UNIVERSITÀ DEGLI STUDI DI NAPOLI "PARTHENOPE"

Progetto Esame di Programmazione III
E
Laboratorio di Programmazione III

Traccia Progetto:
Segreteria Studenti 2

Anno Accademico: 2023/2024

Professore: Angelo Ciaramella

Studente: Andrea Aristarco 0124002166

Sommario

Descrizione del progetto

Il progetto mira a sviluppare un sistema automatizzato in grado di gestire in modo efficiente e completo tutti gli scenari possibili di una segreteria studenti universitaria. Per raggiungere questo obiettivo, si fa uso del linguaggio di programmazione Java, integrato con la tecnologia JavaFX e SceneBuilder per la creazione di interfacce grafiche intuitive e funzionali. Il sistema si concentra sulle diverse azioni eseguibili da parte di segretari, studenti ed insegnanti. Queste operazioni includono prenotazioni agli esami, inserimento di appelli, immatricolazioni di studenti appena iscritti, e altre funzionalità connesse al contesto universitario. L'approccio del progetto è finalizzato a ottimizzare l'esperienza utente, garantendo un'efficace gestione delle pratiche amministrative e semplificando le operazioni quotidiane dei vari utenti coinvolti.

Il sistema prevede tre modalità di accesso: modalità segretario, modalità docente e modalità studente. Il segretario esegue tali operazioni:

1. Inserire un nuovo studente
2. Visualizzare le informazioni di uno studente (ricerca per nome e cognome o per matricola)

3. Visualizzare o stampare gli esiti dei test per singolo corso e per un intero Corso di Laurea (aggregazione delle valutazioni).

Il docente esegue tali operazioni:

1. Inserire un appello (gestire anche le prenotazioni)
2. Inserire il voto di esame ad uno studente. Allo studente viene richiesto di accettare o rifiutare l'esame.

Lo studente esegue tali operazioni:

1. Prenotarsi ad un esame
2. Accettare o rifiutare un esame
3. Effettuare il test di valutazione del corso.

Durante l'evoluzione del progetto, ho incorporato almeno due design pattern fondamentali: Command e Strategy. Questa scelta è stata guidata dai principi SOLID di programmazione, garantendo una struttura flessibile e manutenibile.
Per garantire una comprensione chiara del codice, ho incluso commenti dettagliati che offrono spiegazioni esaustive su ciascuna parte del sistema.
La gestione delle eccezioni è stata implementata con cura, assicurando un comportamento robusto anche in situazioni impreviste.
Inoltre, il sistema sfrutta l'utilizzo di un database per la memorizzazione dei dati, con particolare riferimento al database MySQL. Questa scelta consente una gestione efficiente e sicura delle informazioni necessarie per le varie funzionalità del sistema, contribuendo così alla coerenza e all'affidabilità complessiva.

Diagramma delle classi

Strategy

Il design pattern Strategy è ampiamente adottato nell'ambito dell'ingegneria del software. La sua concezione è finalizzata alla gestione di una famiglia di algoritmi, consentendo la selezione flessibile e dinamica di uno di essi durante l'esecuzione di un programma. Questo pattern favorisce la separazione delle variazioni dell'algoritmo dalla classe principale che lo utilizza, migliorando così la manutenibilità, la flessibilità e la riutilizzabilità del codice. La sua implementazione agevola l'adattamento a nuovi

requisiti o cambiamenti nei comportamenti algoritmici, senza compromettere la coerenza della struttura del programma.

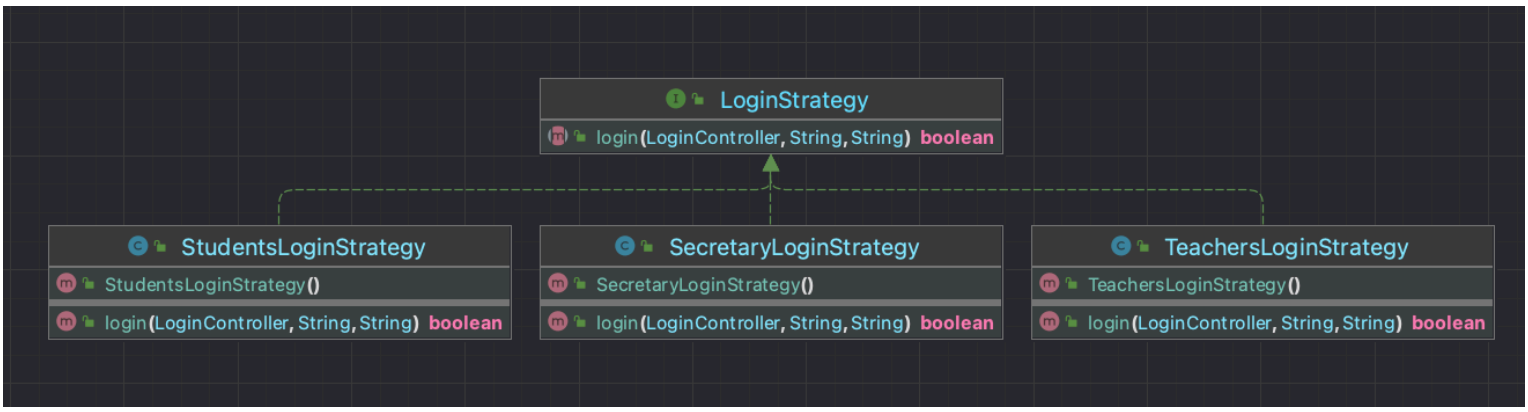
Il design pattern Strategy é utilizzato per gestire i diversi metodi di login. L'interfaccia LoginStrategy definisce una sorta di contratto per i diversi metodi di login. Questa interfaccia specifica il metodo " login(LoginController loginController, String username, String password);", che deve essere implementato da tue le sottoclassi che la estendono.

