En todos los pantallazos, la parte izquierda representa al código sin refactorizar, y el derecho al refactorizado.

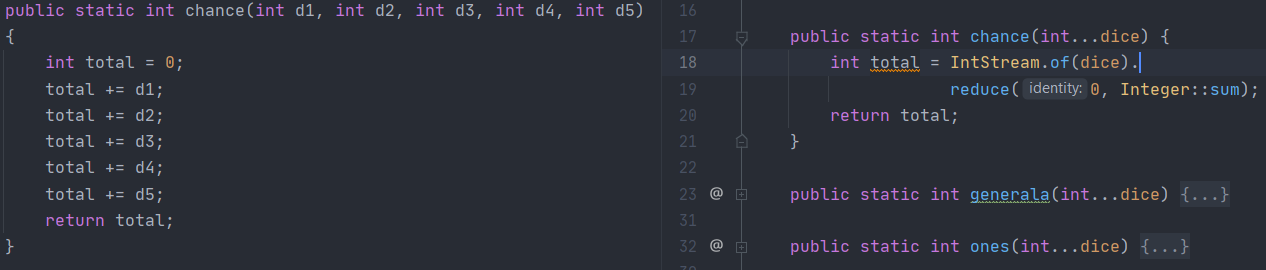
En todos los métodos que tenían parámetros se los cambio para que fuesen dinámicos y así ampliar las opciones de refactoring, por lo que no se vuelve a repetir como un refactorin especifico

# **Constructor**

**Modificador de acceso**: Por el momento nada indica el modificador de acceso “protected” en la variable “dice”, por lo que se cambia a “private”

**Constructor con cuerpo grande:** Las líneas de códigos para la asignación, se pueden ahorrar cambiando los parámetros a dinámicos, y así, solo se asignarían como un arreglo.

# **Chance()**



**Suma repetitiva**: Repetir sumas variable a variable se puede ahorrar con una expresión lambda que sume los términos.

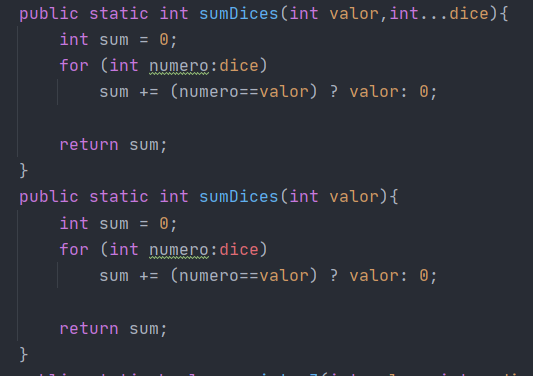
# **Generala()**

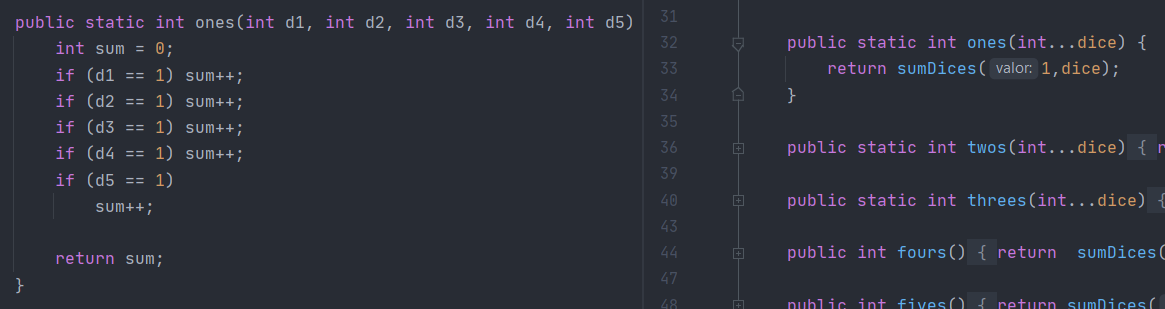
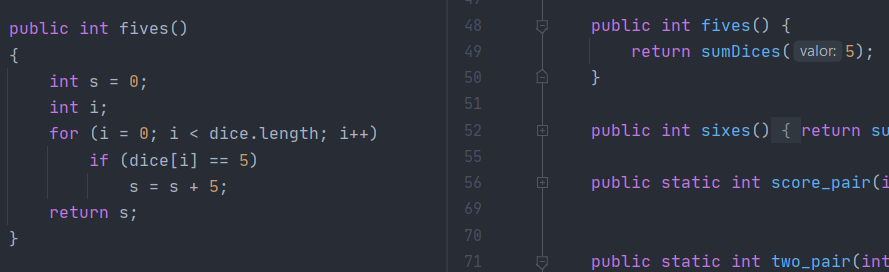
**Complejidad innecesaria:** Al recorrer con un doble For e insertar complejidad que no es necesario provoca una perdida de tiempo al entender el código. Esto se puede mejorar obteniendo un numero cualquiera de “dice”, luego multiplicarlo por el largo del arreglo “dice” y para compararlo con el total, se reutiliza el método chance(), esto debería ser True si es que todos los números son iguales.

# **Ones()Twos()Threes()Fours(), Fives(), Sixes():**

Se agrupan estos métodos, ya que su refactorización es similar en todos. Para esta refactorización se utiliza dos métodos con casi el mismo cuerpo, pero se adapta según su sobrecarga de parámetros, en palabras simples, hacen lo mismo.

Estos métodos son:





**Sumas repetitivas:**