Feature Envy GeneralSearch

Updated `GeneralSearch` **Class**

Remove the direct manipulation of the `SearchLog` from `GeneralSearch`. Instead, delegate logging and usage updates to the `SearchLog` class.

java Copiar código

```java
package org.example.studysearch;

import org.example.studycards.CardManager;
import org.example.studyplanner.HabitTracker;
import org.example.studyplanner.TodoTracker;
import org.example.studyregistry.StudyMaterial;
import org.example.studyregistry.StudyTaskManager;

import java.util.ArrayList;
import java.util.List;

public class GeneralSearch implements Search<String> {
    private SearchLog searchLog = new SearchLog("General Search");

    public GeneralSearch() {}
```

Copiar código

```java
@Override
public List<String> search(String text) {
    return handleSearch(text);
}

public SearchLog getSearchLog() {
    return searchLog;
}

private List<String> handleSearch(String text) {
    List<String> results = new ArrayList<>();
    results.addAll(CardManager.getCardManager().searchInCards(text));
    results.addAll(HabitTracker.getHabitTracker().searchInHabits(text));
    results.addAll(TodoTracker.getInstance().searchInTodos(text));
    results.addAll(StudyMaterial.getStudyMaterial().searchInMaterials(text));
    results.addAll(StudyTaskManager.getStudyTaskManager().searchInRegistries(text)

    // Delegate logging responsibilities to the SearchLog class
    this.searchLog.logSearch(text);

    results.add("\nLogged in: " + this.searchLog.getLogName());
```

Key Changes

1. **Encapsulated Log Updates**:
   - The `SearchLog` class now manages its own updates using the `logSearch` method.
   - `GeneralSearch` no longer directly modifies the search history or usage count.
2. **Simplified `GeneralSearch`**:
   - The `GeneralSearch` class is now focused purely on orchestrating the search and delegating logging responsibilities.

Benefits of Refactoring

- **Encapsulation**: The `SearchLog` class is now responsible for maintaining and managing its own state.
- **Cleaner Code**: The `GeneralSearch` class no longer accesses or modifies the internal state of `SearchLog` directly, reducing coupling.
- **Ease of Maintenance**: Future changes to logging behavior will only require updates to the `SearchLog` class.

Feature Envy MaterialSearch

To fix the **feature envy** code smell in this code using the **Move Method** refactoring technique, we should analyze the methods that are being called from the `StudyMaterial` and `searchLog` objects. Since these methods are heavily reliant on their own classes (`StudyMaterial` and `searchLog`), the responsibilities should be moved to those classes.

Here's the refactored version of the code:

Step 1: Move Responsibility to `StudyMaterial`

Move the `searchInMaterials` logic to a method within the `StudyMaterial` class.

java Copiar código

```java
public class StudyMaterial {
    // Existing methods and fields

    public List<String> searchAndLog(String text, SearchLog searchLog) {
        List<String> results = searchInMaterials(text);
        searchLog.addSearchHistory(text);
        searchLog.incrementUsage();
        results.add("\nLogged in: " + se nLog.getLogName());
        return results;
```

Feature Envy RegistrySearch

como ficari ao codigo completo?

Aqui está o código completo para todas as classes envolvidas no seu sistema de busca. Vou estruturar o código para que seja funcional, com base no que você forneceu e seguindo as melhores práticas.

---

1. RegistrySearch

java Copiar código

```java
package org.example.studysearch;

import org.example.studycards.CardManager;
import org.example.studyplanner.HabitTracker;
import org.example.studyplanner.TodoTrac
import org.example.studyregistry.StudyTaskManager;
```

java Copiar código

```java
package org.example.studysearch;

import org.example.studycards.CardManager;
import org.example.studyplanner.HabitTracker;
import org.example.studyplanner.TodoTracker;
import org.example.studyregistry.StudyTaskManager;

import java.util.ArrayList;
import java.util.List;

public class RegistrySearch implements Search<String> {
    private SearchLog searchLog = new SearchLog("Registry Search");

    public RegistrySearch() {}

    @Override
    public List<String> search(String text) {
        return handleRegistrySearch(text
    }
```

