

Qualidade de Software - Refatoração de Code Smells

Lorena Emanuele Pinheiro Ferreira - 516293

Robson José Dias Venancio - 508487

1- Long Method

1.1 - LeitnerSystem

ChatGPT 4o

```
public String getRandomCard(List<Box> otherBoxes) {
    // ...
}

// ...

// ...
```

refatore o code smell Long method utilizando a técnica de refatoração extract method.

Extract Method Refactor

Responder no chat >

O método original foi refatorado utilizando a técnica de "Extract Method" para reduzir o comprimento e melhorar a legibilidade. Agora, o código está dividido em três métodos: `getRandomCard`, `combineBoxes`, e `buildCardResponse`. Isso facilita a manutenção e o entendimento do código.

Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.

Extract Method Refactor

```
1 public String getRandomCard(List<Box> otherBoxes) {
2     if (otherBoxes == null || otherBoxes.isEmpty()) {
3         return null;
4     }
5
6     Box allBoxes = combineBoxes(otherBoxes);
7     Integer randomCard = allBoxes.getRandomCard();
8
9     if (randomCard == null) {
10        return "No card found";
11    }
12
13    return buildCardResponse(randomCard);
14 }
15
16 private Box combineBoxes(List<Box> otherBoxes) {
17     Box allBoxes = new Box();
18     for (Box box : otherBoxes) {
19         allBoxes.addCards(box.getCards());
20     }
21     return allBoxes;
22 }
23
24 private String buildCardResponse(Integer randomCard) {
25     CardManager manager = CardManager.getCardManager();
26     Card card = manager.getCard(randomCard);
27
28     String response = "[" + randomCard + " ] ";
29     response += "The random question was: " + card.getQuestion() + " | ";
30     response += "The answer is: " + card.getAnswer();
31
32     return response;
33 }
34 }
```

1.2 - KanbanView

ChatGPT 4o

```
1 public String kanbanView() throws Exception {
2     try {
3         if (kanban.isEmpty()) {
4             throw new Exception("No material found");
5         }
6
7         StringBuilder sb = new StringBuilder();
8         appendStateMaterials(sb, State.TODO, "Material todo");
9         appendStateMaterials(sb, State.DOING, "Material in progress");
10        appendStateMaterials(sb, State.DONE, "Material completed");
11
12        sb.append("\n");
13        return sb.toString();
14    } catch (Exception e) {
15        throw new Exception(e.getMessage());
16    }
17 }
18
19 private void appendStateMaterials(StringBuilder sb, State state, String header) {
20     sb.append("[ ");
21     sb.append(header).append(": ");
22     sb.append(System.lineSeparator());
23
24     if (kanban.get(state).isEmpty()) {
25         sb.append("No material found");
26     } else {
27         for (PlannerMaterial material : kanban.get(state)) {
28             sb.append(", ").append(material.toString());
29         }
30     }
31     sb.append(System.lineSeparator());
32 }
33 }
```

refatore o código smell Long method utilizando a técnica de refatoração extract method.

Editado

O método foi refatorado usando a técnica de "Extract Method". As seções relacionadas à montagem de strings para cada estado (TODO, DOING, DONE) foram movidas para um método auxiliar chamado `appendStateMaterials`, reduzindo o comprimento do método principal e aumentando a legibilidade.

Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.

Extract Method Refactor

```
1 public String kanbanView() throws Exception {
2     try {
3         if (kanban.isEmpty()) {
4             throw new Exception("No material found");
5         }
6
7         StringBuilder sb = new StringBuilder();
8         appendStateMaterials(sb, State.TODO, "Material todo");
9         appendStateMaterials(sb, State.DOING, "Material in progress");
10        appendStateMaterials(sb, State.DONE, "Material completed");
11
12        sb.append("\n");
13        return sb.toString();
14    } catch (Exception e) {
15        throw new Exception(e.getMessage());
16    }
17 }
18
19 private void appendStateMaterials(StringBuilder sb, State state, String header) {
20     sb.append("[ ");
21     sb.append(header).append(": ");
22     sb.append(System.lineSeparator());
23
24     if (kanban.get(state).isEmpty()) {
25         sb.append("No material found");
26     } else {
27         for (PlannerMaterial material : kanban.get(state)) {
28             sb.append(", ").append(material.toString());
29         }
30     }
31     sb.append(System.lineSeparator());
32 }
33 }
```

1.3 - TodoTracker

ChatGPT 4o

```
1 public String toString() {
2     StringBuilder sb = new StringBuilder();
3     for (Todo todo : todos) {
4         appendTodoInfo(sb, todo);
5     }
6
7     return sb.toString();
8 }
9
10 private void appendTodoInfo(StringBuilder sb, Todo todo) {
11     String todoInfo = todo.toString();
12     sb.append(todoInfo).append("\n");
13
14     Integer id = todo.getId();
15     List<LocalDateTime> todosDate = this.tracker.get(id);
16     if (todosDate == null) {
17         sb.append("No tracks found\n");
18     } else {
19         appendFormattedDates(sb, todosDate);
20     }
21 }
22
23 private void appendFormattedDates(StringBuilder sb, List<LocalDateTime> todosDate) {
24     String pattern = "yyyy-MM-dd HH:mm:ss";
25     DateTimeFormatter formatter = DateTimeFormatter.ofPattern(pattern);
26     for (LocalDateTime idt : todosDate) {
27         String formattedDate = formatter.format(idt);
28         sb.append(formattedDate).append("\n");
29     }
30 }
```

refatore o código smell Long method utilizando a técnica de refatoração extract method.