Grazie a questa struttura, é possibile aggiungere nuovi metodi di login senza modificare la classe principale che li utilizza. Ogni nuovo metodo deve semplicemente implementare l'interfaccia LoginStrategy, assicurando che il metodo " login(LoginController loginController, String username, String password);" sia adeguatamente gestito.

Le classi SecretaryLoginStrategy, TeachersLoginStrategy, StudentsLoginStrategy implementano l'interfaccia LoginStrategy e definiscono i metodi specifici per effettuare il login al sistema.

Il controller LoginController implementa il pattern Strategy nel metodo loginBTN(). Quando l'utente preme il pulsante "LOGIN", il controller recupera il metodo per effettuare il login specifico per tale utente e utilizza il " login(LoginController loginController, String username, String password);" della strategia scelta passando come parametri l'username e la password dell'utente.

In breve, il pattern Strategy consente la selezione dinamica e l'utilizzo di diversi metodi di login, isolando l'implementazione specifica di ciascun metodo dalla logica di controllo principale.



```java
package application.Strategy;

import application.Controllers.LoginController;

/**
 * Interfaccia Strategy, pattern utilizzato per la fase di login
 *
 * @author andreaaristarco
 * @version 1.0
 */
//Pattern Strategy per il login
3 implementations
public interface LoginStrategy {
    /**
     * Metodo utilizzato per la fase di login
     *
     * @param loginController controller della schermata di login
     * @param username username per l'accesso
     * @param password password per l'accesso
     * @return ritorna il login
     */
        3 implementations
        boolean login(LoginController loginController, String username, String password);
    }
```

```java
public class SecretaryLoginStrategy implements LoginStrategy{

    1 usage
    private final DatabaseSecretaryLogin database = new DatabaseSecretaryLogin();


    /**
     * Metodo utilizzato per effettuare il login
     *
     * @param loginController controller della schermata di login
     * @param username username per l'accesso
     * @param password password per l'accesso
     * @return ritorna la schermata del segretario o false se vi è errore
     */
  @Override
  public boolean login(LoginController loginController, String username, String password) {
      Integer id = Integer.parseInt(username);

      if (database.secretaryLogin(id, password)) {
          AlertUtil.showSuccessAlert("Login riuscito");
          loginController.handleSuccessfulLogin( title: "MENU SEGRETERIA",  resourcePath: "/application/DashboardSecretary.fxml");
          return true;
      }
      return false;
  }
}
```

```java
public class StudentsLoginStrategy implements LoginStrategy{
    1 usage
    private final DatabaseStudentsLogin database = new DatabaseStudentsLogin();

    /**
     * Metodo utilizzato per effettuare il login
     *
     * @param loginController controller della schermata di login
     * @param username username per l'accesso
     * @param password password per l'accesso
     * @return ritorna la schermata dello studente o false se vi é errore
     */
    @Override
    public boolean login(LoginController loginController, String username, String password) {
        Long matricola = Long.parseLong(username);

        if (database.studentsLogin(matricola, password)) {
            AlertUtil.showSuccessAlert("Login riuscito");
            LongSession.getInstance().setId(matricola);
            loginController.handleSuccessfulLogin( title: "MENU STUDENTE",  resourcePath: "/application/DashboardStudent.fxml");
            return true;
        }
        return false;
    }
}
```

```java
public class TeachersLoginStrategy implements LoginStrategy{
    1 usage
    private final DatabaseTeachersLogin database = new DatabaseTeachersLogin();

    /**
     * Metodo utilizzato per effettuare il login
     *
     * @param loginController controller della schermata di login
     * @param username username per l'accesso
     * @param password password per l'accesso
     * @return ritorna la schermata dell'insegnante o false se vi é errore
     */
    @Override
    public boolean login(LoginController loginController, String username, String password) {
        Integer id = Integer.parseInt(username);

        if (database.teachersLogin(id, password)) {
            AlertUtil.showSuccessAlert("Login riuscito");
            LongSession.getInstance().setId(Long.valueOf(id));
            loginController.handleSuccessfulLogin( title: "MENU DOCENTE",  resourcePath: "/application/DashboardTeacher.fxml");
            return true;
        }
        return false;
    }
}
```

```java
@FXML
protected void loginBTN(ActionEvent event) {
    if (loginUser.getText().isEmpty() || loginPassw.getText().isEmpty()) {
        AlertUtil.showErrorAlert("Compila tutti i campi");
    } else {
        String input = loginUser.getText();
        String password = loginPassw.getText();

        LoginStrategy loginStrategy;

        if (databaseSecretary.secretaryLogin(Integer.valueOf(input), password)) {
            loginStrategy = new SecretaryLoginStrategy();
        } else if (databaseStudents.studentsLogin(Long.parseLong(input), password)) {
            loginStrategy = new StudentsLoginStrategy();
        } else if (databaseTeachers.teachersLogin(Integer.parseInt(input), password)) {
            loginStrategy = new TeachersLoginStrategy();
        } else {
            System.out.println("login errato");
            AlertUtil.showErrorAlert("Login errato");
            return;
        }

       if (loginStrategy.login( loginController: this, input, password)) {
            System.out.println("Login riuscito");
        } else {
            System.out.println("Login non riuscito");
        }

    }
}
```

