JAVA Programming Lab (JPL)

(TE EXTC)

Mini Project Report

Sardar Patel Institute of Technology

Munshi Nagar, Andheri (W), Mumbai-400058

University of Mumbai

2023-2024

Project Topic: Password Manager

Under Prof. Nikahat Kazi

Group Members

Sr.No	Name	UID	Batch
1.	Rushikesh Khedekar	2022200069	J1
2.	Krishna Kumavat	2022200075	J1
3.	Amey Parab	2023201019	J1

Work done by all of 3 members are specified as:

Amey Parab – Application Overview and overall GUI design of project.

Rushikesh Khedekar – Custom Data Structures and also Backend work.

Krishna Kumavat – Security and Cryptography for encryption and decryption.

Contents

➢ **Project Overview**:

Project is a desktop application developed in Java, specifically designed to manage passwords and notes. It provides a user-friendly graphical interface built using the Swing library. The application does not use a database, instead, it stores data locally, either in memory or in local files.

Here are the key components and functionalities of your project:

1. **Loading Screen:** The application starts with a loading screen, providing visual feedback to the user that the application is loading.
2. **Password Management**: The application allows users to store their passwords securely. It uses a hash table for efficient storage and retrieval of passwords. The passwords are encrypted using the PBEWithMD5AndDES encryption algorithm for security.
3. **Password Generation:** The application includes a password generator that can generate a random password of a given length. This feature can be useful for users who want to create strong, random passwords.
4. **Notes Management:** In addition to managing passwords, the application also allows users to store and retrieve notes. This could be useful for storing additional information related to the passwords, such as the associated username or the purpose of the password.
5. **User Interface:** The application provides a graphical user interface (GUI) that makes it easy for users to interact with the application. It includes several screens for different functionalities, such as storing passwords, generating passwords, and adding notes.

The application is developed in Visual Studio Code and the recommended Java version for this project is Java SE 18.0.2.1. This ensures that the project can leverage the latest features and improvements in the Java language. The project does not use a database, which makes the application simpler and more portable, though it may have limitations in terms of data persistence and storage size.

➢ <u>**Tech Stack**</u> **:**

1. Language : JAVA
2. Java GUI Toolkit : Swing
3. Development Kit : VS Code

Java is chosen for its portability, robustness, and widespread adoption. Its object-oriented nature allows for organized code structuring, making it suitable for building complex applications**.** Swing is used to create the application's user interface. It provides a wide range of GUI components that are customizable and platform-independent. Swing facilitates the creation of interactive and visually appealing graphical interfaces.The Swing library helps design the user interface with various components like buttons, text areas, labels, etc., for user interaction.

JCA provides cryptographic functionalities for secure communication, data integrity, and confidentiality. In this case, it's used for encrypting and decrypting sensitive information (like passwords) to ensure data security. The custom implementation of a HashTable using linear probing allows efficient storage and retrieval of password and account information. This data structure resolves collisions by linearly probing through the table to find an empty slot for insertion. ‘SecureRandom’ is used for generating cryptographically strong random numbers. In the context of this application, it's employed to generate random passwords with high entropy.

This tech stack combines Java's core capabilities with Swing for the graphical interface, cryptographic features for data security, and custom data structures to build a functional password and notes management application.

➢ <u>Project Structure and Components</u> :

1. **Imports:** Your code begins by importing several Java libraries. These libraries provide classes and methods that your application uses.
2. **SplashScreen Class:** This class is responsible for creating and managing the loading screen of your application.
3. **HashtablePassword Class:** This class implements a hash table for password management. It uses either linear or quadratic probing as a collision resolution technique.
4. **PasswordGenerator Class:** This class is used to generate random passwords. It has a generatePassword method that takes the length of the password as input and returns a randomly generated password of that length.
5. **hashTableMap Interface**: This interface defines the methods that a hash table for account management should have. It includes methods for getting an account (get_Acc), adding an account (add_Acc), and removing an account (remove_Acc).
6. **PasswordManager Class:** This class seems to be the main class for your application. It implements the ActionListener interface, which means it can respond to user actions like button clicks. It has several GUI components like buttons, labels, and text fields for managing passwords and notes. It also has a HashtablePassword object for storing account data.
7. **Main Method:** This is the entry point of your application. It creates a new SplashScreen object to display the loading screen, and then creates a new PasswordManager object to start the application.

➢ **<u>Key Features and Functionality</u>:**

<u>Key feature are as follows:</u>

1. Generate a random Password of a length >4.
2. Store and Retrieve passwords.
3. Delete a password.
4. Encrypt a plain text with a secret key.
5. Decrypt the encrypted text with the secret key.
6. Add a note and view the note added.

Functionality:

1. **User Interface:** The application provides a graphical user interface (GUI) built using the Swing library. This makes it easy for users to interact with the application.
2. **Password Management:** Users can store their passwords in the application. The passwords are likely encrypted for security purposes. Users can add, update, delete, and retrieve their passwords.
3. **Notes Management:** Users can also store notes in the application. This could be useful for storing additional information related to the passwords, such as the associated username or the purpose of the password.
4. **Loading Screen:** When the application is launched, a loading screen is displayed. This includes a progress bar that fills up over time, providing visual feedback to the user that the application is loading.
5. **No Database:** The application does not use a database. This means that all data is stored locally, either in memory or in local files. This makes the application more portable, but it may have limitations in terms of data persistence and storage size.
6. **Desktop Application:** The application is a desktop application, which means it is installed and runs on a user’s personal computer. All data is stored locally on the user’s machine.

