Formal Languages and Compiler Design –Lab 2

https://github.com/baraganio/Formal-languages-and-compiler-design

I had to implement my own data structure for the Symbol Table. I have chosen Java language, and a Hash-Table as the data structure to represent my ST.

For the hashFunction I calculate the HashCode of the value I am going to store and divide it by the size of the HashTable. Then I take the reminder of the division and in that position will be the value stored. If two values have the same position, I will repeat the hashFunction for the second value to store, but this time adding one to the HashCode before dividing it by the HashTable size.

The CustomHashTable class represents the data structure itself and has as attributes an String array and the number of elements it has stored. As methods, I have getters, constructor, insert, findPosition (which calculate the HashValue taking into account if there has been collisions or not), find (given the value, returns its position in the HashTable), toString, fHash (make the operations to calculate the HashValue), isPositivePrime (check if a number is prime, in order to use it to calculate the HashTable size) and nextPrimeNumber (it calculate the most effective size of the HashTable starting from a given size).

As an example, I make some tests:

```
@Test
public void test1() {
    CustomHashTable tabla = new CustomHashTable(11);
    System.out.println(tabla.toString());
    assertEquals("-;-;-;-;-;-;-;-;-;-;-;[Size: 11 Num.Elems.: 0]",tabla.toString());
    // Insert null
    assertEquals(-2,tabla.insert(null));
    // Insert elements
    assertEquals(0,tabla.insert("8"));
    assertEquals(0,tabla.insert("10"));
    System.out.println(tabla.toString());
    assertEquals("-;8;-;-;-;10;-;-;-;-;-;[Size: 11 Num.Elems.: 2]",tabla.toString());
    assertEquals(0,tabla.insert("66"));
    System.out.println(tabla.toString());
    assertEquals("-;8;66;-;-;10;-;-;-;-;-;[Size: 11 Num.Elems.: 3]",tabla.toString());
    // Insert elements with collisions
    assertEquals(0,tabla.insert("77"));
    System.out.println(tabla.toString());
    assertEquals("77;8;66;-;-;10;-;-;-;-;-;[Size: 11 Num.Elems.: 4]",tabla.toString());
    assertEquals(0,tabla.insert("88"));
    System.out.println(tabla.toString());
    assertEquals("77;8;66;-;-;10;-;-;-;-;88;[Size: 11 Num.Elems.: 5]",tabla.toString());
    assertEquals(0,tabla.insert("89"));
    System.out.println(tabla.toString());
    assertEquals("77;8;66;89;-;10;-;-;-;-;88;[Size: 11 Num.Elems.: 6]",tabla.toString());
    // Insert elements
    assertEquals(0,tabla.insert("3"));
    assertEquals(0,tabla.insert("6"));
    assertEquals(0,tabla.insert("7"));
    assertEquals(0,tabla.insert("20"));
    assertEquals(0,tabla.insert("16"));
    System.out.println(tabla.toString());
    assertEquals("77;8;66;89;6;10;7;3;20;16;88;[Size: 11 Num.Elems.: 11]",tabla.toString());
```