

Writing an HTML filter that, given an arbitrary HTML document, produces an HTML document that will not result in the execution of script if loaded into a user's browser, but leaves "basic markup" (fonts, formatting, etc.) intact. Considering the possibility that the input document is not well-formed HTML, and also browser-specific features.

```
public static void main(String[] args) {
    try {
        BufferedReader reader = new BufferedReader(new FileReader(fileName:"sample.html"));
        BufferedWriter writer = new BufferedWriter(new FileWriter(fileName:"filtered.html"));

        String line;
        boolean insideScript = false;
        while ((line = reader.readLine()) != null) {
            if (line.contains(s:"<script") {
                insideScript = true;
            } else if (line.contains(s:"</script>")) {
                insideScript = false;
            } else if (!insideScript) {
                writer.write(line.replaceAll(regex:"<script.*?>|</script>", replacement:""));
                writer.newLine();
            }
        }

        reader.close();
        writer.close();

        System.out.println(x:"Script content and tags removed successfully.");
    }
}
```

RemoveScriptContent, reads an HTML file named "sample.html", removes any script content and script tags from it, and writes the filtered content to a new file named "filtered.html". Here's a breakdown of the code:

1. Import necessary classes:

- java.io.*: This imports the entire java.io package, which contains classes for input and output operations.

2. Define the main class RemoveScriptContent:

- This class contains the main method, which serves as the entry point for the program.

3. Inside the main method:

- Create a BufferedReader named reader to read from "sample.html".
- Create a BufferedWriter named writer to write to "filtered.html".
- Initialize a boolean variable insideScript to keep track of whether the program is inside a script tag.

4. Read each line from "sample.html" using a while loop:

- If the line contains <script, set insideScript to true.

- If the line contains </script>, set insideScript to false.

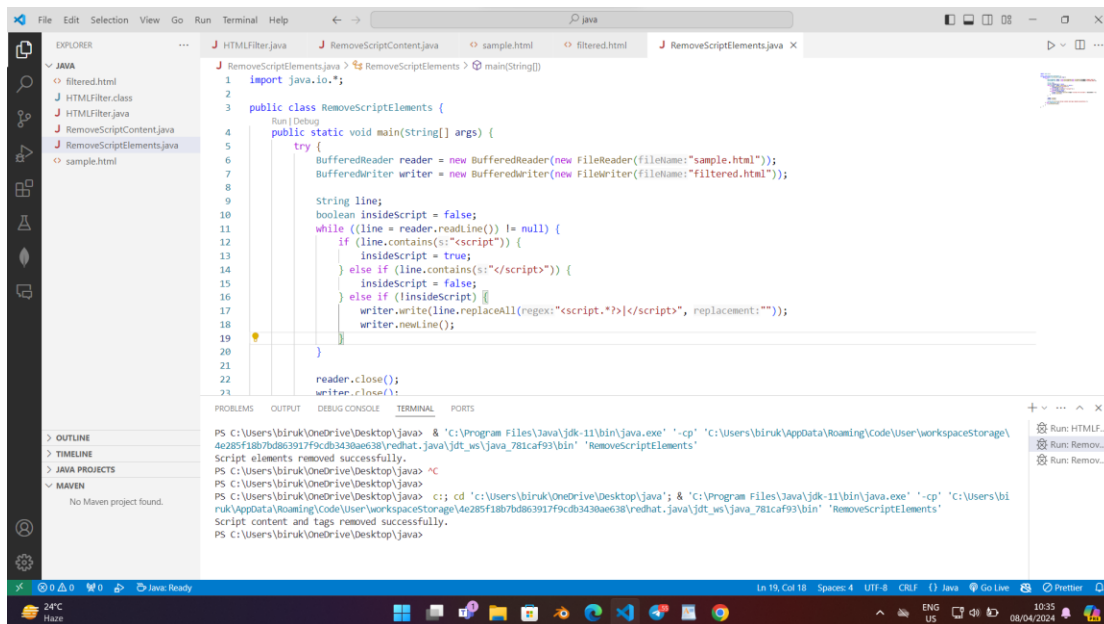
- If not inside a script tag, write the line to "filtered.html" after removing any script content or script tags using replaceAll.

