VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

ISA PROJECT

PASSWORD STRENGTH AND SECURITY, ENCRYPTION OF

BY

K. Sai Ruthwik Reddy 21BCE5539

Faculty: Dr. Abirami S

Abstract of the Project:

The Importance of Adding Security to Passwords and Utilizing Encrypted Negative Passwords for Enhanced Security

In today's technologically driven world, where digital interactions and transactions are pervasive, ensuring robust security measures for sensitive data is of utmost importance. Among the most critical aspects of security lies the protection of passwords, which serve as the primary means of authentication for various online accounts and systems. This project submission abstract highlights the significance of implementing strong security practices for passwords and emphasizes the role of encrypted negative passwords as an advanced security measure.

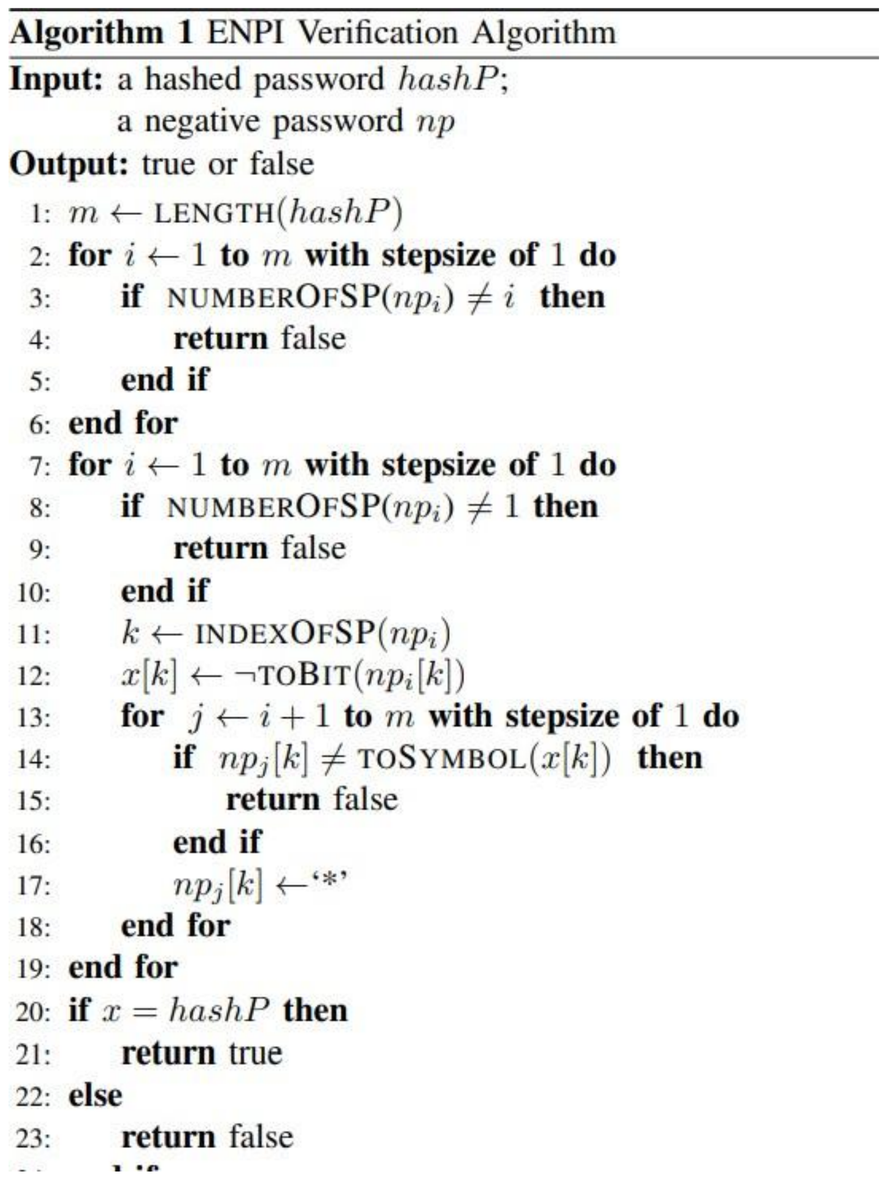
The first section of the project delves into the importance of adding security to passwords. Passwords act as the proverbial keys to access a plethora of personal and confidential information, from email accounts to financial records and social media profiles. By employing strong security measures, such as password complexity rules and multifactor authentication, the vulnerability of passwords to attacks like brute-force, dictionary, or social engineering is significantly reduced. An in-depth analysis of the most common password vulnerabilities and the potential consequences of password breaches will be explored.

The second section of the project focuses on the significance of using encrypted negative passwords as a cutting-edge security solution. Unlike traditional passwords, encrypted negative passwords operate on the concept of obscurity and unpredictability. Instead of containing the actual password, they store encrypted data unrelated to the actual password, thus adding an extra layer of complexity and security. This innovative approach thwarts attempts at password retrieval through various advanced techniques, such as password-guessing attacks or data breaches. By understanding the underlying encryption mechanisms and the benefits it offers, users and organizations can proactively safeguard their valuable information.

In conclusion, this project submission underscores the pivotal role of password security in safeguarding sensitive data and online identities. It advocates for the implementation of robust password security practices, including password complexity and multifactor authentication. Additionally, it highlights the revolutionary potential of encrypted negative passwords, providing an effective deterrent against evolving cyber threats. By embracing these security measures, users and organizations can significantly enhance their digital security posture and mitigate the risks associated with unauthorized access and data breaches.

ALGORITHM USED:

1) For ENP:



Algorithm 1 ENPI Verification Algorithm

Input: a hashed password $hashP$;
a negative password np

Output: true or false

```
$m \leftarrow \text{LENGTH}(hashP)$
for $i \leftarrow 1$ to $m$ with stepsize of $1$ do
    if $\text{NUMBEROFSP}(np_i) \neq i$ then
        return false
    end if
end for
for $i \leftarrow 1$ to $m$ with stepsize of $1$ do
    if $\text{NUMBEROFSP}(np_i) \neq 1$ then
        return false
    end if
    $k \leftarrow \text{INDEXOFSP}(np_i)$
    $x[k] \leftarrow \neg\text{TOBIT}(np_i[k])$
    for $j \leftarrow i + 1$ to $m$ with stepsize of $1$ do
        if $np_j[k] \neq \text{TOSYMBOL}(x[k])$ then
            return false
        end if
        $np_j[k] \leftarrow$ '*'
    end for
end for
if $x = hashP$ then
    return true
else
    return false
```

Steps involved:

The code starts by creating a Scanner object to read input from the user via the console.

The user is prompted to enter a password using System.out.print("Enter the password: "), and the entered password is stored in the password variable using scanner.nextLine().

The startTime variable is initialized using System.currentTimeMillis(). This captures the current time before starting the dictionary attack.

You need to specify the path to the dictionary file by replacing <path_to_dictionary_file> with the actual file path in the dictionaryPath variable.

Inside the try-catch block, a BufferedReader is created to read the dictionary file specified by dictionaryPath. Each line of the file is read and stored in the dictionaryPassword variable.

A boolean variable isPasswordFound is initialized to false. This variable will be used to track whether the password is found in the dictionary.

A while loop is used to iterate through each line of the dictionary file. If a match is found between dictionaryPassword and the user-entered password, isPasswordFound is set to true, and the loop is exited using break.

After reading all lines from the dictionary file, the BufferedReader is closed using br.close().

Based on the value of isPasswordFound, the code displays a corresponding message indicating whether the password was found in the dictionary or not.

The endTime variable is initialized using System.currentTimeMillis(). This captures the current time after completing the dictionary attack.

The elapsed time is calculated by subtracting startTime from endTime, and the result is stored in the elapsedTime variable.

Finally, the elapsed time is displayed on the console using System.out.println("Time taken: " + elapsedTime + " milliseconds").

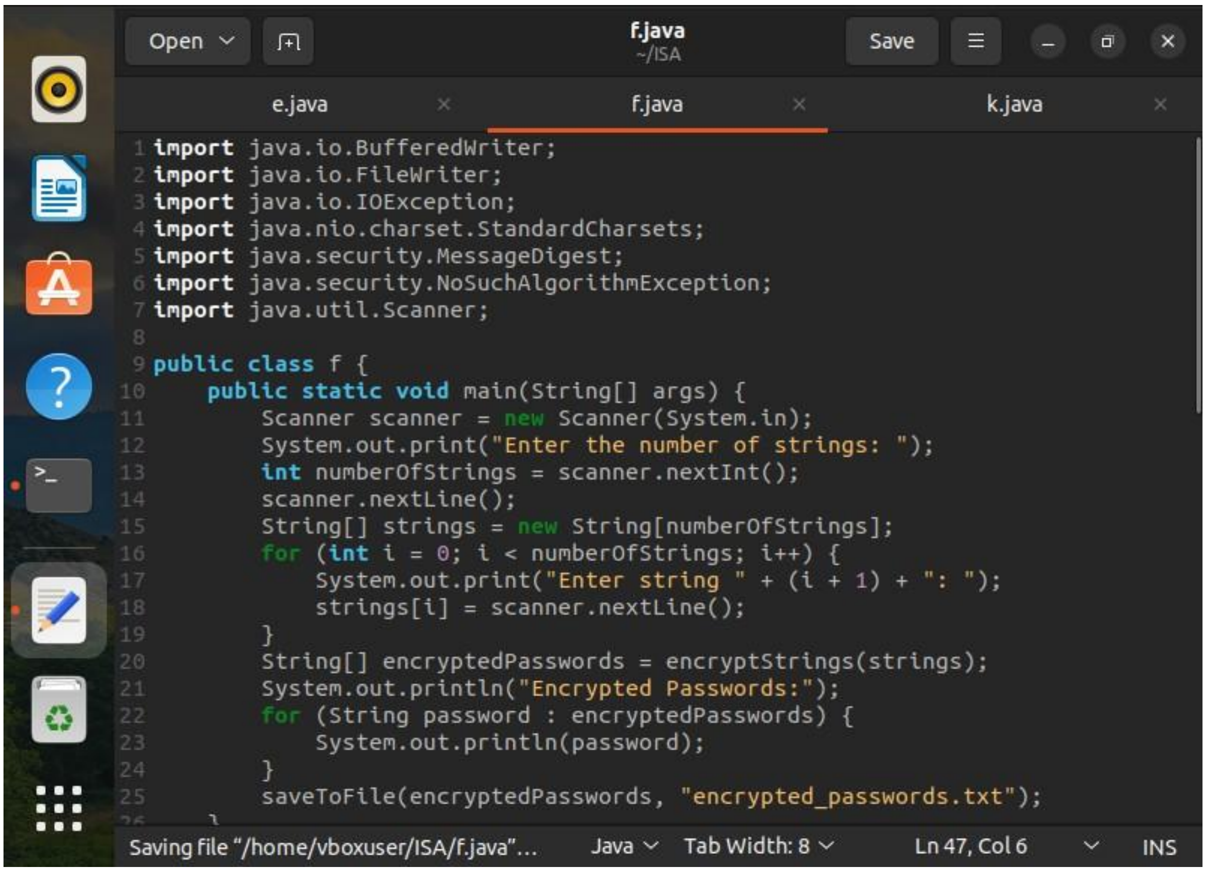
The Scanner object is closed using scanner.close() to release the associated resources.

Remember to replace <path_to_dictionary_file> with the actual file path of your dictionary file.

Conversion of 100 Passwords into Encrypted Negative Passwords:

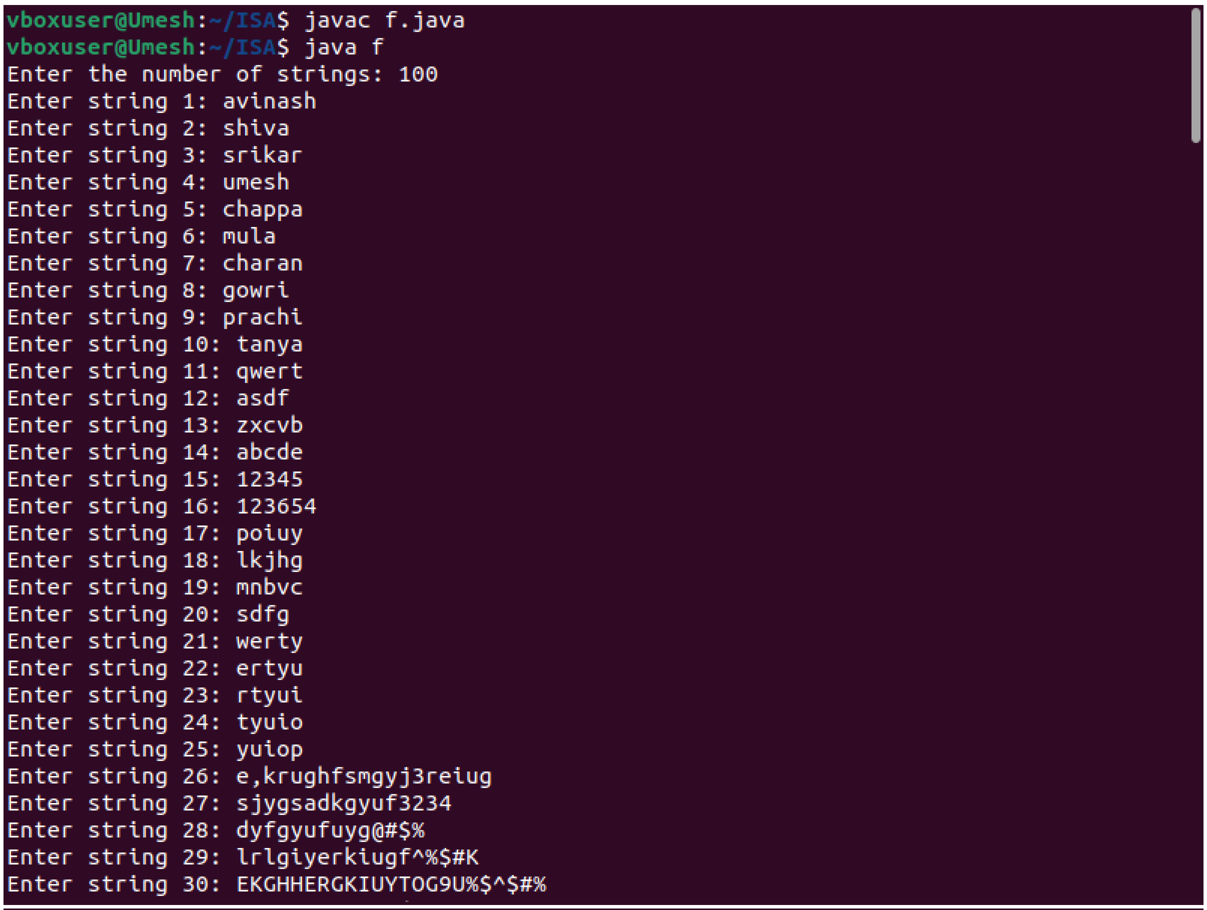
```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

public class f {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of strings: ");
        int numberOfStrings = scanner.nextInt();
        scanner.nextLine();
        String[] strings = new String[numberOfStrings];
        for (int i = 0; i < numberOfStrings; i++) {
            System.out.print("Enter string " + (i + 1) + ": ");
            strings[i] = scanner.nextLine();
        }
        String[] encryptedPasswords = encryptStrings(strings);
        System.out.println("Encrypted Passwords:");
        for (String password : encryptedPasswords) {
            System.out.println(password);
        }
        saveToFile(encryptedPasswords, "encrypted_passwords.txt");
    }
    public static String[] encryptStrings(String[] strings) {
        String[] encryptedPasswords = new String[strings.length];
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            for (int i = 0; i < strings.length; i++) {
                byte[] encodedHash =
digest.digest(strings[i].getBytes(StandardCharsets.UTF_8));
                String encryptedPassword = bytesToHex(encodedHash);
                encryptedPasswords[i] = encryptedPassword;
            }
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return encryptedPasswords;
    }
    public static String bytesToHex(byte[] bytes) {
        StringBuilder sb = new StringBuilder();
        for (byte b : bytes) {
            sb.append(String.format("%02x", b));
        }
        return sb.toString();
    }
    public static void saveToFile(String[] encryptedPasswords, String filename)
{
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(filename))) {
            for (String password : encryptedPasswords) {
                writer.write(password);
                writer.newLine();
            }
            System.out.println("Output saved to " + filename);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

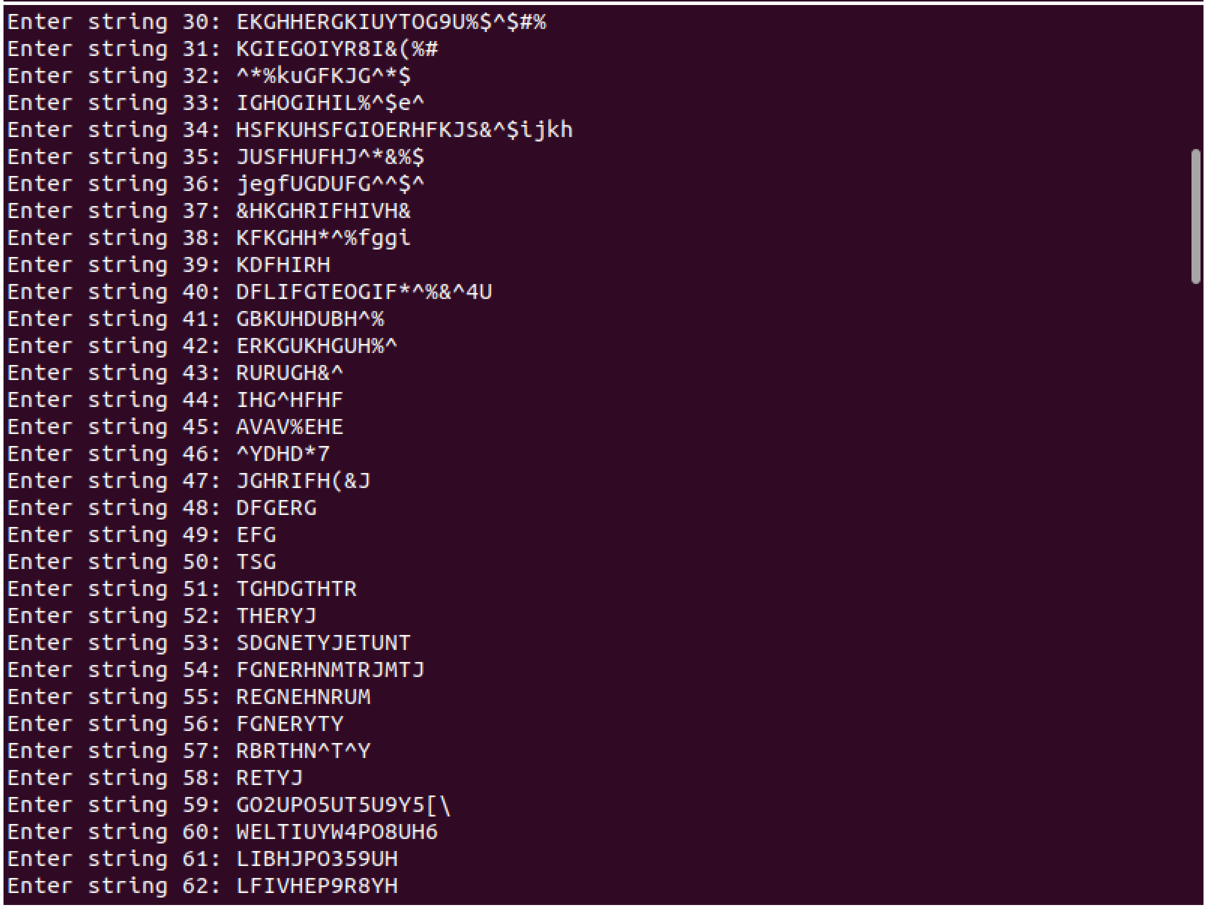


