```java
import java.lang.reflect.Array;
import java.util.ArrayList;

/**
 *  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 Main {
    /**
     * Constructor, not needed for this assignment
     */

    public static void main(String[] args) {
        long[] Hashes= new long[]{ 8952106018744312l4L,750105908431234638L,
111111111115664932L, 977984261343652499L };// Array of values i need to check for
part 2
        long res = 0;
        int i = 0;
        String start;//this is our string which will be set to whats on the
command line argument
        int loopNum = 0; //loop num is just a loop variable that I use.

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

             start = args[8]; // the start string value to one of our arguments

            if (start.length() != 8) {
                System.out.println("Input " + start + " must be 8 characters long
- Exit");
            } else {
                while (loopNum < 10000) {//loop 10,000 times
                    res = hashFunction(start); // set res to the value our
hashfunction spits out when we pass our start string into it
                    start = reductionFunction(res, i);// then pass the hash value
and the increment variable of into our reduction function to get our reduced
string
```

```
 /*System.out.println("Hash result: " + res); // this commented out code is the
solution to problem one it basically prints out the whole chain and the
corresponding hash and reduction values, the final value corresponds to the ones
outlined above
System.out.println("Reduced String: " + start);*/
                    for (int j = 0; j <= 3; j++) { //loop this piece of code
through four times because the array is only 4 long
                        if (Hashes[j] == res) { //check to see if our hash values
collide
                            System.out.println("hash was: " + Hashes[j]);// if
they do print out the hash
                            System.out.println("matching password: " + start);//
and print out the string of the reduction of the matching hash value this is the
solution to problem 2
                        }
                    }
                    i++;//increment i
                    loopNum++;//increment loopnum

                }

            }

        }

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

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

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

        return out.toString();
    }
}
```

Sample output of one of the predefined arguments in this case it was aaaaaaaa:

```
Reduced String: 9qi1h1u9
Hash result: 952162732205577490
Reduced String: zjfNGnDN
Hash result: 1001707981944374395
Reduced String: mbDeyYeT
Hash result: 1022313814808280141
Reduced String: nvotl82N
Hash result: 1009764646863046334
Reduced String: ZS192Bl9
Hash result: 897291847511734652
Reduced String: q4RakPAL
Hash result: 963085161569203503
Reduced String: GFCDGRQp
Hash result: 931874109344049651
Reduced String: mDRc7K6X
Hash result: 938585269842480082
Reduced String: 6ZiCp2vS
Hash result: 965983327557409560
Reduced String: hLY4H15S
Hash result: 908555543756447622
Reduced String: 0tYVdGxr
Hash result: 1004296232632786062
Reduced String: HoMYvB9q
Hash result: 980941288309508832
Reduced String: q0qN8ZFA
Hash result: 940451146217770040
Reduced String: rLtVvpQS

Process finished with exit code 0
```

Passwords for Problem 2:

```
hash was: 977984261343652499
matching password: N2XSQroY

Process finished with exit code 0
```

```
hash was: 895210601874431214
matching password: t0KQFFgh

Process finished with exit code 0
```

one was aaaaaaaa: