CS3780 Project 2

Mark Joyce and Kenan Krijestorac

Environment:

Our chosen environment for this project was Java, this is the language we are most familiar with and it has access to good cryptographic libraries without any special set-up.

How to use:

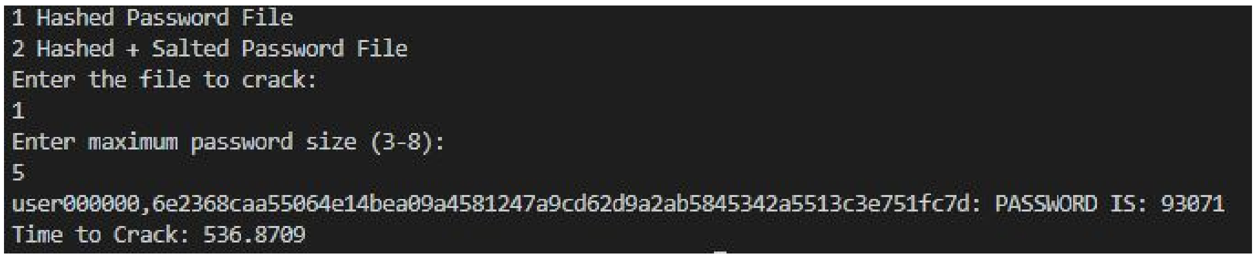
For tasks 1 and 2, run the main method of LoginOrCreate.java and follow the prompts. This will allow you to create an account, authenticate, and generate multiple users for cracking. This class will generate multiple different files (which will be located in the current directory the code runs in ie) ./file.txt) for storing the account information.

For task 3, run the main method of Crack.java and follow the prompts.

For more technical information please refer to the comments within the code.

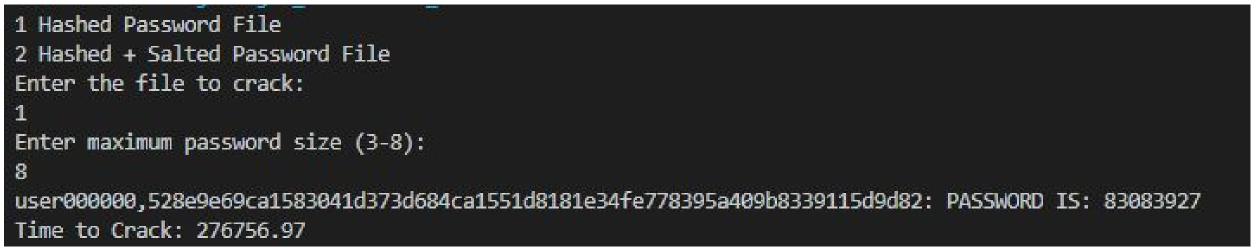
Testing:

For cracking a hashed file with 5 character passwords we found times of around several hundred milliseconds:



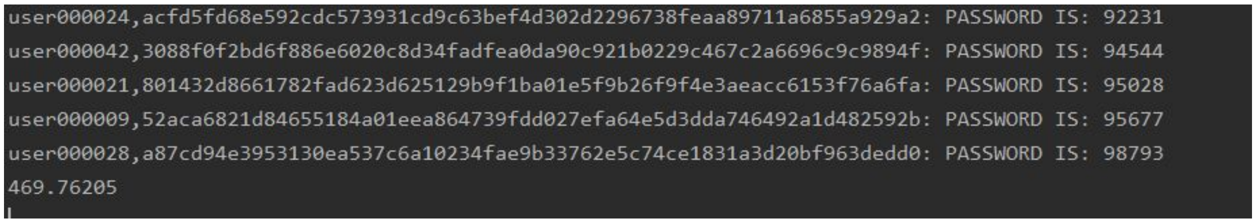
```
1 Hashed Password File
2 Hashed + Salted Password File
Enter the file to crack:
1
Enter maximum password size (3-8):
5
user000000,6e2368caa55064e14bea09a4581247a9cd62d9a2ab5845342a5513c3e751fc7d: PASSWORD IS: 93071
Time to Crack: 536.8709
```

Time to crack hashed file with 8 character password we saw ranges from 30 seconds to several minutes, below is 276 seconds:



```
1 Hashed Password File
2 Hashed + Salted Password File
Enter the file to crack:
1
Enter maximum password size (3-8):
8
user000000,528e9e69ca1583041d373d684ca1551d8181e34fe778395a409b8339115d9d82: PASSWORD IS: 83083927
Time to Crack: 276756.97
```