➢ **Program:**

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.security.SecureRandom;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.AlgorithmParameterSpec;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.KeySpec;
import java.util.Base64;
import javax.crypto.*;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.PBEParameterSpec;


// This class is used to create a loading screen
class SplashScreen {
    JFrame frame;
    JLabel image=new JLabel(new ImageIcon("key-lock.png"));
    JLabel text=new JLabel("PASSWORD & NOTES MANAGER");
    JProgressBar progressBar=new JProgressBar();
    JLabel message=new JLabel();
    SplashScreen()
    {
        createGUI();
        addImage();
        addText();
        addProgressBar();
        runningPBar();
    }
```

```java
    public void createGUI(){
        frame=new JFrame(); // to create a frame
        frame.getContentPane().setLayout(null); // to set the layout of
the frame
        frame.setUndecorated(true);
        frame.setSize(400,400); // to set the size of the frame
        frame.setLocationRelativeTo(null);
        frame.getContentPane().setBackground(new Color(0XFF8787)); //
to set the background color of the frame
        frame.setVisible(true);
    }

    public void addImage(){
        image.setSize(400,200); // to set the size of the image
        frame.add(image);
    }

    public void addText()
    {
        text.setFont(new Font("MV Boli",Font.BOLD,20)); // to set the
font of the text
        text.setBounds(30,200,400,30);
        text.setForeground(Color.black);
        frame.add(text);
    }

    public void addProgressBar(){
        progressBar.setBounds(100,280,200,30); // to set the size of
the progress bar
        progressBar.setBorderPainted(true);
        progressBar.setStringPainted(true);
        progressBar.setBackground(Color.black);
        progressBar.setForeground(new Color(0X38E54D));
        progressBar.setValue(0);
        frame.add(progressBar);
    }
    public void runningPBar(){
        int i=0;//Creating an integer variable and initializing it to 0
        while( i<=100)
```

```
        {
            try{
                Thread.sleep(40);   //Pausing execution for 50
milliseconds
                progressBar.setValue(i);    //Setting value of Progress
Bar
                i++;
                if(i==100)
                    frame.dispose();
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}

//Linear Probing Implementation
class HashtablePassword implements hashTableMap {
    private final int useProbe;    //0 = Linear Probing, 1 = Quadratic
Probing
    private Entry[] entries;       //The array of entries
    private final float loadFactor;     //The load factor
    private int size, used;          //used acquires space for NIL
    private final Entry NIL = new Entry(null, null); //Deleted entries

    private static class Entry{
        Object key, value;
        Entry(Object k, Object v){
            key = k;    value = v;
        }
    }
    public HashtablePassword(int capacity, float loadFactor, int
useProbe){
        entries = new Entry[capacity];
        this.loadFactor = loadFactor;
        this.useProbe = useProbe;
    }


    //Complementary functions
```

```
    public int hash(Object key){
        return (key.hashCode() & 0x7FFFFFFF) % entries.length;
    }

    private int nextProbe(int h, int i){
        return (h+i) % entries.length; //Linear Probing
    }

    private void rehash(){
        Entry[] oldEntries = entries;
        entries = new Entry[2*entries.length+1];
        for (Entry entry : oldEntries) {
            if (entry == NIL || entry == null) continue;
            int h = hash(entry.key);
            for (int x = 0; x < entries.length; x++) {
                int j = nextProbe(h, x);
                if (entries[j] == null) {
                    entries[j] = entry;
                    break;
                }
            }
            used = size;
        }
    }

    @Override
    public int add_Acc(Object Account, Object passwd) {
        if(used > (loadFactor*entries.length))rehash();
        int h = hash(Account);
        for (int i = 0; i < entries.length; i++){
            int j = (h+i) % entries.length;
            Entry entry = entries[j];
            if(entry==null){
                entries[j]= new Entry(Account, passwd);
                ++size;
                ++used;
                return h;
            }
            if(entry == NIL)continue;
            if(entry.key.equals(Account)){
```

```
                Object oldValue = entry.value;
                entries[j].value = passwd;
                return (int) oldValue;
            }
        }
        return h;
    }

    @Override
    public Object get_Acc(Object Account) {
        int h = hash(Account);
        for(int i = 0; i < entries.length; i++){
            int j = nextProbe(h , i);
            Entry entry = entries[j];
            if(entry == null)break;
            if(entry == NIL)continue;
            if(entry.key.equals(Account)) return entry.value;
        }
        return null;
    }

    @Override
    public Object remove_Acc(Object Account) {
        int h = hash(Account);
        for(int i = 0; i < entries.length; i++){
            int j = nextProbe(h,i);
            Entry entry = entries[j];
            if(entry == NIL)continue;
            if(entry.key.equals(Account)){
                Object Value = entry.value;
                entries[j] = NIL;
                size--;
                return Value;
            }
        }
        return null;
    }
}
class CryptoUtil
{
```

```

    Cipher ecipher;
    Cipher dcipher;
    // 8-byte Salt
    byte[] salt = {
        (byte) 0xA9, (byte) 0x9B, (byte) 0xC8, (byte) 0x32,
        (byte) 0x56, (byte) 0x35, (byte) 0xE3, (byte) 0x03
    };
    // Iteration count
    int iterationCount = 19;

    public CryptoUtil() {

    }

    /**
     *
     * @param secretKey Key used to encrypt data
     * @param plainText Text input to be encrypted
     * @return Returns encrypted text
     * @throws java.security.NoSuchAlgorithmException
     * @throws java.security.spec.InvalidKeySpecException
     * @throws javax.crypto.NoSuchPaddingException
     * @throws java.security.InvalidKeyException
     * @throws java.security.InvalidAlgorithmParameterException
     * @throws java.io.UnsupportedEncodingException
     * @throws javax.crypto.IllegalBlockSizeException
     * @throws javax.crypto.BadPaddingException
     *
     */
    public String encrypt(String secretKey, String plainText)
            throws NoSuchAlgorithmException,
            InvalidKeySpecException,
            NoSuchPaddingException,
            InvalidKeyException,
            InvalidAlgorithmParameterException,
            UnsupportedEncodingException,
            IllegalBlockSizeException,
            BadPaddingException {
        //Key generation for enc and desc
```

