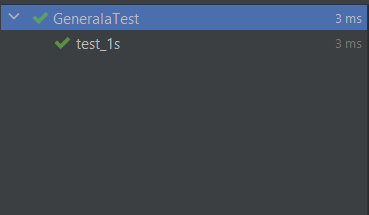
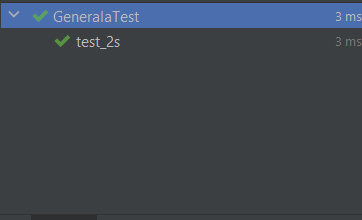
Se corrigieron los métodos “ones”, “twos” y “threes” por la cantidad de condiciones juntas que tienen, se procedió a usar programación funcional para tratar a las variables como valores dinámicos y así poder usar correctamente las funciones lambda, después simplemente se filtró la lista según la condición y se calcularon los resultados con un método sum(), además de que su código esta duplicado en todos los métodos.

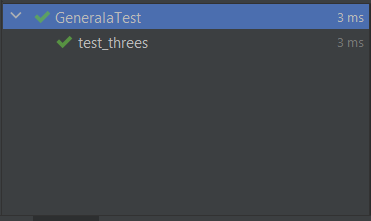
Prueba del método “ones” después de refactorizar



Prueba del método “twos” después de refactorizar

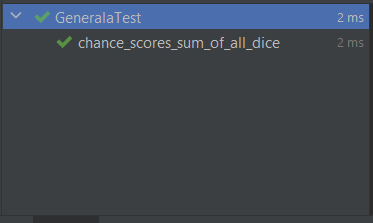


Prueba del método “threes” después de refactorizar



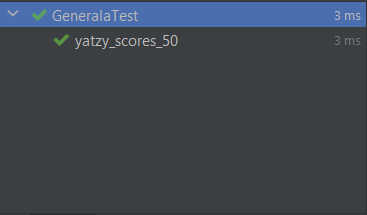
Se corrigió método “chance” por asignaciones repetitivas, con programación funcional se refactorizo el método por completo.

Prueba del método “chance” después de refactorizar



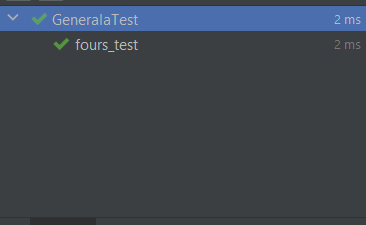
Se corrigió el método “generala”, se usó programación funcional debido que fue más acertado porque nos entregan un “int… dice” entonces se facilita su uso con este tipo de programación, los dos ciclos “for” fueron borrados en su totalidad junto con las condiciones de muy difícil lectura.

Prueba del método “generala” después de refactorizar

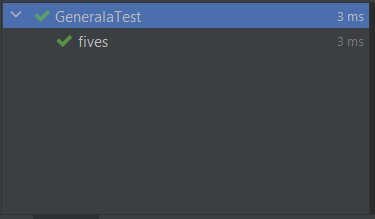


Se corrigieron los métodos “fours”, “fives” y “sixes” poseían ciclo “for” de difícil lectura debido a la costumbre de trabajar con la letra “i” en estos, además de que su código es prácticamente el mismo, se trabaja de la misma manera, pero por alguna razón se cambia el nombre de las variables con las que se trabaja, como si varias personas hubieran hecho un método cada uno.

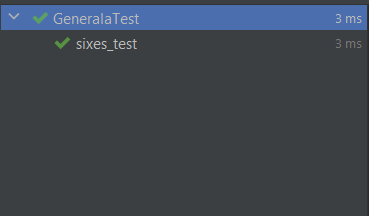
Prueba del método “fours” después de refactorizar



Prueba del método “fives” después de refactorizar

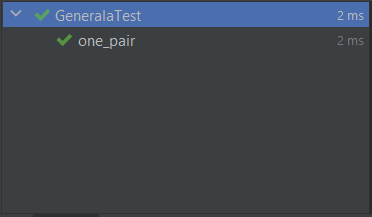


Prueba del método “sixes” después de refactorizar

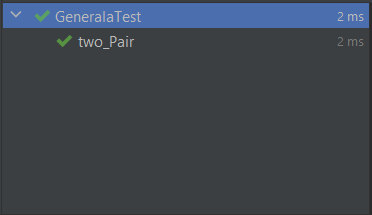


Se corrigieron los métodos “score\_pair” y “two\_pair”, esta vez no se me ocurrió como simplificar mayormente el método con programación funcional, encapsule los contadores en un método llamado “fors” usando un “for” para evitar tantas asignaciones juntas, este método también se usó en “three\_of\_a\_kind” y “four\_of\_a\_kind”.

Prueba del método “score\_pair” después de refactorizar

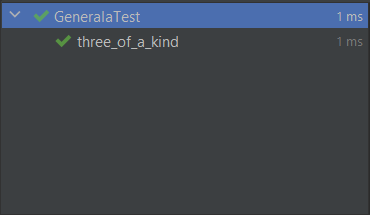


Prueba del método “two\_pair” después de refactorizar

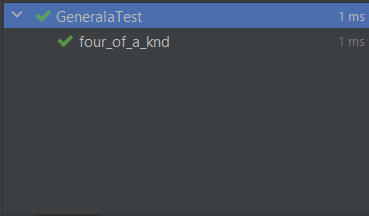


En los métodos “three\_of\_a\_kind” y “four\_of\_a\_kind” había duplicación de código en el ciclo for que usaba el arreglo contador para buscar que numero se repetía tres o cuatro veces, así que cree un método llamado “matches” que simplemente pide un arreglo y el numero de veces que se repite un dado.

Prueba del método “three\_of\_a\_kind” después de refactorizar

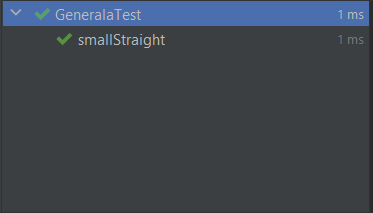


Prueba del método “four\_of\_a\_kind” después de refactorizar

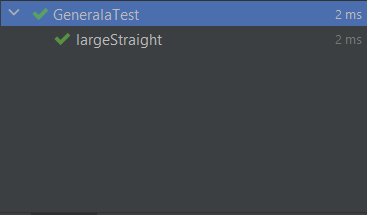


En “smallStraight” se usó programación funcional usando el método “fors” porque este contador estaba repetido otra vez, y se uso un ciclo for para reducir el tamaño de la condición “if” que era muy extensa y repetitiva. En “largeStraight” se copió prácticamente el mismo contenido de “smallStraight” pero en la línea 128 se agregó un +1 para cumplir con su función correctamente según el contexto del juego.

Prueba del método “smallStraight” después de refactorizar



Prueba del método “largeStraight” después de refactorizar



En “fullHouse” se modificaron las variables de confuso nombre, se usó el método “fors” para contar coincidencias, la variable “\_2” se cambió por “isPar” y “\_3” por “isTrio”, a estas se les inicializo con un método llamado “isCondition” que valida si es par o trio según se indique en los parámetros, las variables “\_2\_at” y “\_3\_at” se cambiaron por “contadorPares” y “contadorTrio” y se inicializaron con un método llamado “contadorGeneral” que conto los números que eran par o trio para el correcto cálculo del puntaje.

Prueba del método “fullHouse” después de refactorizar