Command

Il design pattern Command è ampiamente adottato nell'ambito dell'ingegneria del software. La sua concezione si basa sull'incapsulamento di una richiesta come oggetto, consentendo la parametrizzazione dei client con richieste, la possibilità di accodarle, registrarle e supportare operazioni di annullamento. Questo pattern promuove l'incapsulamento, il disaccoppiamento e la flessibilità nell'esecuzione delle operazioni, migliorando così la modularità e la manutenibilità del codice.

Il design pattern Command vi é stato utilizzato in diversi controller: EditStudentController, SecretaryController, QuestController, StudentController, EditExamController, EditVotoController, TeacherController.

Ogni controller al suo interno vi sono implementati metodi che richiamano le classi che hanno incapsulate le operazioni specifiche da eseguire tramite l'utilizzo di bottoni.

In EditStudentController:

1. EditStudent, classe che implementa il pattern Command dove definisce l'azione di modificare i dati di uno studente quando viene invocato il metodo modifyBTNEdit().

```
    @Override
    public void execute() {
        if (tfNomeEdit.getText().isEmpty() || tfCognEdit.getText().isEmpty() || tfResiEdit.getText().isEmpty() || tfPasswEdit.getText().isEmpty()) {
            AlertUtil.showErrorAlert("compila tutti i campi");
        } else {
            //Recupero il valore della matricola dalla riga selezionata
            StudentData selectedStudent = tabellaStudenti.getSelectionModel().getSelectedItem();
            Long matricola = selectedStudent.getMatricola();

            String nome = tfNomeEdit.getText();
            String cognome = tfCognEdit.getText();
            LocalDate localDate = datePickerEdit.getValue();
            Date dataDiNascita = Date.valueOf(localDate);
            String residenza = tfResiEdit.getText();
            String pianoDiStudi = boxStudiEdit.getValue();
            String password = tfPasswEdit.getText();

            // Chiamare il metodo updateRecord del database
            boolean success = databaseSecretaryMenu.updateRecord(matricola, nome, cognome, dataDiNascita, residenza, pianoDiStudi, password);
            if (success) {

                // Ottieni la lista aggiornata di studenti
                ObservableList<StudentData> updatedStudentList = databaseSecretaryMenu.getAllRecords();

                // Aggiorna la TableView con la lista aggiornata
                tabellaStudenti.setItems(updatedStudentList);
                tabellaStudenti.refresh();
                Stage currentStage = (Stage) modificabtnEdit.getScene().getWindow();
                currentStage.close();

            } else {
                AlertUtil.showErrorAlert("Errore durante la modifica dello studente.");
            }
        }
    }
}
```

In SecretaryController:

1. AddStudent, classe che implementa il pattern Command dove definisce l'azione di aggiunta di uno studente addBTN()
2. AddFees, classe che implementa il pattern Command dove definisce l'azione di inserire le tasse ad uno studente appena immatricolato quando viene invocato il metodo insertFeesBTN()
3. ClearStudent, classe che implementa il pattern Command dove definisce l'azione di ripulire i moduli di riempimento dati quando viene invocato il metodo clearBTN()