```
        KeySpec keySpec = new PBEKeySpec(secretKey.toCharArray(), salt,
iterationCount);
        SecretKey key =
SecretKeyFactory.getInstance("PBEWithMD5AndDES").generateSecret(keySpe
c
);
        // Prepare the parameter to the ciphers
        AlgorithmParameterSpec paramSpec = new PBEParameterSpec(salt,
iterationCount);

        //Enc process
        ecipher = Cipher.getInstance(key.getAlgorithm());
        ecipher.init(Cipher.ENCRYPT_MODE, key, paramSpec);
        String charSet = "UTF-8";
        byte[] in = plainText.getBytes(charSet);
        byte[] out = ecipher.doFinal(in);
        String encStr = new String(Base64.getEncoder().encode(out));
        return encStr;
    }

    /**
     * @param secretKey Key used to decrypt data
     * @param encryptedText encrypted text input to decrypt
     * @return Returns plain text after decryption
     * @throws java.security.NoSuchAlgorithmException
     * @throws java.security.spec.InvalidKeySpecException
     * @throws javax.crypto.NoSuchPaddingException
     * @throws java.security.InvalidKeyException
     * @throws java.security.InvalidAlgorithmParameterException
     * @throws java.io.UnsupportedEncodingException
     * @throws javax.crypto.IllegalBlockSizeException
     * @throws javax.crypto.BadPaddingException
     */
    public String decrypt(String secretKey, String encryptedText)
            throws NoSuchAlgorithmException,
            InvalidKeySpecException,
            NoSuchPaddingException,
            InvalidKeyException,
            InvalidAlgorithmParameterException,
            UnsupportedEncodingException,
            IllegalBlockSizeException,
```

```
            BadPaddingException,
            IOException {
        //Key generation for enc and desc
        KeySpec keySpec = new PBEKeySpec(secretKey.toCharArray(), salt,
iterationCount);
        SecretKey key =
SecretKeyFactory.getInstance("PBEWithMD5AndDES").generateSecret(keySpe
c
);
        // Prepare the parameter to the ciphers
        AlgorithmParameterSpec paramSpec = new PBEParameterSpec(salt,
iterationCount);
        //Decryption process; same key will be used for decr
        dcipher = Cipher.getInstance(key.getAlgorithm());
        dcipher.init(Cipher.DECRYPT_MODE, key, paramSpec);
        byte[] enc = Base64.getDecoder().decode(encryptedText);
        byte[] utf8 = dcipher.doFinal(enc);
        String charSet = "UTF-8";
        String plainStr = new String(utf8, charSet);
        return plainStr;
```

```java
    }

}

class PasswordGenerator {
    private static final SecureRandom random = new SecureRandom();
    private static final String caps="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private static final String
small_caps="abcdefghijklmnopqrstuvwxyz";
    private static final String Numeric="1234567890";
    private static final String special_char="~!@#$%^&*(_+{}|:_[?]>=<";
    private static final String dic = caps + small_caps + Numeric +
special_char;

    public String generatePassword(int len) {
        StringBuilder password= new StringBuilder();
        for (int i = 0; i <len ; i++) {
            int index = random.nextInt(dic.length());
            password.append(dic.charAt(index));
        }
        return password.toString();
```

```java
    }

}


interface hashTableMap {

    Object get_Acc(Object Account);
    int add_Acc(Object Account, Object passwd);
    Object remove_Acc(Object Account);
}

class PasswordManager implements ActionListener {

    //Store password class reference
    HashtablePassword data = new HashtablePassword(15,0.5F,0);
    // GUI variables declaration
    JFrame frame;
    JFrame frame2;
    JLabel background;
    Container conn1,conn2;
    JLabel lAcc,lPass;
    JTextArea encryptPasswdArea, genePassArea, searchPassArea;
    JButton PassGeneBtn,PassEncryptBtn, PassStoreBtn, PassSearchBtn,
AccAddBtn, PassDeleteBtn;
    JTextField tAcc,tPass;
    JButton addNoteBtn;
    JLabel addNoteLabel;
    JTextArea tNote;
    JButton addNote;
    JFrame conn3;

    ArrayList<String> notes = new ArrayList<>(); // to store the notes
in an array list of string type

    @Override
    public void actionPerformed(ActionEvent e)  { }

    //Frame settings
```

```
    public static void FrameGUI(JFrame frame){
        frame.setVisible(true);
        frame.setLayout(null);
        frame.setLocationRelativeTo(null);
    }


    //container settings
    public static void ContainerGUI(Container conn){
        conn.setVisible(true);
        conn.setBackground(Color.getHSBColor(20.4f, 10.5f, 12.9f));
        conn.setLayout(null);
    }
```

```java
// buttons settings
public void GUIButtonsSetting(JButton btn){
    btn.setBackground(new Color(0XFB2576));
    btn.setForeground(Color.WHITE);
    btn.setBorder(BorderFactory.createLineBorder(Color.BLACK, 3));
    btn.setFocusable(false);
    Cursor crs = new Cursor(Cursor.HAND_CURSOR);
    btn.setCursor(crs);
    Font fn = new Font("MV Boli", Font.BOLD, 15);
    btn.setFont(fn);
}

//GUI of Store password
public void StoringGUI()
{
    frame2 = new JFrame("Store your passwords");
    frame2.setBounds(1400, 300, 800, 500);
    frame2.setSize(400,400);
    FrameGUI(frame2);
    conn2 = frame2.getContentPane();
    ContainerGUI(conn2);
    Font fn = new Font("MV Boli", Font.BOLD, 20);