Editado

O método foi refatorado utilizando a técnica de "Extract Method". Agora, a lógica foi dividida em três métodos: o principal `toString`, o auxiliar `appendTodoInfo`, e o específico para formatação de datas `appendFormattedDates`. Essa separação melhora a legibilidade e facilita a manutenção do código.

Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.

Extract Method Refactor

```
1 @Override
2 public String toString() {
3     StringBuilder str = new StringBuilder();
4     for (Todo todo : todos) {
5         appendTodoInfo(str, todo);
6     }
7
8     String response = str.toString();
9     if (response.isEmpty()) {
10        return "No Todos found";
11    }
12    return response;
13 }
14
15 private void appendTodoInfo(StringBuilder str, Todo todo) {
16     String todoInfo = todo.toString();
17     str.append(todoInfo).append("\n");
18
19     Integer id = todo.getId();
20     List<LocalDateTime> todosDate = this.tracker.get(id);
21     if (todosDate == null) {
22         str.append("No tracks found\n");
23     } else {
24         appendFormattedDates(str, todosDate);
25     }
26 }
27
28 private void appendFormattedDates(StringBuilder str, List<LocalDateTime> todosDate) {
29     String pattern = "yyyy-MM-dd HH:mm:ss";
30     DateTimeFormatter formatter = DateTimeFormatter.ofPattern(pattern);
31     for (LocalDateTime idt : todosDate) {
32         String formattedDate = formatter.format(idt);
33         str.append(formattedDate).append("\n");
34     }
35 }
36 }
```

1.4 - StudyGoal

ChatGPT 4o

A refatoração reduziu o tamanho do método principal, mas o número de linhas ainda pode ser otimizado para atender à regra NCSS. Vamos ajustar ainda mais o código usando métodos auxiliares para agrupar lógica repetida e reduzir a contagem de linhas no método principal.

Vou refatorar o código.

Editado

O código foi refatorado para reduzir a contagem de linhas no método principal `setGoalSummary`. As verificações condicionais foram simplificadas usando métodos auxiliares genéricos como `appendConditionalSection`, `appendCollectionIfNotNull`, e `appendObjectIfNotNull`. Isso otimiza ainda mais a legibilidade e modularidade, atendendo à regra NCSS.

Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.

Extract Method Refactor

```
1 public String setGoalSummary() {
2     StringBuilder summary = new StringBuilder();
3     appendHeader(summary, "Goal Summary:");
4
5     appendConditionalSection(summary, this.isActive, "Active Goal:", goal);
6     appendConditionalSection(summary, this.isCompleted, "Completed Goal:", goal);
7     appendCollectionIfNotNull(summary, "Requirements:", this.goalRequirements);
8     appendObjectIfNotNull(summary, "Plan:", this.studyPlan);
9     appendObjectIfNotNull(summary, "Objective:", this.studyObjective);
10
11     this.summary = summary.toString();
12     return summary.toString();
13 }
14
15 private void appendHeader(StringBuilder summary, String header) {
16     summary.append(header).append("\n");
17 }
18
19 private void appendConditionalSection(StringBuilder summary, boolean condition, String title, Object content) {
20     if (condition) {
21         appendSection(summary, title, content.toString());
22     }
23 }
24
25 private void appendCollectionIfNotNull(StringBuilder summary, String title, Collection<String> collection) {
26     if (collection != null) {
27         summary.append(title).append("\n");
28         for (String item : collection) {
29             summary.append(item).append(", ");
30         }
31         summary.append("\n");
32     }
33 }
34
35 private void appendObjectIfNotNull(StringBuilder summary, String title, Object obj) {
36     if (obj != null) {
37         appendSection(summary, title, obj.toString());
38     }
39 }
40
41 private void appendSection(StringBuilder summary, String title, String content) {
42     summary.append(title).append("\n").append(content).append("\n\n");
43 }
44 }
```

1.5 - StudyMaterial

ChatGPT 4o

Essa versão melhora a legibilidade e modularidade, reduzindo o tamanho do método principal e facilitando a manutenção.

refatore o code smell Long method utilizando a técnica de refatoração extract method.

Editado

O método foi refatorado usando a técnica de "Extract Method". A lógica para inicializar o mapa, incrementar contagens e lidar com referências específicas foi separada em métodos auxiliares (`initializeResponseMap`, `incrementCount`, `handleVideoReference`, `handleTextReference`), melhorando a legibilidade e modularidade.

Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.

