talvolta con qualche piccola variazione, a livello di codice. Include dettagli generali sul codice stesso (scelte progettuali e di codifica) e, attaverso opportuni diagrammi, informazioni sull'organizzazione del codice in package e una panoramica di tutte le classi (attributi e metodi).

1.1 Object design trade-offs

È stato scelto di usare preferibilmente classi già presenti nella libreria standard di Java, ampiamente documentate e facilmente riutilizzabili.

1.2 Interface documentation guidelines

Durante la scrittura del codice sono state rispettate alcune convenzioni. In particolare:

- Nomi significativi per classi, attributi e metodi.
- Classi con nomi al singolare e iniziale maiuscola e uso della notazione a cammello.
- Opportuni sostantivi per le variabili e opportuni verbi per i metodi, tutti con iniziali minuscole.
- Package tutti in minuscolo per evitare potenziali conflitti di nome.

In generale tutti i file cercano di essere leggibili attraverso l'uso di spazi vuoti, indentazione opportuna, separazione di righe troppo lunghe su più linee e così via. Alcuni esempi di queste ultime convenzioni sono stati tratti dal sito ufficiale di Java nella sezione *Code Conventions for the Java Programming Language*. Le convenzioni di codice sono importanti per i programmatori perché quasi nessun software viene mantenuto per tutta la sua vita dall'autore originale, quindi le convenzioni migliorano la leggibilità del software, consentendo agli ingegneri di comprendere il nuovo codice in modo più rapido e approfondito. Infine tutto il codice sfrutta dove possibile i principi di incapsulamento, ereditarietà e polimorfismo.

1.3 Definitions, acronyms, and abbreviations

Riportiamo nella tabella di seguito definizioni, acronimi e abbreviazioni in modo da rendere il documento uniforme e meno ambiguo.

Termine	Significato
Utente	Generico individuo autenticato che utilizza il Sistema; può essere un Dipendente (di tipo 1, 2, 3 o 4) o il Datore Di Lavoro.
Dipendente	Utilizzatore autenticato dal Sistema che lavora nell'azienda; può essere di quattro tipi (1, 2, 3 o 4) a seconda del servizio assegnatogli e/o svolto.
Datore Di Lavoro	Utilizzatore autenticato dal Sistema che supervisiona il corretto funzionamento dell'azienda e dell'operato dei dipendenti. Abbreviato in "DDL".
Credenziali Login	Insieme di dati necessari all'Utente per autenticarsi e utilizzare il Sistema. Nello specifico sono: Matricola, Password.

Credenziali Entrata/Uscita	Insieme di dati necessari all'Utente per autenticarsi e segnalare la propria entrata/uscita durante un turno ben definito. Nello specifico sono: Matricola, Nome, Cognome. Entrata e uscita vengono normalmente segnalate da un Terminale presente all'ingresso dell'azienda. Tuttavia, l'entrata può essere segnalata dal Dipendente dal Software dell'azienda qualora egli sia in ritardo e il Terminale risulti quindi indisponibile.
Terminale	Dispositivo presente all'ingresso dell'azienda, attraverso il quale i dipendenti possono effettuare la rilevazione dell'entrata e dell'uscita dal turno previsto dello stesso.
Account	Insieme di dati che identificano un Utente autenticato all'interno del Sistema durante una sessione di utilizzo.
Stato del Dipendente Entrata	Fornisce informazioni sul Dipendente rispetto all'ingresso del turno che deve svolgere. Di seguito i possibili stati di un'entrata: Presente, Presente Ritardo, Assente.
Stato del Dipendente Uscita	Fornisce informazioni sul Dipendente rispetto all'uscita del turno che si è appena svolto. Di seguito i possibili stati di un'uscita: Manuale, Automatica.
DBMS (DataBase Management System)	Sistema che organizza in modo strutturato tutti i dati necessari al funzionamento dell'Azienda e fornisce una serie di funzionalità per creare, leggere, aggiornare e rimuovere i dati.

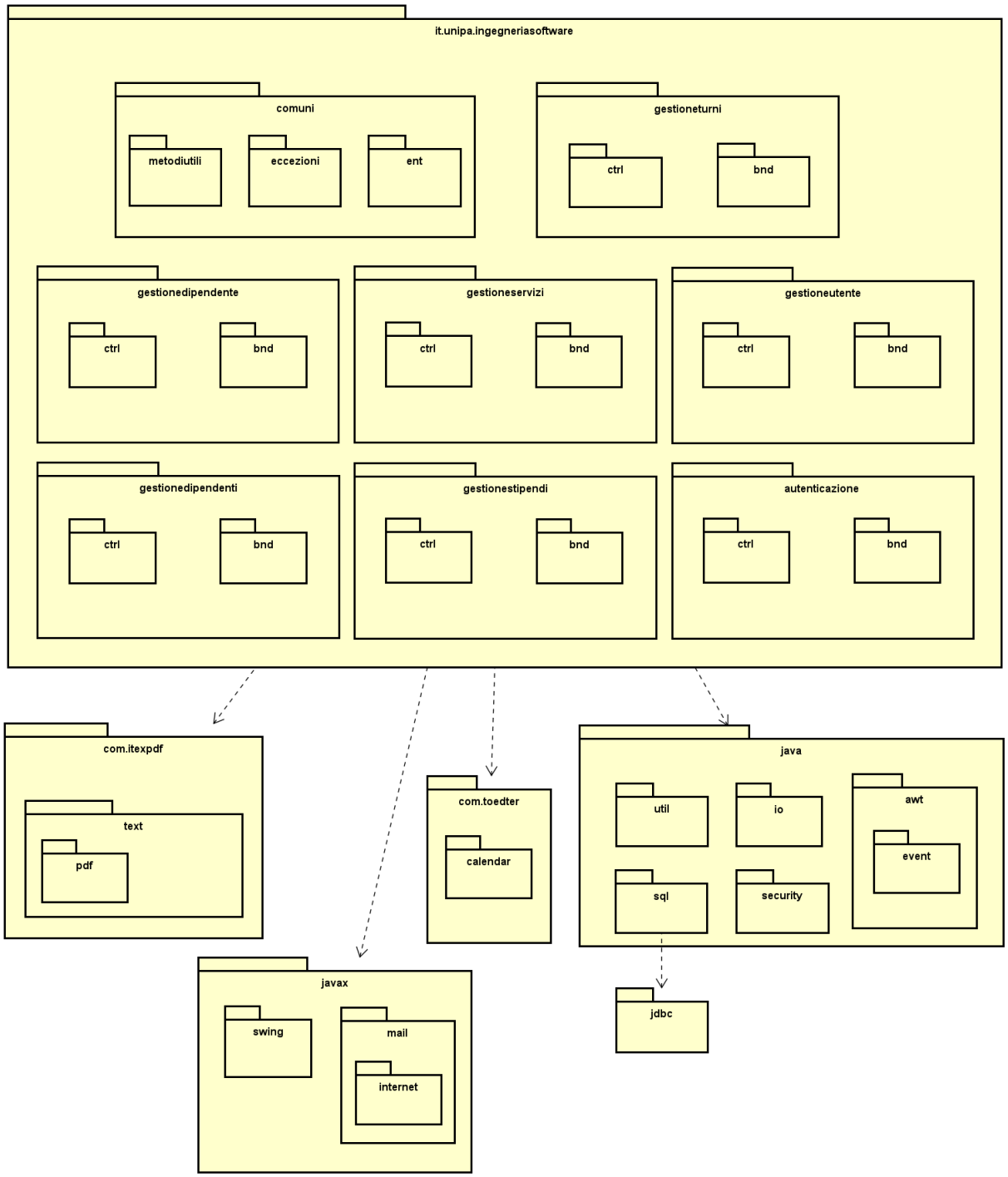
1.4 References

Fonti utili usate durante la scrittura del Object Design Document:

- *Object-Oriented Software Engineering: Using Uml, Patterns and Java* di Bernd Bruegge e Allen H. Dutoit.
- *Software Engineering* di Ian Sommerville.
- Java® Platform, Standard Edition & Java Development Kit Version 19 API Specification: https://docs.oracle.com/en/java/javase/19/docs/api/index.html.

