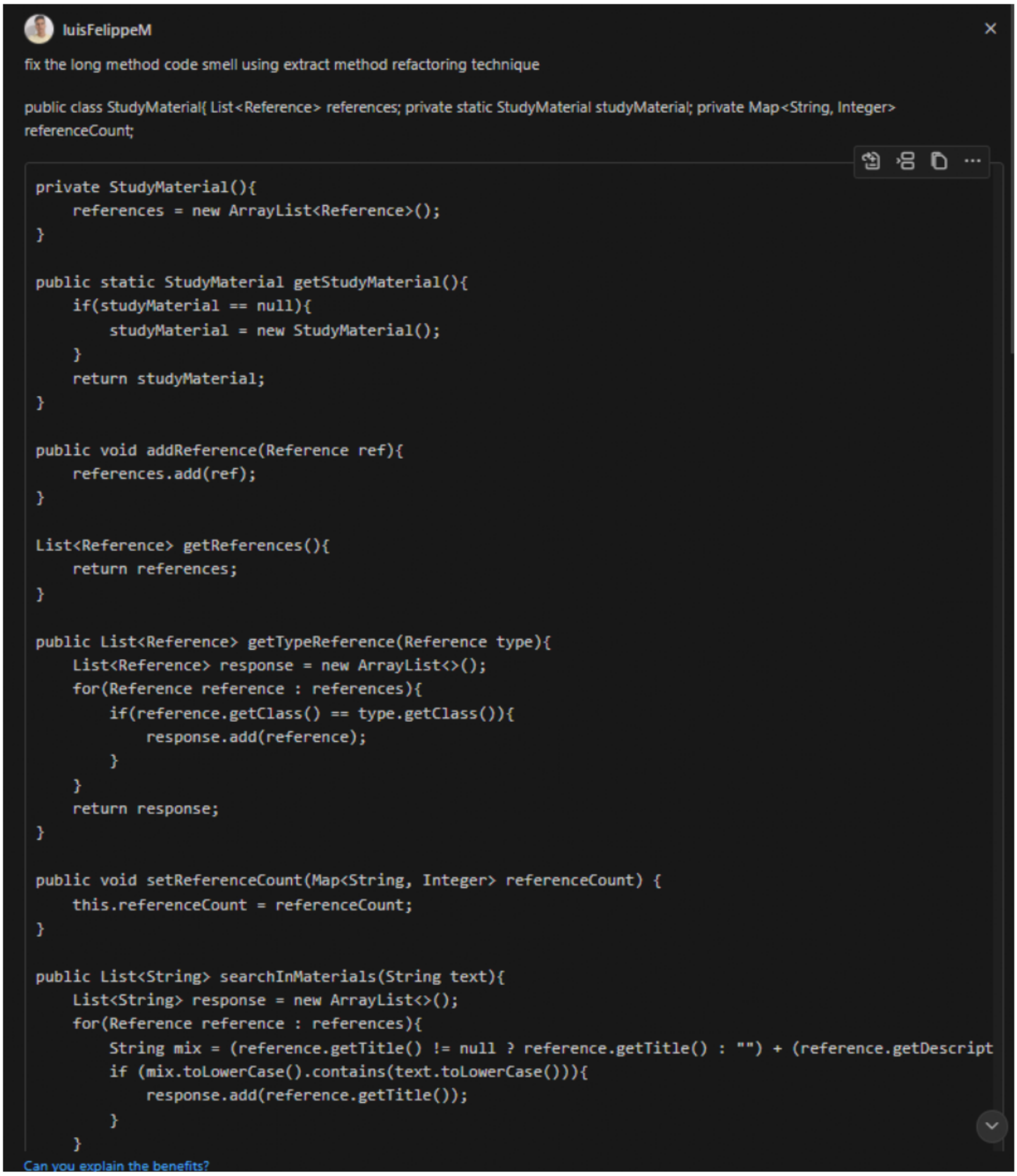


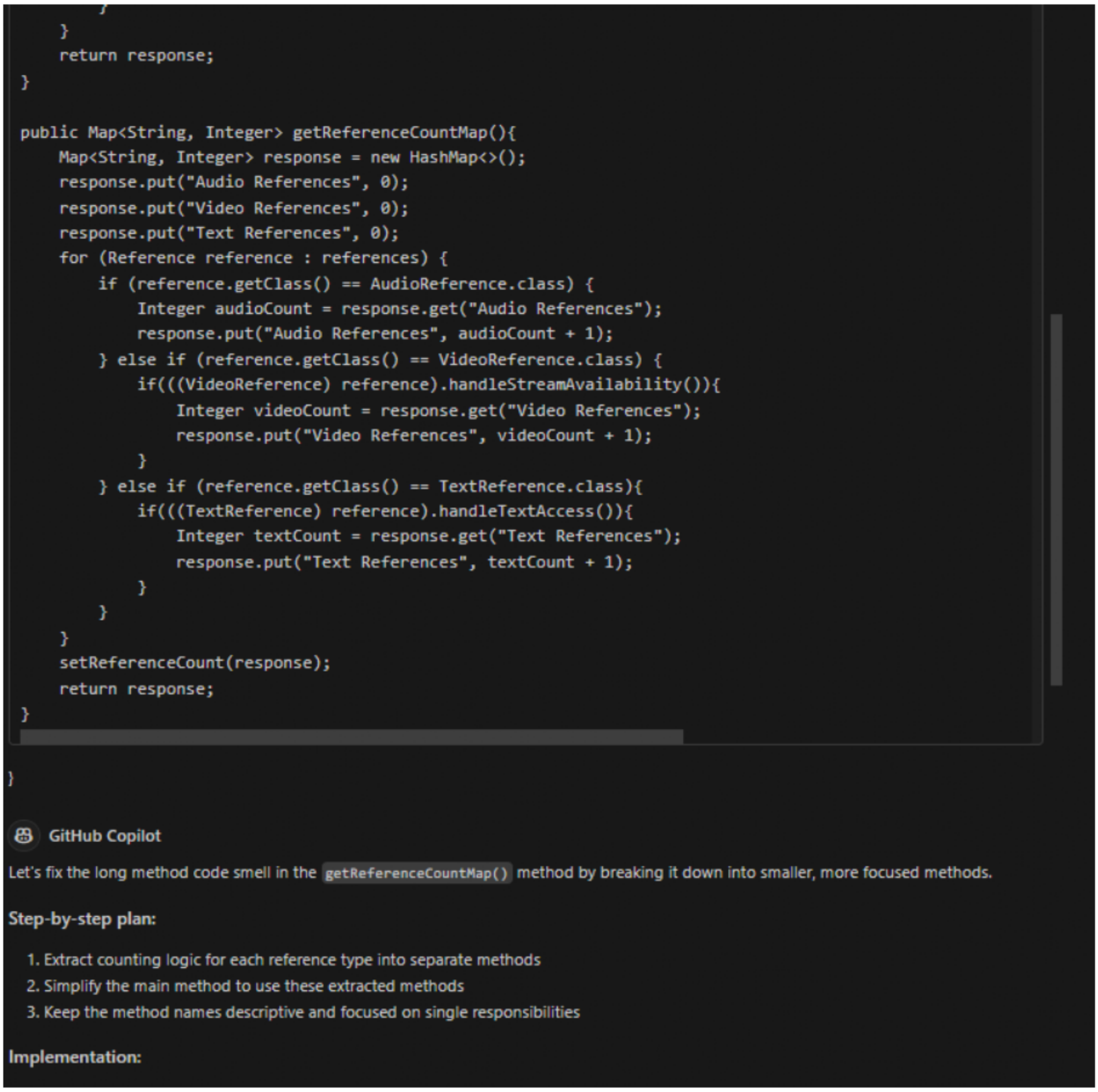
luisFelippeM

fix the long method code smell using extract method refactoring technique

public class StudyMaterial{ List<Reference> references; private static StudyMaterial studyMaterial; private Map<String, Integer> referenceCount;

```
private StudyMaterial(){
    references = new ArrayList<Reference>();
}

public static StudyMaterial getStudyMaterial(){
    if(studyMaterial == null){
        studyMaterial = new StudyMaterial();
    }
    return studyMaterial;
}

public void addReference(Reference ref){
    references.add(ref);
}

List<Reference> getReferences(){
    return references;
}

public List<Reference> getTypeReference(Reference type){
    List<Reference> response = new ArrayList<>();
    for(Reference reference : references){
        if(reference.getClass() == type.getClass()){
            response.add(reference);
        }
    }
    return response;
}

public void setReferenceCount(Map<String, Integer> referenceCount) {
    this.referenceCount = referenceCount;
}

public List<String> searchInMaterials(String text){
    List<String> response = new ArrayList<>();
    for(Reference reference : references){
        String mix = (reference.getTitle() != null ? reference.getTitle() : "") + (reference.getDescript
        if (mix.toLowerCase().contains(text.toLowerCase())){
            response.add(reference.getTitle());
        }
    }
```

- Can you explain the benefits?



```
    }
    return response;
}

public Map<String, Integer> getReferenceCountMap(){
    Map<String, Integer> response = new HashMap<>();
    response.put("Audio References", 0);
    response.put("Video References", 0);
    response.put("Text References", 0);
    for (Reference reference : references) {
        if (reference.getClass() == AudioReference.class) {
            Integer audioCount = response.get("Audio References");
            response.put("Audio References", audioCount + 1);
        } else if (reference.getClass() == VideoReference.class) {
            if(((VideoReference) reference).handleStreamAvailability()){
                Integer videoCount = response.get("Video References");
                response.put("Video References", videoCount + 1);
            }
        } else if (reference.getClass() == TextReference.class){
            if(((TextReference) reference).handleTextAccess()){
                Integer textCount = response.get("Text References");
                response.put("Text References", textCount + 1);
            }
        }
    }
    setReferenceCount(response);
    return response;
}
```

```
}
```