Extract Method Refactor

```
1 public Map<String, Integer> getReferenceCountMap() {
2     Map<String, Integer> response = initializeResponseMap();
3
4     for (Reference reference : references) {
5         if (reference.getClass() == AudioReference.class) {
6             incrementCount(response, "Audio References");
7         } else if (reference.getClass() == VideoReference.class) {
8             handleVideoReference(response, (VideoReference) reference);
9         } else if (reference.getClass() == TextReference.class) {
10            handleTextReference(response, (TextReference) reference);
11        }
12    }
13
14    setReferenceCount(response);
15    return response;
16 }
17
18 private Map<String, Integer> initializeResponseMap() {
19     Map<String, Integer> response = new HashMap<>();
20     response.put("Audio References", 0);
21     response.put("Video References", 0);
22     response.put("Text References", 0);
23     return response;
24 }
25
26 private void incrementCount(Map<String, Integer> response, String key) {
27     response.put(key, response.get(key) + 1);
28 }
29
30 private void handleVideoReference(Map<String, Integer> response, VideoReference videoReference) {
31     if (videoReference.handleStreamAvailability()) {
32         incrementCount(response, "Video References");
33     }
34 }
35
36 private void handleTextReference(Map<String, Integer> response, TextReference textReference) {
37     if (textReference.handleTextAccess()) {
38         incrementCount(response, "Text References");
39     }
40 }
41 }
```

2 - Feature Envy

2.1-TimelineView

refatore o code smell feature envy utilizando a técnica de refatoração move method:



Para refatorar o *code smell* "feature envy" utilizando a técnica de refatoração *Move Method*, vamos identificar o método ou funcionalidade que está acessando excessivamente dados de outra classe (neste caso, parece que o método `habitDateViewAll` está acessando diretamente dados da classe `HabitTracker`).

A ideia é mover a lógica que depende de `HabitTracker` para a própria classe `HabitTracker`. Vou refatorar o código:

Passos:

1. Criar um método na classe `HabitTracker` que encapsula a lógica de construção da visualização dos hábitos.
2. Atualizar o método `habitDateViewAll` para delegar a lógica para o novo método em `HabitTracker`.

Aqui está a refatoração:

Antes:

```
java Copiar código

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

Antes:

```
java Copiar código

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();
}
```

Depois da refatoração (método movido para `HabitTracker`):

```
java Copiar código

// Na classe HabitTracker
public String generateHabitView() {
    StringBuilder response = new StringBuilder();
    for (Habit habit : this.getHabits()) {
        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();
}
```

Depois da refatoração (método movido para `HabitTracker`):

```
java Copiar código

// Na classe HabitTracker
public String generateHabitView() {
    StringBuilder response = new StringBuilder();
    for (Habit habit : this.getHabits()) {
        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();
}
```

E o método `habitDateViewAll` original se torna simples:

```
java Copiar código

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

Benefícios da refatoração:

1. **Encapsulamento aprimorado:** Agora a lógica relacionada a hábitos e seus registros pertence à classe `HabitTracker`, que é responsável por esses dados.
2. **Maior coesão:** As responsabilidades das classes estão mais claras.
3. **Facilidade de manutenção:** Qualquer alteração na forma como os dados são exibidos será feita na própria classe `HabitTracker`.

2.2 - GeneralSearch

Soluções Aplicadas:

1. Delegação da Lógica de Logging para SearchLog:

- Foi identificado que a lógica relacionada ao registro do histórico (`addSearchHistory`), contagem de usos (`setNumUsages`), e criação da mensagem de log (`getLogName`) estava melhor localizada na classe `SearchLog`.
- Criado o método `logSearchAndReturnMessage` em `SearchLog` para encapsular essa lógica.

```
java Copiar código

public class SearchLog {
    public String logSearchAndReturnMessage(String text) {
        this.addSearchHistory(text);
        this.setNumUsages(this.getNumUsages() + 1);
        return "Logged in: " + this.getLogName();
    }
}
```

2. Simplificação do Método handleSearch:

- O método `handleSearch` foi modificado para delegar as responsabilidades relacionadas ao `SearchLog`.
- A lógica de busca permaneceu focada nas chamadas aos gerenciadores, mas a responsabilidade de log foi transferida.

Refatorado:

```
java Copiar código

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));

    // Delegando o logging para SearchLog
    results.add(this.searchLog.logSearchAndReturnMessage(text));

    return results;
}
```

2.3 - StudyCardsController

Refatoração Aplicada:

1. Mover a lógica para a classe `LeitnerSystem`:

- Crie um novo método na classe `LeitnerSystem` que encapsula essa lógica.

```
java Copiar código

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

2. Alterar o método original para delegar a responsabilidade:

- O método na classe atual (`getRandomCardFromBox`) apenas chama o método recém-criado em `LeitnerSystem`.

```
java Copiar código

public String getRandomCardFromBox() {
    return leitnerSystem.getRandomCardFromBox();
}
```

```
response += leitnerSystem.getRandomCard(boxes);
return response;
}
```

refatore o code smell feature envy utilizando a técnica de refatoração move method:



Para refatorar o *code smell Feature Envy* no método `getRandomCardFromBox`, podemos aplicar a técnica *Move Method*. O problema é que o método depende fortemente da lógica de outra classe, `leitnerSystem`, indicando que parte da responsabilidade deve ser movida para lá.

Refatoração

Antes (Código Original)

```
java Copiar código

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


2.4-MaterialSearch

refatore o code smell feature envy utilizando a técnica de refatoração move method:



Vamos aplicar a técnica de refatoração *Move Method* para resolver o *code smell Feature Envy* no método `handleMaterialSearch`. O problema central é que o método acessa repetidamente funcionalidades relacionadas a `SearchLog`, indicando que parte da lógica deveria ser movida para essa classe.