2. Packages

Discutiamo di seguito tutti i dettagli dell'organizzazione dei file del codice.



2.1 it.unipa.ingegneriasoftware

Package principale dell'applicazione. Contiene le seguenti cartelle:

- autenticazione

- comuni
- gestionedipendente
- gestionedipendenti
- gestioneservizi
- gestionestipendi
- gestioneturni
- gestioneutente

autenticazione	
L'Autenticazione è il processo di verifica dell'identità di un utente che richiede l'accesso al sistema. Questo processo implica la verifica dell'identità di un utente attraverso meccanismi di controllo, tramite credenziali e metodi di identificazione a due fattori. In generale, l'autenticazione può essere suddivisa in tre fasi: 1. Identificazione: l'utente fornisce le proprie credenziali (matricola e password) al sistema. 2. Convalida: il sistema verifica le credenziali inserite dall'utente. 3. Conferma dell'identità: il Sistema attua una seconda verifica basata su una One Time Password. 4. Accesso: il Sistema, dopo aver terminato le fasi precedenti, concede o nega l'accesso all'Utente. L'autenticazione serve per garantire che solo gli utenti autorizzati possano accedere al sistema, alle risorse e ai servizi, mantenendo così un livello adeguato di sicurezza.	
Per garantire il corretto funzionamento del sistema abbiamo bisogno delle seguenti Boundary:	• LoginBND • OTP_BND • HomeBND • RecuperoPasswordBND • CreaPasswordBND
Le Control contenute sono invece:	• LoginCTRL • RecuperoPasswordCTRL

Il package comuni contiene le seguenti classi:

- DBMS_BND
- MessaggioASchermo
- Main

Sono inoltre presenti all'interno del package comuni le tre cartelle di seguito descritte:

comuni.eccezioni	
La cartella Eccezioni contiene le eccezioni personalizzate del sistema.	
Attualmente le eccezioni contenute all'interno del package eccezioni sono le seguenti:	• ErroreComunicazioneDBMSException

comuni.ent

La cartella ENT contiene semplici classi di tipo entity i cui attributi permettono di mantenere in memoria, con grande organizzazione, i dati necessari allo svolgimento delle funzionalità previste dal sistema, evitando inoltre interrogazioni ripetute al database.	
Per garantire il corretto funzionamento del sistema abbiamo bisogno delle seguenti Entity:	• UtenteENT • TurnoENT • StipendioENT • InfrastrutturaENT • ProgettoENT • SegnalazioneENT • RecensioneENT

comuni.metodiutili	
La cartella MetodiUtili contiene le classi di utilità per il package it.unipa.ingegneriasoftware, per favorire il corretto funzionamento del software. Al momento tutti i metodi sono contenuti in una singola classe ma in futuro sarà possibile suddividerli in differenti classi, in base al ruolo da essi svolto (esempio: i metodi inerenti alla manipolazione delle stringhe, metodi inerenti all'invio delle email, etc.).	
Attualmente le classi contenute all'interno del package metodiutili sono le seguenti:	• MetodiUtili

gestionedipendente	
La Gestione Dipendente è il processo di amministrazione e gestione delle entrate, delle uscite e delle variazioni richieste dal Dipendente all'interno dell'azienda. In particolare, questo meccanismo gestisce la rilevazione del tempo di lavoro dei Dipendenti, la gestione dei turni assegnati, la gestione di eventuali problemi legati alla rilevazione delle presenze, entro un certo limite, fuori orario tramite presenza da remoto e la possibilità di segnalare variazioni dei turni. La gestione Dipendente serve inoltre a garantire che i Dipendenti siano gestiti in modo efficiente e preciso, in particolare per quanto riguarda i loro orari di lavoro. Inoltre, consente di monitorare la presenza e la puntualità.	
Per garantire il corretto funzionamento del sistema abbiamo bisogno delle seguenti Boundary:	• TempoBND • RilevazioneEntrataUscitaBND • GestioneDipendenteBND • RichiestaVariazioneTurnoBND
Le Control contenute sono invece:	• SegnalaPresenzaCTRL • SegnalaPresenzaRitardoCTRL • SegnalaChiusuraCTRL • RichiestaVariazioneTurnoCTRL

gestionedipendenti	
La Gestione Dipendenti è il processo di amministrazione e gestione dei Dipendenti all'interno dell'azienda da parte del Datore Di Lavoro. La gestione Dipendenti include operazioni come la modifica delle informazioni personali dei Dipendenti (con annessa la gestione del ruolo di questi ultimi), l'assunzione e il licenziamento di un nuovo dipendente e la cronologia delle presenze dell'azienda. In generale, la gestione Dipendenti serve per garantire che le informazioni dei Dipendenti siano protette, sempre aggiornate e, in caso di situazioni impreviste, gestibili.	
Per garantire il corretto funzionamento del sistema abbiamo bisogno delle seguenti Boundary:	• GestioneDipendentiBND • AggiungiDipendenteBND • CercaDipendenteBND • EsportaListaBND • VisualizzaDipendenteBND • ModificaDatiDipendenteBND
Le Control contenute sono invece:	• AggiungiDipendenteCTRL • CercaDipendenteCTRL • EsportaListaCTRL • RimuoviDipendenteCTRL • ModificaDatiDipendenteCTRL • VisualizzaDipendenteCTRL

gestioneservizi	
La Gestione Servizi è il processo che gestisce i servizi messi a disposizione dall'azienda: abbiamo la gestione delle infrastrutture, delle segnalazioni, dei progettti e delle recensioni, mostrando liste e dettagli di queste, garantendo interazione indiretta con i cittadini che si interfacceranno con esse.	
Per garantire il corretto funzionamento del sistema abbiamo bisogno delle seguenti Boundary:	• GestioneInfrastruttureBND • GestioneProgettiBND • GestioneSegnalazioniBND • GestioneRecensioniBND • GestioneServiziBND
Le Control contenute sono invece:	• GestioneInfrastruttureCTRL • GestioneProgettiCTRL • GestioneSegnalazioniCTRL • GestioneRecensioniCTRL • GestioneServiziCTRL

