

# LongParameterList

## AudioReference

```
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() {
        return audioQuality;
    }

    public static AudioQuality audioQualityAdapter(String quality){
        return switch (quality.toLowerCase()) {
            case "low" -> AudioQuality.LOW;
            case "medium" -> AudioQuality.MEDIUM;
            case "high" -> AudioQuality.HIGH;
            case "very_high" -> AudioQuality.VERY_HIGH;
            default -> null;
        };
    }

    public void setAudioQuality(AudioQuality audioQuality) {
        this.audioQuality = audioQuality;
    }

    public void editAudio(AudioQuality audioQuality, boolean isDownloadable, String title,
        String description, String link, String accessRights, String license, String language, int
        rating, int viewCount, int shareCount){
        editBasic(title, description, link);
        this.setAccessRights(accessRights);
        this.setLicense(license);
        this.setAudioQuality(audioQuality);
        editVideoAttributes(rating, language, viewCount, shareCount, isDownloadable);
    }

    public void editAudioAdapter(List<String> properties, List<Integer> intProperties,
        AudioQuality audioQuality, boolean isDownloadable){
        this.editAudio(audioQuality, isDownloadable, properties.get(0), properties.get(1),
        properties.get(2), properties.get(3), properties.get(4), properties.get(5),
        intProperties.get(0), intProperties.get(1), intProperties.get(2));
    }

    private void editVideoAttributes(int rating, String language, int viewCount, int
        shareCount, boolean isDownloadable){
        this.setRating(rating);
        this.setShareCount(shareCount);
        this.setViewCount(viewCount);
        this.setDownloadable(isDownloadable);
        this.setLanguage(language);
    }

    public void editBasic(String title, String description, String link){
        this.setTitle(title);
        this.setDescription(description);
        this.setLink(link);
    }

}

fix the Long parameter list code smell using the "Introduce Parameter Object." refactoring
technique
```

- ✦ Certainly, let's refactor the `editAudio` method in the `AudioReference` class to address the "Long Parameter List" code smell using the "Introduce Parameter Object" technique.

#### 1. Create a new Parameter Object:

```
Java

public class AudioReferenceEditParams {
    private AudioQuality audioQuality;
    private boolean isDownloadable;
    private String title;
    private String description;
    private String link;
    private String accessRights;
    private String license;
    private String language;
    private int rating;
    private int viewCount;
    private int shareCount;

    // Constructor and getters/setters for all fields
}
```

#### 2. Modify the `editAudio` method:

```
Java

public void editAudio(AudioReferenceEditParams params) {
    editBasic(params.getTitle(), params.getDescription(), params.getLink());
    this.setAccessRights(params.getAccessRights());
    this.setLicense(params.getLicense());
    this.setAudioQuality(params.getAudioQuality());
    editVideoAttributes(params.getRating(), params.getLanguage(), params.getViewCount());
}
```

#### 3. Modify the `editAudioAdapter` method:

```
Java

public void editAudioAdapter(List<String> properties, List<Integer> intProperties,
    AudioReferenceEditParams params = new AudioReferenceEditParams();
    params.setTitle(properties.get(0));
    params.setDescription(properties.get(1));
    params.setLink(properties.get(2));
    params.setAccessRights(properties.get(3));
    params.setLicense(properties.get(4));
    params.setLanguage(properties.get(5));
    params.setRating(intProperties.get(0));
    params.setViewCount(intProperties.get(1));
    params.setShareCount(intProperties.get(2));
    params.setAudioQuality(audioQuality);
    params.setDownloadable(isDownloadable);

    this.editAudio(params);
}
```

#### Explanation:

- We created a new class `AudioReferenceEditParams` to encapsulate all the parameters of the `editAudio` method.
- The `editAudio` method now accepts an instance of `AudioReferenceEditParams` instead of individual parameters.
- The `editAudioAdapter` method creates an instance of `AudioReferenceEditParams` and populates it with the necessary values before calling the `editAudio` method.

#### Benefits:

- **Improved Readability:** The code is now more readable as the long parameter list is replaced with a single object.
- **Reduced Complexity:** The `editAudio` method is now simpler and easier to understand.
- **Increased Maintainability:** Changes to the parameters can be made in one place (the `AudioReferenceEditParams` class) without affecting other parts of the code.
- **Enhanced Reusability:** The `AudioReferenceEditParams` object can be reused in other parts of the application.