```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

public class f {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of strings: ");
        int numberOfStrings = scanner.nextInt();
        scanner.nextLine();
        String[] strings = new String[numberOfStrings];
        for (int i = 0; i < numberOfStrings; i++) {
            System.out.print("Enter string " + (i + 1) + ": ");
            strings[i] = scanner.nextLine();
        }
        String[] encryptedPasswords = encryptStrings(strings);
        System.out.println("Encrypted Passwords:");
        for (String password : encryptedPasswords) {
            System.out.println(password);
        }
        saveToFile(encryptedPasswords, "encrypted_passwords.txt");
```

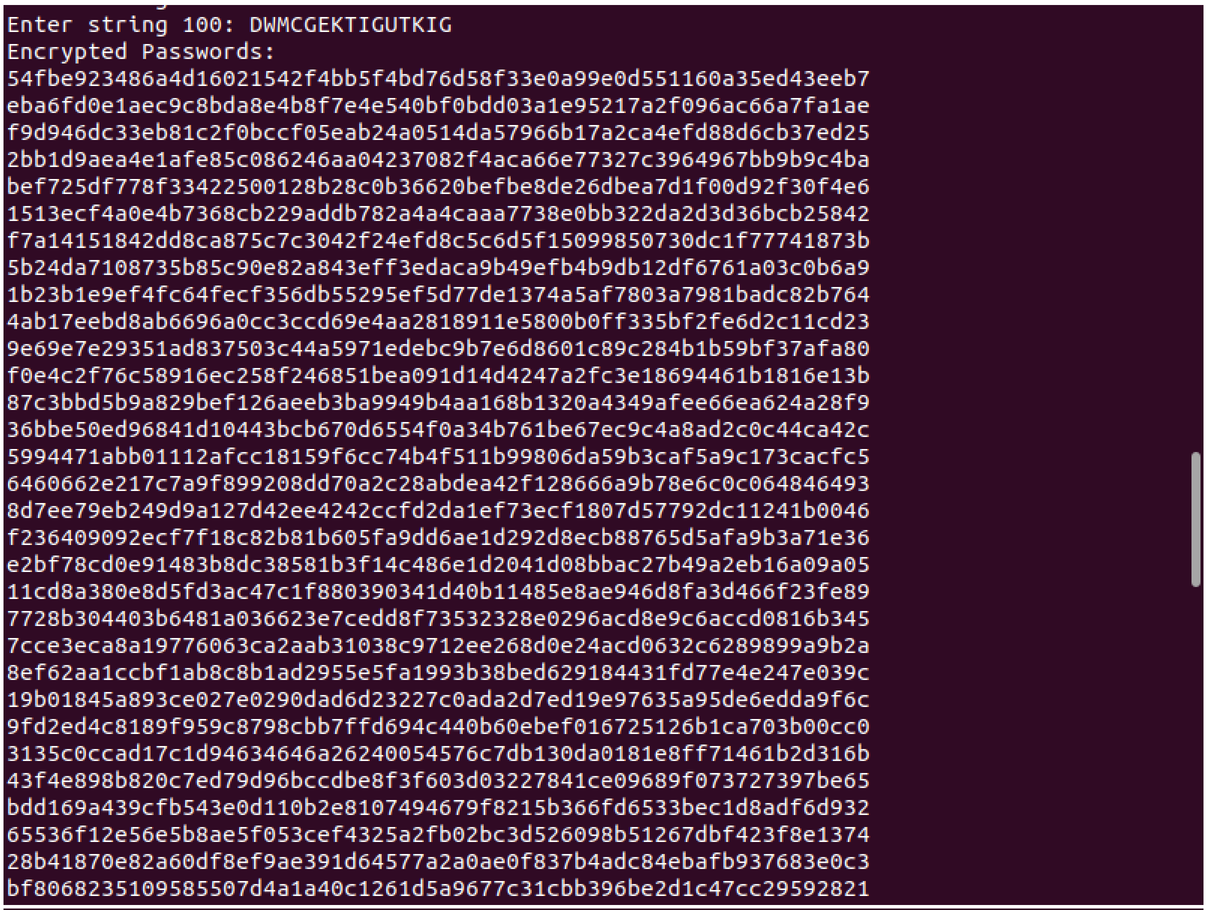
Saving file "/home/vboxuser/ISA/f.java"...   Java   Tab Width: 8   Ln 47, Col 6   INS



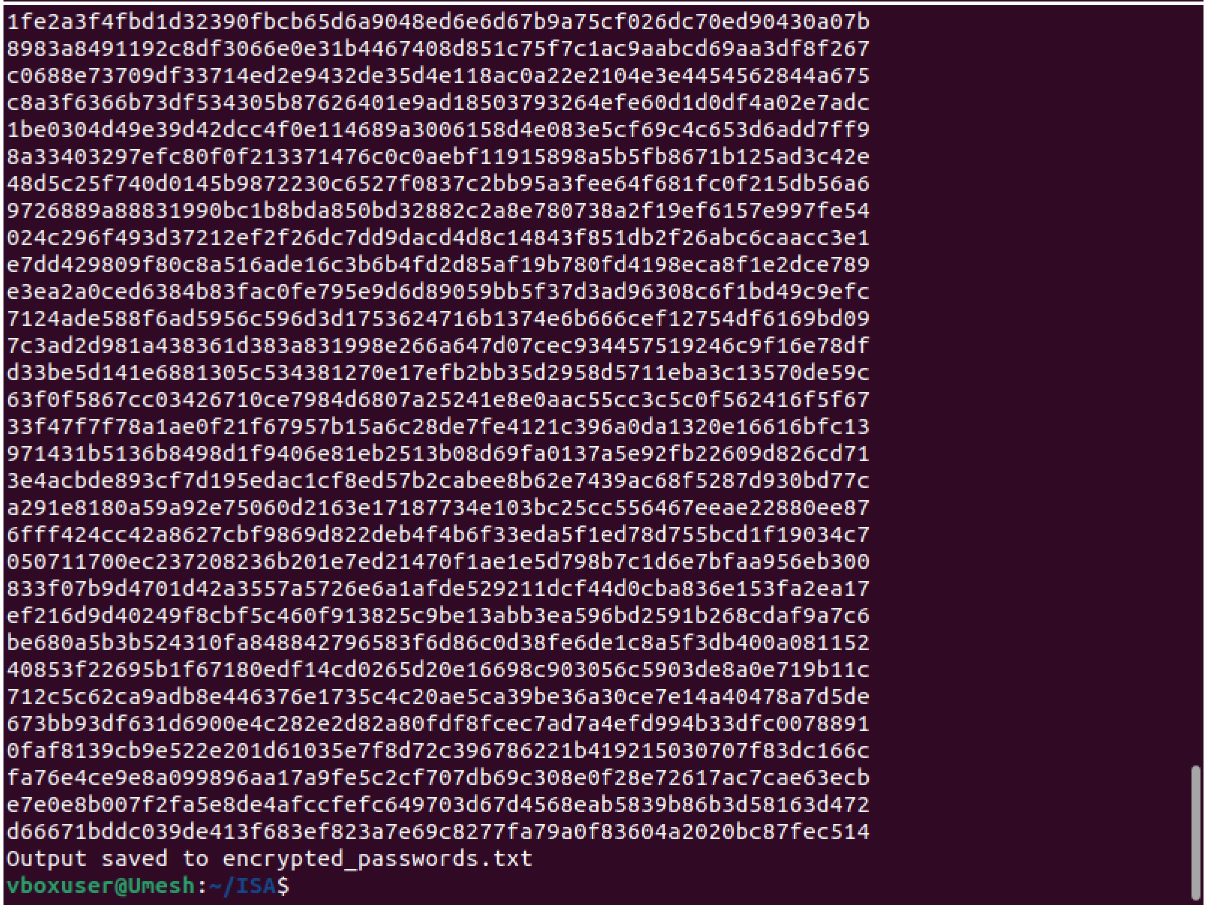
```
vboxuser@Umesh:~/ISA$ javac f.java
vboxuser@Umesh:~/ISA$ java f
Enter the number of strings: 100
Enter string 1: avinash
Enter string 2: shiva
Enter string 3: srikar
Enter string 4: umesh
Enter string 5: chappa
Enter string 6: mula
Enter string 7: charan
Enter string 8: gowri
Enter string 9: prachi
Enter string 10: tanya
Enter string 11: qwert
Enter string 12: asdf
Enter string 13: zxcvb
Enter string 14: abcde
Enter string 15: 12345
Enter string 16: 123654
Enter string 17: poiuy
Enter string 18: lkjhg
Enter string 19: mnbvc
Enter string 20: sdfg
Enter string 21: werty
Enter string 22: ertyu
Enter string 23: rtyui
Enter string 24: tyuio
Enter string 25: yuiop
Enter string 26: e,krughfsmgyj3reiug
Enter string 27: sjygsadkgyuf3234
Enter string 28: dyfgyufuyg@#$%
Enter string 29: lrlgiyerkiugf^%$#K
Enter string 30: EKGHHERGKIUYTOG9U%$^$#%
```