Refatoração

Antes (Código Original)

```
java Copiar código

private List<String> handleMaterialSearch(String text) {
    List<String> results = new ArrayList<>();
    results.addAll(StudyMaterial.getStudyMaterial().searchInMaterials(text));
    this.searchLog.addSearchHistory(text);
    this.searchLog.setNumUsages(this.searchLog.getNumUsages() + 1);
    results.add("Logged in: " + this.searchLog.getLogName());
    return results;
}
```

Refatoração Aplicada:

1. Mover a lógica relacionada ao `SearchLog` para a própria classe `SearchLog`:

- Adicione um novo método na classe `SearchLog` que encapsula as operações relacionadas ao registro de busca.

```
java Copiar código

public class SearchLog {
    public String logSearch(String text) {
        this.addSearchHistory(text);
        this.setNumUsages(this.getNumUsages() + 1);
        return "Logged in: " + this.getLogName();
    }
}
```

2. Modificar o método `handleMaterialSearch` para delegar a responsabilidade de logging:

- Agora o método `handleMaterialSearch` se torna mais simples, focando apenas na lógica de busca de materiais.

```
java Copiar código

private List<String> handleMaterialSearch(String text) {
    List<String> results = new ArrayList<>();
    results.addAll(StudyMaterial.getStudyMaterial().searchInMaterials(text));
    results.add(this.searchLog.logSearch(text)); // Delegação para SearchLog
    return results;
}
```

2.5 - RegistrySearch

refatore o code smell feature envy utilizando a técnica de refatoração move method:

- Para refatorar o *code smell Feature Envy* no método `handleRegistrySearch`, aplicaremos a técnica de *Move Method*. O problema é que o método depende excessivamente da classe `SearchLog`, o que sugere que a lógica relacionada ao registro de histórico e contagem de usos deveria ser movida para a classe `SearchLog`.

Refatoração

Antes (Código Original)

```
java Copiar código

private List<String> handleRegistrySearch(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(StudyTaskManager.getStudyTaskManager().searchInRegistries(text));

    // Lógica relacionada ao SearchLog
    this.searchLog.addSearchHistory(text);
    this.searchLog.setNumUsages(this.searchLog.getNumUsages() + 1);
    results.add("Logged in: " + this.searchLog.getLogName());

    return results;
}
```

Refatoração Aplicada:

1. Mover a lógica de registro de busca para `SearchLog`:

- Adicione um novo método na classe `SearchLog` para encapsular a lógica relacionada ao histórico e contagem de usos.

```
java Copiar código

public class SearchLog {
    public String logSearch(String text) {
        this.addSearchHistory(text);
        this.setNumUsages(this.getNumUsages() + 1);
        return "Logged in: " + this.getLogName();
    }
}
```

2. Alterar o método `handleRegistrySearch` para delegar a responsabilidade ao `SearchLog`:

```
java Copiar código

private List<String> handleRegistrySearch(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(StudyTaskManager.getStudyTaskManager().searchInRegistries(text));

    // Delegação da lógica para SearchLog
    results.add(this.searchLog.logSearch(text));

    return results;
}
```

3 - Data Class

3.1 - Card

refatore o code smell Data Class utilizando a técnica de refatoração move method.

```
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;
    }

    public String getAnswer() {
        return answer;
    }

    public void setAnswer(String answer) {
        this.answer = answer;
    }

    public void edit(String question, String answer) {
        setQuestion(question);
        setAnswer(answer);
    }
}
```

o code smell inicia na linha 3



O code smell "Data Class" ocorre quando uma classe atua apenas como um contêiner de dados sem

Envie uma mensagem para o ChatGPT



O ChatGPT pode cometer erros. Considere verificar informações importantes.



O code smell "Data Class" ocorre quando uma classe atua apenas como um contêiner de dados sem comportamento relevante. Para resolver isso, podemos usar a técnica de **Move Method**, onde os métodos que fazem sentido serem implementados em outra classe são movidos para lá, e a classe original deixa de ser apenas um repositório de dados.

A seguir, vamos identificar o comportamento da classe `Card` e sugerir uma refatoração. Neste caso, a responsabilidade de editar os dados pode ser movida para uma nova classe que manipula os cartões (`CardManager` , por exemplo). Vamos refatorar:

Código Refatorado:

Classe `Card` após a refatoração:

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;
    }

    public String getQuestion() {
        return question;
    }

    public String getAnswer() {
        return answer;
    }
}
```

Nova classe `CardManager` para encapsular comportamento:

java



Copiar código

Envie uma mensagem para o ChatGPT



O ChatGPT pode cometer erros. Considere verificar informações importantes.

3.2 - Reference

```
public void setViewCount(int viewCount) {
    this.viewCount = viewCount;
}

public int getDownloadCount() {
    return downloadCount;
}

public void setDownloadCount(int downloadCount) {
    this.downloadCount = downloadCount;
}

public int getShareCount() {
    return shareCount;
}

public void setShareCount(int shareCount) {
    this.shareCount = shareCount;
}
}
```

refatore o code smell Data Class method utilizando a técnica de refatoração Move Method



Para refatorar o código utilizando a técnica de refatoração **Move Method**, o objetivo é mover métodos que manipulam dados de uma classe para uma classe que seja mais apropriada para esse tipo de operação. Como a classe `Reference` está sendo usada para representar dados, podemos considerar que os métodos de manipulação de dados podem ser movidos para uma classe que trate desses dados, talvez uma classe especializada, para melhorar a coesão da classe `Reference` e diminuir sua responsabilidade.

Aqui está um exemplo de como mover os métodos de acesso (getters e setters) para uma nova classe `ReferenceData`:

Refatoração

1. **Criação da Classe `ReferenceData`**: A nova classe `ReferenceData` será responsável por armazenar e manipular os dados relacionados à `Reference`.
2. **Movendo os métodos**: Os métodos `set` e `get` podem ser movidos para a nova classe `ReferenceData` enquanto a classe `Reference` continua com as propriedades.



Código Refatorado


Envie uma mensagem para o ChatGPT



Código Refatorado

Classe `ReferenceData`

java

 Copiar código

