Password Generator Application Documentation

Table of Contents

Introduction

The Password Generator Application is a Java program designed to create strong, complex passwords and store them in an encrypted form. This application aims to enhance security by ensuring users have unique, hard-to-guess passwords for their various accounts. It uses Java Cryptography Architecture (JCA) to manage encryption and decryption of passwords.

Design Choices

1. Security

Given the importance of security in handling passwords, the application employs strong encryption techniques to protect stored passwords. The JCA provides the necessary tools for this purpose.

2. Usability

The application is designed with a user-friendly text-based interface, allowing users to generate and manage passwords easily. This choice ensures that users with basic computer skills can still use the application effectively.

3. Modularity

An object-oriented approach is used to design the application, making it modular and easy to maintain. Separate classes are responsible for password generation, management, and cryptography.

Implementation Details

The application consists of three main components: the main class (`PasswordGeneratorApp`), the `PasswordGenerator` class, and the `PasswordManager` class.

Main Class

The `PasswordGeneratorApp` class contains the main method, which serves as the entry point of the application. It handles user input and coordinates actions between the `PasswordGenerator` and `PasswordManager` classes.

```
import java.util.Scanner;



public class PasswordGeneratorApp {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        PasswordManager passwordManager = new PasswordManager();



        while (true) {

            System.out.println("Password Generator Application");

            System.out.println("1. Generate Password");

            System.out.println("2. Save Password");

            System.out.println("3. Retrieve Password");
            System.out.println("4. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter desired password length: ");
                    int length = scanner.nextInt();
                    scanner.nextLine(); // Consume newline
                    String password = PasswordGenerator.generatePassword(length);
                    System.out.println("Generated Password: " + password);
                    break;
                case 2:
                    System.out.print("Enter account name: ");
                    String account = scanner.nextLine();
                    System.out.print("Enter password: ");
```

```
                    String pass = scanner.nextLine();
                    passwordManager.savePassword(account, pass);
                    break;
                case 3:
                    System.out.print("Enter account name: ");
                    String acc = scanner.nextLine();
                    String retrievedPassword =
passwordManager.retrievePassword(acc);
                    System.out.println("Retrieved Password: " + retrievedPassword);
                    break;
                case 4:
                    System.out.println("Exiting application.");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
```

```
}
```

PasswordGenerator Class

The `PasswordGenerator` class is responsible for creating strong, random passwords based on the specified length.

```
import java.security.SecureRandom;



public class PasswordGenerator {

    private static final String CHARACTERS =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz01234
56789!@#$%^&*()-_=+";

    private static final SecureRandom RANDOM = new SecureRandom();



    public static String generatePassword(int length) {

        StringBuilder password = new StringBuilder(length);

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


password.append(CHARACTERS.charAt(RANDOM.nextInt(CHARACTERS.l
ength())));

        }

        return password.toString();

    }
}
```