5. Close the reader and writer to release resources.

6. Handle any IOException that may occur during file operations by printing the stack trace.

7. Print a success message if script content and tags are removed successfully.

Overall, this program effectively removes script content and script tags from an HTML file by filtering out lines containing script tags and content before writing the filtered content to a new file.



```
1 import java.io.*;
2
3 public class RemoveScriptElements {
4     public static void main(String[] args) {
5         try {
6             BufferedReader reader = new BufferedReader(new FileReader("sample.html"));
7             BufferedWriter writer = new BufferedWriter(new FileWriter("filtered.html"));
8
9             String line;
10            boolean insideScript = false;
11            while ((line = reader.readLine()) != null) {
12                if (line.contains("<script")) {
13                    insideScript = true;
14                } else if (line.contains("</script>")) {
15                    insideScript = false;
16                } else if (!insideScript) {
17                    writer.write(line.replaceAll(regex: "<script.*></script>", replacement: ""));
18                    writer.newLine();
19                }
20            }
21            reader.close();
22            writer.close();
23        } catch (IOException e) {
24            e.printStackTrace();
25        }
26    }
27 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\biruk\OneDrive\Desktop\java> & 'C:\Program Files\Java\jdk-11\bin\java.exe' '-cp' 'C:\Users\biruk\AppData\Roaming\Code\User\workspacestorage\4e285f18b7bd863917f9c3d3430ae638\redhat.java\jdt_ws\java_781caf93\bin' 'RemoveScriptElements'
Script elements removed successfully.
PS C:\Users\biruk\OneDrive\Desktop\java> cd 'C:\Users\biruk\OneDrive\Desktop\java' & cd 'C:\Users\biruk\OneDrive\Desktop\java' & 'C:\Program Files\Java\jdk-11\bin\java.exe' '-cp' 'C:\Users\biruk\AppData\Roaming\Code\User\workspacestorage\4e285f18b7bd863917f9c3d3430ae638\redhat.java\jdt_ws\java_781caf93\bin' 'RemoveScriptElements'
Script content and tags removed successfully.
PS C:\Users\biruk\OneDrive\Desktop\java>
```

Fig1 java implementation

Here the java code is run and the program executed successfully. below we examined the code that has been optimized so that no script element is uploaded to site there fore it got filtered.