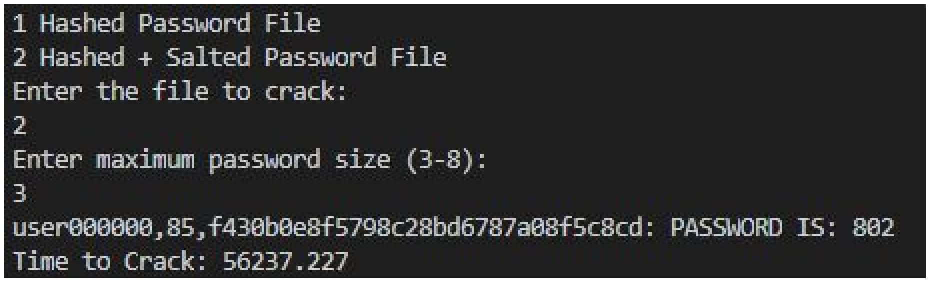
A batch of 50 3 character hash passwords and 50 5 characters hash passwords took only about 500 milliseconds to crack:



```
user000024,acfd5fd68e592cdc573931cd9c63bef4d302d2296738feaa89711a6855a929a2: PASSWORD IS: 92231
user000042,3088f0f2bd6f886e6020c8d34fadfea0da90c921b0229c467c2a6696c9c9894f: PASSWORD IS: 94544
user000021,801432d8661782fad623d625129b9f1ba01e5f9b26f9f4e3aeacc6153f76a6fa: PASSWORD IS: 95028
user000009,52aca6821d84655184a01eea864739fdd027efa64e5d3dda746492a1d482592b: PASSWORD IS: 95677
user000028,a87cd94e3953130ea537c6a10234fae9b33762e5c74ce1831a3d20bf963dedd0: PASSWORD IS: 98793
469.76205
```

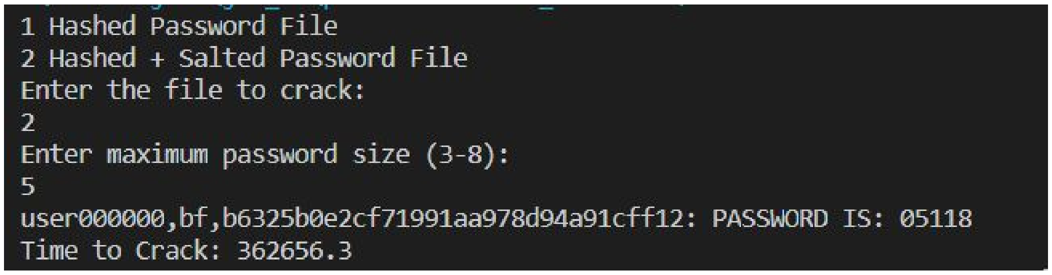
For cracking a password with Salt + Hash we found much longer times:

3 character passwords with salt generally took around 30 seconds to a minute:



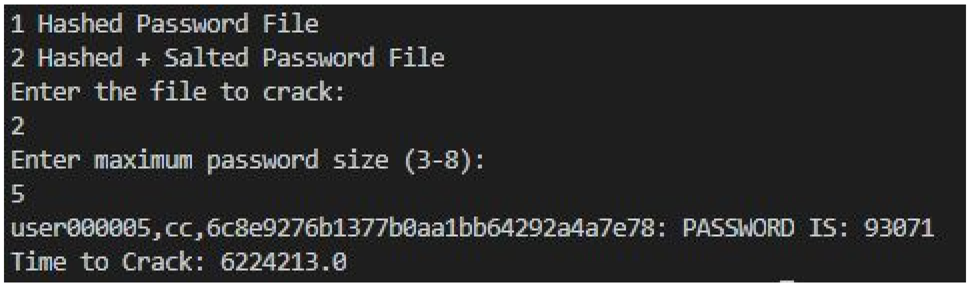
```
1 Hashed Password File
2 Hashed + Salted Password File
Enter the file to crack:
2
Enter maximum password size (3-8):
3
user000000,85,f430b0e8f5798c28bd6787a08f5c8cd: PASSWORD IS: 802
Time to Crack: 56237.227
```

Lower 5 character password with salt took about 6 minutes whereas a higher one took much longer:



```
1 Hashed Password File
2 Hashed + Salted Password File
Enter the file to crack:
2
Enter maximum password size (3-8):
5
user000000,bf,b6325b0e2cf71991aa978d94a91cff12: PASSWORD IS: 05118
Time to Crack: 362656.3
```

Higher 5 character password with salt took about 104 minutes to crack:



```
1 Hashed Password File
2 Hashed + Salted Password File
Enter the file to crack:
2
Enter maximum password size (3-8):
5
user000005,cc,6c8e9276b1377b0aa1bb64292a4a7e78: PASSWORD IS: 93071
Time to Crack: 6224213.0
```

Analysis:

Based on what we have seen when cracking passwords, it would appear that storing passwords with Salt+Hash is much harder to crack than plain text or hash. Most of the time an 8 character numeric only hashed password took less than 5 minutes, while an 8 character Salt+Hash password took so long we were unable to get a time. The implementation of your password cracker will also have a significant impact on performance. Other implementations could use different data structures to store hashed passwords and salted hashed passwords making the comparisons between generated hashes and stored hashes slightly faster, however, the length of time to crack a password would still be relatively similar. Each additional digit added to a password increases the time to crack significantly.

