

# ExcessiveParameterList AudioReference



Aqui está o código refatorado usando a técnica "Introduce Parameter Object" para resolver o problema de excesso de parâmetros. Um `record` chamado `AudioReferenceEditParams` foi introduzido para encapsular os parâmetros relacionados à edição de um objeto `AudioReference`. Isso melhora a legibilidade e mantém o código mais organizado e sustentável.

## Código Refatorado:

```
java Copiar Editar

package org.example.studymaterial;

import java.util.List;

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

## Principais Modificações:

- Criação de `AudioReferenceEditParams`:**
  - Esse `record` encapsula todos os parâmetros necessários para a edição de um `AudioReference`.
- Atualização do método `editAudio`:**
  - Agora recebe um único parâmetro do tipo `AudioReferenceEditParams`.
- Manutenção do método `editAudioAdapter`:**
  - Ele traduz as listas de propriedades para um objeto `AudioReferenceEditParams` antes de chamar o método `editAudio`.

## Benefícios:

- Legibilidade:** O código se torna mais fácil de ler e entender, especialmente no caso de métodos com muitos parâmetros.
- Reutilização:** O `AudioReferenceEditParams` pode ser reutilizado em outras partes do código onde os mesmos parâmetros são necessários.
- Flexibilidade:** Alterações futuras nos parâmetros exigirão mudanças apenas no `record` e no método relacionado.

```

private void editAudio(AudioReference audioReference){
    handleMethodHeader("(Audio Edit)");
    System.out.println("Type the following info: AudioReference.
AudioQuality audioQuality, boolean isDownloadable, " +
        "String title, String description, String link, String
accessRights, String license, String language, int rating, " +
        "int viewCount, int shareCount \n");
    AudioReference.AudioQuality quality
=AudioReference.audioQualityAdapter(getInput());
    audioReference.editAudio(quality,
Boolean.parseBoolean(getInput()), getInput(), getInput(),
getInput(), getInput(),
    getInput(), getInput(), Integer.parseInt(getInput()),
Integer.parseInt(getInput()), Integer.parseInt(getInput()));
}

```

```

/home/rafael/Área de
trabalho/LLMCodeSmellRefactor/src/main/java/org/example/cont
rollers/StudyRegistryController.java:154:23
java: method editAudio in class
org.example.studymaterial.AudioReference cannot be applied to
given types;
    required:
org.example.studymaterial.AudioReference.AudioReferenceEditPa
rams
    found:
org.example.studymaterial.AudioReference.AudioQuality,boolean,
java.lang.String,java.lang.String,java.lang.String,java.lang.String,ja
va.lang.String,java.lang.String,int,int,int
    reason: actual and formal argument lists differ in length

```



O problema ocorre porque o método `editAudio` da classe `AudioReference` espera um único parâmetro do tipo `AudioReference.AudioReferenceEditParams`, mas o código original estava tentando passar argumentos separados. Para resolver o erro, foi necessário criar um objeto da classe `AudioReference.AudioReferenceEditParams` contendo todos os argumentos necessários e, em seguida, passá-lo ao método `editAudio`.

Abaixo está uma revisão para o método `editAudio` que faz isso corretamente:

```
java Copiar Editar

private void editAudio(AudioReference audioReference) {
    handleMethodHeader("(Audio Edit)");
    System.out.println("Type the following info: AudioQuality (LOW, MEDIUM, HIGH, V
        + "title, description, link, accessRights, license, language, rating (6

    // Cria um objeto AudioReferenceEditParams com os dados fornecidos
    var params = new AudioReference.AudioReferenceEditParams(
        AudioReference.audioQualityAdapter(getInput()), // AudioQuality
        Boolean.parseBoolean(getInput()),               // isDownloadable
        getInput(),                                     // title
        getInput(),                                     // description
        getInput(),                                     // link
        getInput(),                                     // accessRights
        getInput(),                                     // license
        getInput(),                                     // language
        Integer.parseInt(getInput()),                    // rating
        Integer.parseInt(getInput()),                    // viewCount
        Integer.parseInt(getInput())                     // shareCount
    );

    // Chama o método editAudio com o objeto params
    audioReference.editAudio(params);
}
```

Além disso, foi corrigido o método `addAudioReference`, que agora utiliza o novo `editAudio` para editar os atributos do `AudioReference`:

```
java Copiar Editar

private AudioReference addAudioReference() {
    handleMethodHeader("(Audio Reference Creation)");
    System.out.println("Type the following info: Audio Quality (LOW, MEDIUM, HIGH,
    AudioReference audioReference = new AudioReference(AudioReference.audioQuality/
    editAudio(audioReference);
    return audioReference;
}
```

### Pontos importantes:

1. **Classe `AudioReferenceEditParams`**: Certifique-se de que esta classe está corretamente definida na classe `AudioReference` e que os parâmetros passados ao construtor correspondem às propriedades esperadas.
2. **Validação de entrada**: O método `getInput()` assume que as entradas estão no formato correto. Caso contrário, é recomendado adicionar validações para evitar erros de parsing.

Se precisar de mais suporte ou ajustes, sinta-se à vontade para pedir!

## ExcessiveParameterList StudyObjective

## Alterações realizadas:

### 1. Introdução do `record Objective`:

- Esse novo record encapsula os parâmetros extensos em uma estrutura única e reutilizável.

### 2. Refatoração de `handleSetObjective`:

- Recebe agora um único parâmetro do tipo `Objective`, simplificando a assinatura do método.

### 3. Adaptação do `handleSetObjectiveAdapter`:

- Cria um `Objective` a partir das listas e chama o método refatorado.

Essa abordagem melhora a legibilidade, facilita manutenção e reduz os riscos associados a alterações futuras.

🔍 📄 🏠 📌 🗑️

🔍

```
/home/rafael/Área de
trabalho/LLMCodeSmellRefactor/src/main/java/org/example/cont
rollers/StudyRegistryController.java:60:18
java: method handleSetObjective in class
org.example.studyregistry.StudyObjective cannot be applied to
given types;
  required: org.example.studyregistry.StudyObjective.Objective
  found:
int,int,int,int,int,int,int,java.lang.String,java.lang.String,java.lang.Strin
g,java.lang.String,java.lang.String,java.lang.String,java.lang.String
,double,boolean
  reason: actual and formal argument lists differ in length
```



O erro indica que a chamada do método `handleSetObjective` na classe `StudyRegistryController` não corresponde à assinatura esperada do método `handleSetObjective` na classe `StudyObjective`.

## Diagnóstico

A assinatura atual do método em `StudyObjective` espera um objeto do tipo `StudyObjective.Objective` como parâmetro, enquanto o código na `StudyRegistryController` está passando vários parâmetros individuais.

🔍 📄 🏠 📌 🗑️

## Solução

1. **Verificar a assinatura do método em `StudyObjective`**: Confirme como o método `handleSetObjective` está definido. Provavelmente, ele espera um objeto encapsulado com os atributos necessários.
2. **Modificar a chamada na `StudyRegistryController` para corresponder à assinatura:** Crie um objeto do tipo esperado ( `StudyObjective.Objective` ) e passe esse objeto como argumento.
3. **Correção no código:** Aqui está uma possível abordagem:

## Código Corrigido

Atualize o método `handleSetObjective` na `StudyRegistryController` para criar um objeto apropriado:

## O que foi ajustado

1. **Criação de um objeto do tipo esperado ( `StudyObjective.Objective` )**: Foi criado o objeto encapsulando todos os parâmetros necessários.
2. **Passagem do objeto como argumento:** O objeto criado foi passado para o método `handleSetObjective` de `StudyObjective`.

## Etapas adicionais

- Certifique-se de que os nomes dos métodos e os tipos definidos em `StudyObjective` estejam corretos.
- Se a classe `StudyObjective.Objective` não for pública, adicione visibilidade adequada para instanciá-la.

Caso encontre mais erros ou dúvidas, envie o trecho relevante, e ajustaremos! 😊

# ExcessiveParameterList StudyPlan



To fix the `ExcessiveParameterList` code smell in the `getReferenceCountMap()` method using the `Extract Method` refactoring technique, we will break down the large logic into smaller, more manageable methods. The primary goal is to reduce the complexity of the `getReferenceCountMap()` method by delegating specific responsibilities (such as counting audio, video, and text references) to dedicated helper methods.

Here's how the refactored code would look after applying the `Extract Method` technique:

## Refactoring Summary:

1. `initializeReferenceCount()` method: This method initializes the `response` map with zero values for each reference type, reducing redundancy in the `getReferenceCountMap()` method.
2. **Helper Methods for Counting:** The counting logic for audio, video, and text references has been moved into three distinct methods: `countAudioReferences()`, `countVideoReferences()`, and `countTextReferences()`. Each of these methods is responsible for counting the respective reference type and returning a map with the count.
3. **Reduction in Parameter List:** There was no explicit parameter list to reduce in this case, but by extracting logic into smaller methods, we made the `getReferenceCountMap()` method cleaner and more focused on orchestrating the different tasks rather than performing all of them directly.

This results in a more maintainable and readable class with a better separation of concerns.