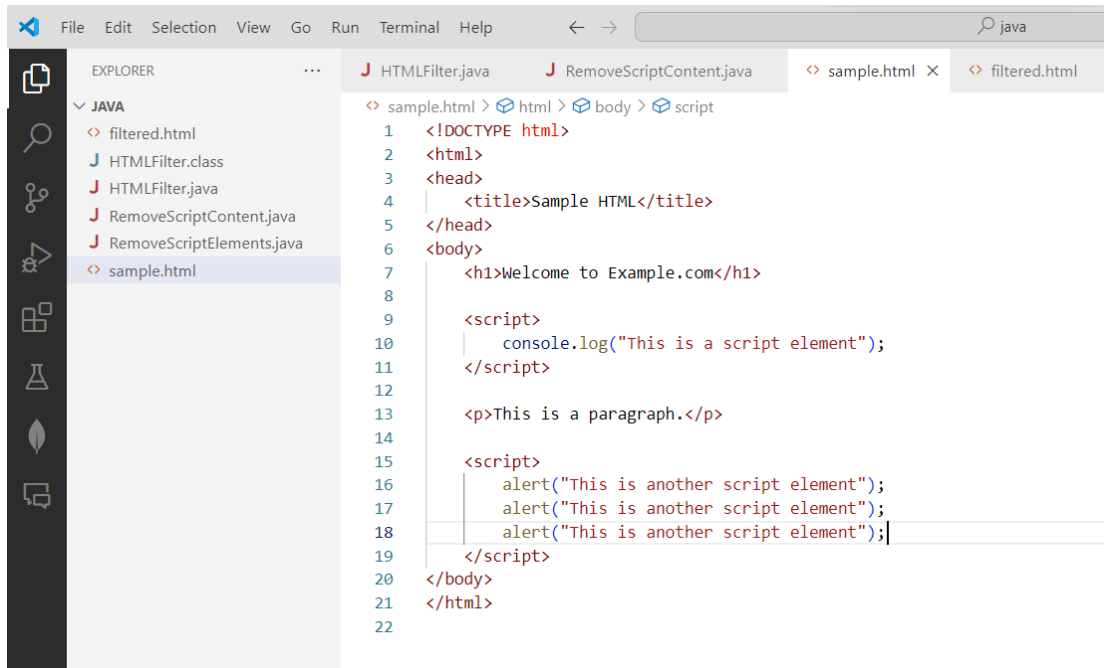


fig2 sample html with scripts

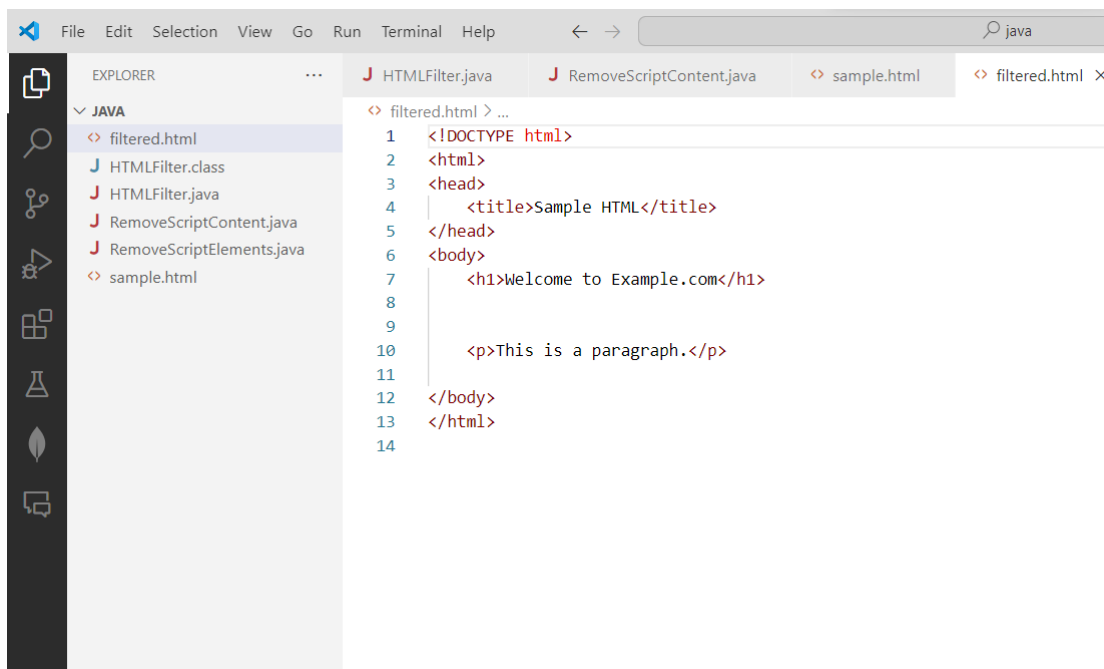


fig3 filtered html without scripts

java code below

```
import java.io.*;
```

```
public class RemoveScriptElements {  
    public static void main(String[] args) {  
        try {  
            BufferedReader reader = new BufferedReader(new FileReader("sample.html"));  
            BufferedWriter writer = new BufferedWriter(new FileWriter("filtered.html"));  
  
            String line;  
            boolean insideScript = false;  
            while ((line = reader.readLine()) != null) {  
                if (line.contains("<script")) {  
                    insideScript = true;  
                } else if (line.contains("</script>")) {  
                    insideScript = false;  
                } else if (!insideScript) {  
                    writer.write(line.replaceAll("<script.*?>|</script>", ""));  
                    writer.newLine();  
                }  
            }  
  
            reader.close();  
            writer.close();  
  
            System.out.println("Script content and tags removed successfully.");  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

programming problem

Implement HTTP digest authorization. Use a password file with salts, and reuse the BasicAuthWebServer from Chapter 9. Implement a program that allows you to add and delete passwords to and from the password file.

Implement HTTP digest authorization.

HTTP Digest Authentication is a method by which a client and a server authenticate each other to establish credentials without sending the actual password over the network. It involves a challenge-response mechanism where the server sends a nonce (a server-generated value), and the client responds with a hash of the username, password, and other information. This hash is calculated using the MD5 algorithm.

When Authenticated:

User Input: The user enters their username and password.

Authentication Check: The program checks if the provided username exists in the users map and if the associated password matches the provided password.

