Professor Nazareno de Oliveira Pacheco
Codificação Segura

Atividade 1: Implementação Básica de Autenticação com Hash e Salt

Objetivo

Desenvolver um sistema de autenticação simples que utilize hash e salt para proteger senhas de usuários. A atividade visa ensinar a implementação básica de técnicas de hashing e a aplicação de salt para melhorar a segurança das senhas armazenadas.

Descrição da Atividade

1. Configuração do Projeto

1. **Criação do Projeto**:
   - Crie um projeto Java no Eclipse ou IDE de sua escolha.
   - Adicione as dependências necessárias para o uso de hashing, se estiver utilizando uma biblioteca específica.
2. **Estrutura do Projeto**:
   - Crie duas classes principais:
     - `Autenticacao`: para gerenciar o registro e o login dos usuários.
     - `SenhaUtil`: para realizar operações de hashing e salt.

2. Implementação da Classe `SenhaUtil`

1. **Hashing com SHA-256 e Salt**:
   - Implemente métodos para gerar um salt aleatório.
   - Crie um método para gerar o hash da senha combinando o salt com a senha original usando o algoritmo SHA-256.

Código Exemplo:

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.Base64;

public class SenhaUtil {
```


```
        private static final String ALGORITMO_HASH = "SHA-256";
        private static final SecureRandom random = new SecureRandom();

        // Gera um salt aleatório
        public static String gerarSalt() {
        byte[] salt = new byte[16];
        random.nextBytes(salt);
        return Base64.getEncoder().encodeToString(salt);
        }

        // Gera o hash da senha usando o salt
        public static String gerarHash(String senha, String salt) throws
NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance(ALGORITMO_HASH);
        md.update(Base64.getDecoder().decode(salt));
        byte[] senhaHash = md.digest(senha.getBytes());
        return Base64.getEncoder().encodeToString(senhaHash);
        }
}
```

3. Implementação da Classe `Autenticacao`

1. **Registro e Login**:
    - Implemente a lógica para registrar um novo usuário, gerando e armazenando o hash e o salt.
    - Implemente a lógica de login para verificar se a senha fornecida corresponde ao hash armazenado.

Código Exemplo:

```
import java.security.NoSuchAlgorithmException;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class Autenticacao {

        private static Map<String, String[]> usuarios = new HashMap<>();

        public static void main(String[] args) throws NoSuchAlgorithmException {
        Scanner scanner = new Scanner(System.in);

        System.out.println("1. Registrar");
        System.out.println("2. Login");
```


```
        int escolha = scanner.nextInt();
        scanner.nextLine(); // Limpar o buffer

        if (escolha == 1) {
        System.out.print("Nome de usuário: ");
        String usuario = scanner.nextLine();
        System.out.print("Senha: ");
        String senha = scanner.nextLine();

        String salt = SenhaUtil.gerarSalt();
        String hash = SenhaUtil.gerarHash(senha, salt);

        usuarios.put(usuario, new String[]{hash, salt});
        System.out.println("Usuário registrado com sucesso!");

        } else if (escolha == 2) {
        System.out.print("Nome de usuário: ");
        String usuario = scanner.nextLine();
        System.out.print("Senha: ");
        String senha = scanner.nextLine();

        String[] dados = usuarios.get(usuario);

        if (dados != null) {
                String hashArmazenado = dados[0];
                String salt = dados[1];

                String hash = SenhaUtil.gerarHash(senha, salt);

                if (hash.equals(hashArmazenado)) {
                System.out.println("Login bem-sucedido!");
                } else {
                System.out.println("Senha incorreta!");
                }
        } else {
                System.out.println("Usuário não encontrado!");
        }
        }
        scanner.close();
        }
}
```

Atividade 2: Implementação Avançada com Proteção de Senhas e Análise de Segurança

Objetivo

Desenvolver uma aplicação que utiliza técnicas avançadas de hashing, como a aplicação de PBKDF2 (Password-Based Key Derivation Function 2) para proteger senhas. A atividade visa explorar técnicas de hashing mais seguras e analisar a segurança das senhas.

Descrição da Atividade

1. Configuração do Projeto

1. **Criação do Projeto**:
   - Crie um projeto Java no Eclipse ou IDE de sua escolha.
   - Adicione as dependências necessárias, se estiver usando bibliotecas específicas para PBKDF2.
2. **Estrutura do Projeto**:
   - Crie duas classes principais:
     - `AutenticacaoAvancada`: para gerenciar o registro e o login dos usuários.
     - `SenhaUtilAvancada`: para realizar operações de hashing com PBKDF2.

2. Implementação da Classe `SenhaUtilAvancada`

1. **Hashing com PBKDF2**:
   - Implemente métodos para gerar o hash da senha usando PBKDF2, que inclui um salt e um número de iterações.

Código Exemplo:

```
import java.security.NoSuchAlgorithmException;
import java.security.spec.KeySpec;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import java.security.SecureRandom;
import java.util.Base64;
```


```
public class SenhaUtilAvancada {

	private static final String ALGORITMO_HASH = "PBKDF2WithHmacSHA256";
	private static final int ITERACOES = 10000;
	private static final int Tamanho_Bytes = 256;

	// Gera o hash da senha usando PBKDF2
	public static String gerarHash(String senha, String salt) throws
NoSuchAlgorithmException {
	try {
	KeySpec spec = new PBEKeySpec(senha.toCharArray(),
Base64.getDecoder().decode(salt), ITERACOES, Tamanho_Bytes);
	SecretKeyFactory factory = SecretKeyFactory.getInstance(ALGORITMO_HASH);
	byte[] hash = factory.generateSecret(spec).getEncoded();
	return Base64.getEncoder().encodeToString(hash);
	} catch (Exception e) {
	throw new NoSuchAlgorithmException("Erro ao gerar hash");
	}
	}

	// Gera um salt aleatório
	public static String gerarSalt() {
	byte[] salt = new byte[16];
	new SecureRandom().nextBytes(salt);
	return Base64.getEncoder().encodeToString(salt);
	}
}
```

3. Implementação da Classe `AutenticacaoAvancada`

1. **Registro e Login**:
   - Implemente a lógica para registrar um novo usuário, gerando e armazenando o hash e o salt usando PBKDF2.
   - Implemente a lógica de login para verificar se a senha fornecida corresponde ao hash armazenado.

```
import java.security.NoSuchAlgorithmException;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class AutenticacaoAvancada {
```


```
private static Map<String, String[]> usuarios = new HashMap<>();

public static void main(String[] args) throws NoSuchAlgorithmException {
Scanner scanner = new Scanner(System.in);

System.out.println("1. Registrar");
System.out.println("2. Login");
int escolha = scanner.nextInt();
scanner.nextLine(); // Limpar o buffer

if (escolha == 1) {
System.out.print("Nome de usuário: ");
String usuario = scanner.nextLine();
System.out.print("Senha: ");
String senha = scanner.nextLine();

String salt = SenhaUtilAvancada.gerarSalt();
String hash = SenhaUtilAvancada.gerarHash(senha, salt);

usuarios.put(usuario, new String[]{hash, salt});
System.out.println("Usuário registrado com sucesso!");

} else if (escolha == 2) {
System.out.print("Nome de usuário: ");
String usuario = scanner.nextLine();
System.out.print("Senha: ");
String senha = scanner.nextLine();

String[] dados = usuarios.get(usuario);

if (dados != null) {
        String hashArmazenado = dados[0];
        String salt = dados[1];

        String hash = SenhaUtilAvancada.gerarHash(senha, salt);

        if (hash.equals(hashArmazenado)) {
        System.out.println("Login bem-sucedido!");
        } else {
        System.out.println("Senha incorreta!");
        }
} else {
        System.out.println("Usuário não encontrado!");
}
}
scanner.close();
}
```

```
}
```

Orientações:

1. **Trabalho Individual ou em Dupla**:
   - A atividade pode ser realizada individualmente ou em duplas, conforme a orientação do instrutor.
2. **Documentação e Comentários**:
   - Documente o código com comentários explicativos para facilitar a compreensão.
   - Inclua um README descrevendo como executar o código e os resultados esperados.
3. **Submissão**:
   - Submeta o código-fonte, os arquivos gerados (se aplicável), e uma breve descrição das implementações e resultados obtidos.