Since we know the format of the file and that the salts are stored within the file, this eases the cracking process. Without knowledge of the size of the salt or the salt in general, the time to crack even a three numerical character password would take an extremely long time.

It is also worth noting that due to the way the password crack works, higher numbers take longer to crack than lower numbers, but this may not be relevant for more complex passwords containing symbols and letters. The usage of symbols, uppercase and lowercase alphabetic characters, and numerical characters with the addition of a salt increases the difficulty of cracking a password, almost making it not even worth the effort to crack.

With what we have observed cracking simple numeric only passwords, it would seem that a password stored in hash+salt form that also contained letters (or even special characters for more security) of a length of at least 8 would take an extremely long time to crack, and thus be secure. For our specific system, a numeric password of a length 8 should be sufficiently safe, though all passwords can *eventually* be cracked.

Source Code:

LoginOrCreate.java - Contains Task 1 and Task 2

```java
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import java.io.*;
import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.KeySpec;
import java.util.concurrent.ThreadLocalRandom;
import java.text.DecimalFormat;
import java.util.*;


public class LoginOrCreate { //Main class, prompt user to Create Account or
Login/Authenticate

   static final int maxLengthAllowed = 10; //max password length

   //File paths
   static final String plainTextFilePath = "./plainTextFile.txt";
   static final String hashFilePath = "./hashFile.txt";
   static final String saltFilePath = "./saltFile.txt";

   public static void main(String[] args) {
       System.out.println("----- | CS3780 Project 2 | -----");
       System.out.println("Would you like to Create an Account or Authenticate?");

       int choice = readChoice(); //Get user input

       int numAccounts = 0;
       int passLength = 0;

       //Determine what user wants to do
       if (choice == 1) createAccount();
       else if (choice == 2) authenticate();
       else if (choice == 3) {
           numAccounts = accNum();
           passLength = pass();
           generateUserPassAcc(numAccounts, passLength);
       }
       else System.out.println("Something went wrong!");

   }

   /**
    * Gets the choice from the user with input validation.
    * @return Returns the users input validated choice.
    **/
   private static int readChoice(){ //Get and validate user choice
       int choice=0; //init choice
       Scanner sc = new Scanner(System.in); //scanner for input

       do {

           try {
               System.out.println("Valid choices are:");
               System.out.println("1 Create Account");
               System.out.println("2 Authenticate");
               System.out.println("3 Create Multiple Accounts");
               System.out.println("Please enter your choice:");

               while (!sc.hasNextInt()) {
                   System.out.println("That's not a valid choice, try again:");
                   System.out.println("1 Create Account");
```

```
                   System.out.println("2 Authenticate");
                   System.out.println("3 Create Multiple Accounts");
                   sc.next();
               }

               choice = sc.nextInt();
           }
           catch(InputMismatchException e){ //catch exception in case, this should
never get called because of the !sc.hasNextInt()
               System.out.println(e);
               System.exit(1);
           }

       } while ((choice != 3) && (choice!=2) && (choice!=1)); //repeat the loop if
the user didn't enter 1 or 2

       return choice; //return validated choice
   }

   /**
    * Handles Authenticating/logging in to an account.
    **/
   private static void authenticate(){
       System.out.println("Authenticate using a username and password.");
       String username = getUsername();
       String password = getPassword();

       if(validatePlain(username,password)){ //Plaintext Validation
           System.out.println("Authenticated with Plaintext.");
       }
       else{
           System.out.println("Plaintext Authentication failed.");
       }

       if(validateHash(username,password)){ //Hash Validation
           System.out.println("Authenticated with Hash.");
       }
       else{
           System.out.println("Hash Authentication failed.");
       }

       if(validateSalt(username,password)){ //Salt + Hash Validation
           System.out.println("Authenticated with Salt + Hash.");
       }
       else{
           System.out.println("Salt + Hash Authentication failed.");
       }



   }


   /**
```

```java
     * Handles Creating an account.
     **/
    private static void createAccount(){ //Create an account

        String filename="./usernames.txt"; //Path to usernames file
        File userFile = new File(filename);
        System.out.println("Create Account.");

        String username = createUsername(); //create a username

        //Check if username is taken
        if(userFile.exists()) {
            try (BufferedReader br = new BufferedReader(new FileReader(userFile))) {
                while (br.ready()) {
                    if(br.readLine().equals(username)) {
                        System.out.println("That username is already taken, please
restart and try again.");
                        System.exit(2);
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        //Add username to a list to check if it exists before creating the files
        try {
            FileWriter fileWriter = new FileWriter(filename, true); //Set true for
append mode
            PrintWriter printWriter = new PrintWriter(fileWriter);
            printWriter.println(username);
            printWriter.close();
        }
        catch(IOException io){
            io.printStackTrace();
        }

        String password = createPassword(); //create a password

        createFiles(username,password); //Create password files


    }


    /**
     * Prompts the user for a username and validates that it fits
     * the criteria.
     * @return Returns the username as a string.
