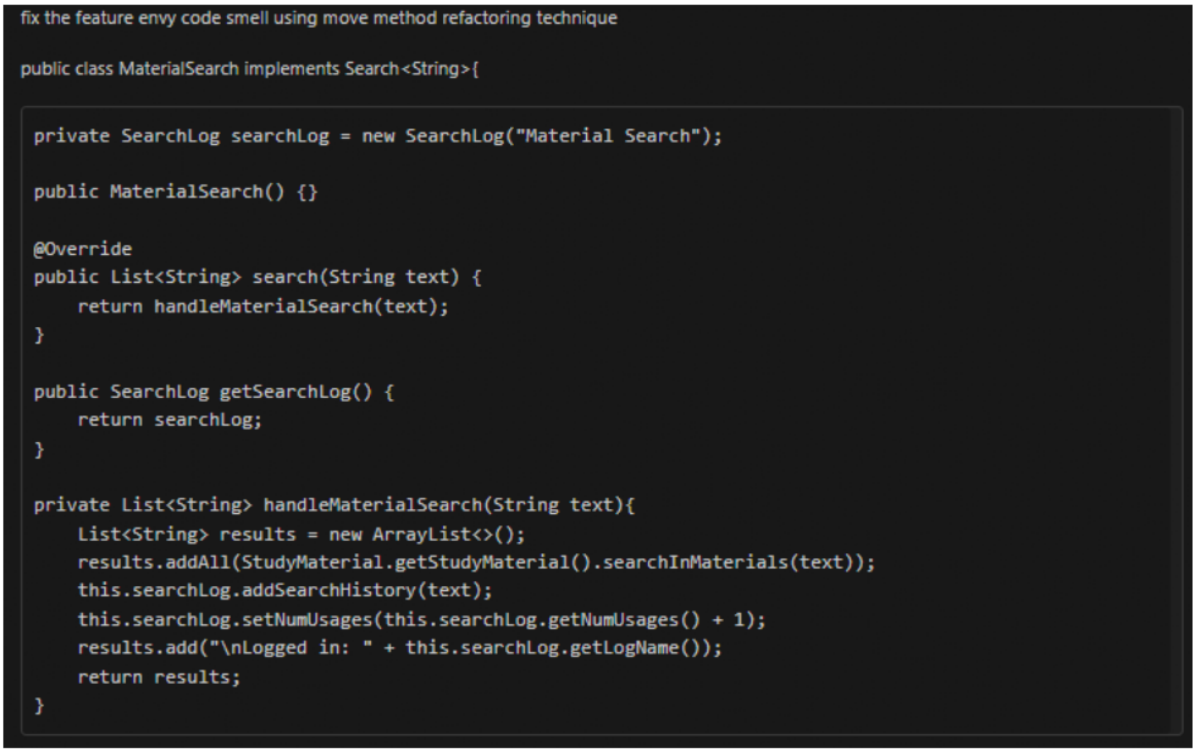