```
// Add some sample users with passwords
users.put(key:"Biruk", value:"Bir123");
users.put(key:"Arsema", value:"Ars123");
```

Nonce Generation: If the authentication is successful, the program generates a nonce (a random value) using generateNonce() method.

```
private static String generateNonce() {
    SecureRandom random = new SecureRandom();
    byte[] nonceBytes = new byte[16];
    random.nextBytes(nonceBytes);
    return Base64.getEncoder().encodeToString(nonceBytes);
}
```

Nonce Transmission: The generated nonce is then printed to simulate transmission to the client for use in subsequent steps of the HTTP Digest Authentication process.

Response Hash Calculation: The program computes the response hash using the username, password, and nonce. It simulates this process by calling the computeResponseHash() method.

```

private static String computeResponseHash(String username, String password, String nonce) {
    try {
        MessageDigest md = MessageDigest.getInstance(algorithm:"MD5");
        String hashInput = username + ":" + "Digest:" + password + ":" + nonce;
        byte[] hashedBytes = md.digest(hashInput.getBytes());
        return Base64.getEncoder().encodeToString(hashedBytes);
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        return null;
    }
}

```

Response Hash Transmission: The computed response hash is printed to simulate sending it to the server for verification.

Complete code compiled view

The screenshot shows an IDE with the following code in `HTTPDigestAuthenticator.java`:

```

1  import java.security.MessageDigest;
2  import java.security.NoSuchAlgorithmException;
3  import java.security.SecureRandom;
4  import java.util.Base64;
5  import java.util.HashMap;
6  import java.util.Map;
7  import java.util.Scanner;
8
9  public class HTTPDigestAuthenticator {
10     private static final Map<String, String> users = new HashMap<>();
11     private static final Map<String, String> nonces = new HashMap<>();
12
13     public static void main(String[] args) {
14         // Add some sample users with passwords
15         users.put(key:"Birik", value:"B1r123");
16         users.put(key:"Arsena", value:"Ars123");
17
18         Scanner scanner = new Scanner(System.in);
19         while (true) {
20             System.out.println("1. Authenticate");
21             System.out.println("2. Exit");
22             System.out.print("Choose an option: ");
23             int choice = scanner.nextInt();
24             scanner.nextLine(); // Consume newline

```

The terminal output shows the following execution:

```

PS C:\Users\Birik\OneDrive\Desktop> java -cp "C:\Program Files\Java\jdk-11\bin\java.exe" -cp "C:\Users\Birik\AppData\Roaming\Code\User\workspaceStorage\44c9432ed734a596eece23f6b7e3da\redhat.java\jdk_vs_java2_8b7942ff\bin" HTTPDigestAuthenticator
1. Authenticate
2. Exit
Choose an option: 1
Enter username: Arsena
Enter password: Ars123
Nonce: v5LHNCa8xfPfeCePlbBPQ==
Response Hash: MUP0IC+bVco3hyIL9eUPA==
Authentication successful!

```

When Not Authenticated:

User Input: The user enters their username and password.

