CercaAreaGeograficaFrame

1. Singleton Pattern

- **Dove si applica:** DatabaseManager
- **Motivazione:** il DatabaseManager gestisce la connessione al database e dovrebbe essere un'unica istanza per garantire la gestione centralizzata delle risorse. L'uso del Singleton Pattern in questo contesto è utile per evitare la creazione di più istanze di DatabaseManager e gestire efficacemente le connessioni al database
- **Codice:**

  istanza del DatabaseManager

2. Observer Pattern

- **Dove si applica:** gestione della cronologia delle ricerche (searchHistory)
- **Motivazione:** la cronologia delle ricerche (searchHistory) viene aggiornata e visualizzata nella JTextArea (historyArea). Applicare l'Observer Pattern permetterebbe di automatizzare l'aggiornamento della vista (historyArea) ogni volta che i dati (searchHistory) cambiano, migliorando la separazione tra la logica di gestione dei dati e la loro visualizzazione
- **Codice rilevante**:

```
private void updateHistory() {
    historyArea.setText(""); // Cancella il contenuto dell'area di testo
    for (int i = searchHistory.size() - 1; i >= 0; i--) {
        historyArea.insert(searchHistory.get(i) + "\n", 0); // Inserisci l'elemento in cima alla pila
    }
}

private void clearSearchHistory() {
    searchHistory.clear();
    updateHistory();
}
```

3. MVC (Model-View-Controller) Pattern

- **Dove si applica:** intera struttura dell'applicazione GUI

- **Motivazione:** separare la logica di business (Modello), l'interfaccia utente (Vista), e il controllo dell'interazione (Controller). Attualmente, il codice mescola la logica dell'applicazione con la logica di presentazione nella classe CercaAreaGeograficaFrame, il che rende il codice meno modulare e più difficile da manutenere
- **Codice Rilevante**

```
// Pannelli per la GUI
private JPanel searchPanel;
private JPanel buttonPanel;
private JPanel historyPanel;

// Costruzione dei pannelli
searchPanel = new JPanel(new GridLayout(3, 2, 10, 10));
buttonPanel = new JPanel();
historyPanel = new JPanel(new BorderLayout());

// Configurazione dei pulsanti e delle azioni
searchButton.addActionListener(this);
clearButton.addActionListener(this);
homeButton.addActionListener(this);
paginaInizialeButton.addActionListener(this);

// Aggiunta dei pannelli al frame
add(searchPanel, BorderLayout.NORTH);
add(buttonPanel, BorderLayout.CENTER);
add(historyPanel, BorderLayout.SOUTH);
```

4. Factory Method Pattern

- **Dove si applica:** creazione di istanze di AreaGeografica
- **Motivazione:** il codice per la creazione di istanze di AreaGeografica dal file CSV si trova nel metodo caricaAreeDaCSV. L'utilizzo del Factory Method Pattern potrebbe isolare la logica di creazione in un unico punto, facilitando la manutenzione e l'estensione

- **Codice rilevante:**

```
private List<AreaGeografica> caricaAreeDaCSV(String filePath) {
    List<AreaGeografica> aree = new ArrayList<>();
    try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
        String line;
        while ((line = br.readLine()) != null) {
            String[] values = line.split(";");
            if (values.length >= 6) {
                String nome = values[1].trim();
                String stato = values[4].trim();
                String[] coordinates = values[5].trim().split(",");
                if (coordinates.length == 2) {
                    double latitudine;
                    double longitudine;
                    try {
                        latitudine = Double.parseDouble(coordinates[0].trim());
                        longitudine = Double.parseDouble(coordinates[1].trim())
                    } catch (NumberFormatException ex) {
                        continue;
                    }
                    AreaGeografica area = new AreaGeografica(nome, stato, latit
                    aree.add(area);
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return aree;
```

5. Strategy Pattern

- **Dove si applica:** metodi di ricerca (searchBynameOrState e searchByCoordinates)
- **Motivazione:** i metodi searchBynameOrState e searchByCoordinates implementano due strategie di ricerca diverse. Applicare il Strategy Pattern permetterebbe di definire queste strategie di ricerca in classi separate e intercambiabili, facilitando l'estensione e la manutenzione

```
private void searchByNameOrState(String searchTerm) {
    List<AreaGeografica> risultati = new ArrayList<>();
    for (AreaGeografica area : aree) {
        if (area.getNome().toLowerCase().contains(searchTerm.toLowerCase()) ||
            area.getStato().toLowerCase().contains(searchTerm.toLowerCase())) {
            risultati.add(area);
        }
    }
    mostraRisultati(risultati);
    searchHistory.add("Ricerca per nome/stato: " + searchTerm);
    updateHistory();
}

private void searchByCoordinates(double latitude, double longitude) {
    AreaGeografica closestArea = null;
    double closestDistance = Double.MAX_VALUE;
    for (AreaGeografica area : aree) {
        double distance = calculateDistance(latitude, longitude, area.getLatitud
        if (distance < closestDistance) {
            closestDistance = distance;
            closestArea = area;
        }
    }
    if (closestArea != null) {
        mostraRisultati(List.of(closestArea));
        searchHistory.add("Ricerca per coordinate: Latitudine=" + latitude + ",
    } else {
        JOptionPane.showMessageDialog(this, "Nessuna area trovata per le coordin
    }
    updateHistory();
}
```

- **Codice rilevante:**

6. Command Pattern

- **Dove si applica:** azioni dei pulsanti (searchButton, clearButton, homeButton e paginaInizialeButton)
- **Motivazione:** per incapsulare le azioni dei pulsanti in oggetti comando, facilitando l'esecuzione delle azioni associate
- **Codice rilevante:**

- **metodo actionPerformed con il controllo su tutti i bottoni che vengono schiacciati**