

## Feature Envy TimeLineView

QualidadeDeSoftware > Feature Envy Refactoring 1 ▾

fix the Feature Envy code smell using the move method refactoring technique

```
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(" ");
            }
        }
    }
}
```

Envie uma mensagem para o ChatGPT



O ChatGPT pode cometer erros. Considere verificar informações importantes.

QualidadeDeSoftware > Feature Envy Refactoring 1 ▾

Here's a refactored version of the code:

### Refactored Code

HabitTracker Class (with new methods)

java

Copiar código

```
package org.example.studyplanner;

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

public class HabitTracker {

    // Existing methods...

    public String formatHabitWithRecords(Habit habit) {
        StringBuilder response = new StringBuilder();
        response.append("[ Habit: ")
            .append(habit.getName())
            .append(". Records: ");
        List<LocalDateTime> records = getHabitRecords(habit.getId());
        for(LocalDateTime record : records){
            response.append(formatHabitDate(record)).append(" ");
        }
    }
}
```

## Feature Envy StudySearch

QualidadeDeSoftware > Feature Envy Refactoring 2

fix the Feature Envy code smell using the move method refactoring technique without adding a class, just changing what is in the move to and method

```
package org.example.studysearch;

import org.example.studycards.CardManager;
import org.example.studyplanner.HabitTracker;
import org.example.studyplanner.TODOTracker;
import org.example.studyregistry.StudyMaterial;
import org.example.studyregistry.StudyTaskManager;

import java.util.ArrayList;
import java.util.List;

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

    public GeneralSearch() {}

    @Override
```

QualidadeDeSoftware > Feature Envy Refactoring 2

extensively. Here's the refactored code:

Refactored Code

java Copiar código

```
package org.example.studysearch;

import org.example.studycards.CardManager;
import org.example.studyplanner.HabitTracker;
import org.example.studyplanner.TODOTracker;
import org.example.studyregistry.StudyMaterial;
import org.example.studyregistry.StudyTaskManager;

import java.util.ArrayList;
import java.util.List;

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

    public GeneralSearch() {}

    @Override
```

## Feature Envy MaterialSearch

QualidadeDeSoftware > Feature Envy Refactoring 3 ▾

fix the Feature Envy code smell using the move method refactoring technique without adding a class, making changes only to the "method" file and adding more things to the class that is already being used

```
package org.example.studysearch;

import org.example.studyregistry.StudyMaterial;

import java.util.ArrayList;
import java.util.List;

public class MaterialSearch implements Search<String>{

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

    public MaterialSearch() {}

    @Override
    public List<String> search(String text) {
```

QualidadeDeSoftware > Feature Envy Refactoring 3 ▾



To fix the **Feature Envy** code smell using the **Move Method** refactoring technique, the `handleMaterialSearch` method from `MaterialSearch` can be moved into the `SearchLog` class. This minimizes the reliance of `MaterialSearch` on `SearchLog` for search handling logic. Below is the refactored code:

### Updated `MaterialSearch` Class

java

Copiar código

```
package org.example.studysearch;

import java.util.List;

public class MaterialSearch implements Search<String> {

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

    public MaterialSearch() {}

    @Override
```

## Feature Envy MaterialSearch

QualidadeDeSoftware > Feature Envy Refactoring 4 ▾

fix the Feature Envy code smell using the move method refactoring technique without adding a class, making changes only to the "method" file and adding more things to the class that is already being used

```
package org.example.studysearch;

import org.example.studycards.CardManager;
import org.example.studyplanner.HabitTracker;
import org.example.studyplanner.TODOTracker;
import org.example.studyregistry.StudyTaskManager;

import java.util.ArrayList;
import java.util.List;

public class RegistrySearch implements Search<String>{
    private SearchLog searchLog = new SearchLog("Registry Search");
    public RegistrySearch(){}

    @Override
    public List<String> search(String text) {
```

QualidadeDeSoftware > Feature Envy Refactoring 4 ▾

eliminate the Feature Envy smell. The `handleRegistrySearch` method's logic is moved to `SearchLog`, leveraging the existing functionality of `SearchLog`. No new classes are introduced, and the changes are restricted to the provided files.