     **/
    private static String createUsername(){ //Create a valid username from input
        String username="";
        Scanner sc = new Scanner(System.in);
        try{
            System.out.println("Usernames may be up to 10 characters long and
consist only of alphanumeric characters.");
            System.out.println("Enter a username:");

            while (!sc.hasNext("[A-Za-z0-9]{1,10}")) { //using regex to validate
                System.out.println("Invalid: Usernames may be up to 10 characters
long and consist only of alphabetic characters. Try again:");
                sc.next();
            }
            username = sc.next();
            System.out.println("Username Entered: " + username);

        }
        catch(InputMismatchException e){
            System.out.println(e);
            System.exit(1);
        }

        return username;
    }

    /**
     * Prompts the user for a password and validates the password fits
     * the criteria.
     * @return Returns the password as a string.
     **/
    private static String createPassword(){
        String password="";
        Scanner sc = new Scanner(System.in);

        try{
            System.out.println("Passwords may be up to " + maxLengthAllowed + " long
and consist only of the numbers 0-9.");
            System.out.println("Enter a password:");

            while (!sc.hasNext("[0-9]{1,"+maxLengthAllowed+"}")) { //using regex to
validate
                System.out.println("Invalid: Passwords may be up to " +
maxLengthAllowed + " long and consist only of the numbers 0-9. Try again:");
                sc.next();
            }
            password = sc.next();
            System.out.println("Password Entered: " + password);

        }
        catch(InputMismatchException e){
            System.out.println(e);
            System.exit(1);
        }

        return password;
    }
    /**
     * Handles creating the password files by calling the appropriate methods.
     **/
    private static void createFiles(String username, String password){
        createPlainTextFile(username, password);
        createHashedFile(username,password);
        createSaltFile(username, password);
    }

    /**
     * Creates or appends username/password to plaintext password file.
     **/
    private static void createPlainTextFile(String username, String password){
        String textToAppend = username + "," + password;

        try {
            FileWriter fileWriter = new FileWriter(plainTextFilePath, true); //Set
true for append mode
            PrintWriter printWriter = new PrintWriter(fileWriter);
            printWriter.println(textToAppend);  //New line
            printWriter.close();
        }
        catch(IOException io){
            io.printStackTrace();
        }
    }

    /**
     * Creates or appends username/password to hashed password file.
     **/
    private static void createHashedFile(String username, String password) {
        String textToAppend="";
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");

            md.update(password.getBytes(StandardCharsets.UTF_8));
            byte[] digest = md.digest();

            String hexPass = byteArrayToHexString(digest);

            textToAppend = username + "," + hexPass;
        }
        catch(NoSuchAlgorithmException n){
            n.printStackTrace();
        }

        try {
            FileWriter fileWriter = new FileWriter(hashFilePath, true); //Set true
for append mode
            PrintWriter printWriter = new PrintWriter(fileWriter);
            printWriter.println(textToAppend);  //New line
            printWriter.close();
        }
        catch(IOException io){
```

```java
            io.printStackTrace();
        }
    }


    /**
     * Creates or appends username/password to Salt password file.
     **/
    private static void createSaltFile(String username, String password){
        String textToAppend="";
        String hexPass;
        String hexSalt;

        SecureRandom random = new SecureRandom(); //Get a secure random number
        byte[] salt = new byte[1];
        random.nextBytes(salt);

        KeySpec spec = new PBEKeySpec(password.toCharArray(), salt, 65536, 128);
        try {
            SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
            byte[] hash = factory.generateSecret(spec).getEncoded();
            hexPass = byteArrayToHexString(hash); //salted hash pass in hex
            hexSalt = byteArrayToHexString(salt); //hexed Salt value
            textToAppend = username + "," + hexSalt + "," + hexPass; //text to add
to file
        }

        catch(NoSuchAlgorithmException n){
            n.printStackTrace();
        }
        catch(InvalidKeySpecException i){
            i.printStackTrace();
        }


        try {
            FileWriter fileWriter = new FileWriter(saltFilePath, true); //Set true
for append mode
            PrintWriter printWriter = new PrintWriter(fileWriter);
            printWriter.println(textToAppend);  //New line
            printWriter.close();
        }
        catch(IOException io){
            io.printStackTrace();
        }
    }

    /**
     * Converts Byte Array to Hex String
     **/
    public static String byteArrayToHexString(byte[] bytes) {
        BigInteger bigInteger = new BigInteger(1, bytes);
        return bigInteger.toString(16);
```

```java
}


/**
 * Converts Hex Strings to Byte Array
 **/
public static byte[] hexStringToByteArray(String hexString) {
    byte[] byteArray = new BigInteger(hexString, 16)
            .toByteArray();
    if (byteArray[0] == 0) {
        byte[] output = new byte[byteArray.length - 1];
        System.arraycopy(
                byteArray, 1, output,
                0, output.length);
        return output;
    }
    return byteArray;
}

/**
 * Get a username for authentication
 **/
private static String getUsername(){
    String username="";
    Scanner sc = new Scanner(System.in);

    try{
        System.out.println("Enter username:");

        while (!sc.hasNext("[A-Za-z0-9]{1,10}")) { //using regex to validate
            System.out.println("Invalid username, try again:");
            sc.next();
        }
        username = sc.next();

    }
    catch(InputMismatchException e){
        System.out.println(e);
        System.exit(1);
    }