4. LogoutSecretary, classe che implementa il pattern Command dove definisce l'azione di effettuare log out in caso di terminato utilizzato del sistema quando viene invocato il metodo logoutBTN() e logout2BTN()
5. SearchStudents, classe che implementa il pattern Command dove definisce l'azione di cercare gli studenti tramite barra di ricerca quando viene invocato il metodo searchBTN()
6. RefreshStudent, classe che implementa il pattern Command dove definisce l'azione di aggiornare la barra di ricerca e la tabella degli studenti quando viene invocato il metodo refreshBTN()
7. DeleteStudent, classe che implementa il pattern Command dove definisce l'azione di eliminare uno studente quando viene invocato il metodo deleteBTN()
8. DeleteFees, classe che implementa il pattern Command dove definisce l'azione di eliminare le tasse quando viene invocato il metodo deleteFeesBTN()
9. ModifyStudent, classe che implementa il pattern Command dove definisce l'azione di aprire una schermata specifica per la modifica dello studente quando viene invocato il metodo modifyBTN()
10. SearchFees e SearchVote, classi che implementano il pattern Command dove definiscono l'azione di ricercare le tasse o i voti per lo studente specifico quando vengono invocati i metodi searchFeesBTN() e searchVoteBTN()
11. RefreshFees e RefreshVote, classi che implementano il pattern Command dove definiscono l'azione di aggiornare le tabelle tasse o voti quando vengono invocati i metodi refreshFeesBTN() e refreshVoteBTN().

```java
@Override
public void execute() {
    if (tfNome.getText().isEmpty() || tfCognome.getText().isEmpty() || tfResidenza.getText().isEmpty() || tfPassw.getText().isEmpty()) {
        AlertUtil.showErrorAlert("Compila tutti i campi");
    } else {
        //Recupero la matricola
        Long matricola = DatabaseSecretaryMenu.generateRandomNumber( length: 10);

        //Richiamo della query per aggiungere lo studente
        boolean success = DatabaseSecretaryMenu.register(matricola, nome, cognome, Date.valueOf(dataDiNascita), residenza, pianoDiStudi, password);

        if (success) {
            studentList = databaseSecretaryMenu.getAllRecords();
            tabellaStudenti.setItems(studentList);
            tabellaStudenti.refresh();

            tfNome.clear();
            tfCognome.clear();
            datePicker.setValue(null);
            tfResidenza.clear();
            tfPassw.clear();
            boxStudi.getSelectionModel().clearSelection();
        } else {
            AlertUtil.showErrorAlert("Errore durante l'aggiunta dello studente.");
        }
    }
}
}
```

```java
    @Override
    public void execute() {
        StudentData selectedStudent = tabellaStudenti.getSelectionModel().getSelectedItem();

        if (selectedStudent != null) {
            //Recupero matricola selezionata
            Long matricola = selectedStudent.getMatricola();

            //Richiamo della query per eliminare lo studente
            boolean success = databaseSecretaryMenu.deleteRecord(matricola);

            if (success) {
                studentList = databaseSecretaryMenu.getAllRecords();
                tabellaStudenti.setItems(studentList);
                tabellaStudenti.refresh();
                AlertUtil.showSuccessAlert("studente eliminato correttamente");
            } else {
                AlertUtil.showErrorAlert("Errore durante l'eliminazione dello studente.");
            }
        }
    }
}
```

```java
    @Override
    public void execute() {
        FeesData selectedFees = tabellaFees.getSelectionModel().getSelectedItem();

        if (selectedFees != null){
            Integer idTassa = selectedFees.getIdTassa();

            boolean success = databaseSecretaryMenu.deleteFees(idTassa);

            if (success){
                feesList = databaseSecretaryMenu.getAllFees();
                tabellaFees.setItems(feesList);
                tabellaFees.refresh();
                AlertUtil.showSuccessAlert("tassa eliminata correttamente");
            }else {
                AlertUtil.showErrorAlert("errore durante l'eliminazione della tassa");
            }
        }
    }
}
```

```java
    @Override
    public void execute() {
        StudentData selectedStudents = tabellaStudenti.getSelectionModel().getSelectedItem();

        if (selectedStudents != null){
            Integer idTassa = Math.toIntExact(DatabaseSecretaryMenu.generateRandomNumber( length: 6));
            Long matricola = selectedStudents.getMatricola();

            boolean success = databaseSecretaryMenu.insertFees(idTassa, matricola);

            if (success){
                feesList = databaseSecretaryMenu.getAllFees();
                tabellaFees.setItems(feesList);
                tabellaFees.refresh();
                AlertUtil.showSuccessAlert("tasse inserite correttamente");
            }else {
                AlertUtil.showErrorAlert("errore durante l'inserimento delle tasse");
            }
        }
    }
}
```

In QuestController:

1. QuestExamEdit, classe che implementa il pattern Command dove definisce l'azione di compilare il questionario quando viene invocato il metodo insertBTN().

```java
    @Override
    public void execute() {
        if (boxQuest1.getItems().isEmpty() || boxQuest2.getItems().isEmpty() || boxQuest3.getItems().isEmpty()){
            AlertUtil.showErrorAlert("compila tutti i campi");
        }else {
            QuestData selectedQuest = tabellaQuest.getSelectionModel().getSelectedItem();
            //Recupero l'id del questionario, la matricola di chi l'ha effettuato, il nome dell'esame scelto e le risposte controllando lo stato se effettuato o meno
            Integer idQuest = selectedQuest.getIdQuest();
            Long matricolaDomanda = LongSession.getInstance().getId();
            String esameDomanda = selectedQuest.getNomeExQuest();
            String domanda1 = boxQuest1.getValue();
            String domanda2 = boxQuest2.getValue();
            String domanda3 = boxQuest3.getValue();
            Boolean effettuato = selectedQuest.getEffettuato();

            //Recupero della query per aggiungere i dati del questionario
            boolean success = databaseStudentsMenu.addQuest(idQuest, matricolaDomanda, esameDomanda, domanda1, domanda2, domanda3);

            if (success){
                //Richiamo della query per aggiornare lo stato del questionario
                databaseStudentsMenu.updateQuest(idQuest, effettuato);
                questList = databaseSecretaryMenu.getAllQuest();
                tabellaQuest.setItems(questList);
                tabellaQuest.refresh();
                AlertUtil.showSuccessAlert("questionario completato");
                Stage currentStage = (Stage) inseriscibtn.getScene().getWindow();
                currentStage.close();
            }else {
                AlertUtil.showErrorAlert("questionario non completato");
            }
        }
    }
}
```

In StudentController:

1. BookExam, classe che implementa il pattern Command dove definisce l'azione di prenotare un appello quando viene invocato il metodo bookBTN()
2. LogoutStudent, classe che implementa il pattern Command dove definisce l'azione di log out in caso di terminato utilizzato del sistema quando viene invocato il metodo logoutBTN()
3. AcceptVote, classe che implementa il pattern Command dove definisce l'azione di accettare il voto di un esame quando viene invocato il metodo acceptBTN()
4. DeclineVote, classe che implementa il pattern Command dove definisce l'azione di rifiutare il voto di un esame quando viene invocato il metodo declineBTN()
5. DeleteBooking, classe che implementa il pattern Command dove definisce l'azione di eliminare una prenotazione quando viene invocato il metodo deleteBTN()
6. PayFees, classe che implementa il pattern Command dove definisce l'azione di pagamento delle tasse quando viene invocato il metodo payBTN()
7. QuestExam, classe che implementa il pattern Command dove definisce l'azione di aprire la schermata per la compilazione del questionario quando viene invocato il metodo questBTN().

```
    /**
     * Metodo utilizzato per accettare il voto dell'esame conseguito
     */
    @Override
    public void execute() {
        VotiData selectedVote = tabellaVoto.getSelectionModel().getSelectedItem();

        if (selectedVote != null){
            //Recupero l'id del voto
            Integer idVoto = selectedVote.getIdVoto();
            //Richiamo della query per l'update del voto
            boolean success = DatabaseStudentsMenu.updateVote(idVoto);

            if (success){
                voteList = databaseStudentsMenu.getAllVote();
                tabellaVoto.setItems(voteList);
                tabellaVoto.refresh();
                AlertUtil.showSuccessAlert("esame confermato");
            }else {
                AlertUtil.showErrorAlert("esame giá confermato");
            }
        }
    }
}
```

```java
    /**
     * Metodo utilizzato per rifiutare un'esame conseguito
     */
    @Override
    public void execute() {
        VotiData selectedVote = tabellaVoto.getSelectionModel().getSelectedItem();

        if (selectedVote != null);
        //Recupero l'id voto
        Integer idVoto = selectedVote.getIdVoto();
        //Recupero della query per rifiutare un voto
        boolean success = DatabaseStudentsMenu.deleteVote(idVoto);

        if (success){
            voteList = databaseStudentsMenu.getAllVote();
            tabellaVoto.setItems(voteList);
            tabellaVoto.refresh();
        }else {
            AlertUtil.showErrorAlert("impossibile rifiutare, esame giá confermato");
        }
    }
}
```

```java
    /**
     * Metodo utilizzato per il pagamento delle tasse
     */
    @Override
    public void execute() {
        FeesData selectedFees = tabellaFees.getSelectionModel().getSelectedItem();

        if (selectedFees != null){
            //Recupero della matricola
            Integer idTassa = selectedFees.getIdTassa();
            //Recupero della query per l'update delle tasse
            boolean success = databaseStudentsMenu.updateFees(idTassa);

            if (success){
                feesList = databaseStudentsMenu.getAllStudentFees(matricola);
                tabellaFees.setItems(feesList);
                tabellaFees.refresh();
                AlertUtil.showSuccessAlert("pagamento tassa andato a buon fine");
            }else {
                AlertUtil.showErrorAlert("Tassa giá pagata");
            }
        }
    }
```

```java
    /**
     * Metodo utilizzato per prenotarsi ad un'esame
     * Prenotandosi ad un'esame creo l'id della prenotazione e lo mostro nella tabella per tenere traccia delle prenotazioni
     */
    @Override
    public void execute() {
        ExamsData selectedExam = tabellaAppelli.getSelectionModel().getSelectedItem();

        System.out.println(selectedExam.getNomeEsame());
        //Recupero id esame e matricola (richiamo il singleton per recuperare l'istanza della matricola)
        Integer idEsame = selectedExam.getIdEsame();
        Long matricola = LongSession.getInstance().getId();
        System.out.println(idEsame.toString() + matricola.toString());

        //Recupero della query per prenotare l'esame
        boolean success = DatabaseStudentsMenu.bookExam(((int) DatabaseSecretaryMenu.generateRandomNumber( length: 5)), idEsame, matricola);
        if (success) {
            ObservableList<ResultJoin> joinList = databaseStudentsMenu.getAllBookingJoin();
            tabellaPre.setItems(joinList);
            tabellaPre.refresh();
            AlertUtil.showSuccessAlert("esame prenotato");
        } else {
            AlertUtil.showErrorAlert("esame giá prenotato");
        }
    }
}
```

