Université d'Ottawa
Faculté de génie

École d'ingénierie et de
technologie de l'information

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Information
Technology and Engineering

Assignment 4 – Devoir 4 (9%, 18 points) CSI2110/2510

Due on December 3 at 23:55 (up to 24hs late accepted with 30%off; not accepted after 24hs)

This is the initial handout; further information may be posted in virtual campus;
it is your responsibility to follow further postings.

Assignments are INDIVIDUAL and the code must be your own independent creation; codes will be compared and plagiarism will not be tolerated (see University Policy on Plagiarism)

Cracking passwords using Hash Maps

You are an ethical hacker employed to educate professionals about issues of security. The CEO and top managers of the company you are visiting explain to you that they have top notch security standards. They use a powerful encryption of passwords that protects their user account database, so they are sure their employees and clients information is safe. Today, you will prove them wrong!

In your previous programming assignment, you played with cryptographic hashes in order to encrypt transactions in a block chain. Another application of this technique is the encryption of passwords required for accessing a software system.

In fact, software systems never store your actual password in their database of users. Instead, they store an encrypted version of it, most often using a cryptographic hash function such as SHA-256. When you log in to the system, the password you specify is also encrypted using the same technique and compared to the hash code stored in the database. The main benefit of this approach is that if a hacker gets access to the database, he/she will only see the encrypted passwords and as was explained before, there is no way to revert back the hash code and retrieve the passwords.

However, if you have a list of encrypted passwords, it could be possible to try to guess (or crack) what these passwords could be. This is possible because users tend to use and reuse the same simple passwords across many systems. The hacker will then encrypt a simple password (such as 12345) and compare it to the list of hashed passwords to see if a match is found. This technique can be very effective if the hacker is able to test a large number of candidate passwords. To be more efficient, the hacker will create a large dictionary of commonly used passwords. These passwords are hashed in advance and stored in the database using the password hash code as the key. When a list of encrypted passwords becomes available, each of these encrypted passwords is searched in the dictionary to see if it matches with the code of one of the candidate passwords.

Create your dictionary of encrypted password

Use a hash map data structure in order to build a dictionary of passwords. The mapped values will be the un-encoded password and the key will be the SHA-1 encrypted string corresponding to this password.

To create your list of candidate passwords, you will use the list of the 10,000 most common passwords that can be found in the file `commonPwd10K.txt` provided.
(Disclaimer: we warn you there are inappropriate words in the list but the list was obtained using an internet resource that claims these are the most common passwords)

You will then augment this list by applying the following rules:

1. Capitalize the first letter of each word starting with a letter, e.g. dragon becomes Dragon
2. Add the current year to the word, e.g. dragon becomes dragon2018
3. Use @ instead of a, e.g. dragon becomes dr@gon
4. Use 3 instead of e, e.g. baseball becomes bas3ball
5. Use 1 instead of i, e.g. michael becomes m1chael

You can also combine these rules to generate even more password.

Specific Tasks for your Implementation:

Any Tester program we use (including Tester.java) will interact with your class PasswordCracker and use one of the database classes (DatabaseStandard and DatabaseMine which implement DatabaseInterface) as well as two types of input files. One input file (commonPwd10K.txt) contains common passwords and the other contains a list of user names and their encrypted passwords (leakedAccounts.txt).

Both password databases will be Hash Maps. The items stored in the hash map are pairs of password and its encrypted version using SHA1. The key is obviously the encrypted password, as the search will be done for encrypted password found from leaked account databases. In Part 1, Java will handle the hash table and the hash function for you. In Part 2 you will implement the hash table from scratch.

PART1) Cracking passwords using java.util.HashMap to store the dictionary of passwords (main methods below are in class PasswordCracker):

Create a simple dictionary of common passwords (set of most common 10000 passwords augmented by combinations of the above rules). The dictionary/database will be in class DatabaseStandard that implements our given interface DatabaseInterface using java.util.HashMap. To do this, please use the following method from class PasswordCracker:

`void createDatabase(ArrayList<String> commonPasswords, DatabaseInterface database)`

This class receives the original passwords in the array list, and is responsible for creating more passwords with augmented rules and insert everything on the given database, which is originally empty.