gestionestipendi
La Gestione Stipendi è il processo di amministrazione e calcolo degli stipendi ai Dipendenti all'interno dell'azienda. L'utente deve potere avere accesso al proprio stipendio mensile e poter visualizzare quelli

dei mesi precedenti. I Dipendenti vengono pagati il primo giorno di ogni mese alle 00:10, tramite il sistema che inizia il calcolo dello stipendio previsto partendo da uno stipendio base e procedendo con la somma delle percentuali previste in base alle ore di lavoro del Dipendente.	
Per garantire il corretto funzionamento del sistema abbiamo bisogno delle seguenti Boundary:	• CalcolaStipendioBND • GestioneStipendiBND • MostraStipendioBND
Le Control contenute sono invece:	• CalcolaStipendioCTRL • MostraStipendioCTRL

gestioneturni	
La Gestione Turni è un processo che consente di pianificare e assegnare i turni lavorativi per l'azienda. Utilizza informazioni sul tempo per garantire che i turni siano assegnati in modo appropriato. Interagisce con il database per la gestione dei dati, in modo da poter tenere traccia dei turni assegnati e dei dipendenti disponibili. Inoltre, ci sono controlli per visualizzare le informazioni sui turni all'interno di una bacheca virtuale, sia per l'intera azienda che per i singoli dipendenti. Ciò consente ai dipendenti di vedere i propri turni e al Datore Di Lavoro di tenere traccia della pianificazione dei turni.	
Per garantire il corretto funzionamento del sistema abbiamo bisogno delle seguenti Boundary:	• GestioneTurniBND • ImpostaGiorniIndisponibiliBND • VisualizzaBachecaTurniBND • TempoBND • VisualizzaTurniPersonaliBND
Le Control contenute sono invece:	• ImpostaGiorniIndisponibiliCTRL • TempoCTRL • VisualizzaBachecaTurniCTRL • VisualizzaTurniPersonaliCTRL

gestioneutente	
La Gestione Utente è il processo di amministrazione delle informazioni degli utenti all'interno di un sistema. Include operazioni come la modifica della password, la modifica dell'iban, la modifica dell'email e il logout. La gestione utente è una parte importante della sicurezza del sistema, poiché permette di avere accesso ai dati sensibili dell'utente e, quindi, deve essere concessa solamente all'utente autorizzato.	
Per garantire il corretto funzionamento del sistema abbiamo bisogno delle seguenti Boundary:	• LogoutBND • GestioneUtenteBND • ModificaDatiAccountBND
Le Control contenute sono invece:	• LogoutCTRL • ModificaDatiAccountCTRL

2.2 java

Java ha una libreria di classi standard, nota come Java API (Application Programming Interface), che fornisce una serie di funzionalità comuni utilizzate per lo sviluppo di applicazioni Java.

I pacchetti della libreria standard di Java usati durante lo sviluppo del progetto sono:

java.awt	
Questo pacchetto fornisce un insieme di classi e interfacce per la creazione di interfacce utente grafiche per le applicazioni Java. Include classi per la creazione di finestre, pulsanti, menu, testo e altri elementi dell'interfaccia utente, nonché classi per la gestione degli eventi e dei layout dei componenti.	
Classi utilizzate:	CardLayout, Color, Dimension, event.ActionEvent, event.ActionListener, EventQueue, Font, Toolkit.

java.util	
Questo pacchetto fornisce una serie di classi e interfacce per la gestione di diverse funzionalità come la gestione delle collezioni, le date e l'elaborazione dei dati. Tra le principali classi del pacchetto, ci sono ArrayList, HashMap, Date e Scanner, utilizzate rispettivamente per la gestione di array dinamici, mappe hash, date e l'acquisizione dei dati.	
Classi utilizzate:	ArrayList, Date, HashMap, List, Map, Random, regex.PatternSyntaxException, ScannerProperties.

java.io	
Questo pacchetto fornisce un insieme di classi e interfacce per la gestione delle operazioni di input/output, come la lettura e la scrittura di file, la lettura da e la scrittura su flussi di dati e la gestione delle eccezioni.	
Classi utilizzate:	File, FileNotFoundException, FileOutputStream.

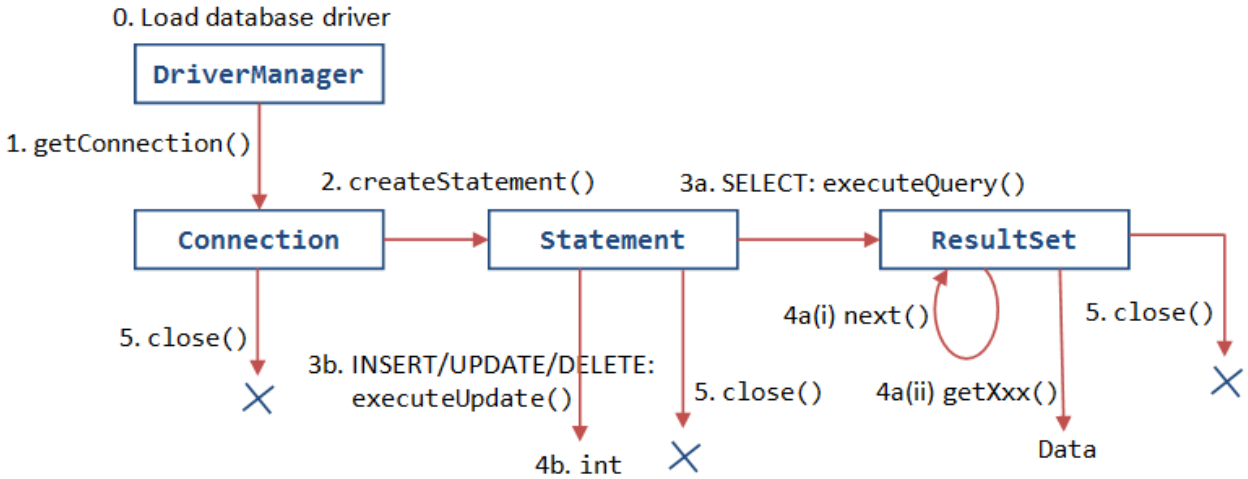
java.security	
Questo pacchetto fornisce un insieme di classi e interfacce per la gestione della sicurezza, come il digest dei messaggi e la generazione dei token.	
Classi utilizzate:	MessageDigest, NoSuchAlgorithmException.

java.time	
Questo pacchetto fornisce un insieme di classi e interfacce per la gestione del tempo.	
Classi utilizzate:	LocalTime, .temporal.ChronoUnit, DayOfWeek.

java.bean	
Questo pacchetto fornisce un insieme di classi e interfacce per ascoltare l'input in JCalendar.	
Classi utilizzate:	PropertyChangeEvent, PropertyChangeListener.

java.sql	
Questo pacchetto fornisce un insieme di classi e interfacce per l'accesso ai database relazionali da parte delle applicazioni Java. Fornisce una serie di classi per connettersi a un database, inviare query e ricevere risultati, queste classi sono utilizzate per connettersi a un database MySQL. Si precisa che il package java.sql si appoggia a un driver chiamato Java DataBase Connectivity (jdbc).	
Classi utilizzate:	Connection, ResultSet, DriverManager, SQLException, PreparedStatement, ResultSetMetaData, TimeStamp.