```
package org.example.studymaterial;

public class ReferenceData {
    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;

    // Getters and Setters
    public void setTitle(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getDescription() {
        return description;
    }

    public void setLink(String link) {
        this.link = link;
    }

    public String getLink() {
        return link;
    }
}
```



Envie uma mensagem para o ChatGPT



3.3 - Todo

```
1 private Integer id;
2 private String title;
3 private String description;
4 private int priority;
5
6 public Todo(Integer id, String title, String description, int priority) {
7     this.id = id;
8     this.title = title;
9     this.description = description;
10    this.priority = priority;
11 }
12
13 @Override
14 public String toString() {
15     return MessageFormat.format("ID: {0}, Title: {1}, Descrição: {2}, Prioridade: {3}",
16         id, title, description, priority);
17 }
18
19 public int getId() { return id; }
20
21 public void setId(Integer id) { this.id = id; }
22
23 public String getTitle() { return title; }
24
25 public void setTitle(String title) { this.title = title; }
26
27 public String getDescription() { return description; }
28
29 public void setDescription(String description) { this.description = description; }
30
31 public int getPriority() { return priority; }
32
33 public void setPriority(int priority) { this.priority = priority; }
```

refatore o code smell data class utilizando a técnica de refatoração move method.



Para refatorar o code smell "Data Class" utilizando a técnica de **Move Method**, você pode transferir responsabilidades para métodos que utilizem os atributos diretamente na própria classe **Todo**. Isso faz com que ela seja mais coesa e encapsule comportamentos relacionados aos dados que ela armazena.

No código apresentado, a classe **Todo** parece estar funcionando apenas como uma estrutura de dados com getters e setters. Vamos transformá-la em uma classe mais orientada a comportamentos ao adicionar métodos que operem diretamente nos atributos dela.

Aqui está a refatoração:

Refatoração do Código:

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;
    private int priority;

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

Envie uma mensagem para o ChatGPT



```

        this.description = description;
        this.priority = priority;
    }

    @Override
    public String toString() {
        return MessageFormat.format(
            "[Priority:{3}] ToDo {0}: {1}, {2}", id, title, description, priority);
    }

    // Behavior added for prioritization
    public boolean isHighPriority() {
        return this.priority > 5;
    }

    public void promotePriority() {
        this.priority += 1; // Simple logic to promote the priority
    }

    public void demotePriority() {
        if (this.priority > 0) {
            this.priority -= 1; // Decrease priority, but ensure it doesn't go below 0
        }
    }

    public void updateDescription(String newDescription) {
        this.description = newDescription.trim(); // Update description with sanitization
    }


    // Getters and Setters
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }

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

    public String getDescription() { return description; }
    public void setDescription(String description) { this.description = description; }

    public int getPriority() { return priority; }
    public void setPriority(int priority) { this.priority = priority; }
}

```

 Copiar código

Alterações Editar

Envie uma mensagem para o ChatGPT



3.4 - Task

```
1 import java.time.LocalDateTime;
2
3 public class Task extends Registry {
4     private String title;
5     private String description;
6     private String author;
7     private LocalDateTime date;
8
9     public Task(String title, String description, String author, LocalDateTime date) {
10         this.title = title;
11         this.name = title;
12         this.description = description;
13         this.author = author;
14         this.date = date;
15     }
16
17     public String getTitle() { return title; }
18     public void setTitle(String title) { this.title = title; }
19     public String getDescription() { return description; }
20     public void setDescription(String description) { this.description = description; }
21     public String getAuthor() { return author; }
22     public void setAuthor(String author) { this.author = author; }
23     public LocalDateTime getDate() { return date; }
24     public void setDate(LocalDateTime date) { this.date = date; }
25 }
```

refatore o code smell Data Class utilizando a técnica de refatoração move method.



A classe `Task` apresentada contém o *code smell* conhecido como "Data Class", onde a classe é apenas um recipiente para dados e não possui comportamento (métodos relevantes). Para eliminar esse *code smell*, podemos mover métodos que realizam comportamentos significativos diretamente relacionados à `Task` para dentro da classe. Por exemplo:

Etapas para aplicar a refatoração:

1. Identificar métodos que interagem com `Task` em outras partes do código.
2. Mover esses métodos para a classe `Task` para encapsular melhor o comportamento e respeitar o princípio de responsabilidade única.

Segue a refatoração inicial:

```
java Copiar código
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; // Verificar se 'name' é necessário
    }
}
```

Envie uma mensagem para o ChatGPT



O ChatGPT pode cometer erros. Considere verificar informações importantes.

// Getters e Setters permanecem apenas se realmente necessários

 Copiar código