Now, use this dictionary to crack a list of encrypted passwords given in a file such as the sample leakedAccounts.txt, where we give you a list of 25 encrypted passwords. The passwords in this file were generated using some of the 10000-list augmented by using a combination of the 5 above rules. Your mission is to crack as many of these as you can (we may test other files as well). The tester program already has code that will read through a file in the same format as leakedAccounts.txt and make multiple calls to the following method in class PasswordCracker that cracks individual passwords using the database of passwords:

String crackPassword(String encryptedPassword, DatabaseInterface database)
You need to implement this method. This task is accomplished by looking up the encryptedPassword in the given database and then retrieve its corresponding original password and return.

PART2) Cracking passwords using your own Hash Table to store passwords.
Perform the same task again, but using an implementation of your own Hash map implemented in class DatabaseMine. Indeed, after DatabaseMine is implemented, repeating the tasks above requires no further effort. Everything above is based on the interface DatabaseInterface, which is implemented by both DatabaseStandard and DatabaseMine. You can see how Part1 and Part2 are performed via appropriate method calls by looking at our tester program given in Tester.java So, to complete part 2 you only need to implement DatabaseMine. Your hash table will use linear probing for collision resolution and the hash function will be key.hashCode() mod N, for N a prime number. There are more details in the instruction section.

Coding and Submission Instructions:

1) Inspect the java classes and the input files provided to you.
2) Go implementing your tasks, incrementally, and testing. You can do part1 first considering the given passwords; later add the part that creates the modified passwords to augment the list. Once this is done, you can move to part 2 where you concentrate on implementing your own hash table. Alternatively, if you are finding a hard time debugging the part that adds modified passwords (BTW study Java String class to help with word manipulation), you may do part 2, also based on the original passwords only. This will allow you to crack a portion of the account passwords, and develop your code incrementally.
3) How each part and function relates to the various classes you need to implement?

CLASSES GIVEN TO YOU THAT MUST NOT BE CHANGED:

Tester.java: our testing class that should not be modified; it assumes your classes will have certain methods, so you must make your other classes to work properly with it.

Sha1.java: this class is the same as assignment#2, with the only change being that the hexa output is now given in upper case (change 1a to 1A)
DatabaseInterface.java: this is an interface that helps us write generic code that can operate with either one of the Hash Map implementations for the password database, namely classes DatabaseStandard or DatabaseMine; these two classes must **implement the interface**.

```
public interface DatabaseInterface {
     public String save(String plainPassword, String encryptedPassword);
     public String decrypt(String encryptedPassword);
     public int size();
     public void printStatistics();
}
```

CLASSES THAT YOU WILL CREATE:

We provide some indications of things that must be present in the class. You are totally free to create any other auxiliary methods, variable members, inner classes, as you need.

For Part1:
DatabaseStandard.java: implements **DatabaseInterface** by having inside an object that is a java.util.HashMap, and making use of its methods to implement the methods required by the interface (see methods above). This class starts like this:

```
public class DatabaseStandard implements DatabaseInterface {
…
public DatabaseStandard() {…}
       // this constructor must create the initial hash map
…
}
```

Method printStatistics() must provide statistics as in the output example below.

PasswordCracker.java: this is the class that will populate the database with passwords from a given list of common passwords and that will decode a given encrypted password by looking it up on the database to then discover its corresponding original password.