```

1. Authenticate
2. Exit
Choose an option: 1
Enter username: Amha
Enter password: Amh321

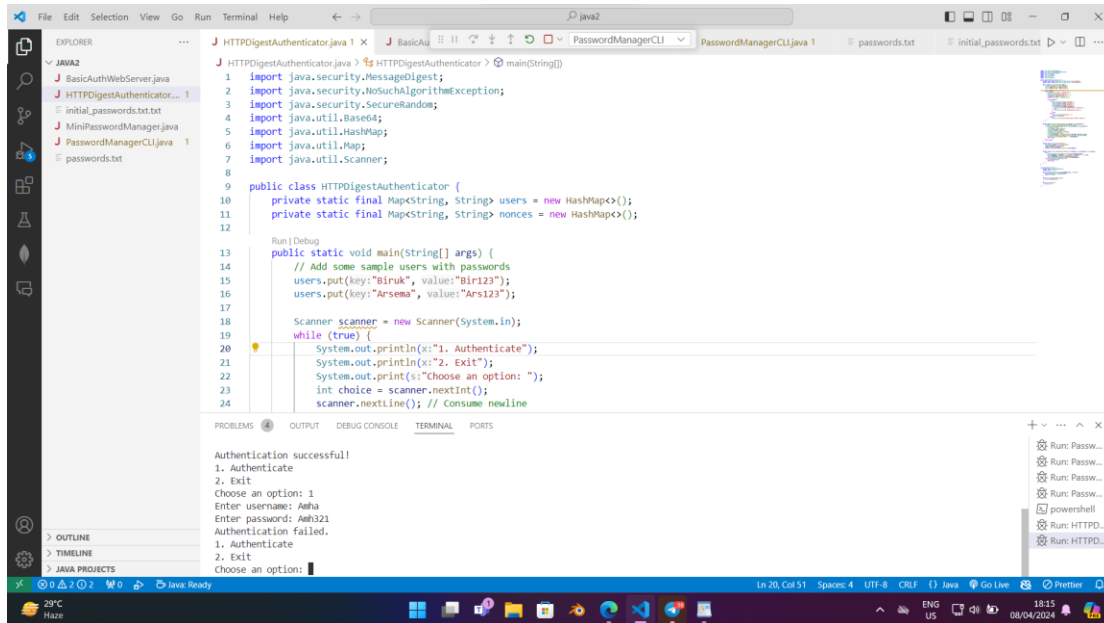
```

Authentication Check: The program checks if the provided username exists in the users map and if the associated password matches the provided password.

Failure Message: If the authentication fails, the program prints "Authentication failed."

```
1. Authenticate
2. Exit
Choose an option: 1
Enter username: Amha
Enter password: Amh321
Authentication failed.
```

complete compiled view



Implementation java code

```
import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.security.SecureRandom;

import java.util.Base64;

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class HTTPDigestAuthenticator {

    private static final Map<String, String> users = new HashMap<>();

    private static final Map<String, String> nonces = new HashMap<>();

    public static void main(String[] args) {

        // Add some sample users with passwords
```

```

users.put("Biruk", "Bir123");

users.put("Arsema", "Ars123");


Scanner scanner = new Scanner(System.in);

while (true) {

    System.out.println("1. Authenticate");

    System.out.println("2. Exit");

    System.out.print("Choose an option: ");

    int choice = scanner.nextInt();

    scanner.nextLine(); // Consume newline


    switch (choice) {

        case 1:

            System.out.print("Enter username: ");

            String username = scanner.nextLine();

            System.out.print("Enter password: ");

            String password = scanner.nextLine();

            if (authenticate(username, password)) {

                System.out.println("Authentication successful!");

            } else {

                System.out.println("Authentication failed.");

            }

            break;

        case 2:

            System.out.println("Exiting...");

            System.exit(0);

        default:

            System.out.println("Invalid option. Please try again.");

    }

}

}

private static boolean authenticate(String username, String password) {

```



```

// Check if the user exists and the password is correct
if (users.containsKey(username) && users.get(username).equals(password)) {

    // Generate a nonce

    String nonce = generateNonce();

    nonces.put(username, nonce);

    // Send the nonce to the client

    System.out.println("Nonce: " + nonce);

    // Compute the response hash

    String responseHash = computeResponseHash(username, password, nonce);

