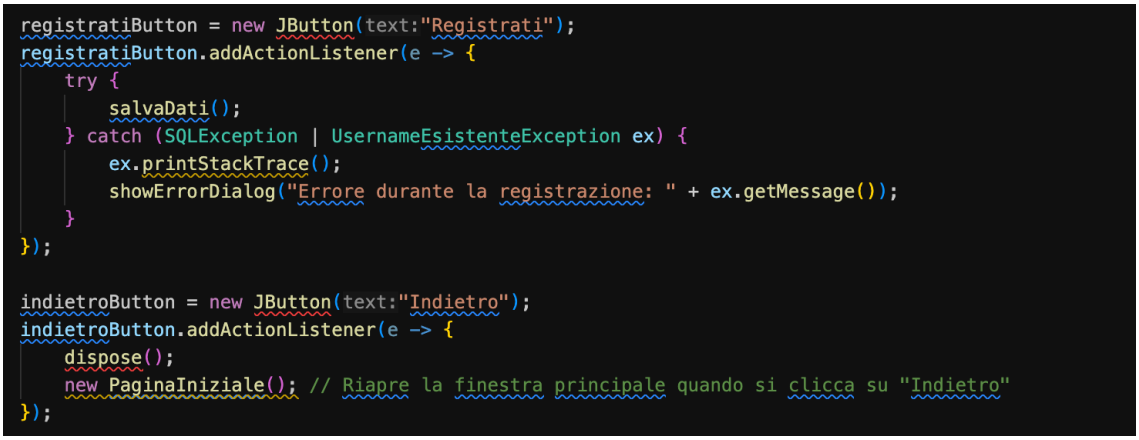
RegistrazioneFrame

- **Command Pattern**
  - **Dove si applica:** gestione delle azioni dei bottoni (registrazione e indietro)
  - **Motivazione:** l'azione da eseguire quando si preme un bottone è incapsulata all'interno di un oggetto ActionListener, separando la logica dell'azione dall'interfaccia utente. Questo è il concetto chiave del Command Pattern, che permette di cambiare o estendere facilmente il comportamento dei bottoni senza alterare la logica della finestra principale
  - codice rilevante:



```
registratiButton = new JButton(text:"Registrati");
registratiButton.addActionListener(e -> {
    try {
        salvaDati();
    } catch (SQLException | UsernameEsistenteException ex) {
        ex.printStackTrace();
        showErrorDialog("Errore durante la registrazione: " + ex.getMessage());
    }
});

indietroButton = new JButton(text:"Indietro");
indietroButton.addActionListener(e -> {
    dispose();
    new PaginaIniziale(); // Riapre la finestra principale quando si clicca su "Indietro"
});
```

- **Singleton Pattern**
  - **Dove si applica:** all’istanza di DatabaseManager
  - **Motivazione:** il databaseManager è implementato come un Singleton per garantire che esista una sola istanza durante l'esecuzione dell'applicazione, gestendo centralmente la connessione al database
  - codice rilevante:
    - private DatabaseManager databaseManager;
    - public RegistrazioneFrame(DatabaseManager databaseManager, ActionListener indietroListener) {
      this.databaseManager = databaseManager;

- **Facade Pattern**
  - **Dove si applica:** interfaccia utente per l'accesso e la gestione dei dati utente
  - **Motivazione:** fornisce una semplice interfaccia per un insieme complesso di classi o operazioni. Un Facade può essere utilizzato per semplificare l'interazione con il sistema di backend, nascondendo la complessità delle operazioni di database e validazione dietro un'interfaccia più semplice
  - codice rilevante:

```
private void salvaDati() throws SQLException, UsernameEsistenteException {
    String nome = nomeField.getText().trim();
    String cognome = cognomeField.getText().trim();
    String codiceFiscale = codiceFiscaleField.getText().trim();
    String email = emailField.getText().trim();
    String username = usernameField.getText().trim();
    String password = new String(passwordField.getPassword()).trim();
    String centroMonitoraggioStr = centroField.getText().trim();

    if (nome.isEmpty() || cognome.isEmpty() || codiceFiscale.isEmpty() || email.isEmpty() || username.isEmpty()
            || password.isEmpty() || centroMonitoraggioStr.isEmpty()) {
        showErrorDialog(message:"Per favore, completa tutti i campi obbligatori.");
        return;
    }

    if (!isValidCodiceFiscale(codiceFiscale)) {
        showErrorDialog(message:"Il codice fiscale non è valido.");
        return;
    }

    if (!isValidEmail(email)) {
        showErrorDialog(message:"L'email non è valida.");
        return;
    }

    if (databaseManager.verificaUsernameEsistente(username)) {
        showErrorDialog(message:"Username già presente nel database.");
        return;
    }

    if (password.length() < 8) {
        showErrorDialog(message:"La password deve essere lunga almeno 8 caratteri.");
        return;
    }

    databaseManager.inserisciDati(nome, cognome, codiceFiscale, email, username, password, centroMonitoraggio);
    JOptionPane.showMessageDialog(this, message:"Registrazione avvenuta con successo!", title:"Successo",
            JOptionPane.INFORMATION_MESSAGE);

    dispose(); // Chiude la finestra di registrazione
    new PaginaIniziale(); // Torna alla pagina iniziale
}
```

Riepilogo

- **Command Pattern**: incapsulamento delle azioni dei bottoni
- **Singleton Pattern**: gestione unica e centralizzata del DatabaseManager

- **Facade Pattern**: Interfaccia semplificata per operazioni complesse.