```
Enter string 30: EKGHHERGKIUYTOG9U%$^$#%
Enter string 31: KGIEGOIYR8I&(%#
Enter string 32: ^*%kuGFKJG^*$
Enter string 33: IGHOGIHIL%^$e^
Enter string 34: HSFKUHSFGIOERHFKJS&^$ijkh
Enter string 35: JUSFHUFHJ^*&%$
Enter string 36: jegfUGDUFG^^$^
Enter string 37: &HKGHRIFHIVH&
Enter string 38: KFKGHH*^%fggi
Enter string 39: KDFHIRH
Enter string 40: DFLIFGTEOGIF*^%&^4U
Enter string 41: GBKUHDUBH^%
Enter string 42: ERKGUKHGUH%^
Enter string 43: RURUGH&^
Enter string 44: IHG^HFHF
Enter string 45: AVAV%EHE
Enter string 46: ^YDHD*7
Enter string 47: JGHRIFH(&J
Enter string 48: DFGERG
Enter string 49: EFG
Enter string 50: TSG
Enter string 51: TGHDGTHTR
Enter string 52: THERYJ
Enter string 53: SDGNETYJETUNT
Enter string 54: FGNERHNMTRJMTJ
Enter string 55: REGNEHNRUM
Enter string 56: FGNERYTY
Enter string 57: RBRTHN^T^Y
Enter string 58: RETYJ
Enter string 59: GO2UPO5UT5U9Y5[\
Enter string 60: WELTIUYW4PO8UH6
Enter string 61: LIBHJPO359UH
Enter string 62: LFIVHEP9R8YH
```



```
Enter string 100: DWMCGEKTIGUTKIG
Encrypted Passwords:
54fbe923486a4d16021542f4bb5f4bd76d58f33e0a99e0d551160a35ed43eeb7
eba6fd0e1aec9c8bda8e4b8f7e4e540bf0bdd03a1e95217a2f096ac66a7fa1ae
f9d946dc33eb81c2f0bccf05eab24a0514da57966b17a2ca4efd88d6cb37ed25
2bb1d9aea4e1afe85c086246aa04237082f4aca66e77327c3964967bb9b9c4ba
bef725df778f33422500128b28c0b36620befbe8de26dbea7d1f00d92f30f4e6
1513ecf4a0e4b7368cb229addb782a4a4caaa7738e0bb322da2d3d36bcb25842
f7a14151842dd8ca875c7c3042f24efd8c5c6d5f15099850730dc1f77741873b
5b24da7108735b85c90e82a843eff3edaca9b49efb4b9db12df6761a03c0b6a9
1b23b1e9ef4fc64fecf356db55295ef5d77de1374a5af7803a7981badc82b764
4ab17eebd8ab6696a0cc3ccd69e4aa2818911e5800b0ff335bf2fe6d2c11cd23
9e69e7e29351ad837503c44a5971edebc9b7e6d8601c89c284b1b59bf37afa80
f0e4c2f76c58916ec258f246851bea091d14d4247a2fc3e18694461b1816e13b
87c3bbd5b9a829bef126aeeb3ba9949b4aa168b1320a4349afee66ea624a28f9
36bbe50ed96841d10443bcb670d6554f0a34b761be67ec9c4a8ad2c0c44ca42c
5994471abb01112afcc18159f6cc74b4f511b99806da59b3caf5a9c173cacfc5
6460662e217c7a9f899208dd70a2c28abdea42f128666a9b78e6c0c064846493
8d7ee79eb249d9a127d42ee4242ccfd2da1ef73ecf1807d57792dc11241b0046
f236409092ecf7f18c82b81b605fa9dd6ae1d292d8ecb88765d5afa9b3a71e36
e2bf78cd0e91483b8dc38581b3f14c486e1d2041d08bbac27b49a2eb16a09a05
11cd8a380e8d5fd3ac47c1f880390341d40b11485e8ae946d8fa3d466f23fe89
7728b304403b6481a036623e7cedd8f73532328e0296acd8e9c6accd0816b345
7cce3eca8a19776063ca2aab31038c9712ee268d0e24acd0632c6289899a9b2a
8ef62aa1ccbf1ab8c8b1ad2955e5fa1993b38bed629184431fd77e4e247e039c
19b01845a893ce027e0290dad6d23227c0ada2d7ed19e97635a95de6edda9f6c
9fd2ed4c8189f959c8798cbb7ffd694c440b60ebef016725126b1ca703b00cc0
3135c0ccad17c1d94634646a26240054576c7db130da0181e8ff71461b2d316b
43f4e898b820c7ed79d96bccdbe8f3f603d03227841ce09689f073727397be65
bdd169a439cfb543e0d110b2e8107494679f8215b366fd6533bec1d8adf6d932
65536f12e56e5b8ae5f053cef4325a2fb02bc3d526098b51267dbf423f8e1374
28b41870e82a60df8ef9ae391d64577a2a0ae0f837b4adc84ebafb937683e0c3
bf8068235109585507d4a1a40c1261d5a9677c31cbb396be2d1c47cc29592821
```

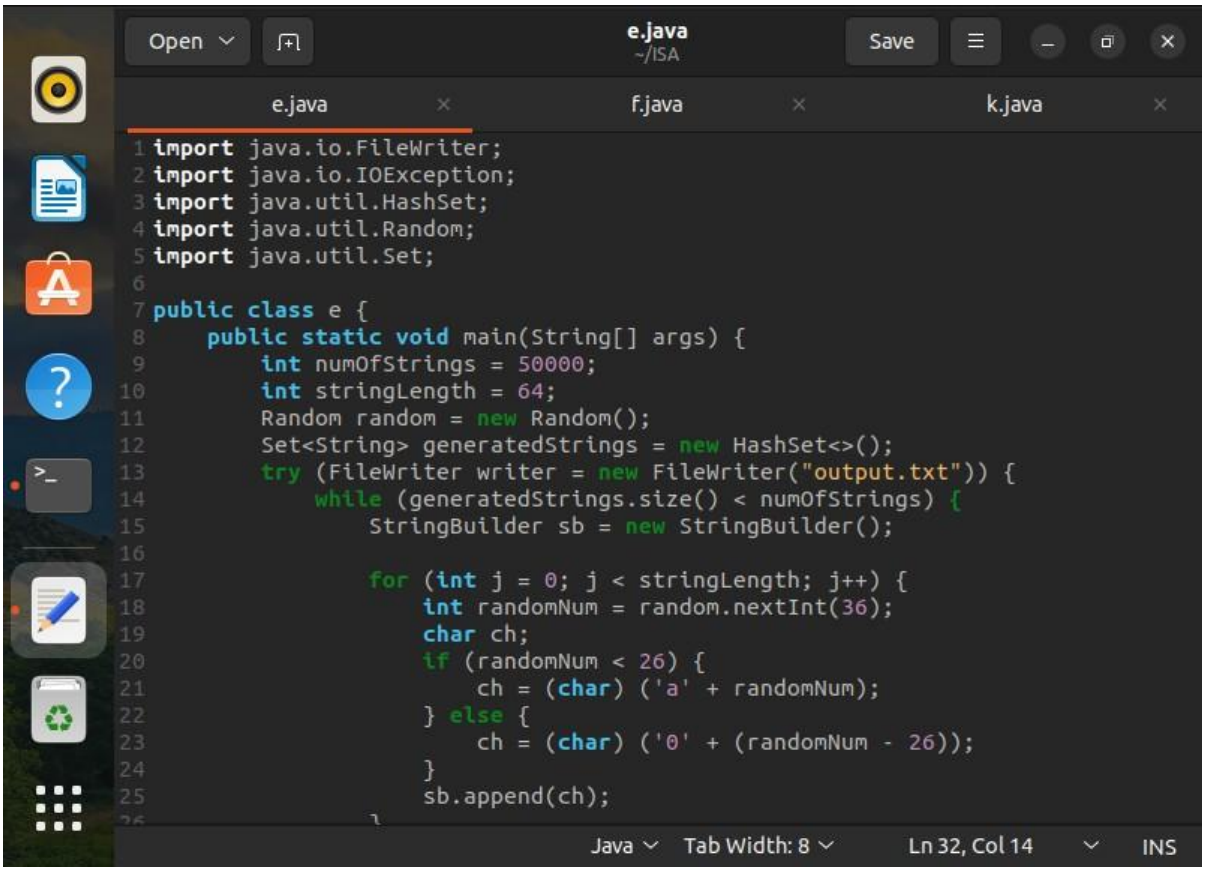