### Refactored `RegistrySearch`

```
java Copiar código

package org.example.studysearch;

import java.util.List;

public class RegistrySearch implements Search<String> {
    private SearchLog searchLog = new SearchLog("Registry Search");

    public RegistrySearch() {}

    @Override
    public List<String> search(String text) {
        // Delegating the search and logging responsibility to SearchLog
        return searchLog.performSearchWithLogging(text);
    }
}
```

## Feature Envy LeitnerSystem

QualidadeDeSoftware > Feature Envy Refactoring 5

LU

fix the Feature Envy code smell using the move method refactoring technique without adding a class, making changes only to the "method" file and adding more things to the class that is already being used

```
public String getRandomCardFromBox() {
    String response = "";
    response += leitnerSystem.getMethodName();
    List<Box> boxes = leitnerSystem.getBoxes();
    response += leitnerSystem.getRandomCard(boxes);
    return response;
}
```

```
package org.example.studycards;
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
```

QualidadeDeSoftware > Feature Envy Refactoring 5



To address the Feature Envy code smell in the `getRandomCardFromBox` method using the Move Method refactoring technique, the method can be moved to the `LeitnerSystem` class. This ensures that the logic operates closer to the data it depends on, thereby improving cohesion and reducing the dependency of one class on the internals of another.

Here's the updated `LeitnerSystem` class with the `getRandomCardFromBox` method added, and the refactored version of the original class:

### Refactored `LeitnerSystem` Class

java

Copiar código

```
package org.example.studycards;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class LeitnerSystem extends StudyMethod {
    List<Box> boxes = null;
```

## Feature Data class Card

QualidadeDeSoftware > Encapsulate Behavior Refactoring 1 ▾

LU

fix the Feature Data class smell using the encapsulate behavior refactoring technique

```
package org.example.studycards;

public class Card {
    private String question;
    private String answer;

    public Card(String question, String answer) {
        this.question = question;
        this.answer = answer;
    }

    public String getQuestion() {
        return question;
    }

    public void setQuestion(String question) {
        this.question = question;
    }
}
```



To address the "Data Class" code smell using the **Encapsulate Behavior** refactoring technique, you should move functionality related to the data stored in the `Card` class into the class itself. This makes the class more responsible for its own data and reduces reliance on external code manipulating its fields directly.

Here's the updated `Card` class with behavior encapsulated:

java

Copiar código

```
package org.example.studycards;

public class Card {
    private String question;
    private String answer;

    public Card(String question, String answer) {
        this.question = question;
        this.answer = answer;
    }
}
```

Envie uma mensagem para o ChatGPT

## Data class Reference

QualidadeDeSoftware > Encapsulate Behavior Refactoring 2


fix the Feature Data class smell using the encapsulate behavior refactoring technique

```
package org.example.studymaterial;

public abstract class Reference {
    private String title;
    private String description;
    private String link;
    private String accessRights;
    private String license;
    private boolean isDownloadable;
    private int rating;
    private String language;
    private int viewCount;
    private int downloadCount;
    private int shareCount;

    public void setTitle(String title) {
        this.title = title;
    }
}
```

QualidadeDeSoftware > Encapsulate Behavior Refactoring 2

 To fix the **Feature Data Class** smell in the `Reference` class using the **Encapsulate Behavior** refactoring technique, we can encapsulate the data by introducing behavior methods that operate on the fields. This change makes the class responsible for more functionality, thus reducing its perception as merely a data container.

Here's the refactored `Reference` class:

```
java Copiar código

package org.example.studymaterial;

public abstract class Reference {
    private String title;
    private String description;
    private String link;
    private String accessRights;
    private String license;
    private boolean isDownloadable;
    private int rating;
    private String language;
    private int viewCount;
}
```

## Data class ToDo

QualidadeDeSoftware > Encapsulate Behavior Refactoring 3 ▾

fix the Feature Data class smell using the encapsulate behavior refactoring technique

```
package org.example.studyplanner;

import java.text.MessageFormat;

public class ToDo implements PlannerMaterial{
    private Integer id;
    private String title;
    private String description;
    private int priority;

    public ToDo(Integer id, String title, String description, int priority) {
        this.id = id;
        this.title = title;
        this.description = description;
        this.priority = priority;
    }
}
```

@Override

Envie uma mensagem para o ChatGPT

QualidadeDeSoftware > Encapsulate Behavior Refactoring 3 ▾

```
},
}
}
```



To fix the "Data Class" smell in the `ToDo` class using the **Encapsulate Behavior** refactoring technique, we need to introduce behaviors (methods) that use the data encapsulated within the class to make it more cohesive. This will reduce its "data carrier" nature and provide more utility for the class.