Di seguito viene riportato uno schema che mostra tutte le classi coinvolte durante l'esecuzione di una query:



Riportiamo adesso alcuni dettagli implementativi sulla nostra DBMS_BND. Tutti i metodi che eseguono query invocano un unico metodo centralizzato chiamato `eseguiCRUD` (Create, Read, Update, Delete). Questo metodo, dopo aver aperto una connessione con il database (creando quindi un oggetto `Connection`) e creato un oggetto `PreparedStatement` a partire dalla connessione stessa e da una query parametrizzata (che come detto nel System Design Document riduce o addirittura previene eventuali SQL *injection*), è in grado di capire se la query è una SELECT oppure una INSERT/UPDATE/DELETE. A questo punto vengono sostituiti nella query i parametri necessari e la query viene eseguita. Se la query è una SELECT, viene creato un oggetto di tipo `ResultSet`. Quest'ultimo viene iterato e viene usato per creare una lista di mappe. Ogni elemento della lista corrisponde a una tupla del risultato e ha per chiave il nome della colonna e per valore l'attributo che essa contiene. Questa particolare struttura dati ci permette, attraverso opportuni costruttori presenti nelle classi del package comuni.ent, di accedere ai risultati della query in maniera semplice e veloce. In questo modo la vita del ResultSet è limitata alla DBMS_BND stessa e le classi CTRL che la sfruttano non devono necessariamente conoscere come Java si interfaccia con un database SQL. Se invece la query non è una SELECT, viene semplicemente memorizzato in un attributo intero il numero totale di tuple coinvolte nell'operazione. In caso di errore di comunicazione o di query sbagliata il metodo eseguiCRUD lancia una ErroreComunicazioneDBMSException con un messaggio personalizzato. Un'ultima precisazione riguardo lo schema rappresentato in figura: a partire da JDK 7 gli oggetti di tipo Connection, Statement e ResultSet vengono automaticamente chiusi se si utilizza un try-with-resources. Si è quindi deciso di sfruttare questa tecnica.

2.3 javax

javax.swing	
Questo pacchetto fornisce un insieme di classi e interfacce per la creazione di interfacce utente grafiche per le applicazioni Java, fornisce un insieme di componenti potenti e flessibili per la creazione di interfacce utente.	
Classi utilizzate:	JFrame, JButton, JPanel, JLabel, JTextField, SwingConstants, JPasswordField, JOptionPane, JScrollPane, border.EmptyBorder, table.DefaultTableModel, RowFilter, table.TableRowSorter, table.TableModel, JRadioButton, JTextArea.

javax.mail	
Questo pacchetto fornisce un insieme di classi e interfacce per l'invio e la ricezione di email, e per la gestione dei MIME, necessario per l'utilizzo anche del file activation.jar.	
Classi utilizzate:	Message, MessagingException, PasswordAuthentication, Session, Transport, internet.InternetAddress, internet.MimeMessage.

2.4 com.itextpdf.text

com.itextpdf.text	
Questo pacchetto fornisce un insieme di classi e interfacce per la creazione e manipolazione di documenti in formato PDF, non fa parte della libreria standard di Java.	
Classi utilizzate:	Document, DocumentException, Paragraph, pdf.PdfWriter, pdf.PdfPTable.

2.5 com.toedter

com.toedter	
Questo pacchetto fornisce un insieme di classi e interfacce per la creazione e manipolazione di calendari, utile nelle interfacce grafiche dove deve essere mostrato un calendario con le rispettive date.	
Classi utilizzate:	calendar.JCalendar.

3. Class Interfaces

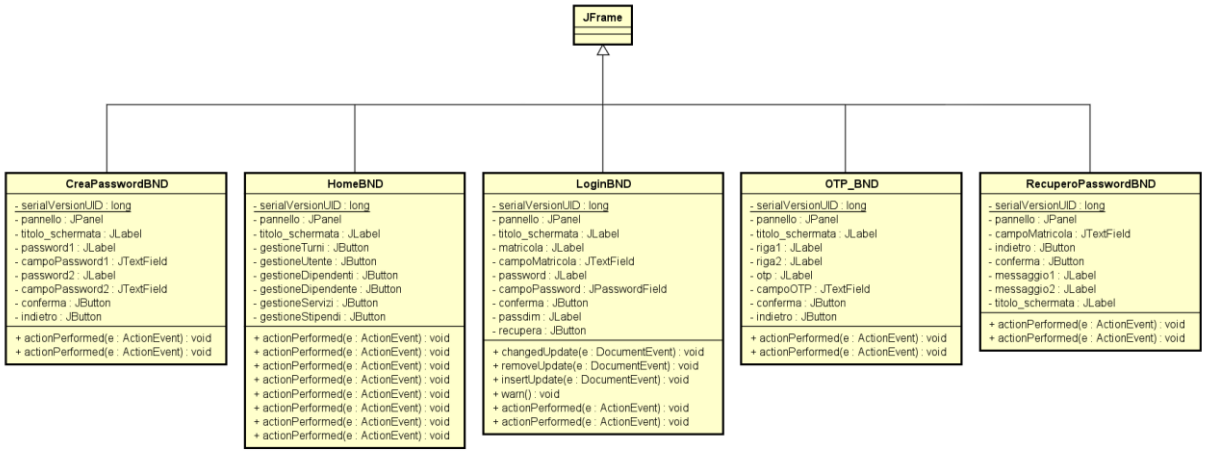
Prima di riportare tutti i diagrammi UML del package principale, vorremmo evidenziare alcune precisazioni riguardo l'implementazione dell'analisi precedentemente avvuenuta nel RAD:

- TerminaleIndisponibileBND è in realtà inglobata in `RilevazioneEntrataUscitaBND`: a partire da un unico *frame* vengono automaticamente alternati i due *panel* in base all'orario corrente.
- Rispetto al RAD è presente un'ulteriore control chiamata `GestioneServiziCTRL`, poiché sono necessari dei controlli prima di aprire la schermata corrispondente e si è quindi deciso di evitare la duplicazione del codice nelle quattro control dei singoli casi d'uso.
- I metodi `actionPerformed` sono utilizzati per il corretto funzionamento degli `ActionListener` che, associati ai pulsanti presenti in ogni schermata, permettono a questi ultimi di attendere indefinitamente un'azione svolta dall'utente su di essi e di avviare successivamente

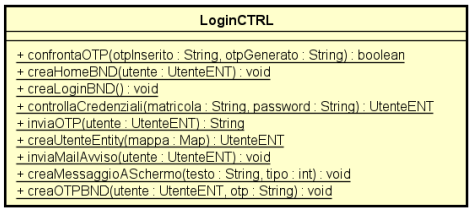
tutte le azioni per le quali è stato ideato. Sono inoltre presenti, in alcune classi, metodi che permettono di *ascoltare* calendari e tabelle.