PasswordManager Class

The `PasswordManager` class handles saving and retrieving encrypted passwords. It uses the JCA for encryption and decryption.

```
import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import java.util.Base64;

import java.util.HashMap;


public class PasswordManager {

    private static final String ALGORITHM = "AES";

    private static SecretKey secretKey;

    private HashMap<String, String> passwordStorage = new HashMap<>();


    static {

        try {

            KeyGenerator keyGen = KeyGenerator.getInstance(ALGORITHM);

            keyGen.init(256);

            secretKey = keyGen.generateKey();

        } catch (Exception e) {

            e.printStackTrace();
        }
    }

    public void savePassword(String account, String password) {
        try {
            Cipher cipher = Cipher.getInstance(ALGORITHM);
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            String encryptedPassword =
Base64.getEncoder().encodeToString(cipher.doFinal(password.getBytes()));
            passwordStorage.put(account, encryptedPassword);
            System.out.println("Password saved successfully.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public String retrievePassword(String account) {
        try {
            if (passwordStorage.containsKey(account)) {
                String encryptedPassword = passwordStorage.get(account);
                Cipher cipher = Cipher.getInstance(ALGORITHM);
                cipher.init(Cipher.DECRYPT_MODE, secretKey);
                return new
String(cipher.doFinal(Base64.getDecoder().decode(encryptedPassword)));
```

```
            } else {
                return "Account not found.";
            }
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Cryptography

Encryption and Decryption

To ensure the security of stored passwords, the application uses AES (Advanced Encryption Standard) for encryption and decryption. AES is a symmetric encryption algorithm, meaning the same key is used for both encryption and decryption. This key is generated using `KeyGenerator` from the JCA.

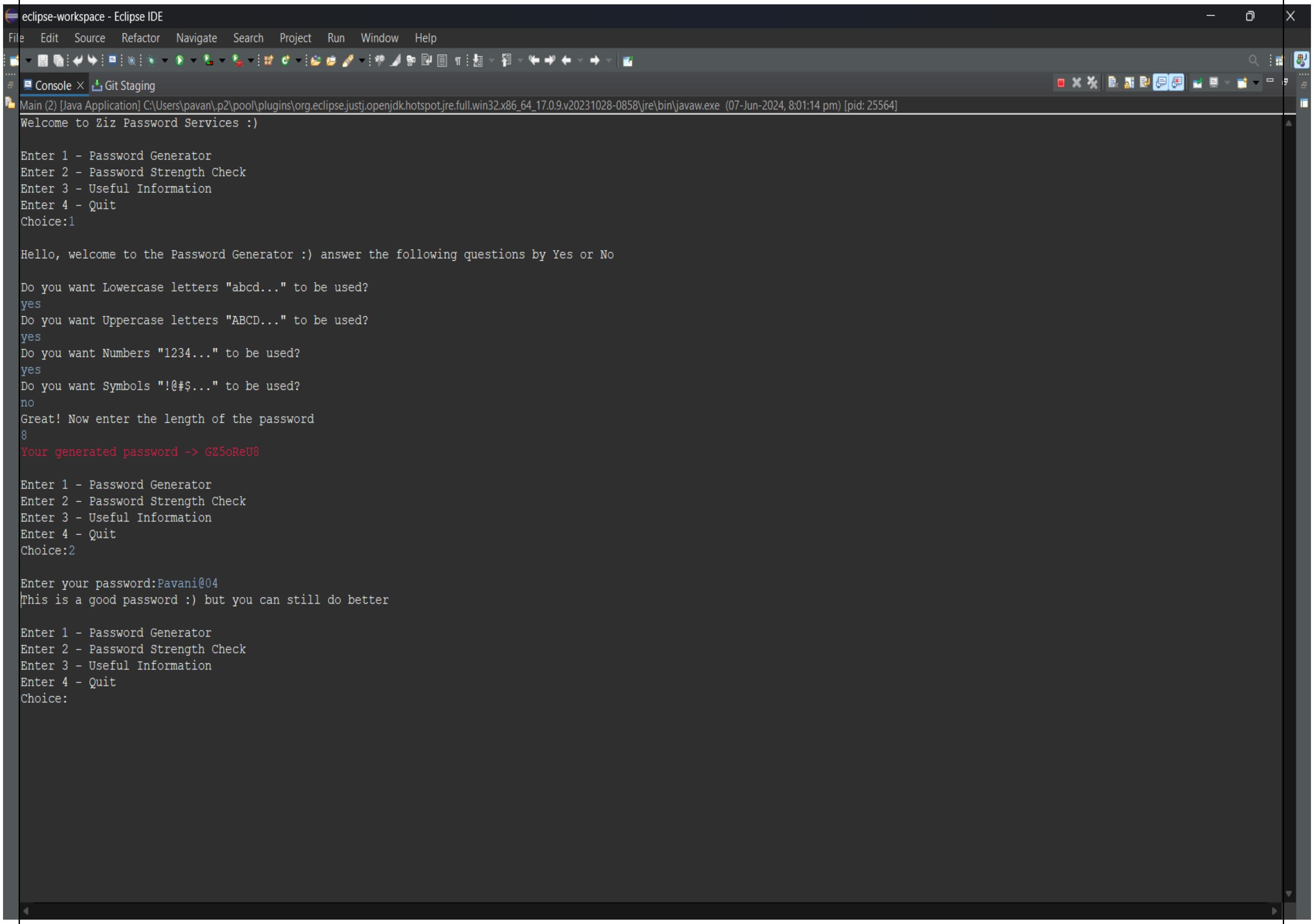
Java Cryptography Architecture

The Java Cryptography Architecture (JCA) is a framework for working with cryptographic operations in Java. It abstracts the implementation details of different cryptographic algorithms and provides a standard interface for developers. In this application, JCA is used to generate keys, encrypt passwords, and decrypt them when needed.

User Interface

The user interface is text-based and interacts with the user via the console. The user is presented with a menu of options to generate passwords, save them, and retrieve them. Each option is selected by entering a corresponding number.

Output:


eclipse-workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Console
Git Staging

```
Main (2) [Java Application] C:\Users\pavan\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe  (07-Jun-2024, 8:01:14 pm) [pid: 25564]
Welcome to Ziz Password Services :)

Enter 1 - Password Generator
Enter 2 - Password Strength Check
Enter 3 - Useful Information
Enter 4 - Quit
Choice:1

Hello, welcome to the Password Generator :) answer the following questions by Yes or No

Do you want Lowercase letters "abcd..." to be used?
yes
Do you want Uppercase letters "ABCD..." to be used?
yes
Do you want Numbers "1234..." to be used?
yes
Do you want Symbols "!@#$..." to be used?
no
Great! Now enter the length of the password
8
Your generated password -> GZ5oReU8

Enter 1 - Password Generator
Enter 2 - Password Strength Check
Enter 3 - Useful Information
Enter 4 - Quit
Choice:2

Enter your password:Pavani@04
This is a good password :) but you can still do better

Enter 1 - Password Generator
Enter 2 - Password Strength Check
Enter 3 - Useful Information
Enter 4 - Quit
Choice:
```

Challenges Faced

1. Cryptography Implementation

Implementing encryption and decryption using the JCA required careful handling of cryptographic keys and ensuring compatibility with the chosen encryption algorithm (AES). Managing the encryption key securely was crucial to maintaining the integrity of the stored passwords.

2. Secure Randomness

Generating secure random passwords necessitated the use of `SecureRandom` instead of the standard `Random` class. `SecureRandom` provides a cryptographically strong random number generator, ensuring the generated passwords are more resistant to prediction.

3. User Input Handling

Handling user input and providing meaningful feedback for invalid inputs was essential. This was managed using a `Scanner` object to read inputs and a switch statement to manage different user choices.

Conclusion

The Password Generator Application provides a secure and user-friendly solution for generating and managing passwords. It leverages the Java Cryptography Architecture to ensure the security of stored passwords, using AES encryption. The text-based interface makes it accessible to a wide range of users. Future enhancements could include features such as password strength validation, integration with external storage solutions, and a graphical user interface for improved usability. The challenges faced during implementation provided valuable insights into cryptography and secure software development practices.