The following methods must be present:

```
public class PasswordCracker {
…
void createDatabase(ArrayList<String> commonPasswords, DatabaseInterface
database)  {
// receives list of passwords and populates database with entries consisting
// of (key,value) pairs where the value is the password and the key is the
// encrypted password (encrypted using Sha1)
// in addition to passwords in commonPasswords list, this class is
```

```
// responsible to add mutated passwords according to rules 1-5.

String crackPassword(String encryptedPassword, DatabaseInterface database) {
//uses database to crack encrypted password, returning the original password
```

An example of usage of both methods in Part 1 is given next, so that you test these classes before integrating them to interact with our Tester.java:

```
  PasswordCracker testCracker=new PasswordCracker();
  DatabaseStandard database1=new DatabaseStandard();
  ArrayList<String> commonPass=new ArrayList<String>();
  commonPass.add("123456");
  commonPass.add("password");
  commonPass.add("12345678");
  commonPass.add("brady");
  testCracker.createDatabase(commonPass,database1);
  database1.printStatistics();
  String code=new String("F35D55B3ACF667911A679B44918F5D88B2400823");
  String discoverPassword=testCracker.crackPassword(code,database1);
  System.out.println("Decrypt: "+code+ " Password: "+discoverPassword+";");

  Our output for the program above is:
  *** DatabaseStandard Statistics ***
  Size is 20 passwords
  Initial Number of Indexes when Created 37
  *** End DatabaseStandard Statistics ***
  Decrypt: F35D55B3ACF667911A679B44918F5D88B2400823 Password: brady;
```

Part 2:

DatabaseMine.java: implements **DatabaseInterface** by creating its own Hash Table inside this class, and using it to implement the methods required by the interface.

For part 2 your Hash Table will be a simple implementation of a hash table using linear probing. You will have a table of entries (key,value), both Strings, where the key is the encrypted password and the value is the original password.

The hash function that determines the home address of a key will use the method hashCode(), already provided in the class String, apply modulo N, where N is a prime number. Note that N is the table size, so it must be large enough for the amount of keys you wish to insert, also taking into account the loading factor.

A list of the first 1 million prime numbers can be found here, so you can experiment choose one that works well for our quantity of data: http://www.mathematical.com/primes0to1000k.html

```
public class DatabaseMine implements DatabaseInterface {
  int N; // this is a prime number that gives the number of addresses
// these constructors must create your hash tables with enough positions N
// to hold the entries you will insert; you may experiment with primes N
```

```
  public DatabaseMine() {…}      // here you pick suitable default N
  public DatabaseMine(int N) {…} // here the N is given by the user
…
  int hashFunction(String key) {
          int address=key.hashCode()%N;
          return (address>=0)?address:(address+N);
  }
…
public void printStatistics() { … }
// important statistics must be collected (here or during construction) and
printed here: size, number of indexes, load factor, average number of probes
and average number of displacements (see output example below)
}
```

Note that the first line in hashFunction method should suffice, if key.hashCode() was always a positive number or if the operation mod (%) would always return a positive number. However, in our version of Java we observed that both proved to be false and for negative numbers the mod operator gave negative numbers greater than –N. The last line corrects this.

You can use a similar tester code as in page 5, changing the second line to:
DatabaseMine database1=new DatabaseMine(37); // this uses constructor that specifies size
Our output for this code, which also displays the result of `printStatistics()` is:

```
*** DatabaseMine Statistics ***
Size is 20 passwords
Number of Indexes is 37
Load Factor is 0.5405405
Average Number of Probes is 1.5
Number of displacements (due to collisions) 4
*** End DatabaseMine Statistics ***
Decrypt: F35D55B3ACF667911A679B44918F5D88B2400823 Password: brady;
```

Note that the number of passwords inserted depend on how the combination of rules 1-5 is implemented.You may have different numbers as us.

WHAT TO SUBMIT: <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

DatabaseInterface.java (as given)
Tester.java (as given)
Sha1.java (as given)
DatabaseStandard.java (part 1)
PasswordCracker.java (the same for part1 and 2)
DatabaseMine.java (part 2)

(MARKING SCHEME TO BE DECIDED)
HAVE FUN, AND CHECK FURTHER INFORMATION/UPDATES IN BRIGHTSPACE AREA FOR ASSIGNMENT#4.