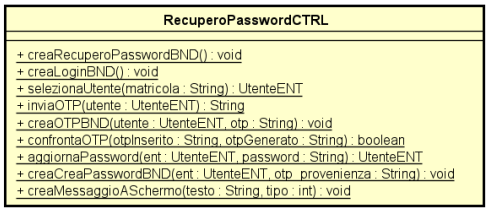
- I metodi `run` contengono le istruzioni da svolgere in un thread diverso da quello da cui è stato avviato il programma. In particolare l'utilizzo di thread separati per lunghe operazioni *in background* come la generazione dei turni, il calcolo degli stipendi e la gestione delle entrate/uscite dei dipendenti evita il congelamento delle schermate, che rimarrebbero altrimenti bloccate fino al completamento delle numerose query necessarie alle suddette operazioni.
- Sebbene non abbiamo usato la serializzazione degli oggetti, tutte le classi serializzabili sono provviste di un serialVersionUID poiché la sua assenza può comportare imprevisti e warning in fase di compilazione.
- Tutte le classi Control (CTRL) sono sprovviste di attributi e costruttori al loro interno e non è pertanto necessario associarle a un particolare oggetto. Inoltre, tutti i metodi che esse contengono sono dichiarati `static` ed è quindi possibile invocarli senza creare una specifica istanza della classe, riferendosi direttamente alla classe stessa.
- Il mock-up inerente al caso d'uso Cerca Dipendente presenta una piccola variazione rispetto all'interfaccia implementata in fase di codifica: i pulsanti raffigurati nel mock-up non sono più presenti in ogni riga, bensì una sola volta, e sono utilizzabili a seguito della selezione del dipendente desiderato. (Mockup Cerca Dipendente).

Autenticazione (BND)



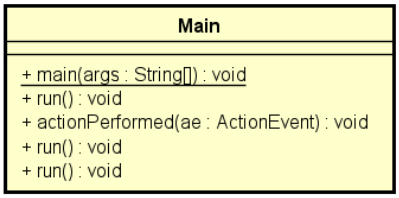
Autenticazione (CTRL)

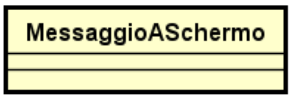

LoginCTRL
+ confrontaOTP(otpInserito : String, otpGenerato : String) : boolean
+ creaHomeBND(utente : UtenteENT) : void
+ creaLoginBND() : void
+ controllaCredenziali(matricola : String, password : String) : UtenteENT
+ inviaOTP(utente : UtenteENT) : String
+ creaUtenteEntity(mappa : Map) : UtenteENT
+ inviaMailAvviso(utente : UtenteENT) : void
+ creaMessaggioASchermo(testo : String, tipo : int) : void
+ creaOTPBND(utente : UtenteENT, otp : String) : void


RecuperoPasswordCTRL
+ creaRecuperoPasswordBND() : void
+ creaLoginBND() : void
+ selezionaUtente(matricola : String) : UtenteENT
+ inviaOTP(utente : UtenteENT) : String
+ creaOTPBND(utente : UtenteENT, otp : String) : void
+ confrontaOTP(otpInserito : String, otpGenerato : String) : boolean
+ aggiornaPassword(ent : UtenteENT, password : String) : UtenteENT
+ creaCreaPasswordBND(ent : UtenteENT, otp_provenienza : String) : void
+ creaMessaggioASchermo(testo : String, tipo : int) : void

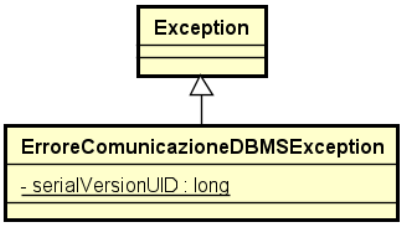
Comuni


DBMS_BND
- insiemeRisultati : List
- totaleTupleAggiornate : int
- tentativo_riconnessione : int
- url : String
- user : String
- pass : String
- eseguiCRUD(query : String, parametri : List) : void
- resultSetToArrayList(rs : ResultSet) : void
+ getInsiemeRisultati() : List
+ getTotaleTupleAggiornate() : int
+ queryVerificaIdentita(utente : UtenteENT) : UtenteENT
+ queryVerificaCredenziali(matricola : String, password : String) : UtenteENT
+ querySelezionaUtente(matricola : String) : UtenteENT
+ queryAggiornaPassword(utente : UtenteENT, nuovaPassword : String) : UtenteENT
+ queryModificaDati(utente : UtenteENT, emailNuova : String, passwordNuova : String, IBANnuovo : String) : UtenteENT
+ queryEsportaLista(data : LocalDate) : List
+ queryInserisciDipendente(utente : UtenteENT) : boolean
+ queryCercaDipendente(testoCercato : String) : List
+ queryRimuoviDipendente(matricola : String) : boolean
+ queryModificaDipendente(utente : UtenteENT) : boolean
+ generaTurni(servizioInt : int) : boolean
+ querySelezionaTurni(utente : UtenteENT, data_inizio : LocalDate, data_fine : LocalDate) : List
+ queryMostraTurni(data : LocalDate, ora_i : LocalTime, ora_f : LocalTime, matricola : String, servizio : Integer) : List
+ queryMostraStipendi(da : LocalDate, a : LocalDate, matricola : String) : List
+ queryCalcolaStipendi() : boolean
+ queryMostraInfrastrutture(codice : String, nome : String, citta : String) : List
+ queryMostraProgetti(codice : String, descrizione : String, stato : String) : List
+ queryMostraSegnalazioni(codice : String, domanda : String, risposta : String, categoria : String) : List
+ queryMostraRecensioni(codice : String, testo : String, punteggio : String) : List
+ queryImpostaGiorniIndisponibili(dataInizio : LocalDate, dataFine : LocalDate, utente : UtenteENT) : LocalDate
+ queryPresenza(matricola : int, turno : TurnoENT) : void
+ queryInTurno(ent : UtenteENT, dataInizio : LocalDate, dataFine : LocalDate) : List
+ queryGiorniIndisponibili(daLD : LocalDate, aLD : LocalDate) : ArrayList
+ queryAccettaFerie(utente : UtenteENT, daLT : LocalTime, aLT : LocalTime, daLD : LocalDate, aLD : LocalDate, motivo : int, testo : String) : int
+ queryGetTurno(ent : UtenteENT, dateTime : LocalDateTime) : TurnoENT
+ queryDipendentiTurno(DateTime : LocalDateTime) : void
+ queryDipendentiPresenti(dateTime : LocalDateTime, servizio : int) : List
+ queryChiudiServizio(servizio : int) : void
+ queryGetAssenti(dateTime : LocalDateTime) : List
+ queryNonInTurno(ent : UtenteENT, dataInizio : LocalDate, dataFine : LocalDate) : List
+ queryAggiornaVariazione(entSelezioanto_codTurno : List) : void
+ queryInTurnoAltriServizi(parametri : List) : List
+ queryNonInTurnoAltriServizi(parametri : List) : List
+ queryGetTurno2(utente : UtenteENT, data : LocalDate) : List
+ queryAggiornaOreDaRecuperare_rimuovi(matricola : Object, cod_turno : Object) : void
+ queryAggiornaOreDaRecuperare_aggiungi(matricola : Object, cod_turno : Object) : void
+ queryGetFineTurno(ent : UtenteENT, dateTime : LocalDateTime) : TurnoENT
+ queryStatoFineTurno(ent : UtenteENT, turno : TurnoENT, tipo : String) : void
+ queryPresenzaGiaEffettuata(utente : UtenteENT, data_ora : LocalDateTime) : int
+ queryTurnoDaEffettuare(parametri : List) : TurnoENT
+ queryAggiornaStato(turno : TurnoENT, utente : UtenteENT) : void
+ queryServizioAperto(servizio : int) : boolean
+ queryApriServizio(servizio : int) : void
+ queryTurniNonChiusi(tempo : LocalDateTime) : List
+ queryServizioVariazione(ent : UtenteENT, adesso : LocalDateTime) : int
+ queryGetTurno3(ent : UtenteENT, adesso : LocalDateTime) : boolean

