VisualizzaAreaGeografica

1. Observer Pattern

- **Dove si applica:** interazione tra la visualizzazione dei dati climatici e l'aggiornamento dell'interfaccia utente
- **Motivazione:** l'idea di aggiornare la vista (la infoTextArea e commentTextArea) in risposta a cambiamenti nello stato (come l'inserimento di un nome di area e la visualizzazione dei dati) si avvicina al concetto di Observer. Quando i dati vengono aggiornati, l'interfaccia utente viene modificata di conseguenza
- **Codice rilevante:**

// Codice nel metodo displayAreaInformation
infoTextArea.setText(sb.toString());

// Codice nel metodo loadComment
commentTextArea.setText(rs.getString("Note"));

2. Producer-Consumer Pattern

- **Dove si applica:** comunicazione tra l'interfaccia utente e il database
- **Motivazione:** il client (interfaccia utente) produce richieste di visualizzazione e salvataggio dei dati, e il server/database consuma queste richieste. La gestione delle richieste di visualizzazione e salvataggio dei commenti segue il modello Producer-Consumer, dove il client produce le richieste e il database gestisce i dati
- **Codice rilevante:**

```
visualizzaButton.addActionListener(e -> {
    String areaName = areaNameField.getText().trim();
    if (areaName.isEmpty()) {
        infoTextArea.setText(t:"Inserisci un nome di area valido.");
    } else {
        displayAreaInformation(areaName);
        loadComment(areaName); // Carica il commento esistente, se presente
    }
});

saveCommentButton.addActionListener(e -> saveComment(areaNameField.getText().trim(), commentTextArea.getText().trim()));
```

3. Strategy Pattern

- **Dove si applica:** elaborazione e visualizzazione delle informazioni climatiche
- **Motivazione:** diverse strategie per calcolare e formattare i dati climatici possono essere implementate e selezionate dinamicamente
- **Codice rilevante:**

```
sb.append("Numero di Rilevazioni: ").append(numRilevazioni).append("\n");
sb.append("Temperatura Media: ").append(temperaturaFormat.format(avgTemperatura)
sb.append("Umidità Media: ").append(umiditaFormat.format(avgUmidita)).append(" %
sb.append("Pressione Atmosferica Media: ").append(pressioneAtmosfericaFormat.for
sb.append("Velocità del Vento Media: ").append(velocitaVentoFormat.format(avgVel
```

4. Singleton Pattern

- **Dove si applica:** Gestione della connessione al database.
- **Motivazione:** Il Singleton Pattern può essere usato per gestire una singola istanza di DatabaseManager, assicurando che solo una connessione al database esista durante l'esecuzione dell'applicazione. Sebbene non venga mostrato esplicitamente nel codice, il pattern Singleton è implicito nella creazione e gestione della connessione al database
- **Codice rilevante**:

```
public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        try {
            DatabaseManager dbManager = new DatabaseManager();
            new VisualizzaAreaGeograficaFrame(dbManager).setVisible(true);
        } catch (SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "Errore di connessione al database: " + e.getMessage(), "Errore", JOptio
        }
    });
}
```