CT255 Assignment 2

Michael Mc Curtin

ID: 21459584

Source Code

```
import java.util.Arrays;

/*  CT255 Assignment 2
 *  This class provides functionality to build rainbow tables (with a
different reduction function per round) for 8 character long strings, which
    consist of the symbols "a .. z", "A .. Z", "0 .. 9", "!" and "#" (64
symbols in total).
    Properly used, it creates the following value pairs (start value - end
value) after 10,000 iterations of hashFunction() and reductionFunction():
          start value  -  end value
          Kermit12        lsXcRAuN
          Modulus!        L2rEsY8h
          Pigtail1        R0NoLf0w
          GalwayNo        9PZjwF5c
          Trumpets        !oeHRZpK
          HelloPat        dkMPG7!U
          pinky##!        eDx58HRq
          01!19!56        vJ90ePjV
          aaaaaaaa        rLtVvpQS
          036abgH#        klQ6IeQJ


 *
 * @author Michael Schukat, Michael McCurtin
 * @version 1.0
 */
public class RainbowTable {
    /**
     * Constructor, not needed for this assignment
     */
    public RainbowTable() {

    }


    public static void main(String[] args) {
        long res = 0;
        int i;
        String start;
```

```
        // array of all start values for problem 2
        String[] startValues = {
                "Kermit12",
                "Modulus!",
                "Pigtail1",
                "GalwayNo",
                "Trumpets",
                "HelloPat",
                "pinky##!",
                "01!19!56",
                "aaaaaaaa",
                "036abgH#"
        };

        // array of all hash values to find matches for in problem 2
        long[] hashValues = {
                8952106018744312l4L,
                750105908431234638L,
                111111111115664932L,
                977984261343652499L
        };


        if (args != null && args.length > 0) { // Check for <input> value
            start = args[0];

            if (start.length() != 8) {
                System.out.println("Input " + start + " must be 8
characters long - Exit");
            } else {
                // My code for problem 1 and 2 starts here

                long hashVal; // holds the value after hashing

                // iterate through the StartValues array
                for (int j = 0; j < 10; j++) {


                    String startVal = startValues[j]; // startVal: the
initial start value
                    String key = startVal; // key: the value fed into the
hash function

                    System.out.printf("Input: %s ", startVal);
```

```
                    // generate a rainbow table with 10000 chain elements
                    for (int k = 0; k < 10000; k++) {
                        hashVal = hashFunction(key);

                        // check if the hash value is in the hashValues
array, then print which password it corresponds to
                        for (int l = 0; l < 4; l++) {
                            if(hashValues[l] == hashVal) {
                                System.out.printf("\nMatch found for %d
(%s): %s\n", hashVal, reductionFunction(hashVal, k), startVal);
                            }
                        }

                        key = reductionFunction(hashVal, k); // reduce the
hash, then repeat
                    }

                    // print the result
                    System.out.printf("Output: %s\n", key);
                }




            }

        } else { // No <input>
            System.out.println("Use: RainbowTable <Input>");
        }
    }


    private static long hashFunction(String s) {
        long ret = 0;
        int i;
        long[] hashA = new long[]{1, 1, 1, 1};

        String filler, sIn;

        int DIV = 65536;

        filler = new
String("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGH");

        sIn = s + filler; // Add characters, now have "<input>HABCDEF..."
        sIn = sIn.substring(0, 64); // // Limit string to first 64
characters

        for (i = 0; i < sIn.length(); i++) {
            char byPos = sIn.charAt(i); // get i'th character
            hashA[0] += (byPos * 17111); // Note: A += B means A = A + B
            hashA[1] += (hashA[0] + byPos * 31349);
            hashA[2] += (hashA[1] - byPos * 101302);
            hashA[3] += (byPos * 79001);
        }

        ret = (hashA[0] + hashA[2]) + (hashA[1] * hashA[3]);
        if (ret < 0) ret *= -1;
        return ret;
    }
```

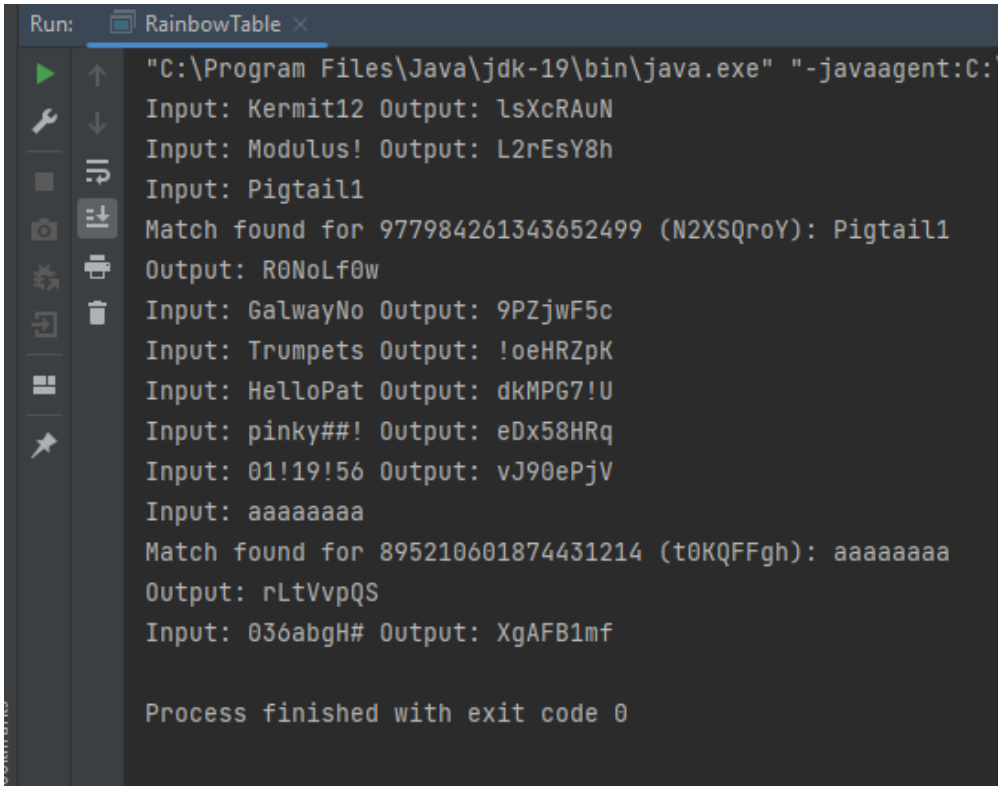
```
    private static String reductionFunction(long val, int round) {  // Note
that for the first function call "round" has to be 0,
        String car, out;                                            // and
has to be incremented by one with every subsequent call.
        int i;                                                      // I.e.
"round" created variations of the reduction function.
        char dat;

        car = new
String("0123456789ABCDEFGHIJKLMNOPQRSTUNVXYZabcdefghijklmnopqrstuvwxyz!#");
        out = new String("");

        for (i = 0; i < 8; i++) {
            val -= round;
            dat = (char) (val % 63);
            val = val / 83;
            out = out + car.charAt(dat);
        }

        return out;
    }
}
```

Program Output



The hash value 895210601874431214 (t0KQFFgh) corresponds to *aaaaaaaa*.

The hash value for 977984261343652499 (N2XSQroY) corresponds to *Pigtail1*.