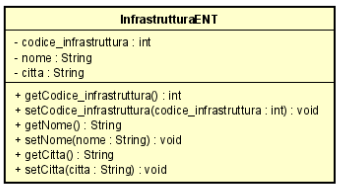


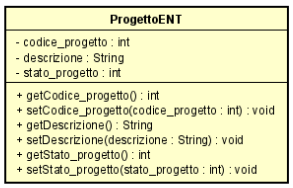


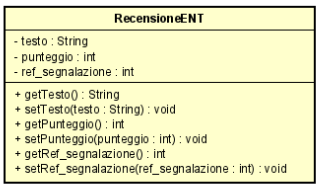
Comuni/Eccezioni

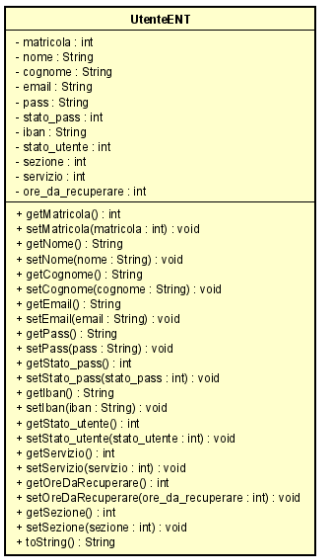


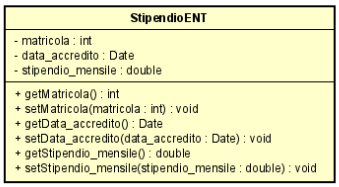
Comuni/ENT

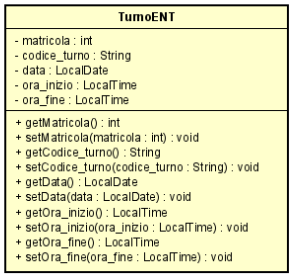


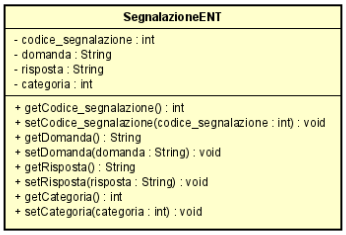




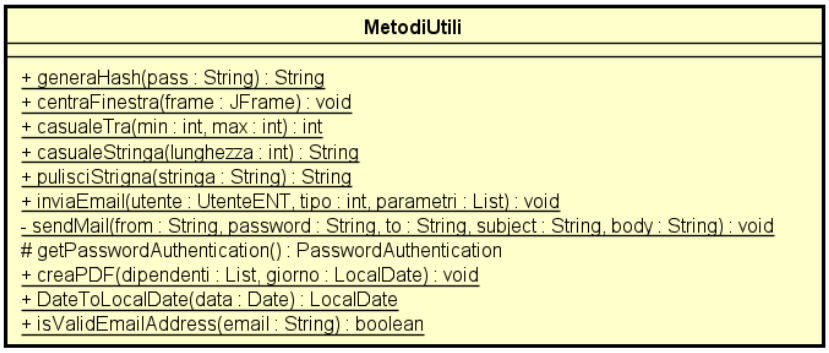