    // Simulate sending the response hash to the server for verification

    System.out.println("Response Hash: " + responseHash);

    return true;
}

return false;
}

private static String generateNonce() {

    SecureRandom random = new SecureRandom();

    byte[] nonceBytes = new byte[16];

    random.nextBytes(nonceBytes);

    return Base64.getEncoder().encodeToString(nonceBytes);
}

private static String computeResponseHash(String username, String password, String nonce) {

    try {

        MessageDigest md = MessageDigest.getInstance("MD5");

        String hashInput = username + ":" + "Digest:" + password + ":" + nonce;

        byte[] hashedBytes = md.digest(hashInput.getBytes());

        return Base64.getEncoder().encodeToString(hashedBytes);

    } catch (NoSuchAlgorithmException e) {

        e.printStackTrace();

        return null;

    }
}

```

```

    }
}

class HashedPasswordTuple {
    private final String hashedPassword;
    private final int salt;

    public HashedPasswordTuple(String hashedPassword, int salt) {
        this.hashedPassword = hashedPassword;
        this.salt = salt;
    }

    public String getHashedPassword() {
        return hashedPassword;
    }

    public int getSalt() {
        return salt;
    }
}

```

Implement a program that allows you to add and delete passwords to and from the password file

Data Storage: The program stores usernames and hashed passwords in a `HashMap<String, String>` named `passwords`.

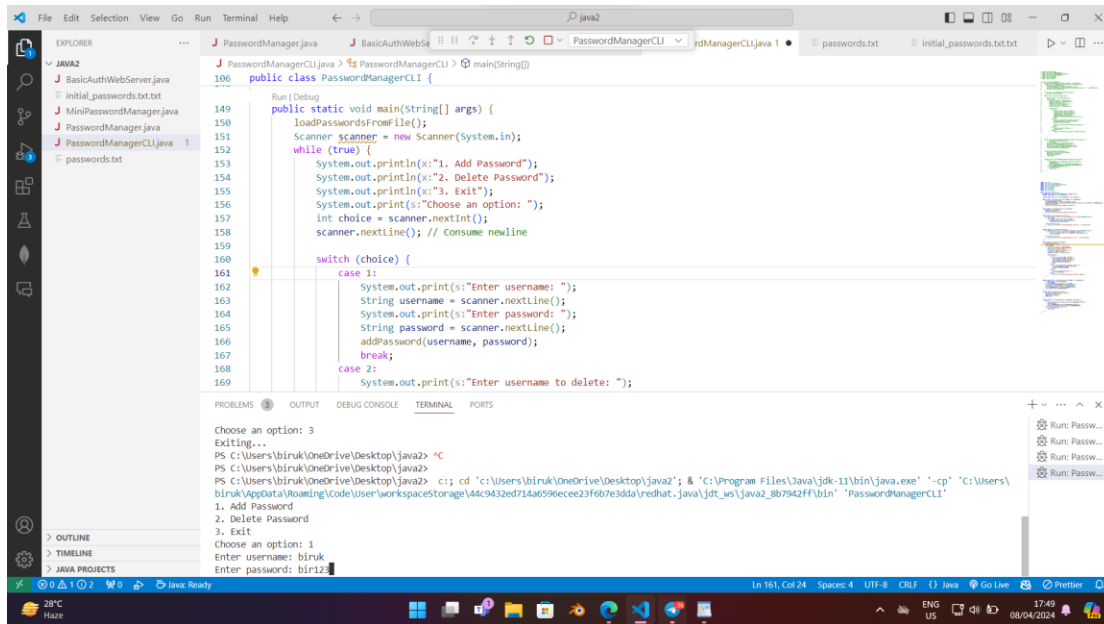
File Operations:

It defines two constants:

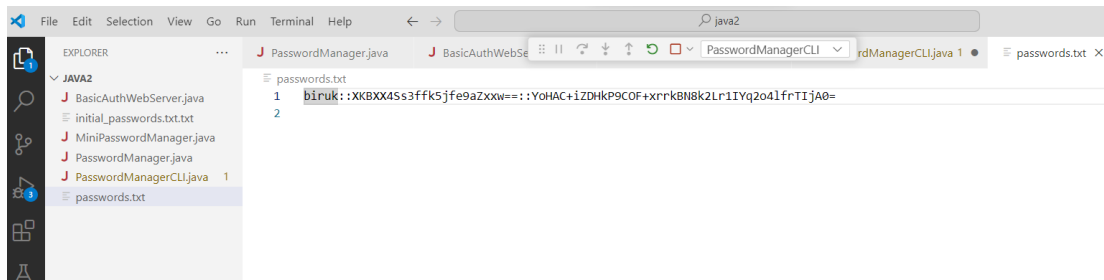
`PASSWORD_FILE`: Specifies the filename for storing passwords.

`SALT_SEPARATOR`: Specifies a separator used when storing the salt and hashed password together in the file.

It provides functions to load passwords from the file into the passwords map (`loadPasswordsFromFile()`) and to save passwords from the passwords map into the file (`savePasswordsToFile()`).



The `addPassword()` function allows users to add passwords for specific usernames. It generates a random salt, hashes the password with the salt using the SHA-256 algorithm, and stores the username along with the salt and hashed password in the passwords map. It then saves the updated passwords to the file.



This shows that the addition with the salt and hash is confirmed!