    //Account textFiled and label
    lAcc = new JLabel("ACCOUNT NAME");
```

```java
    lAcc.setBounds(90, 23, 380, 20);
    lAcc.setFont(fn);
    conn2.add(lAcc);

    tAcc = new JTextField();
    tAcc.setBounds(90,70,200,50);
    tAcc.setFont(fn);
    tAcc.setBorder(BorderFactory.createLineBorder(Color.BLACK, 3));
    tAcc.setForeground(Color.DARK_GRAY);
    conn2.add(tAcc);
```

```
        //Account password textField and label
        lPass = new JLabel("ACCOUNT PASSWORD");
        lPass.setBounds(90, 160, 380, 20);
        lPass.setFont(fn);
        conn2.add(lPass);

        tPass = new JTextField();
        tPass.setBounds(90,200,200,50);
        tPass.setFont(fn);
        tPass.setBorder(BorderFactory.createLineBorder(Color.BLACK,
3));
        tPass.setForeground(Color.DARK_GRAY);
        conn2.add(tPass);

        AccAddBtn = new JButton("STORE");
        AccAddBtn.setBounds(120, 290, 150, 50);
        conn2.add(AccAddBtn);
        GUIButtonsSetting(AccAddBtn);
    }

    //for password generator and encryption
    public void textArea(String Pass,JTextArea TA){
        TA.setText(Pass);
        Font fn = new Font("MV Boli", Font.BOLD, 20);
        TA.setWrapStyleWord(true);
        TA.setLineWrap(true);
        TA.setCaretPosition(0);
        TA.setEditable(false);
        TA.setFont(fn);
```

```
    }

    //GUI of Password Manager
    PasswordManager() {

        frame = new JFrame("Password Manager");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400,650);
        frame.setResizable(false);
        ImageIcon img = new ImageIcon("background.png");
        background = new JLabel("",img,JLabel.CENTER);
        background.setBounds(0,0,400,650);
        background.setVisible(true);
        frame.add(background);

        FrameGUI(frame);

        conn1 = frame.getContentPane();
        ContainerGUI(conn1);

        //Generator buttons settings
        PassGeneBtn = new JButton("GENERATE PASSWORD");
        PassGeneBtn.setBounds(90, 20, 220, 40);
        conn1.add(PassGeneBtn);
        GUIButtonsSetting(PassGeneBtn);

        //generating password
        PassGeneBtn.addActionListener(e -> {
        if(PassGeneBtn ==e.getSource())
        {
            try{
                int len =
Integer.parseInt(JOptionPane.showInputDialog("Enter the password
length"));
                if(len>4)
                {
                    // password generator class reference
                    PasswordGenerator pass = new PasswordGenerator();
                    String passwd = pass.generatePassword(len);
```

```
                    genePassArea = new JTextArea(5,4);
                    textArea(passwd,genePassArea);
                    JOptionPane.showMessageDialog(conn1,new
JScrollPane(genePassArea),"Copy your
password",JOptionPane.INFORMATION_MESSAGE);

                }
                else JOptionPane.showMessageDialog (conn1,"Password
length must be greater than 8!","Invalid Input
Error",JOptionPane.WARNING_MESSAGE);

            }
            catch(Exception
ex){JOptionPane.showMessageDialog(conn1,"Write
something","EXIT!",JOptionPane.ERROR_MESSAGE);}
        }
    }
    );

        // add a encryption button and action
        JButton EncryptBtn = new JButton("ENCRYPT Text");
        EncryptBtn.setBounds(90, 90, 220, 40);
        conn1.add(EncryptBtn);
        GUIButtonsSetting(EncryptBtn);
        EncryptBtn.addActionListener(e -> {
            if(EncryptBtn ==e.getSource())
            {
                try{
                    String text = JOptionPane.showInputDialog("Enter
the text to encrypt");
                    String secretKey =
JOptionPane.showInputDialog("Enter the secret key");
                    if(text.length()>0 && secretKey.length()>0)
                    {
                        // password generator class reference
                        CryptoUtil pass1 = new CryptoUtil();
                        String passwd = pass1.encrypt(secretKey, text);
// encrypting the text
                        genePassArea = new JTextArea(5,4); // text area
for the encrypted text
```

```
                        textArea(passwd,genePassArea); // setting the
```

```
text area
                        JOptionPane.showMessageDialog(conn1,new
JScrollPane(genePassArea),"Copy your
password",JOptionPane.INFORMATION_MESSAGE); // showing the encrypted
text

                    }
                    else JOptionPane.showMessageDialog (conn1,"Write
something","Invalid Input Error",JOptionPane.WARNING_MESSAGE);

                }
                catch(Exception
ex){JOptionPane.showMessageDialog(conn1,"Write
something","EXIT!",JOptionPane.ERROR_MESSAGE);}
            }
        }
        );

        // add a decryption button and action
        JButton DecryptBtn = new JButton("DECRYPT Text");
        DecryptBtn.setBounds(90, 160, 220, 40);
        conn1.add(DecryptBtn);
        GUIButtonsSetting(DecryptBtn);
        DecryptBtn.addActionListener(e -> {
            if(DecryptBtn ==e.getSource())
            {
                try{
                    String text = JOptionPane.showInputDialog("Enter
the text to decrypt"); // getting the encrypted text
                    String secretKey =
JOptionPane.showInputDialog("Enter the secret key"); // getting the
secret key
                    if(text.length()>0 && secretKey.length()>0) //
checking if the text and secret key is not empty
                    {
                        // password generator class reference
                        CryptoUtil pass1 = new CryptoUtil(); //
creating a object of the CryptoUtil class
```

```
                        String passwd = pass1.decrypt(secretKey, text);
// decrypting the text
                        genePassArea = new JTextArea(5,4); // text area
for the decrypted text
                        textArea(passwd,genePassArea); // setting the
text area
                        JOptionPane.showMessageDialog(conn1,new
JScrollPane(genePassArea),"Decrypted
text",JOptionPane.INFORMATION_MESSAGE); // showing the decrypted text