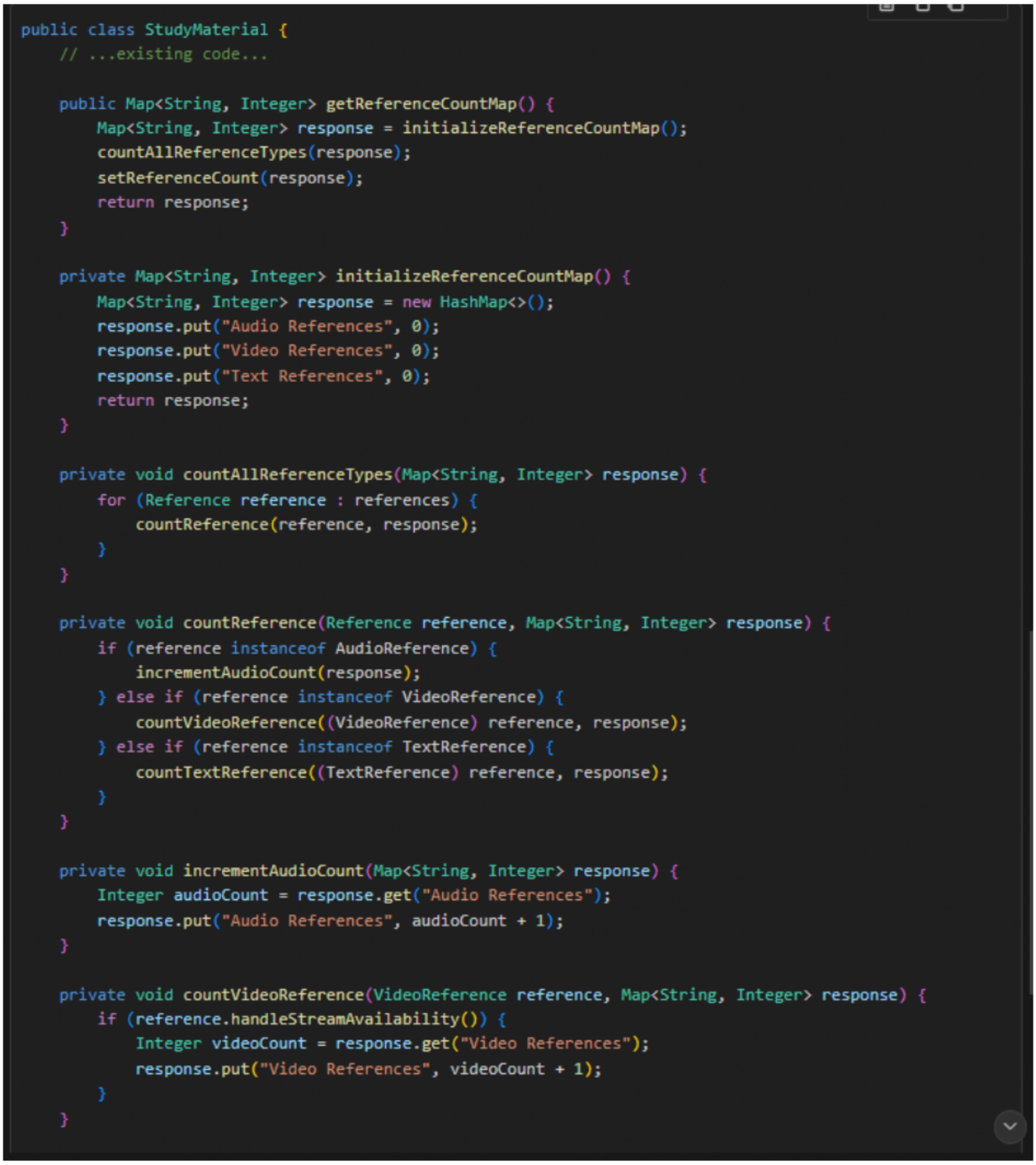
GitHub Copilot

Let's fix the long method code smell in the `getReferenceCountMap()` method