In EditExamController:

1. EditExam, classe che implementa il pattern Command dove definisce l'azione di modificare i dati un appello quando viene invocato il metodo modifyBTNEdit().

```java
    /**
     * Metodo utilizzato per modificare i dati di un appello
     */
    @Override
    public void execute() {
        if(boxEditNome.getItems().isEmpty() || boxEditOrario.getItems().isEmpty() || tfEditAula.getText().isEmpty()){
            AlertUtil.showErrorAlert("compila tutti i campi");
        }else {
            //Recupero il valore del'id esame dalla riga selezionata
            ExamsData selectedExam = tabellaAppelli.getSelectionModel().getSelectedItem();
            Integer idEsame = selectedExam.getIdEsame();
            //Recupero nome, data, orario, aula esame
            String nomeEsame = boxEditNome.getValue();
            LocalDate localDate = datePickerEditEx.getValue();
            Date dataEsame = Date.valueOf(localDate);
            String orarioEsame = boxEditOrario.getValue();
            String aulaEsame = tfEditAula.getText();

            // Richiamo della query per l'update dell'esame
            boolean success = databaseTeacherMenu.updateRecord(idEsame, nomeEsame, dataEsame, orarioEsame, aulaEsame);
            if (success) {

                // Ottieni la lista aggiornata di studenti
                ObservableList<ExamsData> updatedExamList = databaseTeacherMenu.getAllRecords();

                // Aggiorna la TableView con la lista aggiornata
                tabellaAppelli.setItems(updatedExamList);
                tabellaAppelli.refresh();
                Stage currentStage = (Stage) modificabtnEdit.getScene().getWindow();
                currentStage.close();

            } else {
                AlertUtil.showErrorAlert("Errore durante la modifica dell'esame.");
            }
        }
    }
}
```

In EditVotoController:

1. AddVotoEdit, classe che implementa il pattern Command dove definisce l'azione di aggiungere il voto di uno studente quando viene invocato il metodo addBTNVoto().

```java
/**
 * Metodo utilizzato per inserire il voto ad un'esame
 */
@Override
public void execute() {
    if (boxEditVoto.getItems().isEmpty()){
        AlertUtil.showErrorAlert("inserisci voto correttamente");
    }else{
        BookingExamsData selectedBooking = tabellaPrenotazioni.getSelectionModel().getSelectedItem();
        //Recupero l'id della prenotazione, l'id dell'esame, la matricola e il voto
        Integer idPreVo = selectedBooking.getIdPrenotazione();
        Integer idEsameVoto = selectedBooking.getIdEsamePre();
        Long matricolaVoto = selectedBooking.getMatricolaStudente();
        String nomeExVoto = ExamsData.getNomeEsame();
        Integer voto = boxEditVoto.getValue();
        Boolean conferma = false;
        Integer idPrenotazione = selectedBooking.getIdPrenotazione();

        //Richiamo della query per aggiungere il voto
       boolean success = databaseStudentsMenu.addVote(((int) DatabaseSecretaryMenu.generateRandomNumber( length: 3)), idPreVo, idEsameVoto, nomeExVoto, matricolaVoto, voto, conferma);

        if (success) {
            if (voto == 0) {
                //Se il voto é 0, allora é bocciato
                System.out.println("bocciato");
                AlertUtil.showSuccessAlert("voto aggiunto");
                databaseTeacherMenu.deleteRecordPre(idPrenotazione);
            } else if (voto == 31) {
                //Se il voto é 31, allora vale 30 e lode
                System.out.println("30 e lode");
                AlertUtil.showSuccessAlert("voto aggiunto");
            } else{
                System.out.println("il tuo voto é " + voto);
                AlertUtil.showSuccessAlert("voto aggiunto");
            }
            //Richiamo della query per cancellare la prenotazione quando il voto é stato inserito
            boolean deleteSuccess = databaseTeacherMenu.deleteRecordPre(idPreVo);
            if (deleteSuccess){
                AlertUtil.showSuccessAlert("prenotazione cancellata");
            }else{
                AlertUtil.showErrorAlert("errore durante la cancellazione della prenotazione");
            }
            Stage currentStage = (Stage) aggiungibtnVoto.getScene().getWindow();
            currentStage.close();

            ObservableList<BookingExamsData> bookingList = databaseTeacherMenu.getAllBooking();
            tabellaPrenotazioni.setItems(bookingList);
            tabellaPrenotazioni.refresh();
        }else {
            AlertUtil.showErrorAlert("Errore durante l'operazione");
        }
    }
}
}
```