```
1fe2a3f4fbd1d32390fbcb65d6a9048ed6e6d67b9a75cf026dc70ed90430a07b
8983a8491192c8df3066e0e31b4467408d851c75f7c1ac9aabcd69aa3df8f267
c0688e73709df33714ed2e9432de35d4e118ac0a22e2104e3e4454562844a675
c8a3f6366b73df534305b87626401e9ad18503793264efe60d1d0df4a02e7adc
1be0304d49e39d42dcc4f0e114689a3006158d4e083e5cf69c4c653d6add7ff9
8a33403297efc80f0f213371476c0c0aebf11915898a5b5fb8671b125ad3c42e
48d5c25f740d0145b9872230c6527f0837c2bb95a3fee64f681fc0f215db56a6
9726889a88831990bc1b8bda850bd32882c2a8e780738a2f19ef6157e997fe54
024c296f493d37212ef2f26dc7dd9dacd4d8c14843f851db2f26abc6caacc3e1
e7dd429809f80c8a516ade16c3b6b4fd2d85af19b780fd4198eca8f1e2dce789
e3ea2a0ced6384b83fac0fe795e9d6d89059bb5f37d3ad96308c6f1bd49c9efc
7124ade588f6ad5956c596d3d1753624716b1374e6b666cef12754df6169bd09
7c3ad2d981a438361d383a831998e266a647d07cec934457519246c9f16e78df
d33be5d141e6881305c534381270e17efb2bb35d2958d5711eba3c13570de59c
63f0f5867cc03426710ce7984d6807a25241e8e0aac55cc3c5c0f562416f5f67
33f47f7f78a1ae0f21f67957b15a6c28de7fe4121c396a0da1320e16616bfc13
971431b5136b8498d1f9406e81eb2513b08d69fa0137a5e92fb22609d826cd71
3e4acbde893cf7d195edac1cf8ed57b2cabee8b62e7439ac68f5287d930bd77c
a291e8180a59a92e75060d2163e17187734e103bc25cc556467eeae22880ee87
6fff424cc42a8627cbf9869d822deb4f4b6f33eda5f1ed78d755bcd1f19034c7
050711700ec237208236b201e7ed21470f1ae1e5d798b7c1d6e7bfaa956eb300
833f07b9d4701d42a3557a5726e6a1afde529211dcf44d0cba836e153fa2ea17
ef216d9d40249f8cbf5c460f913825c9be13abb3ea596bd2591b268cdaf9a7c6
be680a5b3b524310fa848842796583f6d86c0d38fe6de1c8a5f3db400a081152
40853f22695b1f67180edf14cd0265d20e16698c903056c5903de8a0e719b11c
712c5c62ca9adb8e446376e1735c4c20ae5ca39be36a30ce7e14a40478a7d5de
673bb93df631d6900e4c282e2d82a80fdf8fcec7ad7a4efd994b33dfc0078891
0faf8139cb9e522e201d61035e7f8d72c396786221b419215030707f83dc166c
fa76e4ce9e8a099896aa17a9fe5c2cf707db69c308e0f28e72617ac7cae63ecb
e7e0e8b007f2fa5e8de4afccfefc649703d67d4568eab5839b86b3d58163d472
d66671bddc039de413f683ef823a7e69c8277fa79a0f83604a2020bc87fec514
Output saved to encrypted_passwords.txt
vboxuser@Umesh:~/ISA$
```

Randomly assigns 50000 random passwords of 64 bits length and stores it in “output.txt” file

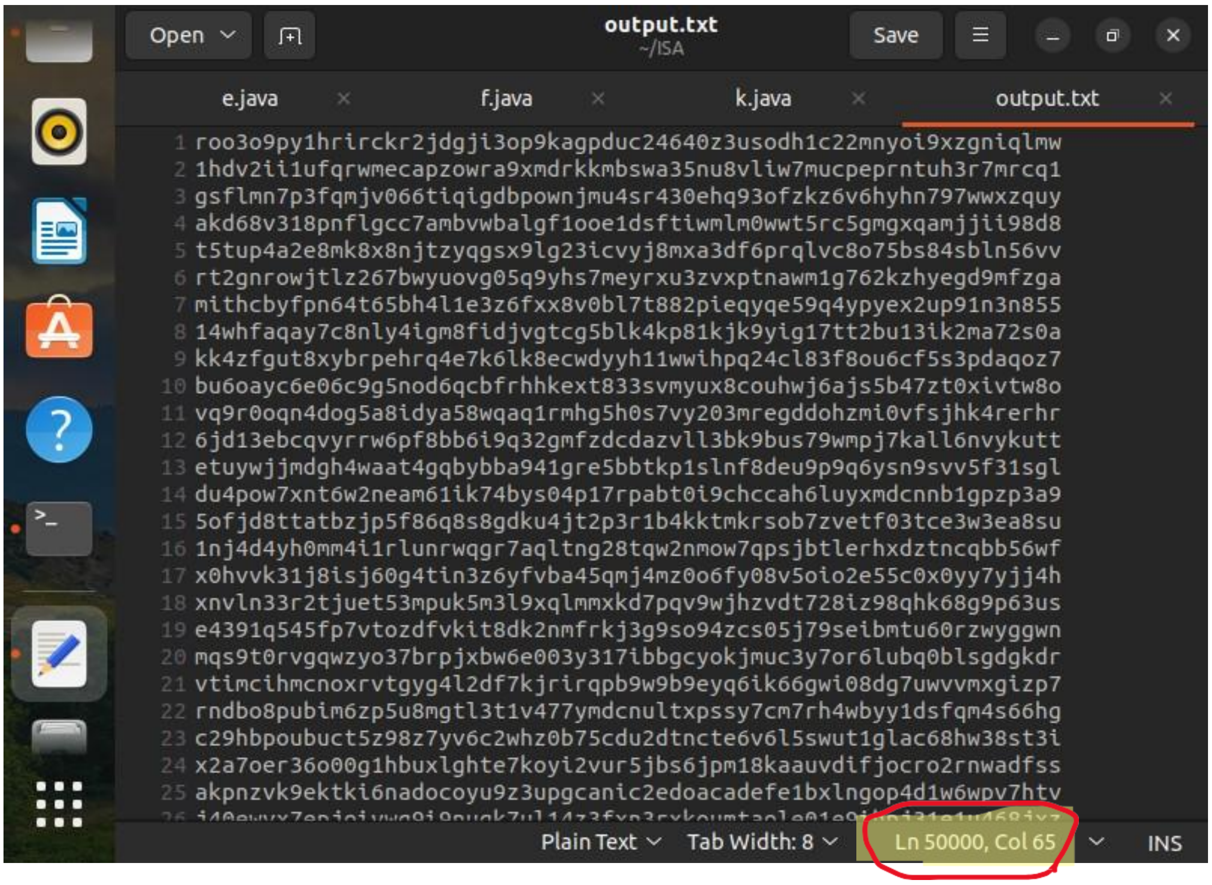
```
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashSet;
import java.util.Random;
import java.util.Set;