                    }
                    else JOptionPane.showMessageDialog (conn1,"Password
length must be greater than 8!","Invalid Input
Error",JOptionPane.WARNING_MESSAGE);

                }
                catch(Exception
ex){JOptionPane.showMessageDialog(conn1,"Write
something","EXIT!",JOptionPane.ERROR_MESSAGE);}
            }
        }
        );

        //storing password using hashtable
        PassStoreBtn = new JButton("STORE PASSWORD");
        PassStoreBtn.setBounds(90, 230, 220, 40);
        conn1.add(PassStoreBtn);
        GUIButtonsSetting(PassStoreBtn);
        //Store password action
        PassStoreBtn.addActionListener(e -> {
            if(PassStoreBtn ==e.getSource())
            {
                try{
                    StoringGUI();
                    // action on the Store btn
                    AccAddBtn.addActionListener(e4 -> {
                        if (AccAddBtn == e4.getSource()) {
                            String account_name = tAcc.getText(); //
getting the account name
```

```
                            String acc_pass = tPass.getText(); //
getting the password
                            if (account_name.isEmpty() &&
acc_pass.isEmpty()) {

JOptionPane.showMessageDialog(conn2,"unable to store your
password!","ERROR",JOptionPane.ERROR_MESSAGE);
                            }
                            else{
                                //calling put method of the
hashtablePassword class
                                data.add_Acc(account_name,acc_pass); //
adding the account name and password to the hashtable
                                JOptionPane.showMessageDialog(conn2,
"Account added Successfully !");
                                tAcc.setText(null);
                                tPass.setText(null);
                            }
                        }
                    }
                );
            }
           catch(Exception ex)
{JOptionPane.showMessageDialog(conn2,"Write
something","EXIT",JOptionPane.ERROR_MESSAGE);}
        }
    }
    );

    //searching password
    PassSearchBtn = new JButton("SEARCH PASSWORD");
    GUIButtonsSetting(PassSearchBtn);
    PassSearchBtn.setBounds(90, 300, 220, 40);
    conn1.add(PassSearchBtn);
    PassSearchBtn.addActionListener(e ->{
        if (PassSearchBtn ==e.getSource()){
            try{
                String acc_name =
JOptionPane.showInputDialog("Enter your Account Name"); // getting the
account name
```

```
                    if (!acc_name.isBlank()) { // checking if the
account name is not empty
                        Object pass =
data.get_Acc(acc_name.toLowerCase()); // getting the password of the
account name
                        if(pass!=null) { // checking if the password is
not null
                            searchPassArea = new JTextArea(4,5); //
text area for the password
                            textArea(String.valueOf(pass),
searchPassArea); // setting the text area
                            JOptionPane.showMessageDialog(conn1, new
JScrollPane(searchPassArea), "Copy your password",
JOptionPane.INFORMATION_MESSAGE);
                        }
                        else JOptionPane.showMessageDialog(conn1,
"Account not Found!");
                    }
                }
                catch (Exception ex){
                    JOptionPane.showMessageDialog(conn1,"Write
something","EXIT",JOptionPane.ERROR_MESSAGE);
                    }
                }
            }
        );

        // deleting password
        PassDeleteBtn = new JButton("DELETE PASSWORD");
        GUIButtonsSetting(PassDeleteBtn);
        PassDeleteBtn.setBounds(90, 370, 220, 40);
        conn1.add(PassDeleteBtn);
        PassDeleteBtn.addActionListener(e -> {
            if (PassDeleteBtn == e.getSource()) {
                try {
                    String acc_name =
JOptionPane.showInputDialog("Enter the Account Name"); // getting the
account name
                    if (!acc_name.isBlank()) {
```

```
                        data.remove_Acc(acc_name.toLowerCase()); //
removing the account name and password from the hashtable
                        JOptionPane.showMessageDialog(conn1, "Delete
successfully!"); // showing the message
                    }
                    else JOptionPane.showMessageDialog(conn1, "Account
not found!", "INFO", JOptionPane.INFORMATION_MESSAGE);
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(conn1, "Write
something", "EXIT", JOptionPane.ERROR_MESSAGE);
                }
            }


        }
        );