In TeacherController:

1. AddExam, classe che implementa il pattern Command dove definisce l'azione di aggiungere un appello quando viene invocato il metodo addBTN()
2. ClearExam, classe che implementa il pattern Command dove definisce l'azione di ripulire i moduli di riempimento dati quando viene invocato il metodo clearBTN()

3. LogoutTeacher, classe che implementa il pattern Command dove definisce l'azione di log out in caso di terminato utilizzato del sistema quando viene invocato il metodo logoutBTN()
4. SearchExam, SearchBooking, classi che implementano il pattern Command dove definiscono l'azione di ricercare gli appelli o le prenotazioni quando vengono invocati i metodi searchBTN() e searchPreBTN()
5. RefreshExam e RefreshBooking, classi che implementano il pattern Command dove definiscono l'azione di aggiornare le tabelle appelli o prenotazioni quando vengono invocati i metodi refreshBTN() e refreshPreBTN().
6. DeleteExam, DeleteBooking, classi che implementano il pattern Command dove definiscono l'azione di eliminare un appello o una prenotazione quando vengono invocati i metodi deleteBTN() e deletePreBTN()
7. ModifyExam, classe che implementa il pattern Command dove definisce l'azione di apertura della schermata per la modifica di un appello quando viene invocato il metodo modifyBTN()
8. AddVoto, classe che implementa il pattern Command dove definisce l'azione di apertura della schermata per l'inserimento del voto quando viene invocato il metodo addVotoBTN().

```java
    /**
     * Metodo utilizzato per aggiungere un appello
     */
    @Override
    public void execute() {
        if(boxEsami.getItems().isEmpty() || boxOrari.getItems().isEmpty() || tfAula.getText().isEmpty()){
            AlertUtil.showErrorAlert("compila tutti i campi");
        }else{
            //Recupero nome, data, orario e aula dell'esame
            String nomeEsame = boxEsami.getValue();
            LocalDate localDate = datePickerEsami.getValue();
            Date dataEsame = Date.valueOf(localDate);
            String orarioEsame = boxOrari.getValue();
            String aulaEsame = tfAula.getText();

            //Richiamo della query per l'aggiunta dell'appello
            boolean success = DatabaseTeacherMenu.register(((int) DatabaseSecretaryMenu.generateRandomNumber( length: 5)), nomeEsame, dataEsame, orarioEsame, aulaEsame);

            if (success) {
                // Aggiorna la TableView con la lista aggiornata degli appelli dal database
                examsList = databaseTeacherMenu.getAllRecords();
                tabellaAppelli.setItems(examsList);
                tabellaAppelli.refresh();
                AlertUtil.showSuccessAlert("appello inserito correttamente");

                // Pulisci i campi del modulo dopo l'aggiunta
                boxEsami.getSelectionModel().clearSelection();
                datePickerEsami.setValue(null);
                boxOrari.getSelectionModel().clearSelection();
                tfAula.clear();

            } else {
                AlertUtil.showErrorAlert("Errore durante l'aggiunta dell'appello.");
            }
        }
    }
}
```

```java
    /**
     * Metodo utilizzato per eliminare un appello
     */
    @Override
    public void execute() {
        ExamsData selectedExam = tabellaAppelli.getSelectionModel().getSelectedItem();

        if (selectedExam != null){
            //Recupero l'id dell'esame
            Integer idEsame = selectedExam.getIdEsame();
            //Richiamo della query per eliminare l'appello
            boolean success = databaseTeacherMenu.deleteRecord(idEsame);

            if (success){
                examsList = databaseTeacherMenu.getAllRecords();
                tabellaAppelli.setItems(examsList);
                tabellaAppelli.refresh();
                AlertUtil.showSuccessAlert("appello eliminato con successo");

            }else {
                AlertUtil.showErrorAlert("errore durante l'eliminazione dell'esame");
            }
        }
    }
}
```

```java
    /**
     * Metodo utilizzato per eliminare una prenotazione
     */
    @Override
    public void execute() {
        BookingExamsData selectedBooking = tabellaPrenotazioni.getSelectionModel().getSelectedItem();

        if (selectedBooking != null){
            //Recupero l'id della prenotazione
            Integer idPrenotazione = selectedBooking.getIdPrenotazione();
            //Recupero della query per eliminare una prenotazione
            boolean success = databaseTeacherMenu.deleteRecordPre(idPrenotazione);

            if (success){
                bookingList = databaseTeacherMenu.getAllBooking();
                tabellaPrenotazioni.setItems(bookingList);
                tabellaPrenotazioni.refresh();
                AlertUtil.showSuccessAlert("prenotazione eliminata con successo");
            }else {
                AlertUtil.showErrorAlert("errore durante l'eliminazione della prenotazione");
            }
        }
    }
```