by breaking it down into smaller, more focused methods.

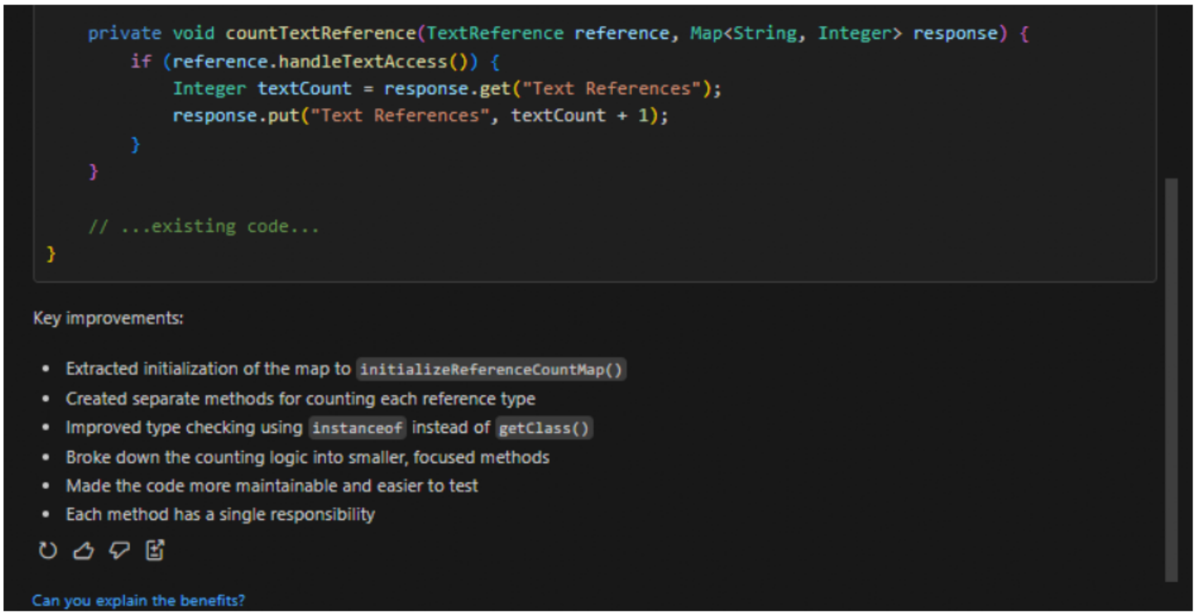
Step-by-step plan:

1. Extract counting logic for each reference type into separate methods
2. Simplify the main method to use these extracted methods
3. Keep the method names descriptive and focused on single responsibilities

Implementation:



```
public class StudyMaterial {
    // ...existing code...

    public Map<String, Integer> getReferenceCountMap() {
        Map<String, Integer> response = initializeReferenceCountMap();
        countAllReferenceTypes(response);
        setReferenceCount(response);
        return response;
    }

    private Map<String, Integer> initializeReferenceCountMap() {
        Map<String, Integer> response = new HashMap<>();
        response.put("Audio References", 0);
        response.put("Video References", 0);
        response.put("Text References", 0);
        return response;
    }

    private void countAllReferenceTypes(Map<String, Integer> response) {
        for (Reference reference : references) {
            countReference(reference, response);
        }
    }

    private void countReference(Reference reference, Map<String, Integer> response) {
        if (reference instanceof AudioReference) {
            incrementAudioCount(response);
        } else if (reference instanceof VideoReference) {
            countVideoReference((VideoReference) reference, response);
        } else if (reference instanceof TextReference) {
            countTextReference((TextReference) reference, response);
        }
    }

    private void incrementAudioCount(Map<String, Integer> response) {
        Integer audioCount = response.get("Audio References");
        response.put("Audio References", audioCount + 1);
    }

    private void countVideoReference(VideoReference reference, Map<String, Integer> response) {
        if (reference.handleStreamAvailability()) {
            Integer videoCount = response.get("Video References");
            response.put("Video References", videoCount + 1);
        }
    }
```