        addNoteBtn = new JButton("ADD NOTE");
        GUIButtonsSetting(addNoteBtn);
        addNoteBtn.setBounds(90, 440, 220, 40);
        conn1.add(addNoteBtn);
        addNoteBtn.addActionListener(e -> {
            if (addNoteBtn == e.getSource()) {
                try {
                    NoteGUI();
                    // action on the add note btn
                    addNote.addActionListener(e4 -> {
                        if (addNote == e4.getSource()) {
                            String note = tNote.getText(); // getting
the note
                            if (note.isEmpty()) {
                                JOptionPane.showMessageDialog(conn3,
"unable to store your note!", "ERROR", JOptionPane.ERROR_MESSAGE);
                            } else {
                                //calling put method of the
hashtablePassword class
                                notes.add(note); // adding the note to
the arraylist
                                JOptionPane.showMessageDialog(conn3,
"Note added Successfully !");
                                conn3.setVisible(false);
```

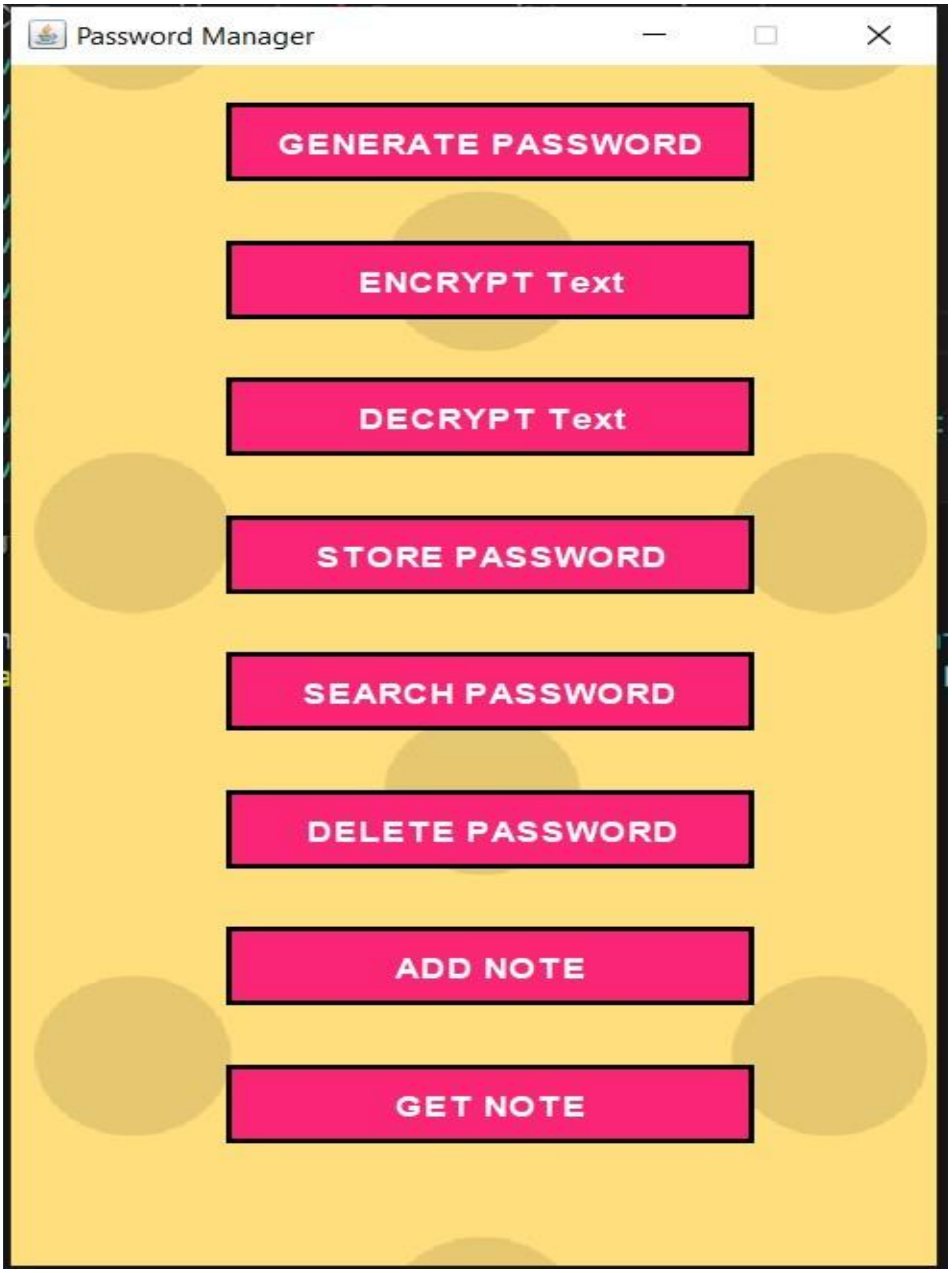
```
                                tNote.setText(null);
                            }
                        }
                    });
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(conn3, "Write
something", "EXIT", JOptionPane.ERROR_MESSAGE);
                }
            }
        }
        );

        //get all notes
        JButton getNoteBtn = new JButton("GET NOTE");
        GUIButtonsSetting(getNoteBtn);
        getNoteBtn.setBounds(90, 510, 220, 40);
        conn1.add(getNoteBtn);
        getNoteBtn.addActionListener(e -> {
            if (getNoteBtn == e.getSource()) {
                try {
                    String allNotes = notes.get(notes.size() - 1); //
getting the last note added
                    if (allNotes.isEmpty()) { // checking if the note
is empty or not
                        JOptionPane.showMessageDialog(conn1, "No note
found!", "INFO", JOptionPane.INFORMATION_MESSAGE); // showing the
message
```

```
                    } else {
                        searchPassArea = new JTextArea(4, 5); // text
area for the note
                        textArea(allNotes, searchPassArea); // setting
the text area
                        JOptionPane.showMessageDialog(conn1, new
JScrollPane(searchPassArea), "Get your notes",
JOptionPane.INFORMATION_MESSAGE); // showing the message
                    }
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(conn1, "Add a note
before trying to retrive", "EXIT", JOptionPane.ERROR_MESSAGE);
                }
```

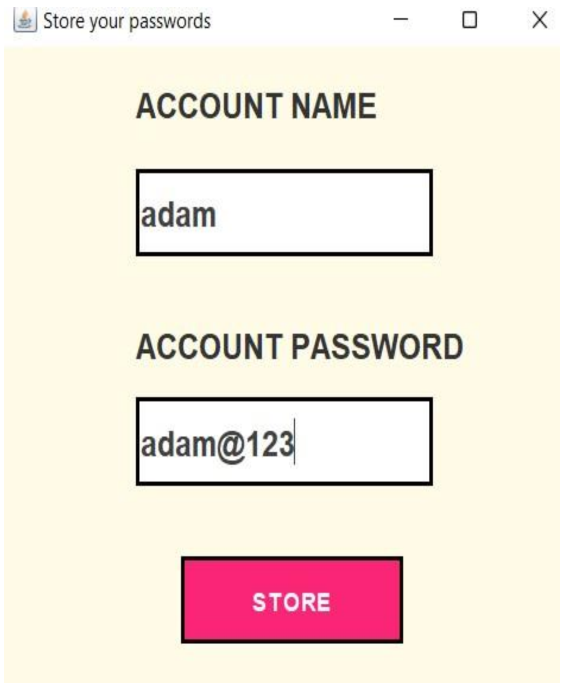
```
        }
    }
    );