Tutti i controller citati hanno un'implementazione simile per l'esecuzione dei comandi: il metodo execute() viene richiamato quando il pulsante corrispondente, separando l'azione specifica.
D'altronde tutti i controller si collegano ai rispettivi oggetti Command eseguendo il comando quando il pulsante viene premuto.

Struttura del Database della Segreteria

Il database della segreteria contiene diverse tabelle, ognuno presenta una scopo specifico per gestire i dati relativi al segretario, ai docenti e agli studenti.

Ecco una panoramica delle tabelle e dei loro campi:

Tabella secretary:
- id: Identificativo univoco del segretario (Chiave primaria).
- username:Username del segretario.
- password: Password del segretario.

Tabella teachers:
- id: Identificativo univoco del docente (Chiave primaria).
- nome: Nome del docente.
- cognome: Cognome del docente.
- pianoDiStudi: Corso di Studio del docente.
- password: Password del docente.

Tabella students:
- matricola: Codice univoco dello studente (Chiave primaria).
- nome: Nome dello studente.
- cognome: Cognome dello studente.
- dataDiNascita: Data di nascita dello studente.
- residenza: Città o via di residenza dello studente.
- pianoDiStudi: Corso di Studio dello studente.
- password: Password dello studente.

Tabella exams:
- idEsame: Identificativo univoco dell'appello (Chiave primaria).
- nomeEsame: Nome dell'appello.
- dataEsame: Data svolgimento appello.
- orarioEsame: Orario svolgimento appello.
- aulaEsame: Numero aula svolgimento esame.

Tabella bookingExams:
- idPrenotazione: Identificativo univoco della prenotazione (Chiave primaria).
- idEsamePre: Identificativo dell'appello prenotato (Chiave esterna).
- matricolaStudente: Codice univoco dello studente (Chiave esterna).

Tabella domandeQuest:
- idDomanda: Identificativo univoco delle domande (Chiave primaria).
- matricolaDomanda: Codice univoco dello studente (Chiave esterna).
- EsameDomanda: Nome dell'appello (Chiave esterna).
- domanda1: Prima domanda del questionario che conterrà la risposta.
- domanda2: Seconda domanda del questionario che conterrà la risposta.
- domanda3: Terza domanda del questionario che conterrà la risposta.

Tabella questionari:
- idQuest: Identificativo univoco del questionario (Chiave primaria).
- nomeExQuest: Nome dell'appello (Chiave esterna).
- effettuato: aggiorna lo stato da false a true se viene effettuato.

Tabella tasse:
- idTassa: Identificativo univoco delle tasse (Chiave primaria).
- matricolaTassa: Codice univoco dello studente (Chiave esterna).
- causale: Motivazione della tassa.
- importo: Ammontare della singola tassa.
- dataScadenza: Data di scadenza per il pagamento della tassa.
- pagata: aggiorna lo stato da false a true se viene pagata.

Tabella voti:
- idVoto: Identificativo univoco del voto (Chiave primaria).
- idPreVoto: Identificativo univoco della prenotazione (Chiave esterna).
- idEsameVoto: Identificativo dell'appello prenotato (Chiave esterna).
- nomeExVoto: Nome dell'appello (Chiave esterna).
- matricolaVoto: Codice univoco dello studente (Chiave esterna).
- voto: Valore intero del voto assegnato.
- conferma: aggiorna lo stato da false a true se viene confermato.

Le tabelle sono progettate per gestire i dati relativi agli appelli creati o prenotati, all' aggiunta di studenti nuovi, al pagamento delle tasse, compilazione dei questionari, e convalida dei voti.
Questa struttura permette una gestione integrata delle informazioni, consentendo al segretario, al docente e allo studente di avere tutte le informazioni necessarie per il compimento delle operazioni da loro richieste.