

```

package org.example.studyplanner;

import java.time.LocalDateTime; import java.time.LocalTime; import java.time.format.DateTimeFormatter; import java.util.*;

public class HabitTracker { private List<Habit> habits; private Map<Integer, List<LocalDateTime>> tracker; private Integer nextId;

    private static HabitTracker instance;

    public static HabitTracker getHabitTracker() {
        if (instance == null) {
            instance = new HabitTracker();
        }
        return instance;
    }

    private HabitTracker(){
        this.habits = new ArrayList<>();
        this.tracker = new HashMap<>();
        this.nextId = 1;
    }

    @Override
    public String toString() {
        StringBuilder response = new StringBuilder();
        for (Habit habit : habits) {
            response.append(habit.toString()).append(", ");
        }
        return "Habits: " + response.toString();
    }

    public Habit getHabitById(Integer id){
        return this.habits.stream()
            .filter(habit -> Objects.equals(habit.getId(), id))
            .findFirst().orElse(null);
    }

    public List<Habit> getHabits() {
        return this.habits;
    }
}

```

```

public String formatHabitDate(LocalDateTime date){
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");

    return date.format(formatter);
}

public List<Integer> getTrackerKeys(){
    return this.tracker.keySet().stream().toList();
}

public int handleAddHabitAdapter(List<String> stringProperties, List<Integer> intProperties, boolean isConcluded) {
    HabitRequest.HabitDetails details = new HabitRequest.HabitDetails()
        .setDailyDedication(intProperties.get(1), intProperties.get(0))
        .setStartDate(LocalDateTime.of(
            intProperties.get(2), // year
            intProperties.get(3), // month
            intProperties.get(4), // day
            intProperties.get(5), // hour
            intProperties.get(6), // minute
            intProperties.get(7) // seconds
        ))
        .setIsConcluded(isConcluded);

    HabitRequest request = new HabitRequest(
        stringProperties.get(0), // name
        stringProperties.get(1), // motivation
        details
    );

    return addHabit(request);
}

// Update the addHabit methods in HabitTracker class

public int addHabit(String name, String motivation) {
    HabitRequest request = new HabitRequest(name, motivation);
    return addHabit(request);
}

```

```

public int addHabit(HabitRequest request) {
    Habit habit = new HabitBuilder(request.getName())
        .withMotivation(request.getMotivation())
        .withDailyDedication(
            request.getDetails().getDailyHoursDedication(),
            request.getDetails().getDailyMinutesDedication())
        .withStartDate(request.getDetails().getStartDate())
        .withId(this.nextId)
        .isConcluded(request.getDetails().getIsConcluded())
        .build();

    this.habits.add(habit);
    int response = nextId;
    this.tracker.put(nextId, new ArrayList<>());
    this.nextId++;
    return response;
}

public void addHabitRecord(Integer id){
    tracker.get(id).add(LocalDateTime.now());
}

public void toggleConcludeHabit(Integer id) {
    for (Habit habit : this.habits) {
        if (habit.getId().equals(id)) {
            habit.setIsConcluded(!habit.getIsConcluded());
        }
    }
}

public void removeHabit(Integer id) {
    this.habits.removeIf(habit -> habit.getId().equals(id));
    this.tracker.remove(id);
}

public List<LocalDateTime> getHabitRecords(Integer id) {
    return this.tracker.get(id);
}

public List<String> searchInHabits(String search){
    List<String> habits = new ArrayList<>();
    for (Habit habit : this.habits) {
        if (habit.getName().toLowerCase().contains(search.toLowerCase()) || habit.getMotivation().toLowerCase().contains(search.toLowerCase())) {
            habits.add(habit.toString());
        }
    }
}

```

What about testing the refactor?

This refactoring:

1. Moves timeline view logic to HabitTracker where the data lives
2. Reduces coupling between classes
3. Eliminates feature envy by moving the method closer to the data it uses
4. Maintains single responsibility principle
5. Simplifies TimelineView to be a thin wrapper/delegation layer

fix the feature envy code smell using move method refactoring technique

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

```
    public GeneralSearch() {}

    @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));
        this.searchLog.addSearchHistory(text);
        this.searchLog.setNumUsages(this.searchLog.getNumUsages() + 1);
        results.add("\nLogged in: " + this.searchLog.getLogName());
        return results;
    }
}
```

```
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);
    }

    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;
}

```

 GitHub Copilot

Let me help you fix the feature envy code smell using the move method refactoring technique.

Step-by-step plan:

1. Identify feature envy: The `GeneralSearch` class frequently accesses `SearchLog`'s methods
2. Move relevant method logic from `handleSearch` into `SearchLog`
3. Create a new method in `SearchLog` to handle search logging
4. Modify `GeneralSearch` to use the new method

Implementation:

```
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...
}
```

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

    public GeneralSearch() {}

    @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));
        this.searchLog.logSearch(text, results);
        return results;
    }
}
```

The changes:

1. Created new `logSearch` method in `SearchLog` that encapsulates logging logic
2. Simplified `handleSearch` in `GeneralSearch` by moving logging responsibility to `SearchLog`
3. Improved encapsulation by keeping search log related operations in `SearchLog` class

This refactoring reduces the feature envy smell by moving the logging-related operations to the class that owns the data.