Here's the refactored code:

### Refactored `ToDo` Class

java

Copiar código

```
package org.example.studyplanner;

import java.text.MessageFormat;

public class ToDo implements PlannerMaterial {
    private Integer id;
    private String title;
    private String description;
```



## Data class Registry

QualidadeDeSoftware > Encapsulate Behavior Refactoring 4 LU

fix the Feature Data class smell using the encapsulate behavior refactoring technique

```
package org.example.studyregistry;

import java.time.LocalDateTime;

public class Task extends Registry{
    private String title;
    private String description;
    private String author;
    private LocalDateTime date;


    public Task(String title, String description, String author,
LocalDateTime date) {
    this.title = title;
    this.name = title;
    this.description = description;
    this.author = author;
    this.date = da
    }
}
```

Envie uma mensagem para o ChatGPT

QualidadeDeSoftware > Encapsulate Behavior Refactoring 4 LU

]

]

 To address the smell of the `Task` class being a "Data Class," we can encapsulate behavior into it. A data class often serves only to store data and provides getter and setter methods without having meaningful methods or behavior. By applying the "Encapsulate Behavior" refactoring technique, we add methods that perform actions related to the data, reducing the reliance on external manipulation of its state.

Here is the refactored `Task` class:

**Refactored Code**

java Copiar código

```
package org.example.studyregistry;

import java.time.LocalDateTime;

public class Task extends Registry {
    private String title;
```

# LongMethod

QualidadeDeSoftware > Extract Method Refactoring

fix the Feature Long Method smell using the extract method refactoring use this method but without changing the behavior of other classes, okay?

```
package org.example.studyplanner;

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

public class TodoTracker {
    private List<ToDo> toDos = new ArrayList<>();
    private Map<Integer, List<LocalDateTime>> tracker;
    private Integer nextId;
    private static TodoTracker instance;

    private TodoTracker() {
        this.tracker = new HashMap<>();
        this.toDos = new ArrayList<>();
    }
}
```

Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.

POR 21:51  
PTB2 13/01/2025

QualidadeDeSoftware > Extract Method Refactoring

To address the "Feature Long Method" smell in the `toString()` method of the `TodoTracker` class, we will use the **Extract Method** refactoring technique. This involves breaking down the `toString()` method into smaller, well-named helper methods without changing the behavior of the class or affecting other classes.

Here's the refactored code:

**Refactored `TodoTracker` Class:**

```
java Copiar código

package org.example.studyplanner;

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

public class TodoTracker {
    private List<ToDo> toDos = new ArrayList<>();
}
```

Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.

POR 21:51  
PTB2 13/01/2025

## Long Method

QualidadeDeSoftware > Refactoring Long Methods

fix the Feature Long Method smell using the extract method refactoring use this method but without changing the behavior of other classes, okay?

```
package org.example.studyplanner;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class KanbanView {
    public enum State{
        TODO, DOING, DONE;
    }

    HabitTracker habitTracker = null;
    TodoTracker todoTracker = null;
    Map<State, List<PlannerMaterial>> kanban = null;

    public void nublir KanbanView(HabitTracker habitTracker, TodoTracker
```

Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.

QualidadeDeSoftware > Refactoring Long Methods

To fix the **Feature Long Method** smell in the **kanbanView** method while preserving the behavior of other classes, we will refactor the method using the **Extract Method** technique. This will involve breaking down the logic into smaller, reusable helper methods to improve readability and maintainability.

Here's the refactored code:

```
java Copiar código

public String kanbanView() throws Exception {
    try {
        if (kanban.isEmpty()) {
            throw new Exception("No material found");
        }
        StringBuilder sb = new StringBuilder();
        sb.append("[ Material ToDo: ");
        sb.append(System.lineSeparator());
        appendStateMaterials(sb, State.TODO, "No material found");
        sb.append(System.lineSeparator());
    }
}
```

Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.

## Long Method

QualidadeDeSoftware > Extract Method Refactoring

fix the Feature Long Method smell using the extract method refactoring use this method but without changing the behavior of other classes, okay?

```
package org.example.studyCards;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class LeitnerSystem extends StudyMethod {
    List<Box> boxes = null;


    public LeitnerSystem(String methodName) {
        super(methodName);
        boxes = new ArrayList<>(Arrays.asList(new Box(), new Box(), new Box(), new Box(), new Box(), new Box(), new Box(), new Box(), new Box(), new Box()));
    }
}
```

Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.

POR 21:51  
PTB2 13/01/2025

QualidadeDeSoftware > Extract Method Refactoring

 To address the **Feature Long Method** smell and reduce the complexity of the `getRandomCard(List<Box> otherBoxes)` method, we can use the **Extract Method** refactoring technique. This involves breaking the method into smaller, more focused helper methods while maintaining the original functionality and ensuring other classes remain unaffected.