}

// method for setting the buttons and GUI for adding notes
private void NoteGUI() {

    conn3 = new JFrame("Add Note");
    conn3.setSize(500, 500);
      conn3.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    conn3.setLocationRelativeTo(null);
    conn3.setLayout(null);
    conn3.setVisible(true);
    conn3.setResizable(false);

    //add note label
     addNoteLabel = new JLabel("Add Note");
    addNoteLabel.setBounds(200, 20, 100, 30);
    conn3.add(addNoteLabel);

    //add note text area
     tNote = new JTextArea(10, 10);
```

```java
    tNote.setBounds(100, 60, 300, 300);
    conn3.add(tNote);

    //add note button
     addNote = new JButton("ADD NOTE");
    GUIButtonsSetting(addNote);
    addNote.setBounds(140, 380, 220, 30);
    conn3.add(addNote);
}

// main method to run the application
public static void main(String[] args) {
    //loading screen class
    new SplashScreen();
    try {
        new PasswordManager();
        }catch (Exception ex) { ex.printStackTrace(); }
 }
}
```

- **Output / Interface:**



- **Accepting the user input and storing the details :**

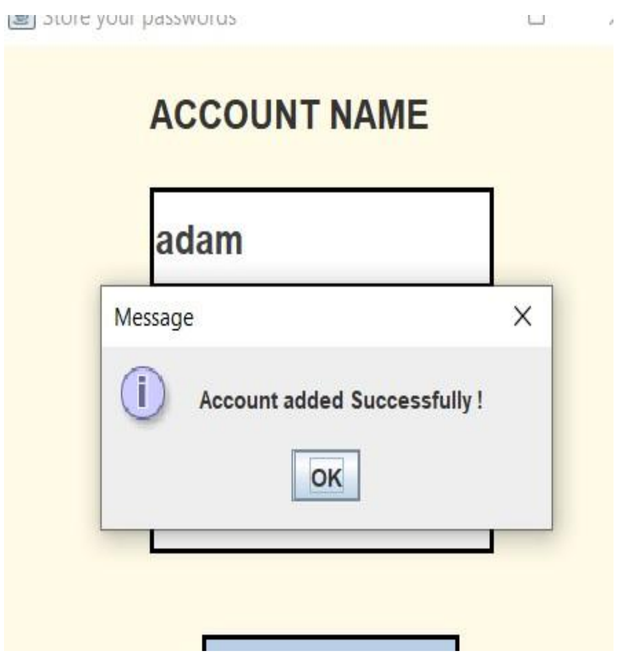

Store your passwords

ACCOUNT NAME

adam

ACCOUNT PASSWORD

adam@123


Store your passwords

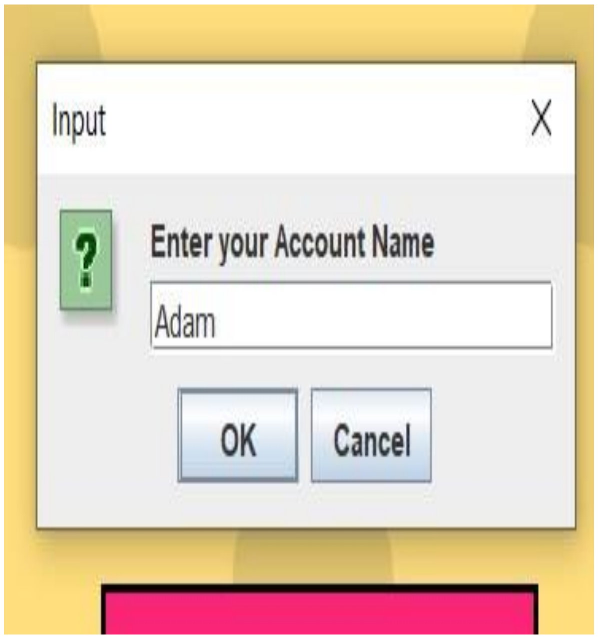
ACCOUNT NAME

adam

Message

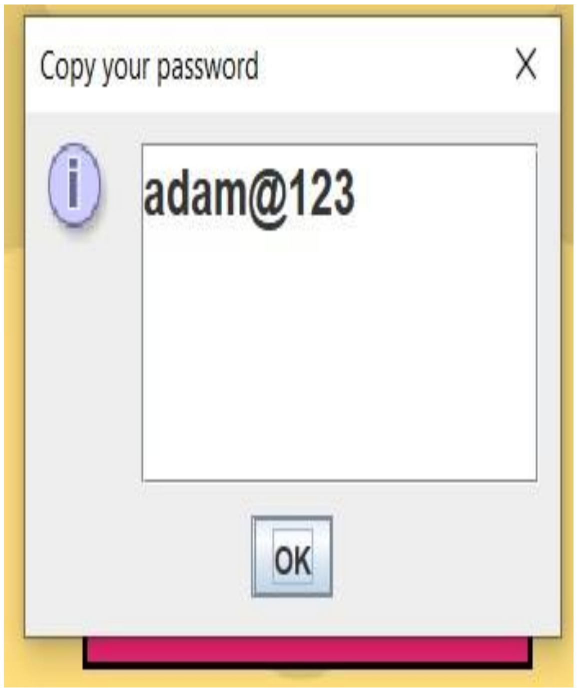
Account added Successfully !

- **Retrieving your Stored Password :**


Input

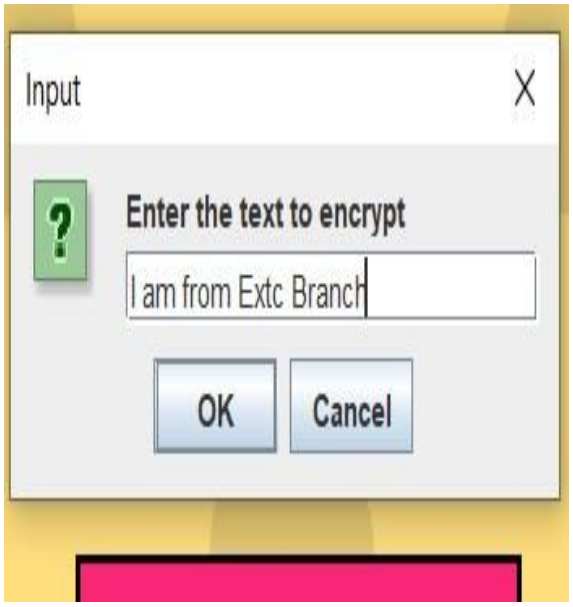
Enter your Account Name

Adam