```
public class PasswordManagerCLI {  
    public static void main(String[] args) {  
        loadPasswordsFromFile();  
        Scanner scanner = new Scanner(System.in);  
        while (true) {  
            System.out.println("1. Add Password");  
            System.out.println("2. Delete Password");  
            System.out.println("3. Exit");  
            System.out.print("Choose an option: ");  
            int choice = scanner.nextInt();  
            scanner.nextLine(); // Consume newline  
  
            switch (choice) {  
                case 1:  
                    System.out.print("Enter username: ");  
                    String username = scanner.nextLine();  
                    System.out.print("Enter password: ");  
                    String password = scanner.nextLine();  
                    addPassword(username, password);  
                    break;  
                case 2:  
                    System.out.print("Enter username to delete: ");  
                    String usernameToDelete = scanner.nextLine();  
                    deletePassword(usernameToDelete);  
                    break;  
                case 3:  
                    System.out.println("Exiting...");  
                    return;  
            }  
        }  
    }  
}
```

Enter password: bir123
Password added successfully.
1. Add Password
2. Delete Password
3. Exit
Choose an option: 2
Enter username to delete: biruk
Password deleted successfully.
1. Add Password
2. Delete Password
3. Exit
Choose an option:

The deletePassword() function allows users to delete passwords for specific usernames. It removes the entry for the given username from the passwords map and saves the updated passwords to the file.

```
1
```

Enter password: bir123
Password added successfully.
1. Add Password
2. Delete Password
3. Exit
Choose an option: 2
Enter username to delete: biruk
Password deleted successfully.
1. Add Password
2. Delete Password
3. Exit
Choose an option:

this shows that the deletion is confirmed!

```

public static boolean checkPassword(String username, String password) {
    if (!passwords.containsKey(username))
        return false;
    String storedPassword = passwords.get(username);
    String[] parts = storedPassword.split(SALT_SEPARATOR);
    byte[] salt = Base64.getDecoder().decode(parts[0]);
    String hashedPassword = hashPassword(password, salt);
    return parts[1].equals(hashedPassword);
}

private static byte[] generateSalt() {
    SecureRandom random = new SecureRandom();
    byte[] salt = new byte[16];
    random.nextBytes(salt);
    return salt;
}

private static String hashPassword(String password, byte[] salt) {
    try {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(salt);
        byte[] hashedPassword = md.digest(password.getBytes());
        return Base64.getEncoder().encodeToString(hashedPassword);
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        return null;
    }
}

```

The `checkPassword()` function is provided to verify if a provided password matches the stored password for a given username. It retrieves the stored hashed password and salt from the passwords map, hashes the provided password with the salt, and compares it with the stored hashed password.

Passwords are hashed using the SHA-256 algorithm with a randomly generated salt, which enhances security by making it computationally expensive to reverse-engineer the original passwords.

Java implementation of code

