Ejercicio 3

```
public class Document {

    List<String> words; // (1)
    //(2)
    public long characterCount() {
          long count = this.words //(4)
        .stream()
        .mapToLong(w -> w.length())
        .sum();
     return count;
    }
    //(2)
    public long calculateAvg() { // (3)
         long avgLength = this.words // (4)
        .stream()
        .mapToLong(w -> w.length())
        .sum() / this.words.size();
          return avgLength;
    }
// Resto del código que no importa
}
```

Tareas:

1. Enumere los code smell y que refactorings utilizará para solucionarlos.
2. Aplique los refactorings encontrados, mostrando el código refactorizado luego de aplicar cada uno.
3. Analice el código original y detecte si existe un problema al calcular las estadísticas. Explique cuál es el error y en qué casos se da ¿El error identificado sigue presente luego de realizar los refactorings? En caso de que no esté presente, ¿en qué momento se resolvió? De acuerdo a lo visto en la teoría, ¿podemos considerar esto un refactoring?

Code Smells:
(1): Rompe el encapsulamiento:
Refactoring: Aplicar encapsulamiento de campo: poner atributo con modificador de acceso privado

```
Consultar: Cuando hago un refactor sobre violacion de encapsulamiento, hace falta siempre agregar los metodos setters y getters? en este caso en especifico no tengo el contexto de si es necesario o no tener getters o setters para la lista...
```

```java
public class Document {
    private List<String> words;

    public long characterCount() {
          long count = this.words
        .stream()
        .mapToLong(w -> w.length())
        .sum();
     return count;
    }

    public long calculateAvg() {
         long avgLength = this.words
        .stream()
        .mapToLong(w -> w.length())
        .sum() / this.words.size();
          return avgLength;
    }

     public List<String> getWords() {
            return new ArrayList<>(this.words);
        }

        public void setWords(List<String> words) {
            this.words = new ArrayList<>(words);
        }
// Resto del código que no importa
}
```

(2): Codigo duplicado: Los metodos de characterCount y calculateAvg usan la misma logica para sumar la longiutud de las palabras. Lo mejor seria tener un metodo en comun para poder reutilizarlo

Refactor: Extraer metodo

```java
public class Document {
    private List<String> words;
```

```
    public long characterCount() {
          long count = this.words
        .stream()
        .mapToLong(w -> w.length())
        .sum();
     return count;
    }

    public long calculateAvg() {
        long avgLength = this.characterCount() / this.words.size();
        return avgLength;
    }

     public List<String> getWords() {
            return new ArrayList<>(this.words);
        }

        public void setWords(List<String> words) {
            this.words = new ArrayList<>(words);
        }
// Resto del código que no importa
}
```

(3): Nombre de metodo poco autoexplicativo: El nombre calculateAvg no deja en claro sobre que dato se calcula el promedio
Refactoring: Renombrar metodo

```
public class Document {
    private List<String> words;

    public long characterCount() {
          long count = this.words
        .stream()
        .mapToLong(w -> w.length())
        .sum();
     return count;
    }

    public long calculateAverageCharacterCount() {
        long avgLength = this.characterCount() / this.words.size();
        return avgLength;
    }

     public List<String> getWords() {
            return new ArrayList<>(this.words);
```

```
        }

        public void setWords(List<String> words) {
            this.words = new ArrayList<>(words);
        }
// Resto del código que no importa
}
```

(4): Variable temporales innecesarias: Se puede retornar el valor directamente sin la necesidad de tener una variable temporal.
Refactoring: Eliminar variable temporal

```
public class Document {
    private List<String> words;

    public long characterCount() {
        return this.words
            .stream()
            .mapToLong(w -> w.length())
            .sum();
    }

    public long calculateAverageCharacterCount() {
        return this.characterCount() / this.words.size();
    }

     public List<String> getWords() {
            return new ArrayList<>(this.words);
        }

        public void setWords(List<String> words) {
            this.words = new ArrayList<>(words);
        }
// Resto del código que no importa
}
```

Errores del codigo que no se pueden solucionar con refactoring ya que son errores de logica:

Error 1:

Posible division por cero en el metodo `calculateAverageCharacterCount` si la lista no contiene ninguna palabra.

Error 2:

El metodo `calculateAverageCharacterCount` devuelve un tipo de dato `long` , lo cual es distinto al tipo de dato que se produce luego de hacer:
`this.characterCount() / this.words.size()` esto devuelve un `double`