    return username;
}


/**
 * Get a password for authentication
 **/
private static String getPassword(){
    String password="";
    Scanner sc = new Scanner(System.in);

    try{
            System.out.println("Enter password:");

            while (!sc.hasNext("[0-9]{1,"+maxLengthAllowed+"}")) { //using regex to
validate
                System.out.println("Invalid password, try again:");
                sc.next();
            }
            password = sc.next();

        }
        catch(InputMismatchException e){
            System.out.println(e);
            System.exit(1);
        }

        return password;
    }

    /**
     * Validates against the plain text password file.
     * @return true if validated, false if not.
     */
    private static boolean validatePlain(String username, String password){
        boolean validated = false;
        Scanner scanner = null;
        try {
            scanner = new Scanner(new File(plainTextFilePath));
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                if(line.equals(username+","+password)){
                    validated = true;
                    break;
                }
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        return validated;
    }


    /**
     * Validates against the hash password file.
     * @return true if validated, false if not.
     */
    private static boolean validateHash(String username, String password){
        boolean validated = false;
        String hashPassword="";

        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");

            md.update(password.getBytes(StandardCharsets.UTF_8));
            byte[] digest = md.digest();

            hashPassword = byteArrayToHexString(digest);


        }
        catch(NoSuchAlgorithmException n){
            n.printStackTrace();
        }


        Scanner scanner = null;
        try {
            scanner = new Scanner(new File(hashFilePath));
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                if(line.equals(username+","+hashPassword)){
                    validated = true;
                    break;
                }
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        return validated;
    }


    /**
     * Validates against the salt + hash password file.
     * @return true if validated, false if not.
     */
    private static boolean validateSalt(String username, String password){
        boolean validated = false;
        File file = new File(saltFilePath);
        List<String> authList = null; //array containing the username,salt,password
        ArrayList<String> authArrayList = null;

        //get user salt and pass from file
        if(file.exists()) {
            try (BufferedReader br = new BufferedReader(new FileReader(file))) {
                while (br.ready()) {
                    String line = br.readLine();
                    if(line.startsWith(username)) {
                        String[] splitLine = line.split(",");
                        authList = Arrays.asList(splitLine); //store csv row to
array

                        authArrayList = new ArrayList(authList);
                        break;
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        byte[] byteArrSalt = hexStringToByteArray(authArrayList.get(1));

        KeySpec spec = new PBEKeySpec(password.toCharArray(), byteArrSalt, 65536,
128);
        try {
            SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
            byte[] hash = factory.generateSecret(spec).getEncoded();
            String hexPass = byteArrayToHexString(hash); //salted hash pass in hex
            if(hexPass.equals(authArrayList.get(2))){
                validated = true;
            }
        }

        catch(NoSuchAlgorithmException n){
            n.printStackTrace();
        }
        catch(InvalidKeySpecException i){
            i.printStackTrace();
        }
        return validated;
    }

    //Asks for user to input the number of accounts to be generated
    private static int accNum() {

        int numAcc = 0;

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of accounts: ");
        numAcc = sc.nextInt();

        while(numAcc < 0 || numAcc > Integer.MAX_VALUE) {
            System.out.println("Please enter a valid number of accounts.");
            numAcc = sc.nextInt();
        }

        return numAcc;
    }

    //Asks for user to input the length of the password
    private static int pass() {

        int length = 0;

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the password length: ");
        length = sc.nextInt();
```

```
       while(length < 3 || length > 8) {
           System.out.println("Please enter a valid password length.");
           length = sc.nextInt();
       }

       sc.close();

       return length;
   }

   //
   private static void generateUserPassAcc(int num, int passWord) {

       String base = "user";
       String pass_word;

       for(int i = 0; i < num; i++) {

           String username = "";

           //0000001 format - converts i to string to store
           String temp = Integer.toString(i);

           if(i < 10) {
               username = base + "00000" + temp;
           }
           else {
               username = base + "0000" + temp;
           }

           pass_word = passwords(passWord);

           createFiles(username,pass_word); // from your code so we can send those
newly generated accounts to the three files

       }
   }

   private static String passwords(int length) {

       String password = "";
       String temp = "9";      //placeholder for the largest digit for the range ie
9xxxx - depends on user length (len=4 => 9999)
       String temp2 = "";

       for(int i = 0; i < length - 1; i++) {
           //creates the temp strings as mentioned above
           temp += "9";
       }


       for(int i = 0; i < length; i++) {
           temp2 += "0";
       }
```

```
        //Converts strings to integers for range
        int newNum2 = Integer.parseInt(temp);

        //Generates random number in user's length range
        int randomNum = ThreadLocalRandom.current().nextInt(0, newNum2);

        DecimalFormat df = new DecimalFormat(temp2);

        password = Integer.toString(randomNum);

        password = df.format(randomNum);

        return password;
    }

}
```