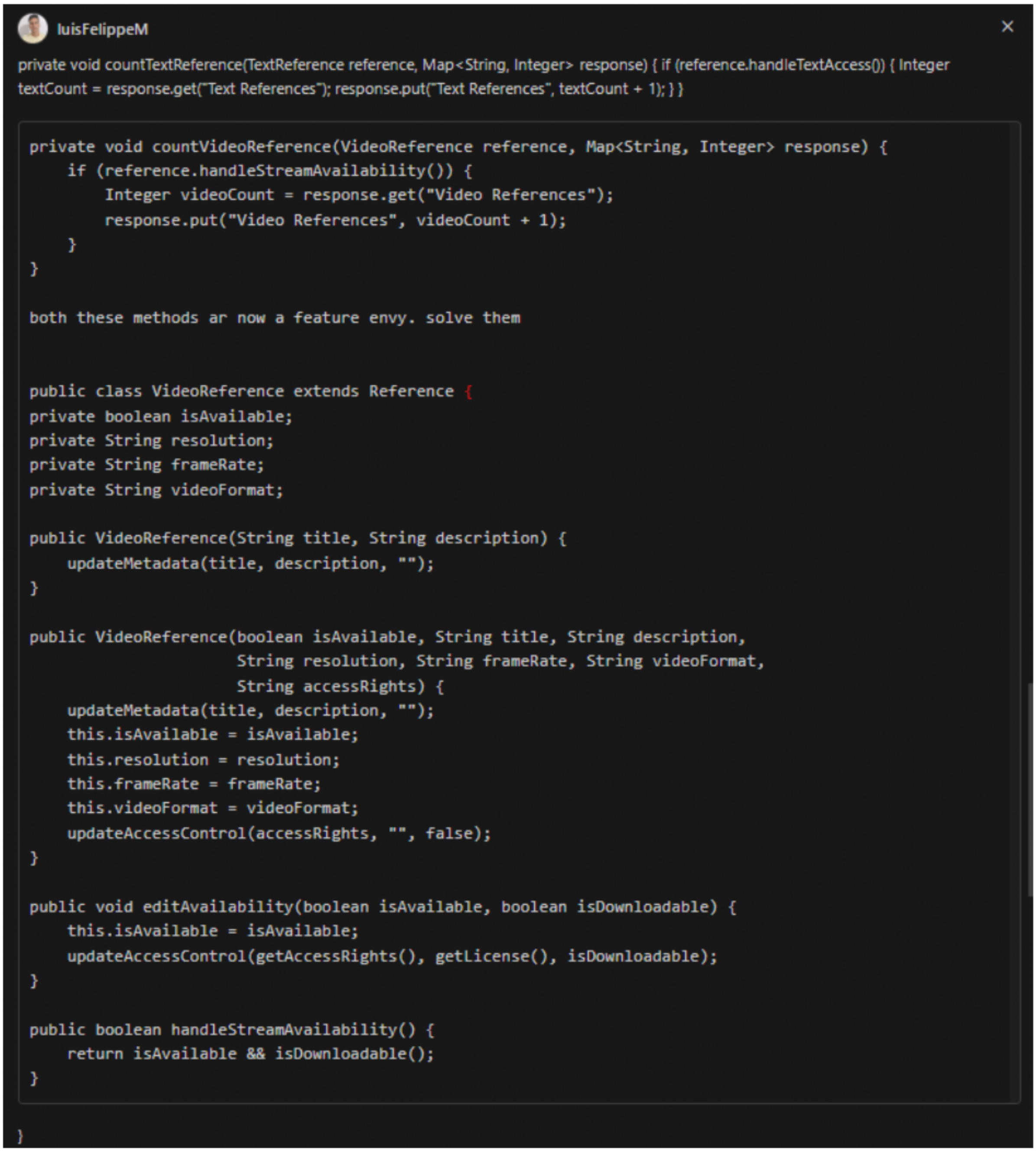


```
    private void countTextReference(TextReference reference, Map<String, Integer> response) {
        if (reference.handleTextAccess()) {
            Integer textCount = response.get("Text References");
            response.put("Text References", textCount + 1);
        }
    }

    // ...existing code...
}
```

Key improvements:

- Extracted initialization of the map to `initializeReferenceCountMap()`
- Created separate methods for counting each reference type
- Improved type checking using `instanceof` instead of `getClass()`
- Broke down the counting logic into smaller, focused methods
- Made the code more maintainable and easier to test
- Each method has a single responsibility

Can you explain the benefits?