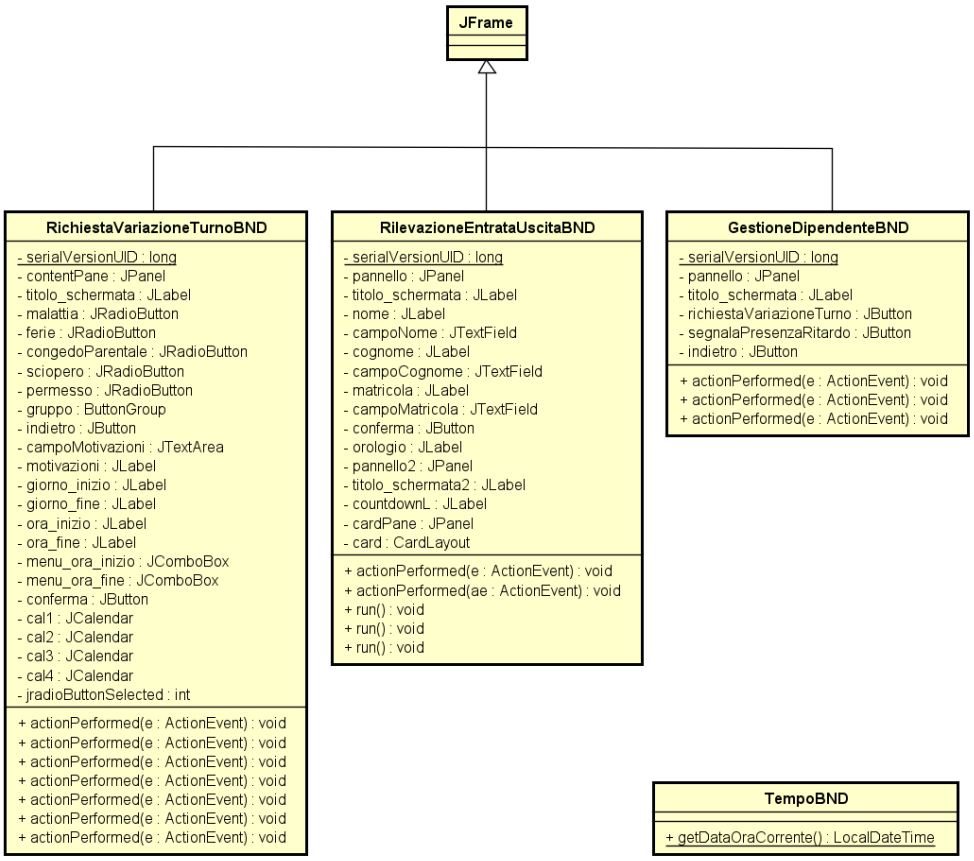




Comuni/MetodiUtili

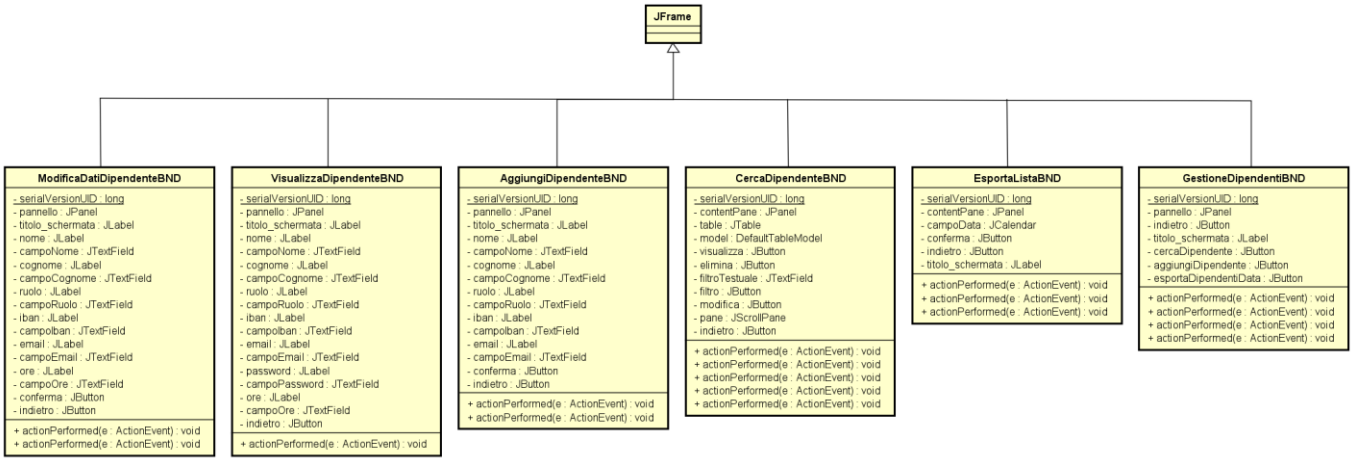


Gestione Dipendente (BND)

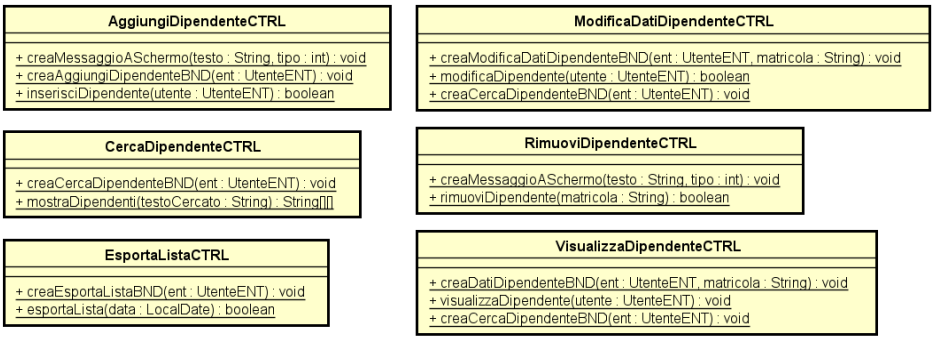


Gestione Dipendente (CTRL)

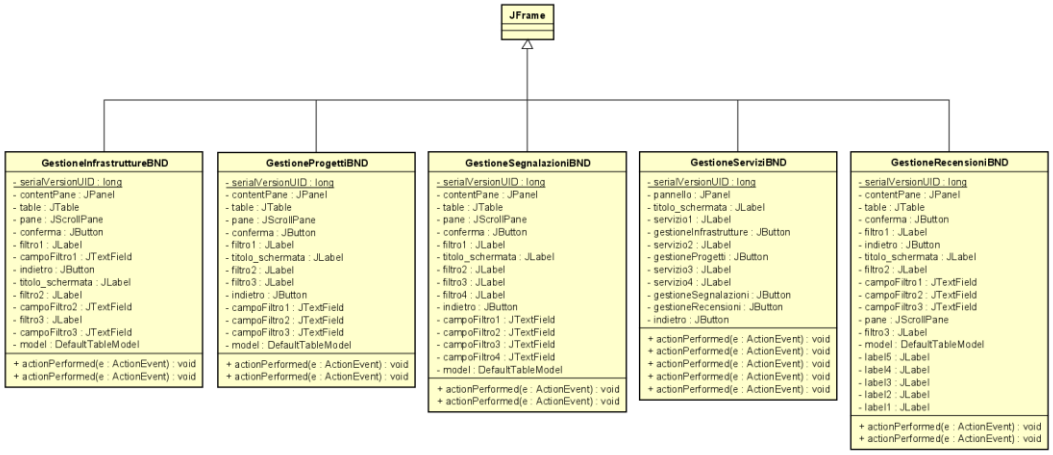
Gestione Dipendenti (BND)



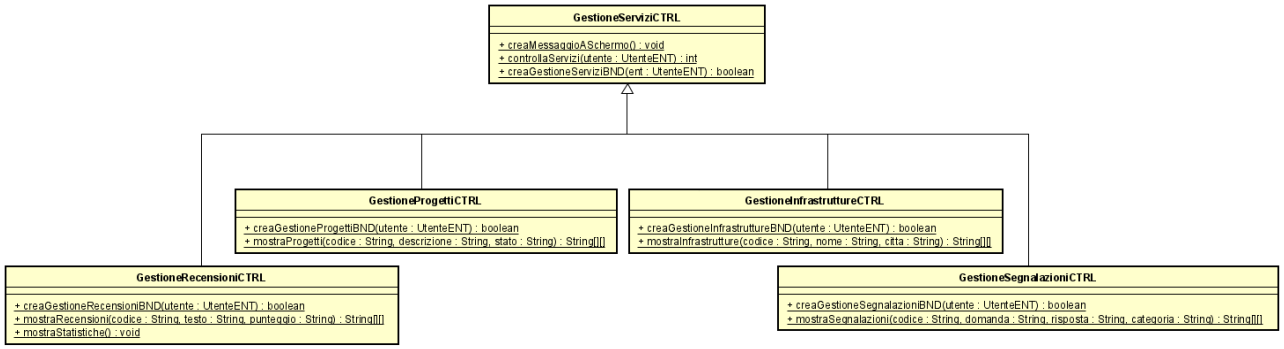
Gestione Dipendenti (CTRL)



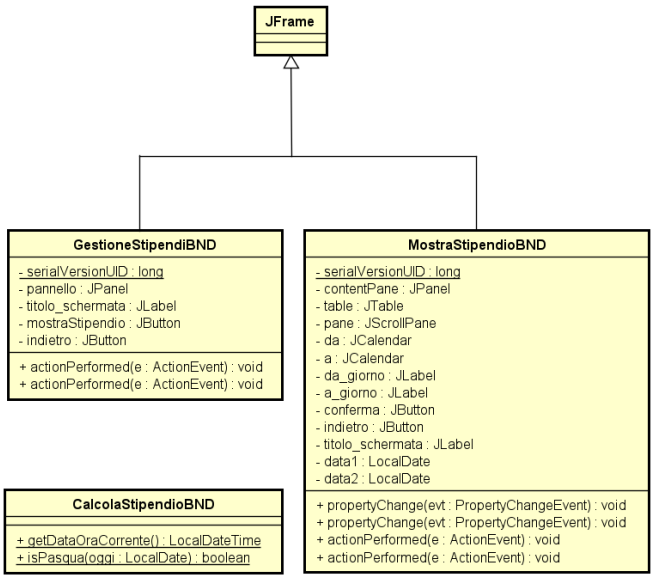
Gestione Servizi (BND)