```
public String getTitle() {
    return title;
}

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

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public String getAuthor() {
    return author;
}

public void setAuthor(String author) {
    this.author = author;
}

public LocalDateTime getDate() {
    return date;
}

public void setDate(LocalDateTime date) {
    this.date = date;
}

// Comportamentos adicionados para encapsular lógica dentro da classe
public boolean isOverdue() {
    return LocalDateTime.now().isAfter(this.date);
}

public String summarizeTask() {
    return String.format("Task: %s | Author: %s | Due: %s", title, author, date);
}

public void postponeTask(int days) {
    this.date = this.date.plusDays(days);
}
```

Envie uma mensagem para o ChatGPT



O ChatGPT pode cometer erros. Considere verificar informações importantes.

4 - Long Parameter List

4.1 - HabitTracker

```
}  
    return habits;  
}  
  
}
```

refatore o code smell Long Parameter List utilizando a técnica de refatoração introduce parameter.

< 2/2 >



Para refatorar o code smell "Long Parameter List" no método `addHabit` da classe `HabitTracker`, podemos utilizar a técnica de introduzir um parâmetro por meio de um objeto que encapsule os argumentos relacionados. Vamos criar uma classe auxiliar chamada `HabitData` que agrupará os parâmetros necessários para criar um hábito.

Aqui está a refatoração:

java

Copiar código

```
package org.example.studyplanner;  
  
import java.time.LocalDateTime;  
import java.time.LocalTime;  
  
public class HabitData {  
    private String name;  
    private String motivation;  
    private Integer dailyMinutesDedication;  
    private Integer dailyHoursDedication;  
    private LocalDateTime startDate;  
    private Boolean isConcluded;  
  
    public HabitData(String name, String motivation, Integer dailyMinutesDedication, Integer dailyHoursDedication, LocalDateTime startDate, Boolean isConcluded) {  
        this.name = name;  
        this.motivation = motivation;  
        this.dailyMinutesDedication = dailyMinutesDedication;  
        this.dailyHoursDedication = dailyHoursDedication;  
        this.startDate = startDate;  
        this.isConcluded = isConcluded;  
    }  
}
```

Envie uma mensagem para o ChatGPT



O ChatGPT pode cometer erros. Considere verificar informações importantes.

```

    public String getName() {
        return name;
    }

    public String getMotivation() {
        return motivation;
    }

    public Integer getDailyMinutesDedication() {
        return dailyMinutesDedication;
    }

    public Integer getDailyHoursDedication() {
        return dailyHoursDedication;
    }

    public LocalDateTime getStartDate() {
        return startDate;
    }

    public Boolean getIsConcluded() {
        return isConcluded;
    }
}

```

Refatorando o método `addHabit`

Agora, alteramos o método `addHabit` para aceitar uma instância de `HabitData` em vez de uma longa lista de parâmetros.

```

java Copiar código

public int addHabit(HabitData habitData) {
    LocalDateTime dailyDedication = LocalDateTime.of(habitData.getDailyHoursDedication(), habitData.getDailyMinutesDedication());
    Habit habit = new Habit(habitData.getName(), habitData.getMotivation(), dailyDedication, habitData.getStartDate(), habitData.getIsConcluded());
    this.habits.add(habit);
    this.tracker.put(nextId, new Appointment());
}

```

Envie uma mensagem para o ChatGPT



O ChatGPT pode cometer erros. Considere verificar informações importantes.

4.2 - StudyObjective

```
    },  
    }  
    return habits;  
  }  
}
```

refatore o code smell Long Parameter List utilizando a técnica de refatoração introduce parameter.

< 2/2 >



Para refatorar o code smell "Long Parameter List" no método `addHabit` da classe `HabitTracker`, podemos utilizar a técnica de introduzir um parâmetro por meio de um objeto que encapsule os argumentos relacionados. Vamos criar uma classe auxiliar chamada `HabitData` que agrupará os parâmetros necessários para criar um hábito.

Aqui está a refatoração:

java

Copiar código

```
package org.example.studyplanner;  
  
import java.time.LocalDateTime;  
import java.time.LocalTime;  
  
public class HabitData {  
    private String name;  
    private String motivation;  
    private Integer dailyMinutesDedication;  
    private Integer dailyHoursDedication;  
    private LocalDateTime startDate;  
    private Boolean isConcluded;  
  
    public HabitData(String name, String motivation, Integer dailyMinutesDedication, Integer dailyHoursDedication, LocalDateTime startDate, Boolean isConcluded) {  
        this.name = name;  
        this.motivation = motivation;  
        this.dailyMinutesDedication = dailyMinutesDedication;  
        this.dailyHoursDedication = dailyHoursDedication;  
        this.startDate = startDate;  
        this.isConcluded = isConcluded;  
    }  
}
```

Envie uma mensagem para o ChatGPT



O ChatGPT pode cometer erros. Considere verificar informações importantes.