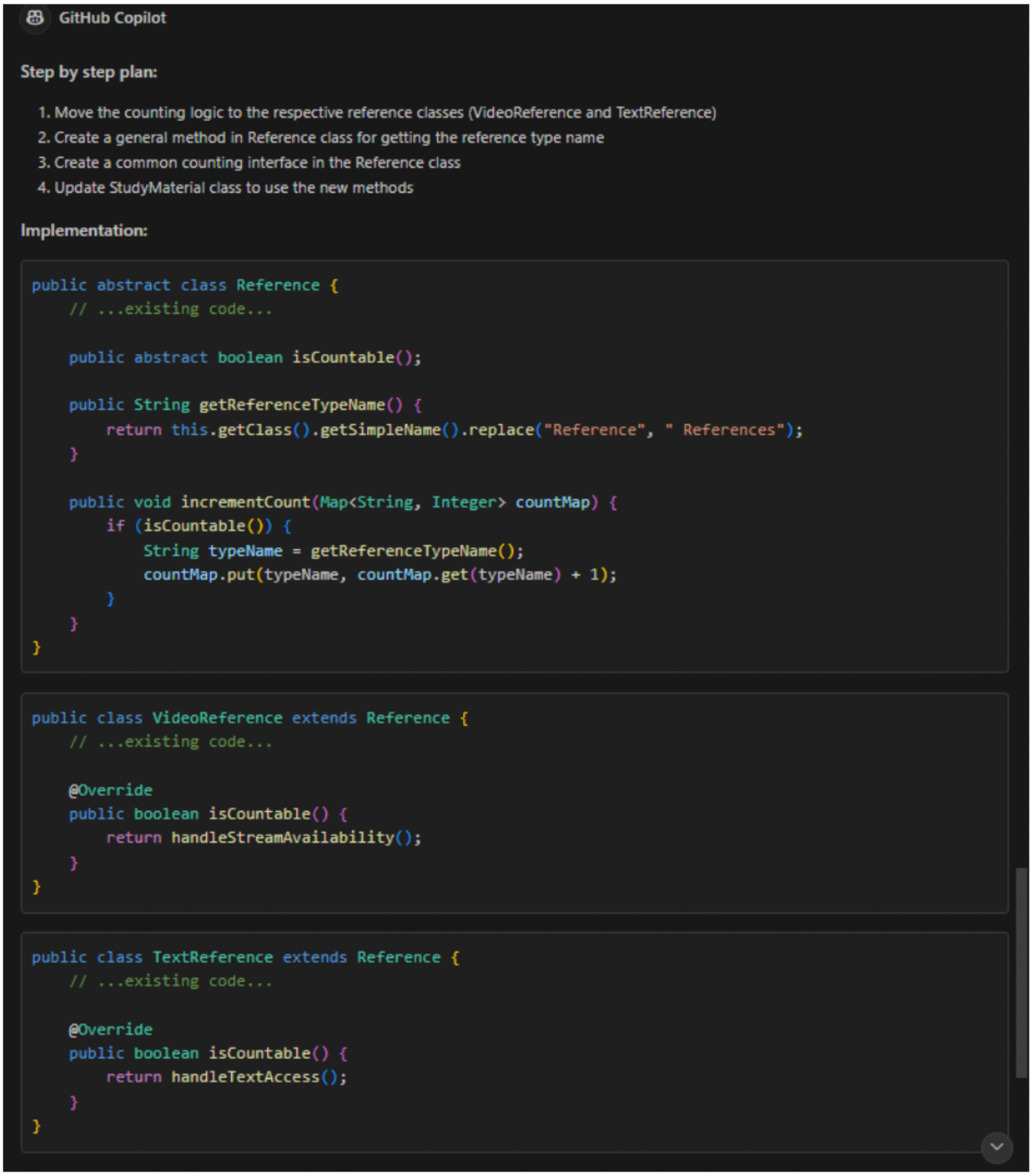


luisFelippeM

private void countTextReference(TextReference reference, Map<String, Integer> response) { if (reference.handleTextAccess()) { Integer textCount = response.get("Text References"); response.put("Text References", textCount + 1); } }

```
private void countVideoReference(VideoReference reference, Map<String, Integer> response) {
    if (reference.handleStreamAvailability()) {
        Integer videoCount = response.get("Video References");
        response.put("Video References", videoCount + 1);
    }
}

both these methods ar now a feature envy. solve them


public class VideoReference extends Reference {
private boolean isAvailable;
private String resolution;
private String frameRate;
private String videoFormat;

public VideoReference(String title, String description) {
    updateMetadata(title, description, "");
}

public VideoReference(boolean isAvailable, String title, String description,
                      String resolution, String frameRate, String videoFormat,
                      String accessRights) {
    updateMetadata(title, description, "");
    this.isAvailable = isAvailable;
    this.resolution = resolution;
    this.frameRate = frameRate;
    this.videoFormat = videoFormat;
    updateAccessControl(accessRights, "", false);
}

public void editAvailability(boolean isAvailable, boolean isDownloadable) {
    this.isAvailable = isAvailable;
    updateAccessControl(getAccessRights(), getLicense(), isDownloadable);
}

public boolean handleStreamAvailability() {
    return isAvailable && isDownloadable();
}
```

```
}
```