```
/home/rafael/Área de
trabalho/LLMCodeSmellRefactor/src/main/java/org/example/cont
rollers/StudyRegistryController.java:60:18
java: method handleSetObjective in class
org.example.studyregistry.StudyObjective cannot be applied to
given types;
  required: org.example.studyregistry.StudyObjective.Objective
  found:
int,int,int,int,int,int,int,java.lang.String,java.lang.String,java.lang.Strin
g,java.lang.String,java.lang.String,java.lang.String,java.lang.String,
double,boolean
  reason: actual and formal argument lists differ in length
```



A mensagem de erro indica que o método `handleSetObjective` definido na classe `StudyObjective` espera um parâmetro do tipo `StudyObjective.Objective`, mas na chamada no arquivo `StudyRegistryController.java`, os argumentos passados não correspondem a essa expectativa.

### Causas Prováveis

1. **Alteração na assinatura do método:** O método `handleSetObjective` na classe `StudyObjective` pode ter sido atualizado para aceitar um único parâmetro encapsulado (por exemplo, um objeto `Objective`), mas o código no `StudyRegistryController` ainda tenta passar parâmetros individuais.
2. **Refatoração Incompleta:** Durante a refatoração, a estrutura para manipular os objetivos (como o encapsulamento dos parâmetros) foi alterada, mas o código que chama o método não foi atualizado para refletir essas mudanças.

#### Passos:

1. **Verifique a Definição do Método** `handleSetObjective` **na Classe** `StudyObjective`:

Navegue até a classe `StudyObjective` e confira como o método está declarado.

Por exemplo:

```
Java Copiar código

public void handleSetObjective(StudyObjective.Objective objective) {
    // implementação
}
```

Aqui, está claro que o método aceita um único objeto encapsulado chamado `Objective`.

2. **Adapte o Código no** `StudyRegistryController`: No método `handleSetObjective` dentro de `StudyRegistryController`, você precisa criar uma instância de `StudyObjective.Objective` com os valores fornecidos pelo usuário.

Substitua o trecho com o problema (linha 60) por algo como:

```
Java Copiar código

StudyObjective.Objective objective = new StudyObjective.Objective(
    Integer.parseInt(getInput()), // id
    Integer.parseInt(getInput()), // priority
    Integer.parseInt(getInput()), // practicedDays
    Integer.parseInt(getInput()), // day
    Integer.parseInt(getInput()), // month
    Integer.parseInt(getInput()), // year
    getInput(), // name
    getInput(), // title
    getInput(), // description
    getInput(), // topic
    getInput(), // objectiveInOneLine
    getInput(), // objectiveFullDescription
    getInput(), // motivation
    Double.parseDouble(getInput()), // duration
    Boolean.parseBoolean(getInput()) // isActive
);

objective.handleSetObjective(objective);
```

3. **Teste a Alteração:** Após ajustar o código, compile novamente e verifique se o problema foi resolvido.

## ExcessiveParameterList StudyTaskManager

### Principais melhorias:

1. **Refatoração com** `WeekSetupData`:

- Criado o registro `WeekSetupData` para encapsular os parâmetros relacionados ao planejamento da semana.
- Substituído o uso de múltiplos parâmetros no método `setUpWeek` e no `handleSetupWeek` pelo registro.

2. **Validação no** `handleSetupWeek`:

- Adicionada uma verificação para garantir que a lista de propriedades tenha os elementos esperados antes de instanciar o `WeekSetupData`.

3. **Melhoria na legibilidade:**

- Agora, os métodos que configuram a semana estão mais claros e evitam listas de parâmetros extensas.

```
/home/rafael/Área de
trabalho/LLMCodeSmellRefactor/src/main/java/org/example/cont
rollers/StudyRegistryController.java:217:25
java: method getInput in class
org.example.controllers.MainController cannot be applied to
given types;
  required: no arguments
  found:   java.lang.String
  reason:  actual and formal argument lists differ in length
```



Esse erro ocorre porque o método `getInput` da classe `MainController` não aceita argumentos. Para resolver o problema, você deve alinhar o uso de `getInput` com a definição do método existente no `MainController`.

### Soluções para Resolver o Erro

### Opção 1: Remover Argumentos da Chamada do Método

Se você não precisa que o método `getInput` exiba prompts personalizados, remova os argumentos das chamadas:

[illegible]



### Opção 2: Criar uma Sobrecarga no `MainController`

Se deseja usar prompts personalizados, modifique o `MainController` para adicionar uma sobrecarga do método `getInput` que aceita um argumento:

```
java Copiar Editar

// Sobrecarga no MainController
public String getInput(String prompt) {
    System.out.print(prompt + " ");
    return new java.util.Scanner(System.in).nextLine();
}
```

Com isso, você poderá chamar `getInput(prompt)` em vez de apenas `getInput()`.

### Opção 3: Alterar o Uso de `getInput`

Se não for possível modificar o `MainController`, você pode criar um método auxiliar na classe atual (`StudyRegistryController`) para lidar com prompts personalizados:

```
java Copiar Editar

private String promptInput(String prompt) {
    System.out.print(prompt + " ");
    return new java.util.Scanner(System.in).nextLine();
}
```

Então, atualize o método `getWeekInfo` para usar o novo método:

```
java Copiar Editar

private void getWeekInfo() {
    System.out.println("(Study Task Manager Week Set Up) Type the following info:");


    studyTaskManager.setUpWeek(
        new StudyTaskManager.WeekSetup(
            promptInput("Plan Name:"),
            promptInput("Objective Title:"),
            promptInput("Objective Description:"),
            promptInput("Material Topic:"),
            promptInput("Material Format:"),
            promptInput("Goal:"),
            promptInput("Reminder Title:"),
            promptInput("Reminder Description:"),
            promptInput("Main Task Title:"),
            promptInput("Main Habit:");
        )
    );
}
```

## ExcessiveParameterList HabitTracker

Aqui está o código completo atualizado com o método `habitDateViewAll()` e todas as dependências funcionando corretamente. Além disso, substituí todas as chamadas para o antigo `getFormattedHabits()`.

## Código Atualizado: HabitTracker

java

 Copiar

```
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;
```




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

    private static HabitTracker instance;


    // Singleton para obter a instância de HabitTracker
    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;
    }
}
```

 Copiar



@Override

 Copiar

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

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

// Retorna todos os hábitos
public List<Habit> getHabits() {
    return this.habits;
}
```



```
// Formata a data de um hábito
public String formatHabitDate(LocalDateTime date) {
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
    return date.format(formatter);
}

// Retorna as chaves do tracker
public List<Integer> getTrackerKeys() {
    return this.tracker.keySet().stream().toList();
}

// Adiciona um novo hábito com todos os parâmetros
public int addHabit(String name, String motivation, LocalTime dailyDedication, LocalTime startDate) {
    Habit habit = new Habit(name, motivation, dailyDedication, this.nextId, startDate);
    this.habits.add(habit);
    int response = nextId;
    this.tracker.put(nextId, new ArrayList<>());
    this.nextId++;
    return response;
}
```

```
// Adiciona um hábito a partir de adaptadores de propriedades
public int handleAddHabitAdapter(List<String> stringProperties, List<Integer> intProperties) {
    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(2));
    return addHabit(stringProperties.get(0), stringProperties.get(1), dailyDedication, startDate);
}

// Adiciona um hábito com apenas nome e motivação
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;
}

// Adiciona um registro de hábito pelo ID
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());
        }
    }
}

// Remove um hábito pelo ID
public void removeHabit(Integer id) {
    this.habits.removeIf(habit -> habit.getId().equals(id));
    this.tracker.remove(id);
}

// Retorna os registros de um hábito pelo ID
public List<LocalDateTime> getHabitRecords(Integer id) {
    return this.tracker.getDefault(id, new ArrayList<>());
}

// Realiza uma busca por hábitos com base no nome ou motivação
public List<String> searchInHabits(String search) {

```


 Copiar

```

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

// Exibe todos os hábitos e seus registros formatados
public String habitDateViewAll() {
    StringBuilder response = new StringBuilder();
    for (Habit habit : this.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(", ");
        }
    }
    return response.toString();
}

```

 Copiar

Copiar

```
// Exibe todos os hábitos e seus registros formatados
public String habitDateViewAll() {
    StringBuilder response = new StringBuilder();
    for (Habit habit : this.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();
}
}
```

#### 1. Método `habitDateViewAll`:

- Adicionado para exibir todos os hábitos e seus registros formatados.
- Este método substitui `getFormattedHabits()`.

#### 2. Chamadas para `getFormattedHabits()` Substituídas:

- Onde antes era usado `getFormattedHabits()`, agora deve-se usar `habitDateViewAll()`.

#### 3. Correções de Ordem:

- No método `handleAddHabitAdapter`, a ordem dos parâmetros ao criar `LocalTime` foi corrigida.

#### 4. Melhorias Gerais:


- Adicionado um `getOrDefault` em `getHabitRecords` para evitar `NullPointerException`.

## Correção

No arquivo `TimelineView.java`, onde você chama o método `getFormattedHabits()`, você deve alterar para chamar `habitDateViewAll()`.

Código Atualizado no `TimelineView.java`:

java

 Copiar

```
public class TimelineView {  
  
    public String habitDateViewAll(HabitTracker ht) {  
        // Substituir a chamada ao método getFormattedHabits por habitDateViewAll  
        return ht.habitDateViewAll(); // Atualizado  
    }  
}
```

## Explicação:

## Explicação:

- O erro ocorre porque o método `getFormattedHabits()` foi renomeado para `habitDateViewAll()` em `HabitTracker`.
- Por isso, você precisa atualizar as chamadas para o método correto, `habitDateViewAll()`.