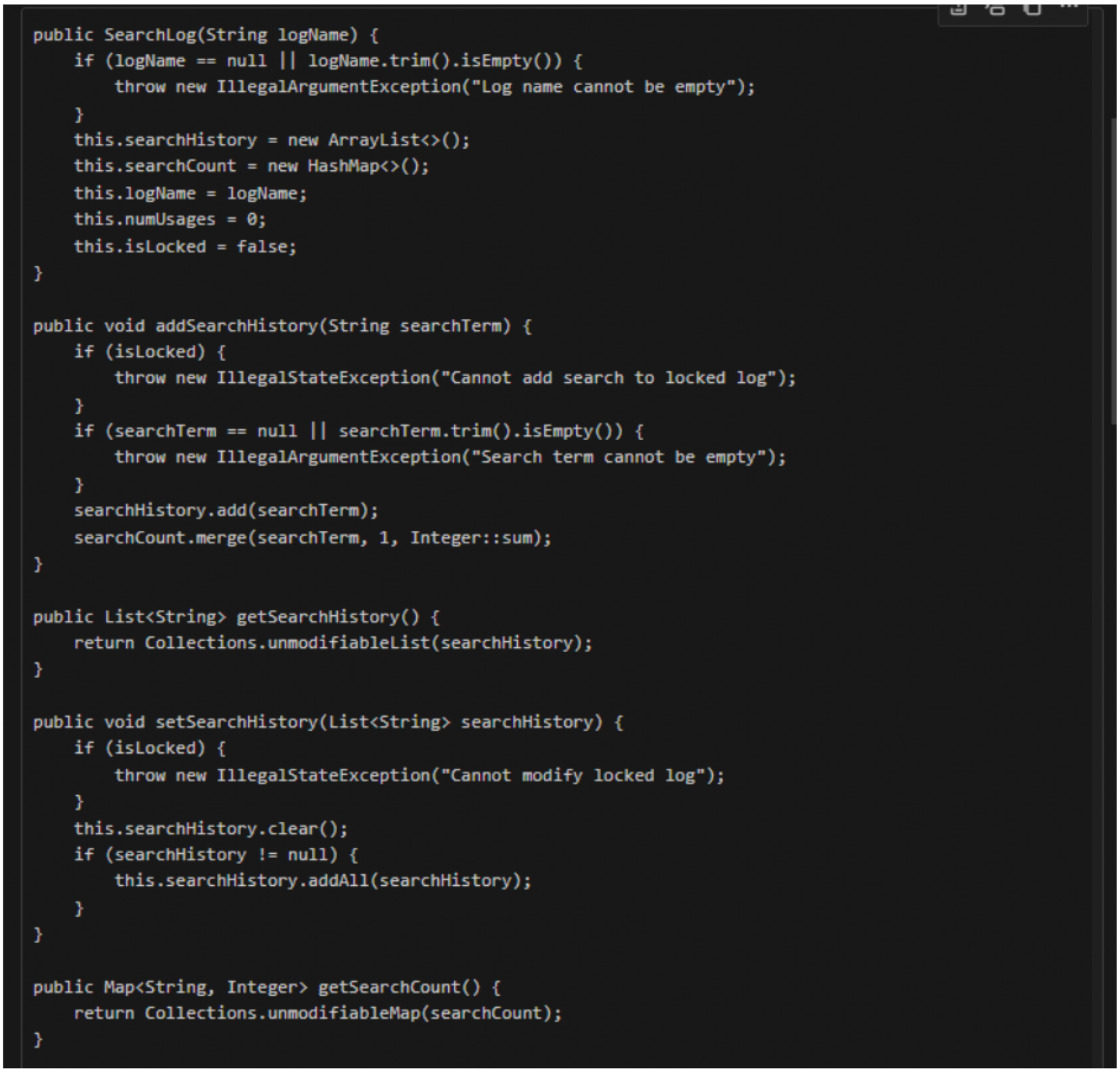