This refactoring significantly improves the code's maintainability and readability by addressing the "Long Parameter List" code smell effectively.



use record instead, since dataclasses in this scenario are code smells



You're right, using `record` in Java is a more concise and potentially more performant alternative to creating a separate class for the parameter object in this case.

Here's the refactored code using `record`:

Java



```
public record AudioReferenceEditParams(
    AudioQuality audioQuality,
    boolean isDownloadable,
    String title,
    String description,
    String link,
    String accessRights,
    String license,
    String language,
    int rating,
    int viewCount,
    int shareCount
) {
}

public void editAudio(AudioReferenceEditParams params) {
    editBasic(params.title(), params.description(), params.link());
    this.setAccessRights(params.accessRights());
    this.setLicense(params.license());
    this.setAudioQuality(params.audioQuality());
    editVideoAttributes(params.rating(), params.language(), params.viewCount(), pa
}

public void editAudioAdapter(List<String> properties, List<Integer> intProperties,
    var params = new AudioReferenceEditParams(
        audioQuality,
        isDownloadable,
        properties.get(0),
        properties.get(1),
        properties.get(2),
        properties.get(3),
        properties.get(4),
        properties.get(5),
        intProperties.get(0),
        intProperties.get(1),
        intProperties.get(2)
    );

    this.editAudio(params);
}
```

#### Key improvements:

- **Conciseness:** `record` provides a more concise syntax for defining immutable classes.
- **Readability:** The code is more readable as the parameter object definition is more compact.
- **Performance:** `record` can potentially offer better performance compared to manually defining a class with getters and setters.

This refactored version effectively addresses the "Long Parameter List" code smell while maintaining good code quality and potentially improving performance.

nice, now please refactor this related method :

```
private void editAudio(AudioReference audioReference){
    handleMethodHeader("(Audio Edit)");
    System.out.println("Type the following info: AudioReference. AudioQuality audioQuality,...
```



Certainly, let's refactor the `editAudio` method to improve its readability and maintainability.

### 1. Use a Parameter Object:

We can leverage the `AudioReferenceEditParams` record we created earlier:

```
Java

private void editAudio(AudioReference audioReference) {
    handleMethodHeader("(Audio Edit)");
    System.out.println("Type the following info: AudioReference. AudioQuality aud:
        \"String title, String description, String link, String accessRights, S
        \"int viewCount, int shareCount \\n\");

    AudioReferenceEditParams params = new AudioReferenceEditParams(
        AudioReference.audioQualityAdapter(getInput()),
        Boolean.parseBoolean(getInput()),
        getInput(),
        getInput(),
        getInput(),
        getInput(),
        getInput(),
        Integer.parseInt(getInput()),
        Integer.parseInt(getInput()),
        Integer.parseInt(getInput())
    );

    audioReference.editAudio(params);
}
```

### 2. Extract Input Logic:

To further improve readability, we can extract the input gathering logic into a separate helper method:

```
Java

private AudioReferenceEditParams getAudioEditParamsFromUserInput() {
    System.out.println("Type the following info: AudioReference. AudioQuality aud:
        \"String title, String description, String link, String accessRights, S
        \"int viewCount, int shareCount \\n\");

    return new AudioReferenceEditParams(
        AudioReference.audioQualityAdapter(getInput()),
        Boolean.parseBoolean(getInput()),
        getInput(),
        getInput(),
        getInput(),
        getInput(),
        getInput(),
        Integer.parseInt(getInput()),
        Integer.parseInt(getInput()),
        Integer.parseInt(getInput())
    );
}

private void editAudio(AudioReference audioReference) {
    handleMethodHeader("(Audio Edit)");
    AudioReferenceEditParams params = getAudioEditParamsFromUserInput();
    audioReference.editAudio(params);
}
```

### 3. Consider Using a Builder Pattern (Optional):

For even more flexibility and potential future enhancements, you could consider using a Builder pattern to construct the `AudioReferenceEditParams` object.

#### Key Improvements:

- **Reduced Method Complexity:** The original `editAudio` method is now significantly shorter and easier to understand.
- **Improved Readability:** The code is more readable due to the use of the `AudioReferenceEditParams` record and the extracted `getAudioEditParamsFromUserInput` method.
- **Increased Maintainability:** Changes to input handling or parameter validation can be made in the `getAudioEditParamsFromUserInput` method without affecting the main `editAudio` logic.

This refactored version provides a more maintainable and readable solution while effectively addressing the concerns of the original code.