```
//Oisin Mc Laughlin - 22441106
/* 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
* @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, j;
      String start;
      String temp;
      String temp2;
      /*
      String[] tests = {"Kermit12", "Modulus!", "Pigtail1", "GalwayNo", "Trumpets",
"HelloPat", "pinky##!", "01!19!56", "aaaaaaaa", "036abgH#"};
      long[] hashes = {8952106018744312l4L, 750105908431234638L,
111111111115664932L, 977984261343652499L};
      */
      long hash1 = 895210601874431214L;
      long hash2 = 750105908431234638L;
      long hash3 = 111111111115664932L;
      long hash4 = 977984261343652499L;
```

```
        System.out.println("\n\n\n");
        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 {
                //Setting temp = to the input
                temp = start;

                for (i = 0; i < 10000; i++) {
                    //Calling hashFunction with temp
                    res = hashFunction(temp);
                    //Reducing the hashed value that was found
                    temp2 = reductionFunction(res, i);

                    //System.out.println(temp + " => Hash => " + res + " => Txt => " + temp2);

                    /*
                    for (j = 0; j < 10; j++) {
                        if (hashFunction(tests[j]).equals(Long.toString(hashes[i]))) {
                            System.out.println("Match Found for " + hashes[i] + " => " + start + " at " +
(i+1));
                        }
                    }
                    */
                    if(res == hash1) {
                        System.out.println("Match Found for " + hash1 + " => " + start + " at " + i) ;
                    }
                    if(res == hash2) {
                        System.out.println("Match Found for " + hash1 + " => " + start + " at " + i);
                    }
                    if(res == hash3) {
                        System.out.println("Match Found for " + hash1 + " => " + start + " at " + i);
                    }
                    if(res == hash4) {
                        System.out.println("Match Found for " + hash1 + " => " + start + " at " + i);
                    }

                    //Setting temp = reduced value
                    temp = temp2;
                }
            }
        }
        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("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEF
GH");

    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,
    // and has to be incremented by one with every subsequent call.
    int i; // I.e. "round" created variations of the reduction function.
    char dat;
    String car, out;

    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;
    }
}
```