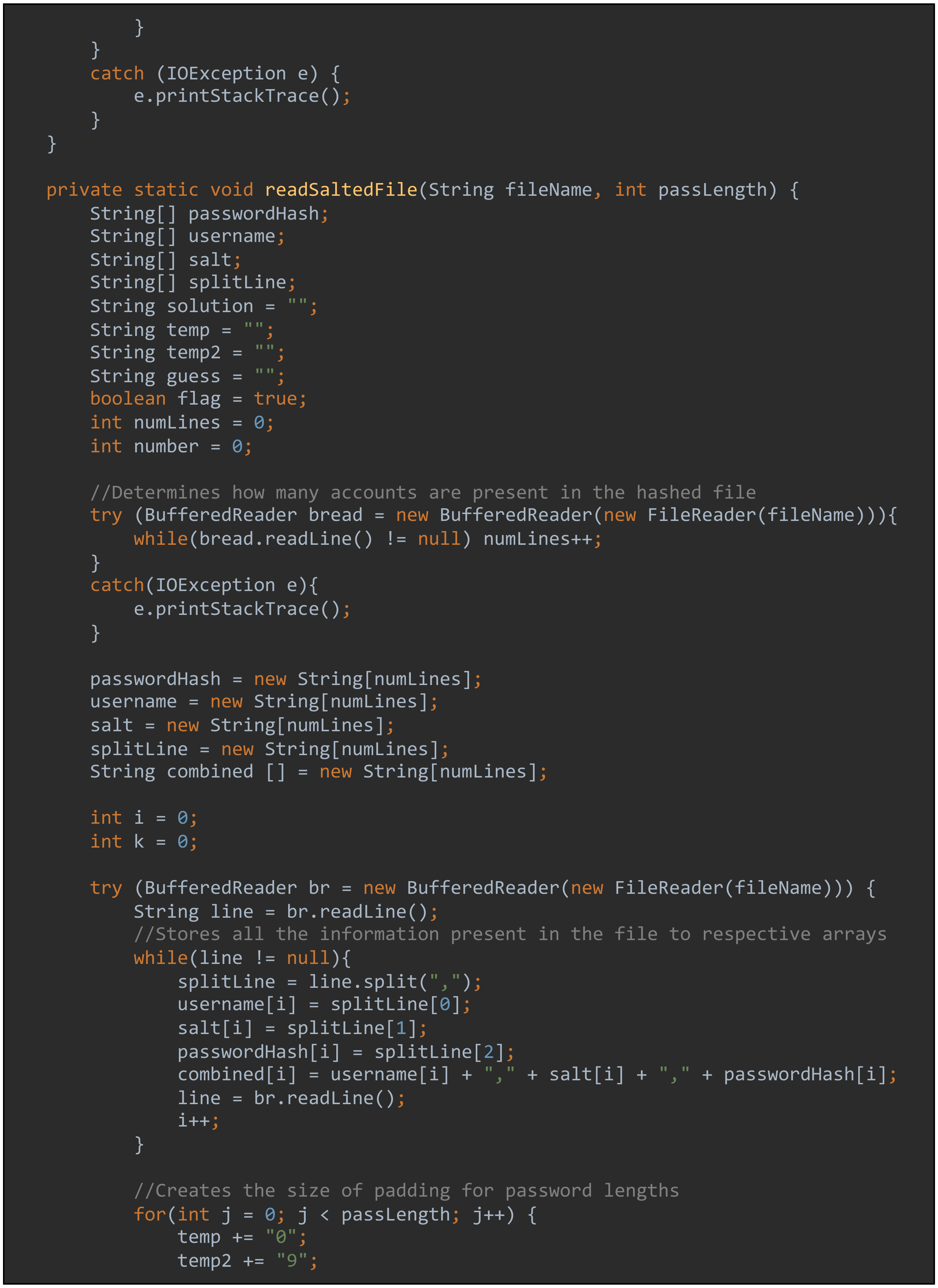
Crack.java - contains Task 3

```
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import java.io.*;
import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.KeySpec;
import java.text.DecimalFormat;
import java.util.*;

public class Crack {

   static final String hashFilePath = "./hashFile.txt";
   static final String saltFilePath = "./saltFile.txt";

   public static void main(String[] args) throws Exception {

       int file = readFileChoice();
       int length = readPasswordLength();

       //Values to count execution time of cracking a hashed file or salted file
       float startTime = 0;
       float endTime = 0;
```

```
        if(file == 1) {
            startTime = System.nanoTime();
            readHashedFile(hashFilePath, length);
            endTime = System.nanoTime();
        }
        else if (file == 2) {
            startTime = System.nanoTime();
            readSaltedFile(saltFilePath, length);
            endTime = System.nanoTime();
        }
        System.out.println(“Time to crack: “ + (endTime - startTime) / 1000000);
    }


    //Reads user input for which file they would like to crack and verifies input
    private static int readFileChoice() {
        int choice = 0;

        Scanner sc = new Scanner(System.in);

        do {
            System.out.println("1 Hashed Password File");
            System.out.println("2 Hashed + Salted Password File");
            System.out.println("Enter the file to crack: ");

            while(!sc.hasNextInt()) {
                System.out.println("That's not a valid choice, try again!");
                System.out.println("1 Hashed Password File");
                System.out.println("2 Hashed + Salted Password File");
                sc.next();
            }

            choice = sc.nextInt();
        } while((choice != 1) && choice !=2);

        return choice;
    }

    /*
     *Retrieves the set length of the password... i.e. user would like to crack all
3 digit passwords
     *in hashed file or salted file
     */
    private static int readPasswordLength() {
        int length = 0;

        Scanner sc = new Scanner(System.in);

        do {
            System.out.println("Enter maximum password size (3-8): ");

            while(!sc.hasNextInt()) {
                System.out.println("That's not a valid choice, try again!");
```

```java
               System.out.println("Enter maximum password size (3-8): ");
               sc.next();
           }

           length = sc.nextInt();
       }while((length < 3) || (length > 8));

       return length;
   }

   private static void readHashedFile(String fileName, int passLength) {

       String hashPassword = "";
       int numLines = 0;
       int i = 0;
       String temp = "";
       String guess = "";
       int number = 0;
       String solution = "";
       String temp2 = "";

       //Determines how many accounts are present in the hashed file
       try (BufferedReader bread = new BufferedReader(new FileReader(fileName))){
           while(bread.readLine() != null) numLines++;
       }
       catch(IOException e){
           e.printStackTrace();
       }


       try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
           String username[] = new String[numLines];
           String splitLine[] = new String[numLines];
           String hash[] = new String[numLines];
           String line = br.readLine();
           String combined [] = new String[numLines];

           //Stores all the information present in the file to respective arrays
           while(line != null){
               splitLine = line.split(",");
               username[i] = splitLine[0];
