```
/**
 *  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       IsXcRAuN
 Modulus!       L2rEsY8h
 Pigtail1       R0NoLf0w
 GalwayNo       9PZjwF5c
 Trumpets       !oeHRZpK
 HelloPat       dkMPG7!U
 pinky##!       eDx58HRq
 01!19!56       vJ90ePjV
 aaaaaaaa       rLtVvpQS
 036abgH#       kIQ6IeQJ


 *
 * @author Conor McGovern
 * @version 1.1
 * ID 18343763
 */
public class RainbowTable
{
        /**
        * Constructor, not needed for this assignment
        */
        public RainbowTable() {

        }

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

        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 starts here
        for (i = 0; i<10000; i++) { // Iterates 10000 times
        res = hashFunction(start); // calling hash function
        start = reductionFunction(res, i); // calling un-hash function

        // My code for problem 2 starts here
        if (res == 895210601874431214L || res == 750105908431234638L || res ==
111111111115664932L || res == 977984261343652499L) { // Checking if Hash is equal to
hash in if
        System.out.printf("Match found: %d\n", res);
        }
        }
        System.out.printf("%s", start);
    }
    }
    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("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEF
GHABCDEFGH");

    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;
        }
}
```