

FEATURE ENVY

TimeLineView


```
package org.example.studyplanner;
```

```
import java.time.LocalDateTime;  
import java.util.List;
```

```
public class TimelineView {
```

```
    public String habitDateViewAll(HabitTracker ht){  
        List<Habit> habits = ht.getHabits();  
        StringBuilder response = new StringBuilder();  
        for(Habit habit : habits){  
            response.append("[ Habit: ")  
                .append(habit.getName())  
                .append(" Records: ");  
            List<LocalDateTime> records =  
                ht.getHabitRecords(habit.getId());  
            for(LocalDateTime record : records){  
                response.append(ht.formatHabitDate(record)).append("  
                , ");  
            }  
            response.append("]");  
        }  
    }
```

```
    return response.toString();  
}
```

```
}
```

```
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 addHabit(String name, String motivation,  
        LocalTime dailyDedication, LocalDateTime startDate,  
        Boolean isConcluded) {  
        Habit habit = new Habit(name, motivation,  
            dailyDedication, this.nextId, startDate, isConcluded);  
        this.habits.add(habit);  
        int response = nextId;  
        this.tracker.put(nextId, new ArrayList<>());  
        this.nextId++;  
        return response;  
    }
```

```
    public int handleAddHabitAdapter(List<String>  
        stringProperties, List<Integer> intProperties, boolean  
        isConcluded){  
        LocalDateTime startDate =  
            LocalDateTime.of(intProperties.get(2),  
                intProperties.get(3), intProperties.get(4),  
                intProperties.get(5), intProperties.get(6),  
                intProperties.get(7));  
        LocalTime dailyDedication =  
            LocalTime.of(intProperties.get(1), intProperties.get(0));  
        // Corrected order  
        return addHabit(stringProperties.get(0),  
            stringProperties.get(1), dailyDedication, startDate,  
            isConcluded);  
    }
```

```
    public int addHabit(String name, String motivation) {
```

```
        Habit habit = new Habit(name, motivation, this.nextId);  
        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.toLower  
                Case()) ||  
                habit.getMotivation().toLowerCase().contains(search.toL  
                owerCase())) {  
                habits.add(habit.toString());  
            }  
        }  
        return habits;  
    }
```

```
}
```

```
fix the feature envy code smell using the move method  
refactoring technique
```

✦ The analysis tool didn't find any errors in the provided code. The fix for the feature envy code smell using the move method refactoring technique would involve identifying the methods that access the private attributes of another class more than their own and moving them to the class that owns the data.

In the given code, the `TimelineView` class relies heavily on the methods of the `HabitTracker` class to access and process habit data. Here's how we can refactor the code using the move method technique:

1. Move the `habitDateViewAll` method from `TimelineView` to `HabitTracker`:

```
Java

public class HabitTracker {

    // Existing methods of HabitTracker

    public String habitDateViewAll() {
        List<Habit> habits = this.getHabits();
        StringBuilder response = new StringBuilder();
        for (Habit habit : habits) {
            response.append("[ Habit: ")
                .append(habit.getName())
                .append(" Records: ");
            List<LocalDateTime> records = this.getHabitRecords(habit.getId());
            for (LocalDateTime record : records) {
                response.append(this.formatHabitDate(record)).append(", ");
            }
            response.append("]");
        }

        return response.toString();
    }

}
```

2. Modify the `TimelineView` class to call the new `habitDateViewAll` method of `HabitTracker`:

```
Java

public class TimelineView {

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

}
```

By applying these changes, the `TimelineView` class no longer needs to access the private attributes of `HabitTracker` directly. It relies on the public methods provided by `HabitTracker` to retrieve the necessary information. This improves code maintainability and reduces the risk of unintended side effects.

This refactoring addresses the feature envy code smell and promotes better encapsulation.