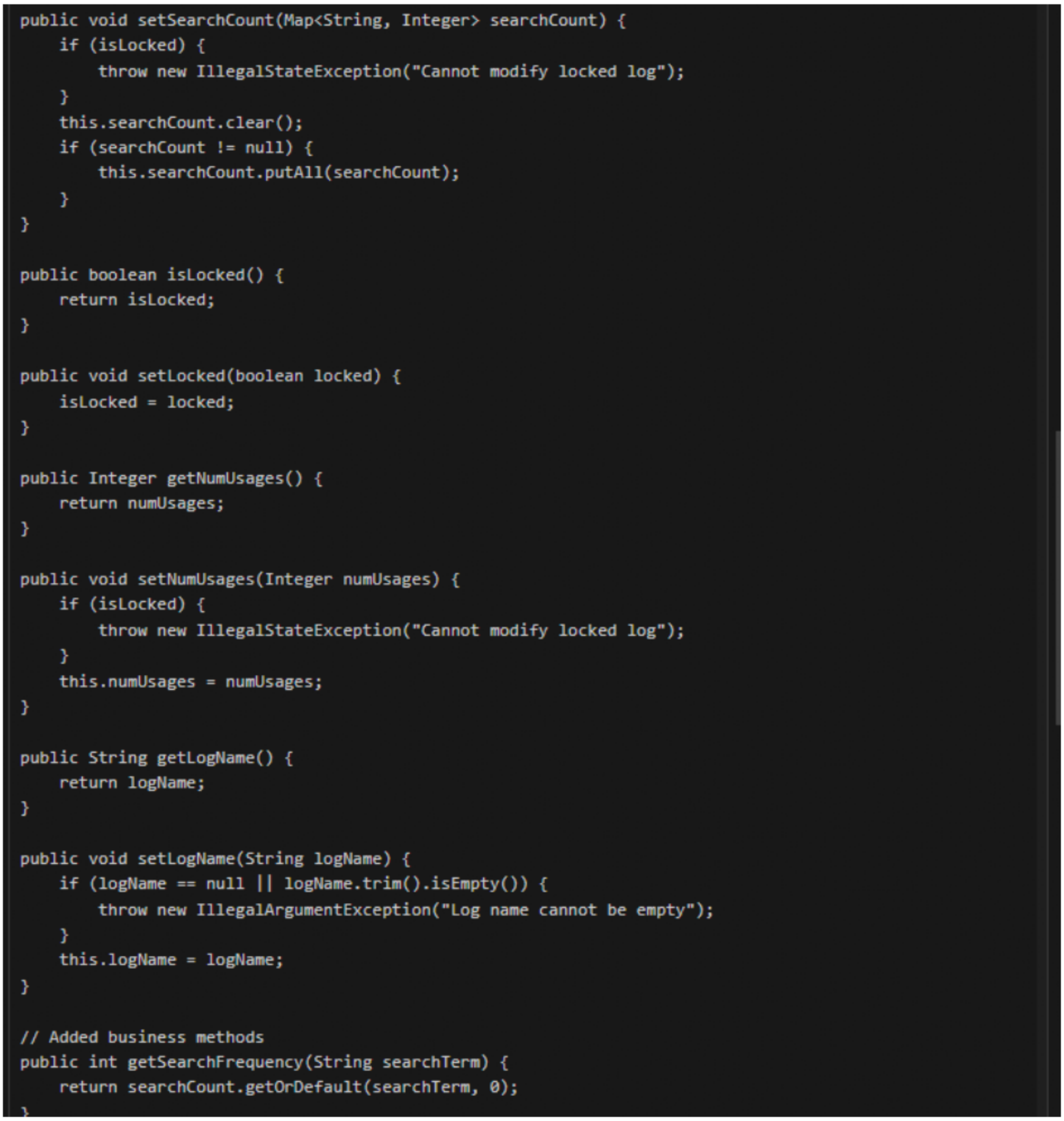
fix the feature envy code smell using move method refactoring technique