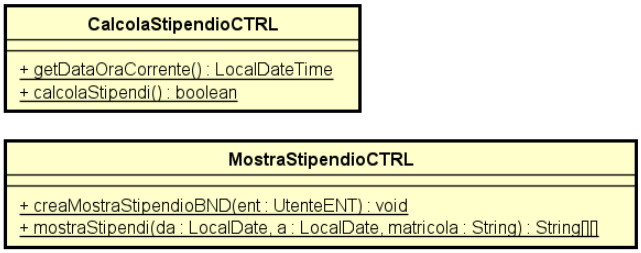
Gestione Servizi (CTRL)



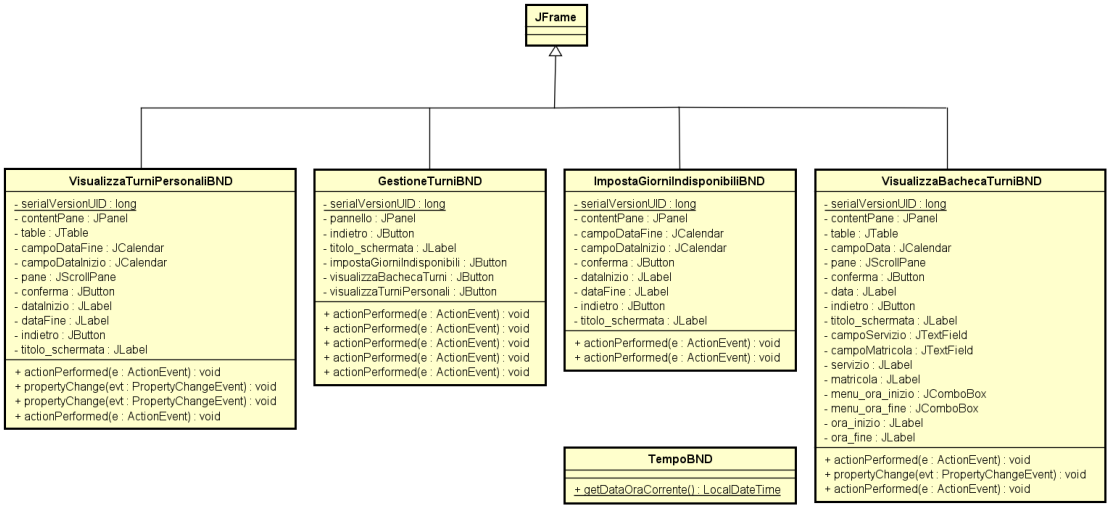
Gestione Stipendi (BND)



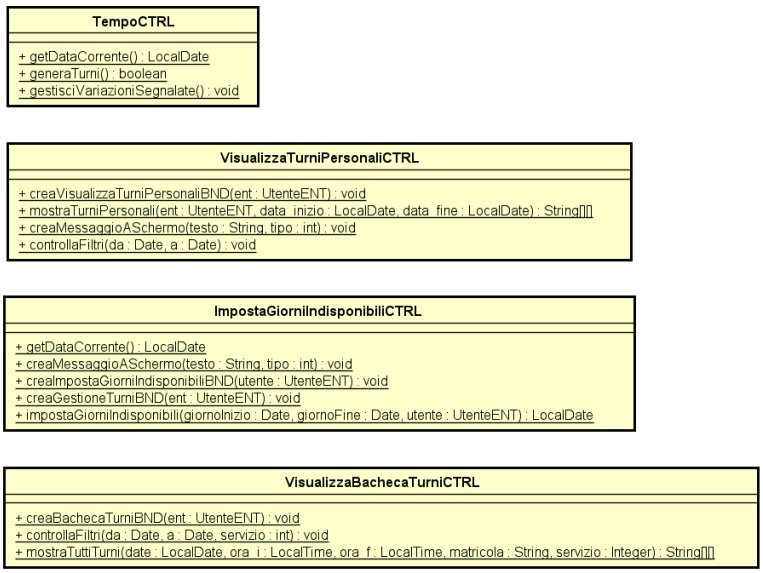
Gestione Stipendi (CTRL)



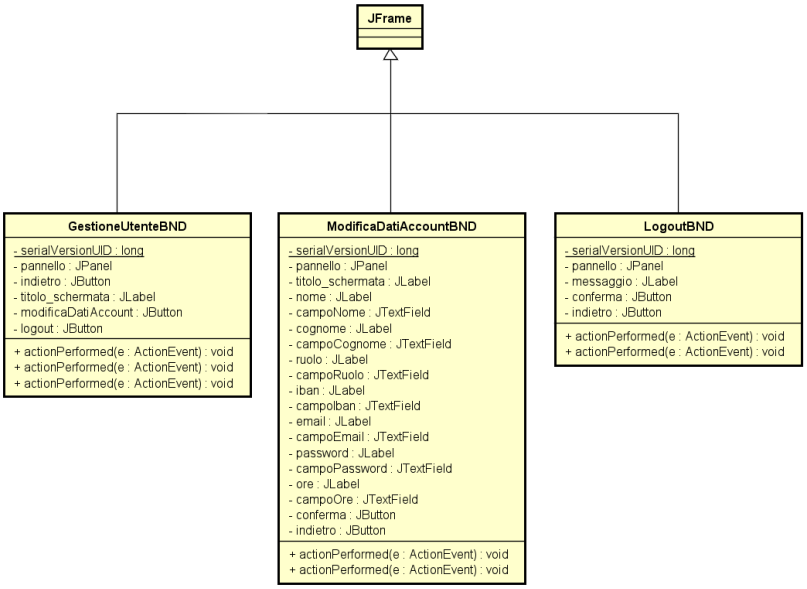
Gestione Turni (BND)



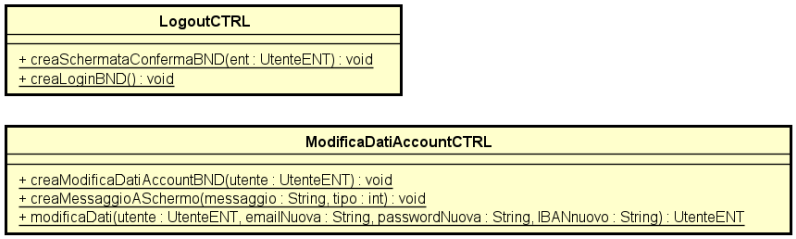
Gestione Turni (CTRL)



Gestione Utente (BND)



Gestione Utente (CTRL)



4. Glossary

Le definizioni del glossario (al momento solo in lingua inglese) sono *riusate* dal libro *Object-Oriented Software Engineering: Using Uml, Patterns and Java* di Bernd Bruegge e Allen H. Dutoit. Abbiamo selezionato quelle che fanno principalmente riferimento all'ODD. Altri termini utili possono essere trovati nel RAD e nel SDD.

O	
Object Design Document (ODD)	A document describing the object design model. The object design model is often generated from comments embedded in the source code.
P	
Package (…)	A UML grouping concept denoting that a set of objects or classes are related. Packages are used in use case and class diagrams to deal with the complexity associated with large numbers of use cases or classes