               hash[i] = splitLine[1];
               combined[i] = username[i] + "," + hash[i];
               line = br.readLine();
               i++;
           }

           try {

               //Creates padding for the digits. i.e. if length is 5 digits 123
would be 00123
               for(int j = 0; j < passLength; j++) {
                   temp += "0";
                   temp2 += "9";
```

```
                }


                DecimalFormat df = new DecimalFormat(temp);
                int temp_2 = Integer.parseInt(temp2);

                int f = 0;
                while(true) {

                    //Number being guessed is changed to string and then padded with
zeroes
                    guess = Integer.toString(number);
                    guess = df.format(number);

                    //Used to exit while loop once there are no more values to check
                    if (f >= username.length){
                        break;
                    }

                    //Hashes the guessed password with SHA-256
                    MessageDigest md = MessageDigest.getInstance("SHA-256");
                    md.update(guess.getBytes(StandardCharsets.UTF_8));
                    byte[] digest = md.digest();
                    hashPassword = byteArrayToHexString(digest);

                    //Checks the current guessed hashed password to all the hashes
in the file
                    for(int q = 0; q < hash.length; q++){
                        solution = hash[q];
                        //If guessed password = hashed password then display and
increment to next hash
                        if (solution.equals(hashPassword)) {
                            for(int j = 0; j < combined.length; j++){
                                if(combined[j].contains(hashPassword)){
                                    System.out.println(combined[j] + ": PASSWORD IS:
" + guess);
                                }
                            }
                            f++;
                        }
                    }


                    number++;

                    //Used to break from loop if guessed number is greater than
digits its seeking
                    if(number > temp_2) {
                        break;
                    }
                }
            }
            catch(NoSuchAlgorithmException n) {
                n.printStackTrace();
```



```java
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void readSaltedFile(String fileName, int passLength) {
        String[] passwordHash;
        String[] username;
        String[] salt;
        String[] splitLine;
        String solution = "";
        String temp = "";
        String temp2 = "";
        String guess = "";
        boolean flag = true;
        int numLines = 0;
        int number = 0;

        //Determines how many accounts are present in the hashed file
        try (BufferedReader bread = new BufferedReader(new FileReader(fileName))){
            while(bread.readLine() != null) numLines++;
        }
        catch(IOException e){
            e.printStackTrace();
        }

        passwordHash = new String[numLines];
        username = new String[numLines];
        salt = new String[numLines];
        splitLine = new String[numLines];
        String combined [] = new String[numLines];

        int i = 0;
        int k = 0;

        try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
            String line = br.readLine();
            //Stores all the information present in the file to respective arrays
            while(line != null){
                splitLine = line.split(",");
                username[i] = splitLine[0];
                salt[i] = splitLine[1];