```

import java.io.*;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.security.SecureRandom;

import java.util.Base64;

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

```

```

public class PasswordManagerCLI {

    private static final String PASSWORD_FILE = "passwords.txt";

    private static final String SALT_SEPARATOR = "::";

    private static Map<String, String> passwords = new HashMap<>();

    // Function to add a password for a given username
    public static void addPassword(String username, String password) {

        // Generate a salt
        byte[] salt = generateSalt();

        // Hash the password with the salt
        String hashedPassword = hashPassword(password, salt);

        // Store the salt and hashed password in the passwords map
        passwords.put(username, Base64.getEncoder().encodeToString(salt) + SALT_SEPARATOR + hashedPassword);

        // Save the passwords to file
        savePasswordsToFile();

        System.out.println("Password added successfully.");
    }

    // Function to delete a password for a given username
    public static void deletePassword(String username) {

        // Remove the password entry for the given username
        passwords.remove(username);

        // Save the passwords to file
        savePasswordsToFile();

        System.out.println("Password deleted successfully.");
    }

    // Function to load passwords from the file into the passwords map
    private static void loadPasswordsFromFile() {

        try (BufferedReader reader = new BufferedReader(new FileReader(PASSWORD_FILE))) {

            String line;

```

```

        while ((line = reader.readLine()) != null) {

            String[] parts = line.split(SALT_SEPARATOR);

            passwords.put(parts[0], parts[1]);

        }

    } catch (IOException e) {

        System.err.println("Error loading passwords from file: " + e.getMessage());

    }

}

// Function to save passwords from the passwords map into the file
private static void savePasswordsToFile() {

    try (BufferedWriter writer = new BufferedWriter(new FileWriter(PASSWORD_FILE))) {

        for (Map.Entry<String, String> entry : passwords.entrySet()) {

            writer.write(entry.getKey() + SALT_SEPARATOR + entry.getValue());

            writer.newLine();

        }

    } catch (IOException e) {

        System.err.println("Error saving passwords to file: " + e.getMessage());

    }

}

// Main function
public static void main(String[] args) {

    // Load passwords from file into the passwords map
    loadPasswordsFromFile();

    Scanner scanner = new Scanner(System.in);

    while (true) {

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

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

        System.out.println("3. Exit");

        System.out.print("Choose an option: ");

        int choice = scanner.nextInt();

        scanner.nextLine(); // Consume newline
    }
}

```

```

switch (choice) {
    case 1:
        System.out.print("Enter username: ");
        String username = scanner.nextLine();
        System.out.print("Enter password: ");
        String password = scanner.nextLine();
        addPassword(username, password);
        break;

    case 2:
        System.out.print("Enter username to delete: ");
        String userToDelete = scanner.nextLine();
        deletePassword(userToDelete);
        break;

    case 3:
        System.out.println("Exiting...");
        System.exit(0);

    default:
        System.out.println("Invalid option. Please try again.");
}
}
}

// Function to check if a provided password matches the stored password for a given username
public static boolean checkPassword(String username, String password) {
    if (!passwords.containsKey(username))
        return false;

    String storedPassword = passwords.get(username);
    String[] parts = storedPassword.split(SALT_SEPARATOR);
    byte[] salt = Base64.getDecoder().decode(parts[0]);
    String hashedPassword = hashPassword(password, salt);
    return parts[1].equals(hashedPassword);
}

```



```
// Function to generate a random salt

private static byte[] generateSalt() {

    SecureRandom random = new SecureRandom();

    byte[] salt = new byte[16];

    random.nextBytes(salt);

    return salt;

}


// Function to hash a password using SHA-256 algorithm and the provided salt

private static String hashPassword(String password, byte[] salt) {

    try {

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

        md.update(salt);

        byte[] hashedPassword = md.digest(password.getBytes());

        return Base64.getEncoder().encodeToString(hashedPassword);

    } catch (NoSuchAlgorithmException e) {

        e.printStackTrace();

        return null;

    }

}
```