Step by step plan:

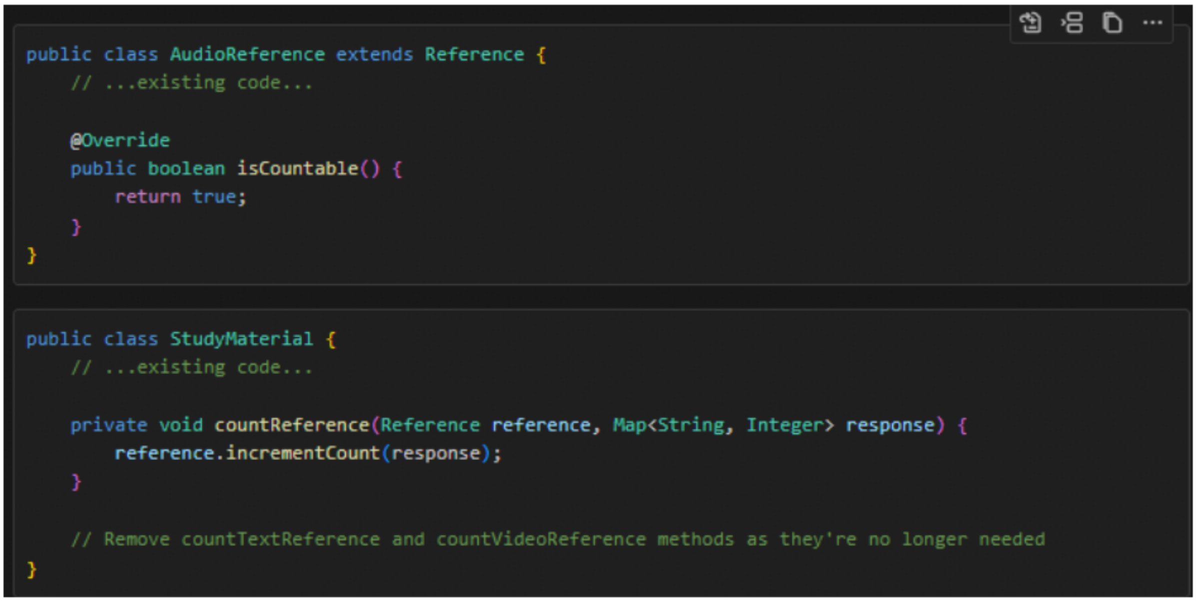
1. Move the counting logic to the respective reference classes (VideoReference and TextReference)
2. Create a general method in Reference class for getting the reference type name
3. Create a common counting interface in the Reference class
4. Update StudyMaterial class to use the new methods

Implementation:

```
public abstract class Reference {
    // ...existing code...

    public abstract boolean isCountable();

    public String getReferenceTypeName() {
        return this.getClass().getSimpleName().replace("Reference", " References");
    }

    public void incrementCount(Map<String, Integer> countMap) {
        if (isCountable()) {
            String typeName = getReferenceTypeName();
            countMap.put(typeName, countMap.get(typeName) + 1);
        }
    }
}
```

```
public class VideoReference extends Reference {
    // ...existing code...

    @Override
    public boolean isCountable() {
        return handleStreamAvailability();
    }
}
```

```
public class TextReference extends Reference {
    // ...existing code...

    @Override
    public boolean isCountable() {
        return handleTextAccess();
    }
}
```



```
public class AudioReference extends Reference {
    // ...existing code...

    @Override
    public boolean isCountable() {
        return true;
    }
}
```

```
public class StudyMaterial {
    // ...existing code...

    private void countReference(Reference reference, Map<String, Integer> response) {
        reference.incrementCount(response);
    }

    // Remove countTextReference and countVideoReference methods as they're no longer needed
}
```