```

        this.startDate = startDate;
        this.isConcluded = isConcluded;
    }

    public String getName() {
        return name;
    }

    public String getMotivation() {
        return motivation;
    }


    public Integer getDailyMinutesDedication() {
        return dailyMinutesDedication;
    }

    public Integer getDailyHoursDedication() {
        return dailyHoursDedication;
    }

    public LocalDateTime getStartDate() {
        return startDate;
    }

    public Boolean getIsConcluded() {
        return isConcluded;
    }
}

```

 Copiar código


Refatorando o método `addHabit`

Agora, alteramos o método `addHabit` para aceitar uma instância de `HabitData` em vez de uma longa lista de parâmetros.

```

java
public int addHabit(HabitData habitData) {
    LocalDateTime dailyDedication = LocalDateTime.of(habitData.getDailyHoursDedication(), habitData.getDailyMinutesDedication());
    Habit habit = new Habit(habitData.getName(), habitData.getMotivation(), dailyDedication, habitData.getStartDate(), habitData.getIsConcluded());
    this.habits.add(habit);
    this.tracker.put(nextId, new ArrayList<>());
    return nextId++;
}

```

 Copiar código

Envie uma mensagem para o ChatGPT



O ChatGPT pode cometer erros. Considere verificar informações importantes.

4.3 - StudyTask

```
public void handleAssignSteps(List<String> stringProperties,
Integer numberOfSteps, boolean isImportant, LocalDateTime
startDate, LocalDateTime endDate){
    assignSteps(stringProperties.get(0), stringProperties.get(1),
stringProperties.get(2), stringProperties.get(3), stringProperties.get(4),
stringProperties.get(5), stringProperties.get(6), stringProperties.get(7),
stringProperties.get(8), numberOfSteps, isImportant, startDate,
endDate);
}

}
```

refatore o code smell Long Parameter List utilizando a técnica de refatoração introduce parameter.



Para refatorar o code smell **Long Parameter List**, podemos introduzir um novo objeto chamado **StepDetails** que encapsula todos os parâmetros relacionados às etapas do plano de estudo. Isso torna o código mais legível e organizado.

Aqui está a refatoração do código:

Criação da classe **StepDetails**

java

Copiar código

```
package org.example.studyregistry;

import java.time.LocalDateTime;

public class StepDetails {
    private String firstStep;
    private String resetStudyMechanism;
    private String consistentStep;
    private String seasonalSteps;
    private String basicSteps;
    private String mainObjectiveTitle;
    private String mainGoalTitle;
    private String mainMaterialTopic;
```

Envie uma mensagem para o ChatGPT



O ChatGPT pode cometer erros. Considere verificar informações importantes.


```

public StepDetails(String firstStep, String resetStudyMechanism, String consistentStep,
String mainObjectiveTitle, String mainGoalTitle, String mainMaterialTopic,
Integer numberOfSteps, boolean isImportant, LocalDateTime startDate,
LocalDateTime endDate) {
    this.firstStep = firstStep;
    this.resetStudyMechanism = resetStudyMechanism;
    this.consistentStep = consistentStep;
    this.seasonalSteps = seasonalSteps;
    this.basicSteps = basicSteps;
    this.mainObjectiveTitle = mainObjectiveTitle;
    this.mainGoalTitle = mainGoalTitle;
    this.mainMaterialTopic = mainMaterialTopic;
    this.mainTask = mainTask;
    this.numberOfSteps = numberOfSteps;
    this.isImportant = isImportant;
    this.startDate = startDate;
    this.endDate = endDate;
}

// Getters
public String getFirstStep() {
    return firstStep;
}

public String getResetStudyMechanism() {
    return resetStudyMechanism;
}

public String getConsistentStep() {
    return consistentStep;
}

public String getSeasonalSteps() {
    return seasonalSteps;
}

public String getBasicSteps() {
    return basicSteps;
}

public String getMainObjectiveTitle() {
    return mainObjectiveTitle;
}

public String getMainGoalTitle() {
    return mainGoalTitle;
}

```

Envie uma mensagem para o ChatGPT



O ChatGPT pode cometer erros. Considere verificar informações importantes.

```
public String getResetStudyMechanism() {  
    return resetStudyMechanism;  
}  
  
public String getConsistentStep() {  
    return consistentStep;  
}  
  
public String getSeasonalSteps() {  
    return seasonalSteps;  
}  
  
public String getBasicSteps() {  
    return basicSteps;  
}  
  
public String getMainObjectiveTitle() {  
    return mainObjectiveTitle;  
}  
  
public String getMainGoalTitle() {  
    return mainGoalTitle;  
}  
  
public String getMainMaterialTopic() {  
    return mainMaterialTopic;  
}  
  
public String getMainTask() {  
    return mainTask;  
}  
  
public Integer getNumberOfSteps() {  
    return numberOfSteps;  
}  
  
public boolean isImportant() {  
    return isImportant;  
}  
  
public LocalDateTime getStartDate() {  
    return startDate;  
}
```



Envie uma mensagem para o ChatGPT



O ChatGPT pode cometer erros. Considere verificar informações importantes.

4.4 - Study Plan

assignSteps(stringPropertie
s.get(0),
stringProperties.get(1),
stringProperties.get(2),
stringProperties.get(3),
stringProperties.get(4),
stringProperties.get(5),
stringProperties.get(6),
stringProperties.get(7),
stringProperties.get(8),
numberOfSteps,
isImportant, startDate,
endDate);
}

}

refatore o code smell Long
Parameter List utilizando a
técnica de refatoração
introduce parameter.

Study Plan Refactor

Responder no chat >

A refatoração foi realizada introduzindo a classe `StudyPlanStepsDetails`, que encapsula todos os parâmetros relacionados às etapas do plano de estudo. Isso reduz significativamente o número de parâmetros nos métodos `assignSteps` e `handleAssignSteps`, melhorando a legibilidade e manutenção do código. Caso precise de mais ajustes ou explicações, estou à disposição!

Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.

Study Plan Refactor