public class e {
    public static void main(String[] args) {
        int numOfStrings = 50000;
        int stringLength = 64;
        Random random = new Random();
        Set<String> generatedStrings = new HashSet<>();
        try (FileWriter writer = new FileWriter("output.txt")) {
            while (generatedStrings.size() < numOfStrings) {
                StringBuilder sb = new StringBuilder();

                for (int j = 0; j < stringLength; j++) {
                    int randomNum = random.nextInt(36);
                    char ch;
                    if (randomNum < 26) {
                        ch = (char) ('a' + randomNum);
                    } else {
                        ch = (char) ('0' + (randomNum - 26));
                    }
                    sb.append(ch);
                }
                String generatedString = sb.toString();
                if (!generatedStrings.contains(generatedString)) {
                    generatedStrings.add(generatedString);
                    writer.write(generatedString + System.lineSeparator());
                }
            }
            System.out.println("Strings written to output.txt successfully.");
        } catch (IOException e) {
            System.out.println("An error occurred while writing to the file: " +
e.getMessage());
        }
    }
}
```

Output:

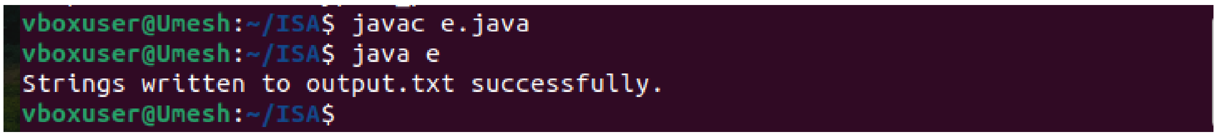


```java
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashSet;
import java.util.Random;
import java.util.Set;

public class e {
    public static void main(String[] args) {
        int numOfStrings = 50000;
        int stringLength = 64;
        Random random = new Random();
        Set<String> generatedStrings = new HashSet<>();
        try (FileWriter writer = new FileWriter("output.txt")) {
            while (generatedStrings.size() < numOfStrings) {
                StringBuilder sb = new StringBuilder();

                for (int j = 0; j < stringLength; j++) {
                    int randomNum = random.nextInt(36);
                    char ch;
                    if (randomNum < 26) {
                        ch = (char) ('a' + randomNum);
                    } else {
                        ch = (char) ('0' + (randomNum - 26));
                    }
                    sb.append(ch);
```

50000 random passwords:



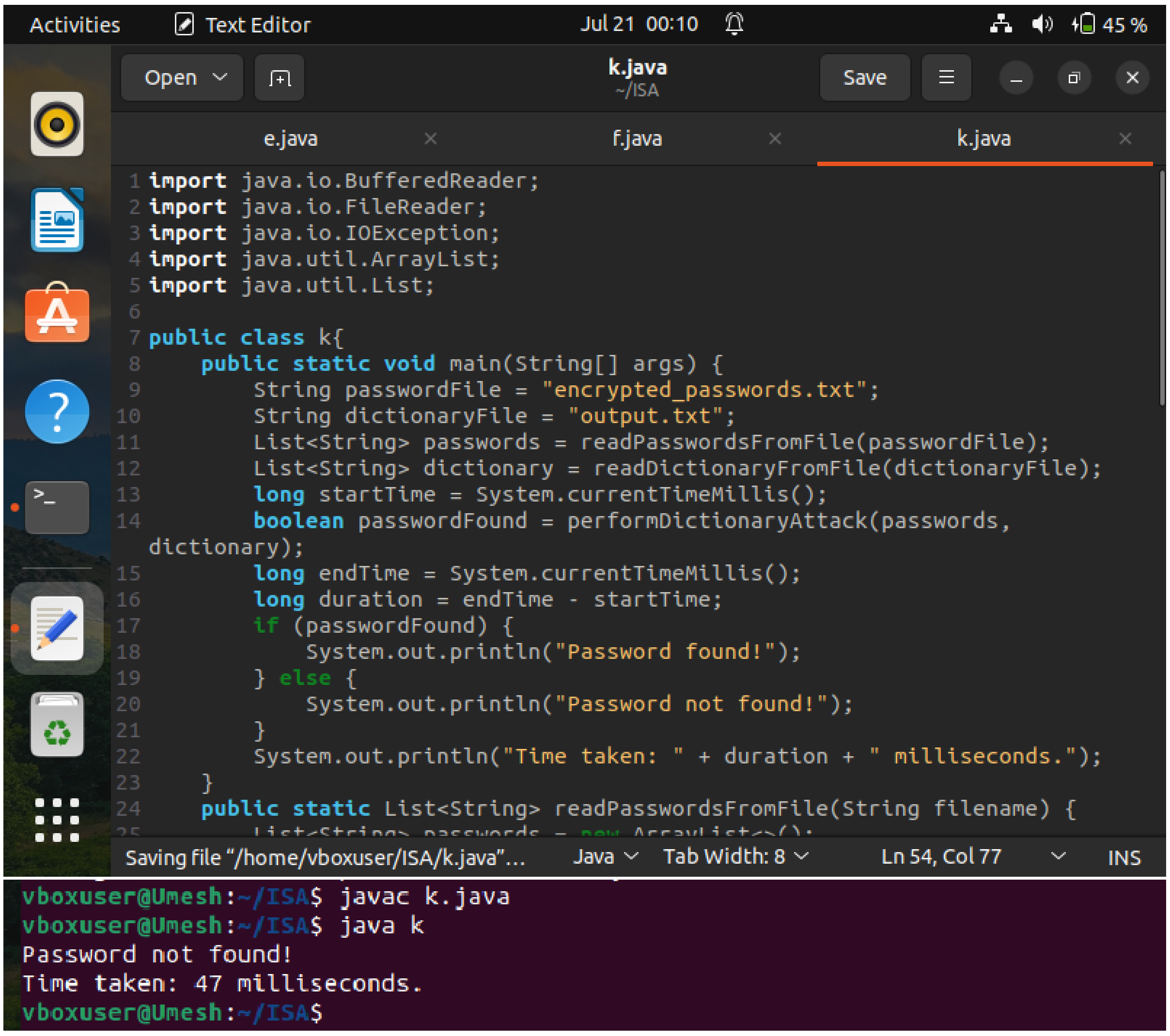
```
roo3o9py1hrirckr2jdgji3op9kagpduc24640z3usodh1c22mnyoi9xzgniqlmw
1hdv2ii1ufqrwmecapzowra9xmdrkkmbswa35nu8vliw7mucpeprntuh3r7mrcq1
gsflmn7p3fqmjv066tiqigdbpownjmu4sr430ehq93ofzkz6v6hyhn797wwxzquy
akd68v318pnflgcc7ambvwbalgf1ooe1dsftiwmlm0wwt5rc5gmgxqamjjii98d8
t5tup4a2e8mk8x8njtzyqgsx9lg23icvyj8mxa3df6prqlvc8o75bs84sbln56vv
rt2gnrowjtlz267bwyuovg05q9yhs7meyrxu3zvxptnawm1g762kzhyegd9mfzga
mithcbyfpn64t65bh4l1e3z6fxx8v0bl7t882pieqyqe59q4ypyex2up91n3n855
14whfaqay7c8nly4igm8fidjvgtcg5blk4kp81kjk9yig17tt2bu13ik2ma72s0a
kk4zfgut8xybrpehrq4e7k6lk8ecwdyyh11wwihpq24cl83f8ou6cf5s3pdaqoz7
bu6oayc6e06c9g5nod6qcbfrhhkext833svmyux8couhwj6ajs5b47zt0xivtw8o
vq9r0oqn4dog5a8idya58wqaq1rmhg5h0s7vy203mregddohzmi0vfsjhk4rerhr
6jd13ebcqvyrrw6pf8bb6i9q32gmfzdcdazvll3bk9bus79wmpj7kall6nvykutt
etuywjjmdgh4waat4gqbybba941gre5bbtkp1slnf8deu9p9q6ysn9svv5f31sgl
du4pow7xnt6w2neam61ik74bys04p17rpabt0i9chccah6luyxmdcnnb1gpzp3a9
5ofjd8ttatbzjp5f86q8s8gdku4jt2p3r1b4kktmkrsob7zvetf03tce3w3ea8su
1nj4d4yh0mm4i1rlunrwqgr7aqltng28tqw2nmow7qpsjbtlerhxdztncqbb56wf
x0hvvk31j8isj60g4tin3z6yfvba45qmj4mz0o6fy08v5oio2e55c0x0yy7yjj4h
xnvln33r2tjuet53mpuk5m3l9xqlmmxkd7pqv9wjhzvdt728iz98qhk68g9p63us
e4391q545fp7vtozdfvkit8dk2nmfrkj3g9so94zcs05j79seibmtu60rzwyggwn
mqs9t0rvgqwzyo37brpjxbw6e003y317ibbgcyokjmuc3y7or6lubq0blsgdgkdr
vtimcihmcnoxrvtgyg4l2df7kjrirqpb9w9b9eyq6ik66gwi08dg7uwvvmxgizp7
rndbo8pubim6zp5u8mgtl3t1v477ymdcnultxpssy7cm7rh4wbyy1dsfqm4s66hg
c29hbpoubuct5z98z7yv6c2whz0b75cdu2dtncte6v6l5swut1glac68hw38st3i
x2a7oer36o00g1hbuxlghte7koyi2vur5jbs6jpm18kaauvdifjocro2rnwadfss
akpnzvk9ektki6nadocoyu9z3upgcanic2edoacadefe1bxlngop4d1w6wpv7htv
```