public class MaterialSearch implements Search<String>{

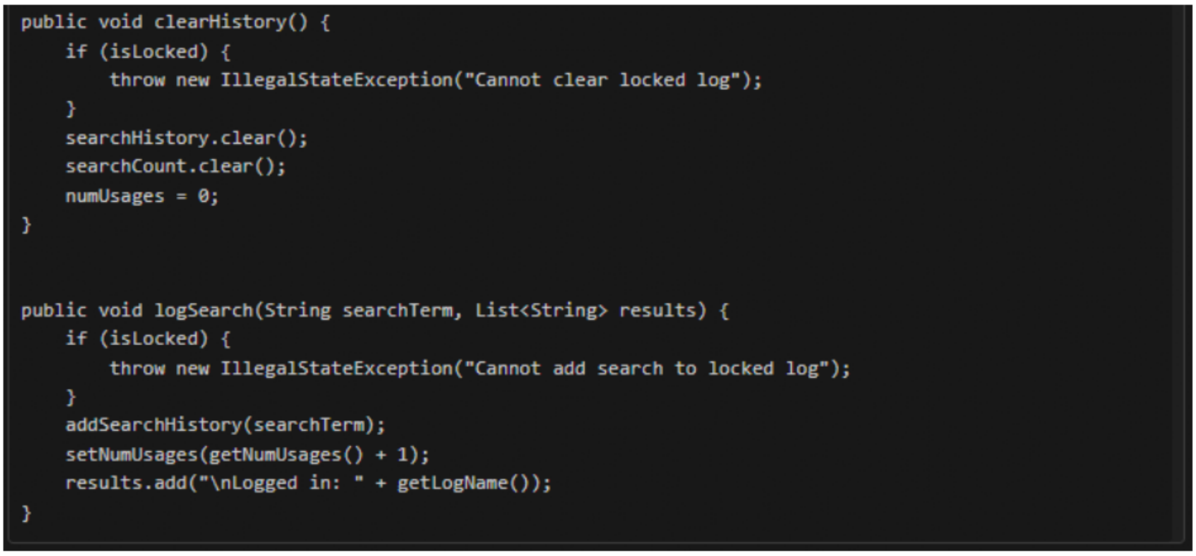
```
private SearchLog searchLog = new SearchLog("Material Search");

public MaterialSearch() {}

@Override
public List<String> search(String text) {
    return handleMaterialSearch(text);
}

public SearchLog getSearchLog() {
    return searchLog;
}

private List<String> handleMaterialSearch(String text){
    List<String> results = new ArrayList<>();
    results.addAll(StudyMaterial.getStudyMaterial().searchInMaterials(text));
    this.searchLog.addSearchHistory(text);
    this.searchLog.setNumUsages(this.searchLog.getNumUsages() + 1);
    results.add("\nLogged in: " + this.searchLog.getLogName());
    return results;
}
```