                passwordHash[i] = splitLine[2];
                combined[i] = username[i] + "," + salt[i] + "," + passwordHash[i];
                line = br.readLine();
                i++;
            }

            //Creates the size of padding for password lengths
            for(int j = 0; j < passLength; j++) {
                temp += "0";
                temp2 += "9";
```

```
           }

           //Creates padding for the digits. i.e. if length is 5 digits 123 would
be 00123
           DecimalFormat df = new DecimalFormat(temp);
           int temp_2 = Integer.parseInt(temp2);

           while(flag){

               if(k >= passwordHash.length){
                   break;
               }

               //Formats int to string with padding so that it can be hashed
               guess = Integer.toString(number);
               guess = df.format(number);
               solution = passwordHash[k];



               byte[] byteArrSalt = hexStringToByteArray(salt[k]);
               KeySpec spec = new PBEKeySpec(guess.toCharArray(), byteArrSalt,
65536, 128);

               try {
                   //Decrypts salt
                   SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
                   byte[] hash = factory.generateSecret(spec).getEncoded();
                   String hexPass = byteArrayToHexString(hash);

                   //Compares the guessed salted password hash to all of hashes in
the file
                   for(int q = 0; q < passwordHash.length; q++){
                       solution = passwordHash[q];
                       //If hash in salted file matches guessed salted hash then
display
                       if (solution.equals(hexPass)) {
                           for(int j = 0; j < combined.length; j++){
                               if(combined[j].contains(hexPass)){
                                   System.out.println(combined[j] + ": PASSWORD IS:
" + guess);
                               }
                           }
                           //Resets the incrementer so that it can continue
guessing values from XXX-passlength
                           number = 0;
                           k++;
                       }
                   }
                   number++;
                   //System.out.println(number);
               }
               catch(NoSuchAlgorithmException n) {
                   n.printStackTrace();
               }
               catch(InvalidKeySpecException m) {
                   m.printStackTrace();
               }
               //If incrementor is greater than acceptable pass length for guessing
then increment the salt and reset counter
               if(number > temp_2){
                   k++;
                   number = 0;
                   //Breaks from loop if there are no more passwords of set length
                   if(k >= passwordHash.length){
                       flag = false;
                   }
               }
           }
       }
       catch(IOException e){
           e.printStackTrace();
       }

   }

   //Converts Byte Array to Hex String
   public static String byteArrayToHexString(byte[] bytes) {
       BigInteger bigInteger = new BigInteger(1, bytes);
       return bigInteger.toString(16);
   }

   // Converts Hex Strings to Byte Array
   public static byte[] hexStringToByteArray(String hexString) {
       byte[] byteArray = new BigInteger(hexString, 16)
               .toByteArray();
       if (byteArray[0] == 0) {
           byte[] output = new byte[byteArray.length - 1];
           System.arraycopy(
                   byteArray, 1, output,
                   0, output.length);
           return output;
       }
       return byteArray;
   }

}
```