Here's the refactored code:

**Refactored Code**

```
java Copiar código

public String getRandomCard(List<Box> otherBoxes) {
    if (isInvalidBoxList(otherBoxes)) {
        return "No card found";
    }

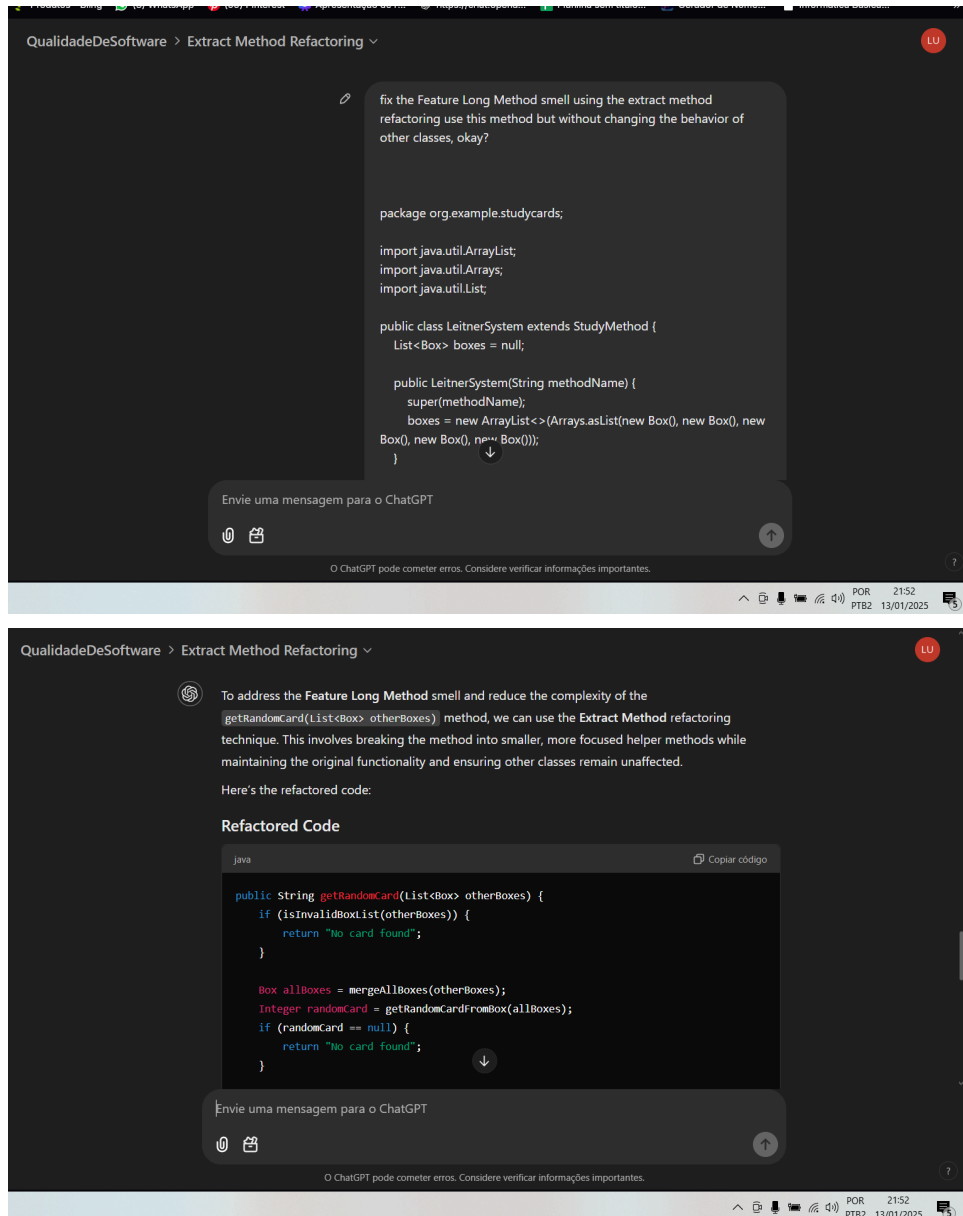
    Box allBoxes = mergeAllBoxes(otherBoxes);
    Integer randomCard = getRandomCardFromBox(allBoxes);
    if (randomCard == null) {
        return "No card found";
    }
}
```