Copy your password

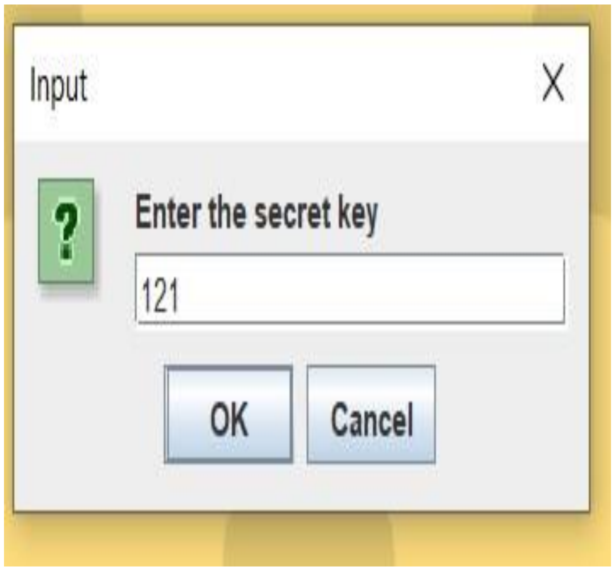
adam@123

- **Entering Text to Encrypt and its Secret Key:**


Input

Enter the text to encrypt

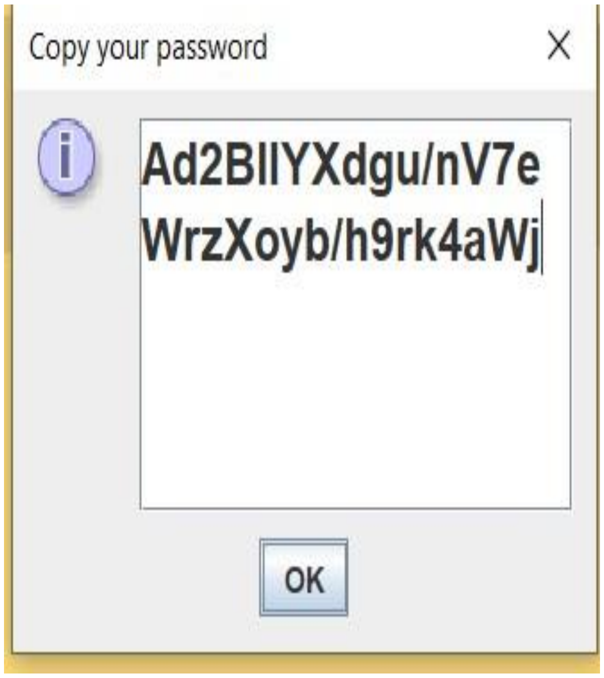
I am from Extc Branch


Input

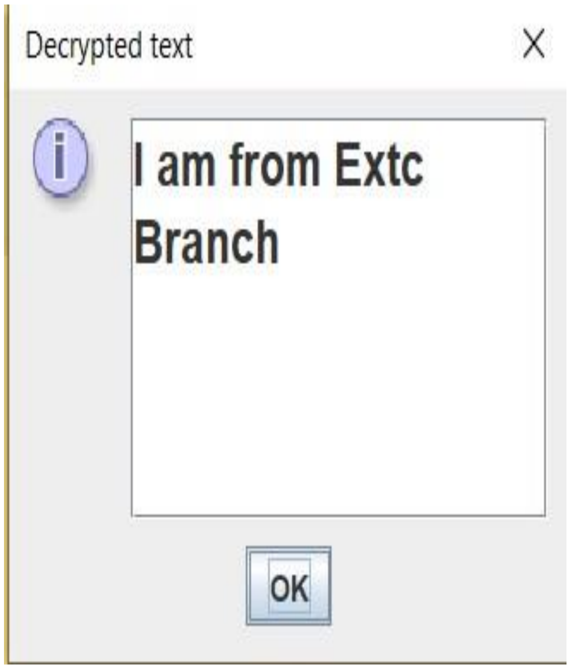
Enter the secret key

121

- **Getting the Encrypted Text and from that again the original Text :**


Copy your password

```
Ad2BIlYXdgu/nV7e
WrzXoyb/h9rk4aWj
```


Decrypted text

I am from Extc
Branch

➢ **<u>Conclusion</u>:**

- Implemented robust password generation functionality with a minimum length of 4 characters, enhancing security measures.
- Successfully integrated password storage and retrieval features, providing users with a seamless and organized experience.
- Enabled password deletion functionality, offering users control and flexibility over their stored credentials.
- Implemented strong encryption and decryption mechanisms using a secret key, ensuring the confidentiality of sensitive information.
- Utilized Java's Swing GUI toolkit to create a user-friendly interface, enhancing the overall user experience.
- Leveraged the VS Code development kit for efficient and streamlined project development.
- Integrated a note-taking feature, allowing users to add and view notes alongside their passwords for better organization.
- Overall, the project successfully combines security, usability, and functionality through Java, Swing, and VS Code, providing a comprehensive solution for password management and note-taking needs.