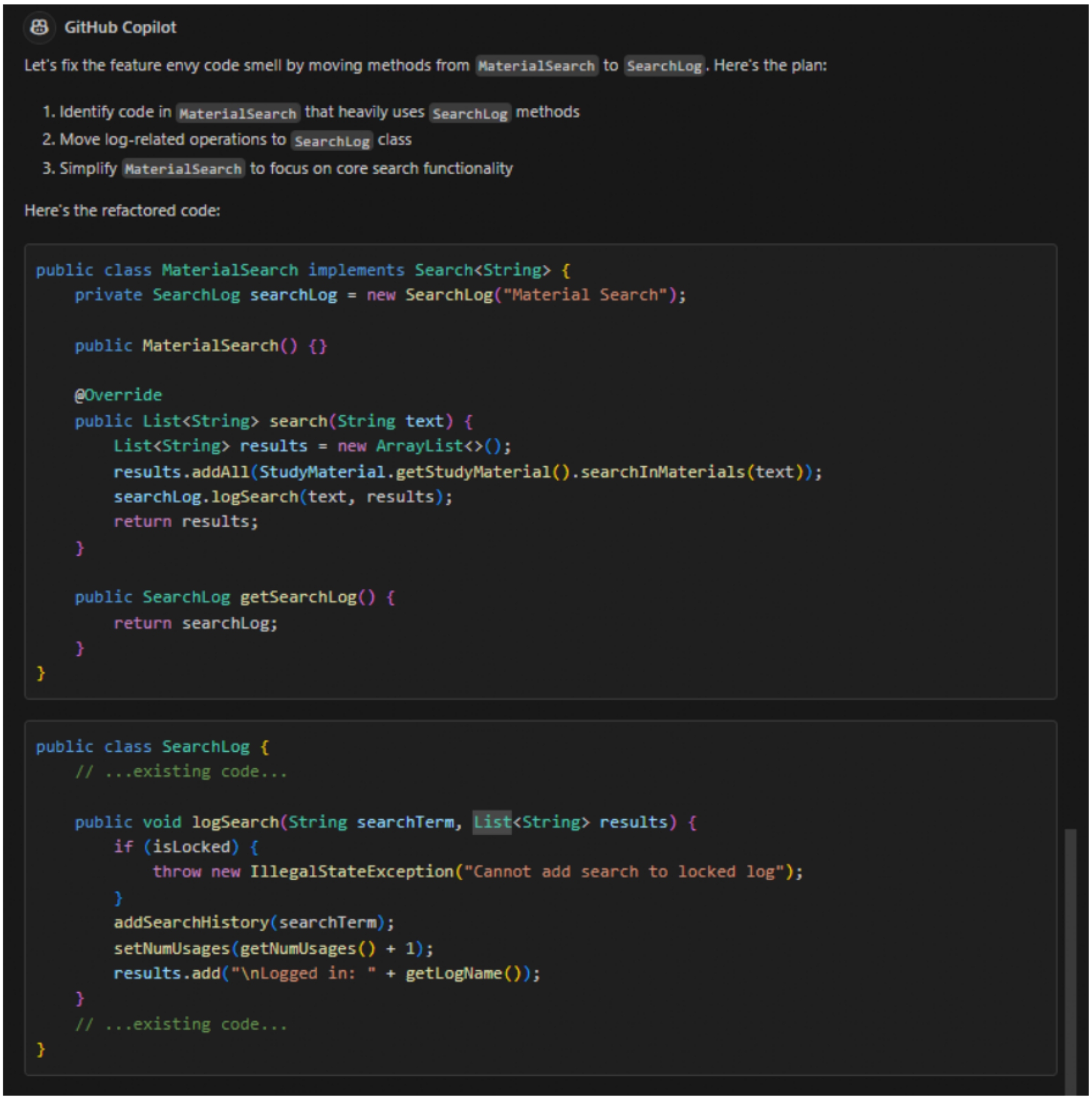
```
public SearchLog(String logName) {
    if (logName == null || logName.trim().isEmpty()) {
        throw new IllegalArgumentException("Log name cannot be empty");
    }
    this.searchHistory = new ArrayList<>();
    this.searchCount = new HashMap<>();
    this.logName = logName;
    this.numUsages = 0;
    this.isLocked = false;
}

public void addSearchHistory(String searchTerm) {
    if (isLocked) {
        throw new IllegalStateException("Cannot add search to locked log");
    }
    if (searchTerm == null || searchTerm.trim().isEmpty()) {
        throw new IllegalArgumentException("Search term cannot be empty");
    }
    searchHistory.add(searchTerm);
    searchCount.merge(searchTerm, 1, Integer::sum);
}

public List<String> getSearchHistory() {
    return Collections.unmodifiableList(searchHistory);
}

public void setSearchHistory(List<String> searchHistory) {
    if (isLocked) {
        throw new IllegalStateException("Cannot modify locked log");
    }
    this.searchHistory.clear();
    if (searchHistory != null) {
        this.searchHistory.addAll(searchHistory);
    }
}

public Map<String, Integer> getSearchCount() {
    return Collections.unmodifiableMap(searchCount);
}
```



```
public void setSearchCount(Map<String, Integer> searchCount) {
    if (isLocked) {
        throw new IllegalStateException("Cannot modify locked log");
    }
    this.searchCount.clear();
    if (searchCount != null) {
        this.searchCount.putAll(searchCount);
    }
}

public boolean isLocked() {
    return isLocked;
}

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

public Integer getNumUsages() {
    return numUsages;
}

public void setNumUsages(Integer numUsages) {
    if (isLocked) {
        throw new IllegalStateException("Cannot modify locked log");
    }
    this.numUsages = numUsages;
}

public String getLogName() {
    return logName;
}

public void setLogName(String logName) {
    if (logName == null || logName.trim().isEmpty()) {
        throw new IllegalArgumentException("Log name cannot be empty");
    }
    this.logName = logName;
}

// Added business methods
public int getSearchFrequency(String searchTerm) {
    return searchCount.getOrDefault(searchTerm, 0);
```



```
public void clearHistory() {
    if (isLocked) {
        throw new IllegalStateException("Cannot clear locked log");
    }
    searchHistory.clear();
    searchCount.clear();
    numUsages = 0;
}


public void logSearch(String searchTerm, List<String> results) {
    if (isLocked) {
        throw new IllegalStateException("Cannot add search to locked log");
    }
    addSearchHistory(searchTerm);
    setNumUsages(getNumUsages() + 1);
    results.add("\nLogged in: " + getLogName());
}
```