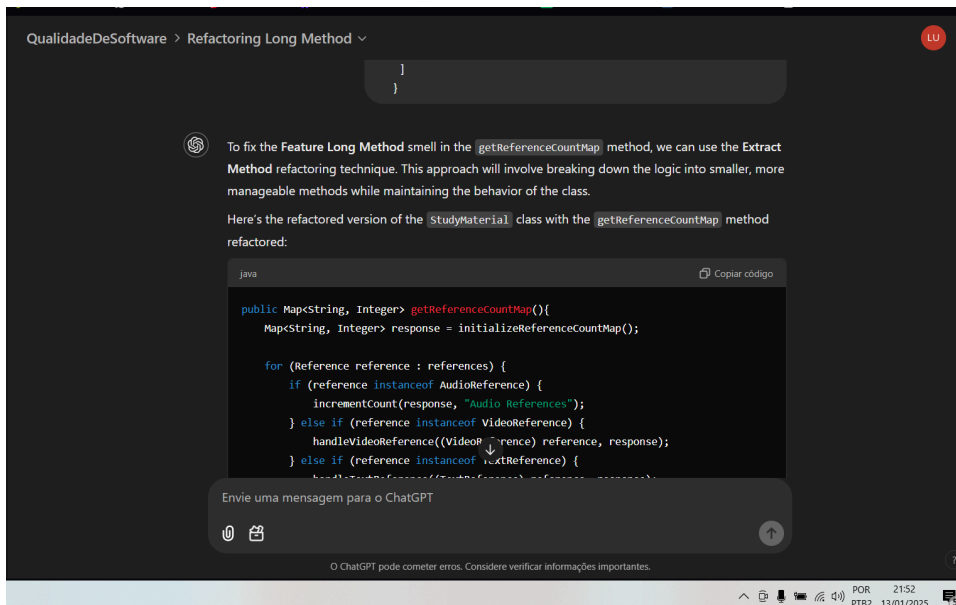
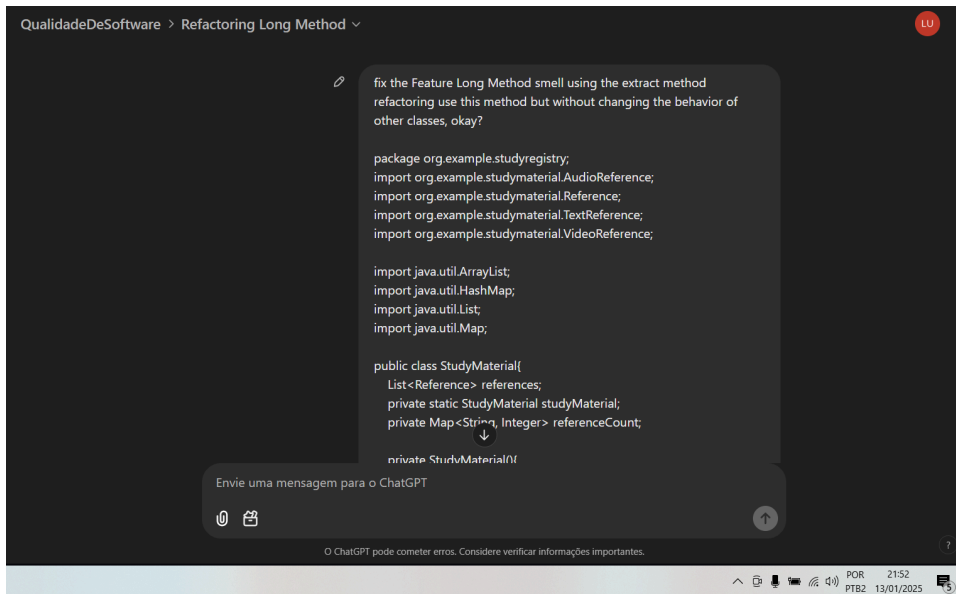
Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.

POR 21:51  
PTB2 13/01/2025

## Long Method





## Long Parameter List

QualidadeDeSoftware > Builder Pattern Refactoring audio reference 1 ▾

fix the Feature Long Parameter List smell using the builder pattern refactoring method

try not to create more code smells

```
package org.example.studymaterial;
```

```
import java.util.List;
```

```
public class AudioReference extends Reference {  
    public enum AudioQuality {  
        LOW, MEDIUM, HIGH, VERY_HIGH;  
    }  
    private AudioQuality audioQuality;
```

```
    public AudioReference(AudioQuality quality){  
        this.audioQuality = quality;  
    }
```

```
    public AudioQuality getAudioQuality() {
```

QualidadeDeSoftware > Builder Pattern Refactoring audio reference 1 ▾

LU



To resolve the **Feature Long Parameter List** smell using the **Builder Pattern**, we can refactor the `editAudio` method in the `AudioReference` class to encapsulate the parameters into a builder class.