```
1 package org.example.studyregistry;  
2  
3 import java.time.LocalDate;  
4 import java.time.format.DateTimeFormatter;  
5 import java.util.ArrayList;  
6 import java.util.Arrays;  
7 import java.util.List;  
8  
9 public class StudyPlan extends Registry {  
10     private StudyObjective objective;  
11     private List<String> steps;  
12  
13     public StudyPlan(String planName, StudyObjective objective, List<StudyMaterial> materials) {  
14         this.name = planName;  
15         this.objective = objective;  
16         this.steps = new ArrayList<>();  
17     }  
18  
19     @Override  
20     public String toString() {  
21         return "Plan: " + name + ",\nObjective: " + objective.getDescription() + ",\nSteps: " + String.join(", ", steps);  
22     }  
23  
24     public List<String> getSteps() {  
25         return steps;  
26     }  
27  
28     public StudyObjective getObjective() {  
29         return objective;  
30     }  
31  
32     public void assignObjective(StudyObjective objective) {  
33         this.objective = objective;  
34     }  
35  
36     public void addSingleStep(String toAdd) {  
37         steps.add(toAdd);  
38     }  
39  
40     public void assignSteps(StudyPlanStepsDetails stepsDetails) {  
41         DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");  
42  
43         this.steps = new ArrayList<>(Arrays.asList(  
44             stepsDetails.getFirstStep(),  
45             stepsDetails.getResetStudyMechanism(),  
46             stepsDetails.getConsistentStep(),  
47             stepsDetails.getSeasonalSteps(),  
48             stepsDetails.getBasicSteps(),  
49             "Number of steps: " + stepsDetails.getNumberOfSteps().toString(),  
50             "Is it important to you? " + stepsDetails.isImportant(),  
51             stepsDetails.getStartDate().format(formatter),  
52             stepsDetails.getEndDate().format(formatter),  
53             stepsDetails.getMainObjectiveTitle(),  
54             stepsDetails.getMainGoalTitle(),  
55             stepsDetails.getMainMaterialTopic(),  
56             stepsDetails.getMainTask()  
57         ));  
58     }  
59  
60     public void handleAssignSteps(StudyPlanStepsDetails stepsDetails) {
```

refatore o code smell Long Parameter List utilizando a técnica de refatoração introduce parameter.

Responder no chat >

A refatoração foi realizada introduzindo a classe `StudyPlanStepsDetails`, que encapsula todos os parâmetros relacionados às etapas do plano de estudo. Isso reduz significativamente o número de parâmetros nos métodos `assignSteps` e `handleAssignSteps`, melhorando a legibilidade e manutenção do código. Caso precise de mais ajustes ou explicações, estou à disposição!

Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.

```

54         stepsDetails.getMainGoalTitle(),
55         stepsDetails.getMainGoalTitle(),
56         stepsDetails.getMainMaterialTopic(),
57         stepsDetails.getMainTask());
58     });
59 }
60 public void handleAssignSteps(StudyPlanStepsDetails stepsDetails) {
61     assignSteps(stepsDetails);
62 }
63 }
64
65 class StudyPlanStepsDetails {
66     private String firstStep;
67     private String resetStudyMechanism;
68     private String consistentStep;
69     private String seasonalSteps;
70     private String basicSteps;
71     private String mainObjectiveTitle;
72     private String mainGoalTitle;
73     private String mainMaterialTopic;
74     private String mainTask;
75     private Integer numberOfSteps;
76     private boolean isImportant;
77     private LocalDateTime startDate;
78     private LocalDateTime endDate;
79
80     public StudyPlanStepsDetails(String firstStep, String resetStudyMechanism, String consistentStep, String seasonalSteps,
81                                 String basicSteps, String mainObjectiveTitle, String mainGoalTitle, String mainMaterialTopic,
82                                 String mainTask, Integer numberOfSteps, boolean isImportant, LocalDateTime startDate, LocalDateTime endDate) {
83         this.firstStep = firstStep;
84         this.resetStudyMechanism = resetStudyMechanism;
85         this.consistentStep = consistentStep;
86         this.seasonalSteps = seasonalSteps;
87         this.basicSteps = basicSteps;
88         this.mainObjectiveTitle = mainObjectiveTitle;
89         this.mainGoalTitle = mainGoalTitle;
90         this.mainMaterialTopic = mainMaterialTopic;
91         this.mainTask = mainTask;
92         this.numberOfSteps = numberOfSteps;
93         this.isImportant = isImportant;
94         this.startDate = startDate;
95         this.endDate = endDate;
96     }
97
98     public String getFirstStep() {
99         return firstStep;
100     }
101
102     public String getResetStudyMechanism() {
103         return resetStudyMechanism;
104     }
105
106     public String getConsistentStep() {
107         return consistentStep;
108     }
109
110     public String getSeasonalSteps() {
111         return seasonalSteps;
112     }
113
114     public String getBasicSteps() {

```