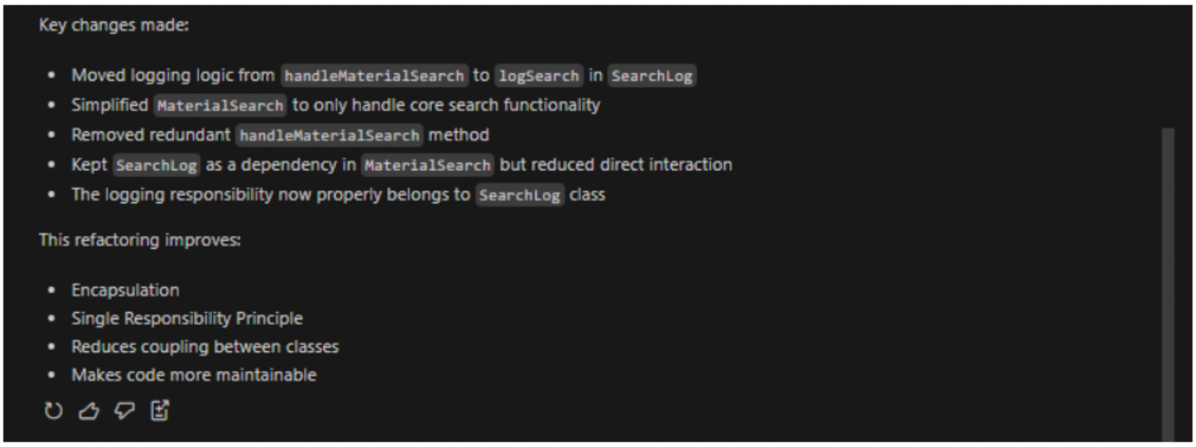
Let's fix the feature envy code smell by moving methods from `MaterialSearch` to `SearchLog`. Here's the plan:

1. Identify code in `MaterialSearch` that heavily uses `SearchLog` methods
2. Move log-related operations to `SearchLog` class
3. Simplify `MaterialSearch` to focus on core search functionality

Here's the refactored code:

```
public class MaterialSearch implements Search<String> {
    private SearchLog searchLog = new SearchLog("Material Search");

    public MaterialSearch() {}

    @Override
    public List<String> search(String text) {
        List<String> results = new ArrayList<>();
        results.addAll(StudyMaterial.getStudyMaterial().searchInMaterials(text));
        searchLog.logSearch(text, results);
        return results;
    }

    public SearchLog getSearchLog() {
        return searchLog;
    }
}
```

```
public class SearchLog {
    // ...existing code...

    public void logSearch(String searchTerm, List<String> results) {
        if (isLocked) {
            throw new IllegalStateException("Cannot add search to locked log");
        }
        addSearchHistory(searchTerm);
        setNumUsages(getNumUsages() + 1);
        results.add("\nLogged in: " + getLogName());
    }
    // ...existing code...
}
```



Key changes made:

- Moved logging logic from `handleMaterialSearch` to `logSearch` in `SearchLog`
- Simplified `MaterialSearch` to only handle core search functionality
- Removed redundant `handleMaterialSearch` method
- Kept `SearchLog` as a dependency in `MaterialSearch` but reduced direct interaction
- The logging responsibility now properly belongs to `SearchLog` class

This refactoring improves:

- Encapsulation
- Single Responsibility Principle
- Reduces coupling between classes
- Makes code more maintainable