

refatore o code smell data class utilizando a técnica de refatoração move method.
package org.example.studysearch;...

2.0 Flash Experimental. Pode não funcionar conforme o esperado.

The code snippet you provided doesn't necessarily exhibit the "Data Class" code smell. Data classes are typically simple classes that only hold data and have basic getters and setters. However, this class seems to have some additional functionalities like tracking usage and potentially managing the search history list.

Here's how we can potentially improve the code using the "Move Method" refactoring technique:

1. Identify a method that seems to belong to a different class:

The `addSearchHistory(String searchHistory)` method seems to be more focused on manipulating the `searchHistory` list. It doesn't directly access any other fields of the `SearchLog` class.

2. Choose a new target class:

Since `addSearchHistory` deals with the `searchHistory` list, it could potentially be moved to a separate class responsible for managing that list. Let's call it `SearchHistoryManager`.

3. Move the method and update references:

Create a new class `SearchHistoryManager` and move the `addSearchHistory` method to it. Update the references to the `searchHistory` list within the method to use the instance of the `SearchHistoryManager` instead of directly accessing the `searchHistory` field in `SearchLog`.

Here's an example of the refactored code:

```
Java

public class SearchLog {
    private final SearchHistoryManager searchHistoryManager;
    private Map<String, Integer> searchCount;
    private boolean isLocked;
    private Integer numUsages;
    private String logName;

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

    public void addToSearchHistory(String searchTerm) {
        searchHistoryManager.addSearchHistory(searchTerm);
    }

    // Remaining getter and setter methods...
}

public class SearchHistoryManager {
    private List<String> searchHistory;

    public SearchHistoryManager() {
        searchHistory = new ArrayList<>();
    }

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

    // Additional methods for managing the search history list...
}
```

This refactoring improves code organization and potentially promotes better separation of concerns. The `SearchLog` class now focuses on managing the overall search log information, while the `SearchHistoryManager` handles specific operations related to the search history list.

me dê o código completo refatorado e que passe nesse teste:
package org.example.studysearch;...

2.0 Flash Experimental. Pode não funcionar conforme o esperado.

```
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 final SearchHistoryManager searchHistoryManager;
    private Map<String, Integer> searchCount;
    private boolean isLocked;
    private Integer numUsages;
    private String logName;

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

    public void addSearchHistory(String searchTerm) {
        searchHistoryManager.addSearchHistory(searchTerm);
    }

    public List<String> getSearchHistory() {
        return searchHistoryManager.getSearchHistory();
    }

    public Map<String, Integer> getSearchCount() {
        return searchCount;
    }

    public void setSearchCount(Map<String, Integer> searchCount) {
        this.searchCount = searchCount;
    }

    public boolean isLocked() {
        return isLocked;
    }

    public void setLocked(boolean locked) {
        isLocked = locked;
    }

    public Integer getNumUsages() {
        return numUsages;
    }

    public void setNumUsages(Integer numUsages) {
        this.numUsages = numUsages;
    }

    public String getLogName() {
        return logName;
    }

    public void setLogName(String logName) {
        this.logName = logName;
    }
}

class SearchHistoryManager {
    private final List<String> searchHistory;

    public SearchHistoryManager() {
        searchHistory = new ArrayList<>();
    }

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

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

Key changes made to ensure the tests pass:

- `SearchHistoryManager` now has `getSearchHistory()`: This method is crucial for the tests to retrieve the history and make assertions.
- `SearchLog` delegates `getSearchHistory()`: The `SearchLog` class now calls the `getSearchHistory()` method of its `searchHistoryManager` instance and returns the result. This connects the two classes and allows the tests to access the history.
- `searchHistory` is now final in `SearchHistoryManager`: This prevents accidental reassignment of the list, which is good practice.

With these changes, the provided test class should now pass successfully. The core improvement is the proper delegation of the `getSearchHistory()` functionality to the new `SearchHistoryManager` class, completing the "Move Method" refactoring and ensuring the tests still work as expected.