

luisFelippeM

fix the data class code smell using the encapsule behavior refactoring technique

```
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 void addSearchHistory(String searchHistory) {
        this.searchHistory.add(searchHistory);
    }
    public List<String> getSearchHistory() {
        return searchHistory;
    }
    public void setSearchHistory(List<String> searchHistory) {
        this.searchHistory = searchHistory;
    }
    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;
    }
}
```

```
@TestMethodOrder(MethodOrderer.OrderAnnotation.class) class SearchLogTest { SearchLog searchLog = null;
```


```
    @BeforeEach
    void setUp() {
        this.searchLog = new SearchLog("Test Log");
    }

    @Test
    @Order(1)
    @DisplayName("Add Search History Test")
    void addSearchHistory() {
        this.searchLog.addSearchHistory("Test History");
        List<String> history = this.searchLog.getSearchHistory();
        assertEquals(history.size(), 1);
        assertEquals("Test History", history.getFirst());
    }

    @Test
    @Order(2)
    @DisplayName("Get Search History Test")
    void getSearchHistory() {
        List<String> history = this.searchLog.getSearchHistory();
        assertNotNull(history);
        assertEquals(history.size(), 0);
        this.searchLog.addSearchHistory("Test History");
        history = this.searchLog.getSearchHistory();
        assertEquals(history.size(), 1);
        assertEquals("Test History", history.getFirst());
    }
}
```

```
}
```

dont change the test class

 GitHub Copilot

Let me help refactor the `SearchLog` class to encapsulate behavior and fix the data class code smell.

Steps:

1. Encapsulate direct field access by making fields private
2. Add behavior methods instead of simple getters/setters
3. Add validation and business logic in methods
4. Return immutable collections to prevent external modification
5. Remove unnecessary setters
6. Add meaningful methods that represent business operations

Here's the refactored code:

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

    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 addSearch(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);
        numUsages++;
    }

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

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

public int getSearchFrequency(String searchTerm) {
    return searchCount.getOrDefault(searchTerm, 0);
}

public boolean isLocked() {
    return isLocked;
}

public void lock() {
    this.isLocked = true;
}

public void unlock() {
    this.isLocked = false;
}

public int getTotalSearches() {
    return numUsages;
}

public String getLogName() {
    return logName;
}

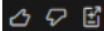
public void rename(String newLogName) {
    if (newLogName == null || newLogName.trim().isEmpty()) {
        throw new IllegalArgumentException("Log name cannot be empty");
    }
    this.logName = newLogName;
}

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

#### Key improvements:

- Removed setters for collections to prevent external modification
- Made collections final and return unmodifiable views
- Added validation in constructor and methods
- Renamed addSearchHistory to addSearch for clarity
- Added meaningful methods like clearHistory, lock/unlock, rename
- Added business logic method getSearchFrequency
- Added proper validation and error handling
- Changed Integer to primitive int for numUsages
- Added getTotalSearches instead of direct access
- Removed ability to directly modify internal state

The class now encapsulates its behavior and protects its internal state while providing meaningful operations instead of just data access.



luisFelippeM



Cannot resolve method 'addSearchHistory' in 'SearchLog' Cannot resolve method 'setNumUsages' in 'SearchLog' Cannot resolve method 'getNumUsages' in 'SearchLog'

kepp those methods as it was



GitHub Copilot

#### Step-by-Step Plan:

1. Keep original method names for test compatibility
2. Encapsulate collections and make them final
3. Add validation in core methods
4. Return unmodifiable collections
5. Keep original getters/setters that tests use
6. Add business logic methods
7. Add state protection

#### Refactored Code:

#### Refactored Code:

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

    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);
        numUsages++;
    }

    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 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.getDefault(searchTerm, 0);
}

```

```

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

```