Feature Envy TimelineView

```java
package org.example.studyplanner;

public class TimelineView {

    public String habitDateViewAll(HabitTracker ht) {
        return ht.getFormattedHabits();
    }
}
```

A lógica que originalmente estava no método `habitDateViewAll` da classe `TimelineView` foi movida para a classe `HabitTracker` na forma de um método chamado `getFormattedHabits`. Isso foi feito porque o código anterior dependia muito de dados e métodos de `HabitTracker`, o que é um indicativo de **Feature Envy**. Agora, a responsabilidade pela formatação pertence à classe que contém os dados, promovendo coesão e encapsulamento.

Feature Envy SearchLog

Aqui está a refatoração, explicando o problema de **Feature Envy** e como utilizei a técnica de **Move Method** para resolver:

Código Refatorado

java Copiar código

```java
package org.example.studysearch;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class SearchLog {
    private List<String> searchHistory;
```

Copiar código

```java
public class SearchLog {
    private List<String> searchHistory;
    private Map<String, Integer> searchCount;
    private boolean isLocked;
    private Integer numUsages;
    private String logName;

    public SearchLog(String logName) {
        searchHistory = new ArrayList<>();
        searchCount = new HashMap<>();
        this.logName = logName;
        numUsages = 0;
        isLocked = false;
    }

    public List<String> logSearchAndSummary(String text) {
        logSearch(text); // Registrar busca
        return generateSummary(); // Delegar para o método encapsulado
    }

    public List<String> generateSummary() {
```

Copiar código

```java
    public List<String> generateSummary() {
        List<String> summary = new ArrayList<>();
        summary.add("Log Name: " + logName);
        summary.add("Number of Searches: " + numUsages);
        summary.add("Search History: " + searchHistory.toString());
        return summary;
    }

    public void logSearch(String text) {
        addSearchHistory(text);
        numUsages++;
    }

    public void addSearchHistory(String searchHistory) {
        this.searchHistory.add(searchHistory);
    }

    public List<String> getSearchHistory() {
        return searchHistory;
    }
```

```java
    public List<String> getSearchHistory() {
        return searchHistory;
    }

    public String getLogName() {
        return logName;
    }
}
```

Justificativa da Refatoração

1. **Problema de Feature Envy:**
   - Métodos como `generateLogEntry` e `updateAndGetName` manipulavam vários dados e métodos relacionados ao histórico de busca (`searchHistory`, `numUsages`, etc.) sem agregar muito valor à classe. Isso indica **Feature Envy**, onde a lógica está em um lugar que não pertence.
2. **Solução: Move Method**
   - A lógica de resumo e manipulação de busca foi consolidada na classe `SearchLog`. Criei o método `generateSummary` para encapsular a lógica de criação do resumo, promovendo coesão.
3. **Resultados:**
   - O código agora é mais conciso, com responsabilidade claramente definida.
   - As operações relacionadas ao histórico de buscas e ao resumo permanecem na classe que contém os dados (`SearchLog`), o que melhora o encapsulamento.

Feature Envy StudyCardsController

StudyCardsController

java Copiar código

```java
public class StudyCardsController {
    private LeitnerSystem leitnerSystem;

    public void handleGetRandomCardFromBox() {
        try {
            // Delegates to LeitnerSystem
            String response = leitnerSystem.getRandomCardFromBox();
            System.out.println(response);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Benefits of This Refactoring

1. **Cleaner Design**:
   - `LeitnerSystem` now encapsulates its behavior, and the controller only coordinates actions.
2. **Ease of Maintenance**:
   - Changes to how `LeitnerSystem` retrieves random cards or combines data only need to be updated in `LeitnerSystem`.
3. **Reusability**:
   - The `getRandomCardFromBox` method in `LeitnerSystem` can now be reused in other parts of the application without depending on `StudyCardsController`.

This refactoring eliminates the **Feature Envy** and adheres to the **Single Responsibility Principle**, making the codebase cleaner and easier to maintain. 😊