```
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;
}
```

fix the LongMethod code smell using the ExtractMethod refactoring technique

Java

```java
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);

    response.putAll(countAudioReferences());
    response.putAll(countVideoReferences());
    response.putAll(countTextReferences());

    setReferenceCount(response);
    return response;
}

private Map<String, Integer> countAudioReferences() {
    Map<String, Integer> audioCounts = new HashMap<>();
    audioCounts.put("Audio References", 0);

    for (Reference reference : references) {
        if (reference.getClass() == AudioReference.class) {
            audioCounts.put("Audio References", audioCounts.get("Audio References"
        }
    }

    return audioCounts;
}

private Map<String, Integer> countVideoReferences() {
    Map<String, Integer> videoCounts = new HashMap<>();
    videoCounts.put("Video References", 0);

    for (Reference reference : references) {
        if (reference.getClass() == VideoReference.class && ((VideoReference) refe
            videoCounts.put("Video References", videoCounts.get("Video References"
        }
    }

    return videoCounts;
}

private Map<String, Integer> countTextReferences() {
    Map<String, Integer> textCounts = new HashMap<>();
    textCounts.put("Text References", 0);

    for (Reference reference : references) {
        if (reference.getClass() == TextReference.class && ((TextReference) refere
            textCounts.put("Text References", textCounts.get("Text References") + 
        }
    }

    return textCounts;
}
```

Explanation:

1. **Extract Methods:**
   - The original `getReferenceCountMap()` method has been broken down into three smaller, more focused methods:
     - `countAudioReferences()`
     - `countVideoReferences()`
     - `countTextReferences()`
   - Each of these extracted methods handles the counting logic for a specific type of reference (Audio, Video, Text).
2. **Improved Readability:**
   - The original method was quite long and complex, making it difficult to understand.
   - By extracting the counting logic into separate methods, the `getReferenceCountMap()` method becomes much more concise and easier to read.
3. **Increased Maintainability:**
   - If you need to change the way any of the reference counts are calculated, you only need to modify the corresponding extracted method.
   - This makes the code easier to maintain and less prone to errors.
4. **Improved Reusability:**
   - The extracted methods can potentially be reused in other parts of your code if needed.

This refactoring addresses the "Long Method" code smell by breaking down the original method into smaller, more manageable units. It improves the overall code quality by enhancing readability, maintainability, and reusability.