```
vboxuser@Umesh:~/ISA$ javac e.java
vboxuser@Umesh:~/ISA$ java e
Strings written to output.txt successfully.
vboxuser@Umesh:~/ISA$
```

Comparison of the given 100 inputs to the 50000 random generated strings:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class k{
    public static void main(String[] args) {
        String passwordFile = "encrypted_passwords.txt";
        String dictionaryFile = "output.txt";
        List<String> passwords = readPasswordsFromFile(passwordFile);
        List<String> dictionary = readDictionaryFromFile(dictionaryFile);
        long startTime = System.currentTimeMillis();
        boolean passwordFound = performDictionaryAttack(passwords,
dictionary);
        long endTime = System.currentTimeMillis();
        long duration = endTime - startTime;
        if                (passwordFound)                {
            System.out.println("Password found!");
        } else {
            System.out.println("Password not found!");
        }
        System.out.println("Time taken: " + duration + " milliseconds.");
    }
    public static List<String> readPasswordsFromFile(String filename) {
        List<String> passwords = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new
FileReader(filename))) {
            String line;
            while ((line = reader.readLine()) != null) {
                passwords.add(line.trim());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return passwords;
    }
    public static List<String> readDictionaryFromFile(String filename) {
        List<String> dictionary = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new
FileReader(filename))) {
            String line;
            while ((line = reader.readLine()) != null) {
                dictionary.add(line.trim());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return dictionary;
    }
    public static boolean performDictionaryAttack(List<String> passwords,
List<String> dictionary) {
        boolean   passwordFound   =   false;
        for (String password : passwords) {
            for (String  word  :  dictionary)  {
                if  (password.equals(word))  {
                    long startTime = System.currentTimeMillis();
                    while   (System.currentTimeMillis()   -   startTime   <   10000)   {
                    }
                    passwordFound  = true;
                    break;
                }
            }
            if (passwordFound) {
                break;
            }
        }
        return passwordFound;
    }
}
```

Output:



```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class k{
    public static void main(String[] args) {
        String passwordFile = "encrypted_passwords.txt";
        String dictionaryFile = "output.txt";
        List<String> passwords = readPasswordsFromFile(passwordFile);
        List<String> dictionary = readDictionaryFromFile(dictionaryFile);
        long startTime = System.currentTimeMillis();
        boolean passwordFound = performDictionaryAttack(passwords,
dictionary);
        long endTime = System.currentTimeMillis();
        long duration = endTime - startTime;
        if (passwordFound) {
            System.out.println("Password found!");
        } else {
            System.out.println("Password not found!");
        }
        System.out.println("Time taken: " + duration + " milliseconds.");
    }
    public static List<String> readPasswordsFromFile(String filename) {
```

```
vboxuser@Umesh:~/ISA$ javac k.java
vboxuser@Umesh:~/ISA$ java k
Password not found!
Time taken: 47 milliseconds.
vboxuser@Umesh:~/ISA$
```

Conclusion:

In conclusion, using encrypted negative passwords with the SHA256 encryption algorithm can provide a strong defence against dictionary attacks. SHA256 is a widely used cryptographic hash function known for its strength and resistance to brute-force attacks.

By hashing commonly used and easily guessable passwords with SHA256 and storing them in a blacklist, the system can prevent users from selecting weak passwords that are vulnerable to dictionary attacks. During the login process, the system compares the hash of the user's entered password with the blacklist. If there is a match, the login attempt is denied, regardless of the actual password.

The strength of SHA256 lies in its collision resistance, meaning it is highly unlikely for two different inputs to produce the same hash output. This property makes it difficult for attackers to reverse-engineer the original password from the hash, even with advanced techniques

However, it's important to note that the overall strength of encrypted negative passwords using SHA256 also depends on the quality and comprehensiveness of the blacklist. Regular updates to the blacklist are necessary to account for new passwords that may become commonly used over time.

Additionally, while SHA256 is a strong hash function, it's important to implement additional security measures such as password complexity requirements, salting, and proper storage and handling of hashed passwords to ensure robust password security.

In summary, utilizing encrypted negative passwords with the SHA256 encryption algorithm can significantly enhance protection against dictionary attacks, but it should be complemented with other security measures for comprehensive password security.