This approach improves readability and makes the code easier to maintain without introducing additional code smells.

Here's the refactored version of the code:

### Updated `AudioReference` Class

java

Copiar código

```
package org.example.studymaterial;  
  
public class AudioReference extends Reference {  
    public enum AudioQuality {  
        LOW, MEDIUM, HIGH, VERY_HIGH;  
    }  
    private AudioQuality audioQuality;  
  
    public AudioReference(AudioQuality quality) {  
        this.audioQuality = quality;  
    }
```

Envie uma mensagem para o ChatGPT

## Long Parameter List

QualidadeDeSoftware > Refactoring HabitTracker Builder ▾

fix the Feature Long Parameter List smell using the builder pattern refactoring use this method but without changing the behavior of other classes, okay?

```
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();
        }
    }
}
```

Envie uma mensagem para o ChatGPT

QualidadeDeSoftware > Refactoring HabitTracker Builder ▾

here's the refactored code.

</> Habit Tracker Refactor

```
1 // Builder Class for Habit
2 package org.example.studyplanner;
3
4 import java.time.LocalDateTime;
5 import java.time.LocalTime;
6
7 public class HabitBuilder {
8     private String name;
9     private String motivation;
10    private LocalTime dailyDedicationTime;
11    private Integer id;
12    private LocalDateTime startDate;
13    private Boolean isConcluded;
14}
```

Responder no chat >

Explanation of Changes

1. Added a `HabitBuilder` Class:

2. Encapsulates the habit creation process



## Long Parameter List

QualidadeDeSoftware > Builder Pattern Refactoring ▾

fix the Feature Long Parameter List smell using the builder pattern refactoring use this method but without changing the behavior of other classes, okay?

```
package org.example.studyregistry;

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

public class StudyObjective extends Registry{
    private String title;
    private String description;
    private String topic;
    private Integer practicedDays;
    private LocalDateTime startDate;
    private Double duration;
    private String objectiveInOneLine;
    private String objectiveFullDescription;

    public String get...e() {
        return title;
    }
}
```

QualidadeDeSoftware > Builder Pattern Refactoring ▾



To address the "Feature Long Parameter List" smell using the Builder Pattern in the `StudyObjective` class without altering the behavior of other classes, follow these steps:

### Refactored Code

Here's the refactored version of the `StudyObjective` class:

```
java Copiar código

package org.example.studyregistry;

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

public class StudyObjective extends Registry {
    private String title;
    private String description;
    private String topic;
    private Integer practicedDays;
    private LocalDateTime startDate;
    private Double duration;
}
```

## Long Parameter List

QualidadeDeSoftware > Refatoração Builder Pattern ▾

LU

fix the Feature Long Parameter List smell using the builder pattern refactoring method

Do not change existing methods

Resolve the problem sem gerar erros em outras classes  
Reasons for the Problem

A long list of parameters might happen after several types of algorithms are merged in a single method. A long list may have been created to control which algorithm will be run and how.

Long parameter lists may also be the byproduct of efforts to make classes more independent of each other. For example, the code for creating specific objects needed in a method was moved from the method to the code for calling the method, but the created objects are passed to the method as parameters. Thus the original class no longer knows about the relationships between objects, and dependency has decreased. But if several of these objects are created, each of them will require its own parameter, which means a longer parameter list.



QualidadeDeSoftware > Refatoração Builder Pattern ▾

java

Copiar código

```
package org.example.studyregistry;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class StudyTaskManager {
    private static StudyTaskManager instance;
    private StudyMaterial studyMaterial = StudyMaterial.getStudyMaterial();
    List<Registry> registryList;
    List<String> weekResponsibilities = List.of();

    private StudyTaskManager() {
        this.registryList = new ArrayList<>();
    }

    public static StudyTaskManager getStudyTaskManager() {
        if (instance == null) {
            instance = new StudyTaskManager();
        }
    }
}
```

## Long Parameter List

QualidadeDeSoftware > Refactor HabitTracker Builder ▾



fix the Feature Long Parameter List smell using the builder pattern refactoring use this method but without changing the behavior of other classes, okay?

```
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();
        }
    }
}
```

Environment: org.eclipse.jdt.core - ChatGPT