Brilliant Solutions, Inc. – Test Solutions (Java)

Date: 7 Apr 25
Author: Alexander La Barge
Company: Brilliant Solutions, Inc.

Overview

Below are sample Java solutions for two coding challenges:

1. **FileEncryptor** – Encrypts a given file (AES/GCM).
2. **FileStatsReader** – Reads a text file and calculates line count, word count, and average word length.

Each block includes a top doc comment describing the file, author, and date, followed by the Java implementation code.

FileEncryptor.java

```java
/*
 * FileEncryptor.java
 * Author: Alexander La Barge
 * Date: 7 Apr 25
 * Company: Brilliant Solutions, Inc.
 *
 * SAMPLE SOLUTION for the File Encryption Utility Challenge
 */

import javax.crypto.Cipher;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.PBEKeySpec;
```

```java
import javax.crypto.spec.SecretKeySpec;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.SecureRandom;
import java.security.spec.KeySpec;

public class FileEncryptor {
    private static final int AES_KEY_SIZE = 128;       // 128-bit key
    private static final int GCM_TAG_LENGTH = 128;     // 128-bit auth tag
    private static final int SALT_SIZE = 16;           // 16 bytes
    private static final int IV_SIZE = 12;             // 12 bytes recommended
for GCM
    private static final int PBKDF2_ITERATIONS = 65536;

    public static void main(String[] args) {
        if (args.length < 3) {
            System.err.println("Usage: java FileEncryptor <inputFile>
<outputFile> <password>");
            return;
        }

        String inputFilePath = args[0];
        String outputFilePath = args[1];
        String password = args[2];

        File inputFile = new File(inputFilePath);
        if (!inputFile.exists()) {
            System.err.println("Input file does not exist: " + inputFilePath);
            return;
        }

        try {
            // Generate a random salt
            byte[] salt = new byte[SALT_SIZE];
            SecureRandom random = new SecureRandom();
            random.nextBytes(salt);

            // Derive key from password
            SecretKeySpec keySpec = deriveKeyFromPassword(password, salt);

            // Generate random IV for GCM
```

```
            byte[] iv = new byte[IV_SIZE];
            random.nextBytes(iv);

            // Encrypt file
            encryptFile(inputFilePath, outputFilePath, keySpec, iv, salt);
            System.out.println("Encryption successful. Encrypted file: " +
outputFilePath);

        } catch (Exception e) {
            System.err.println("Error encrypting file: " + e.getMessage());
        }
    }

    private static SecretKeySpec deriveKeyFromPassword(String password, byte[]
salt) throws Exception {
        KeySpec spec = new PBEKeySpec(password.toCharArray(), salt,
PBKDF2_ITERATIONS, AES_KEY_SIZE);
        SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
        byte[] keyBytes = factory.generateSecret(spec).getEncoded();
        return new SecretKeySpec(keyBytes, "AES");
    }

    private static void encryptFile(String inputFilePath, String outputFilePath,
                                    SecretKeySpec keySpec, byte[] iv, byte[]
salt) throws Exception {

        Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
        GCMParameterSpec gcmSpec = new GCMParameterSpec(GCM_TAG_LENGTH, iv);
        cipher.init(Cipher.ENCRYPT_MODE, keySpec, gcmSpec);

        try (FileInputStream fis = new FileInputStream(inputFilePath);
             FileOutputStream fos = new FileOutputStream(outputFilePath)) {

            // Write salt + IV at the start of the output for future decryption
            fos.write(salt);
            fos.write(iv);

            byte[] buffer = new byte[4096];
            int bytesRead;
            while ((bytesRead = fis.read(buffer)) != -1) {
                byte[] encrypted = cipher.update(buffer, 0, bytesRead);
                if (encrypted != null) {
```

```
                    fos.write(encrypted);
                }
            }
            // Finalize encryption
            byte[] finalBytes = cipher.doFinal();
            if (finalBytes != null) {
                fos.write(finalBytes);
            }
        } catch (IOException e) {
            throw new IOException("I/O error: " + e.getMessage(), e);
        }
    }
}
```

Key Points:

- **AES/GCM** is used, ensuring authenticated encryption.
- **Salt** (16 bytes) and **IV** (12 bytes) are written at the start of the file to allow future decryption with the same password.
- Uses **PBKDF2** with HMAC-SHA256 to derive the AES key from the password and salt.

FileStatsReader.java

```java
/*
 * FileStatsReader.java
 * Author: Alexander La Barge
 * Date: 7 Apr 25
 * Company: Brilliant Solutions, Inc.
 *
 * SAMPLE SOLUTION for the File Stats Reader Challenge
 */

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileStatsReader {
```

```
    public static void main(String[] args) {
        if (args.length < 1) {
            System.err.println("Usage: java FileStatsReader <filePath>");
            return;
        }

        String filePath = args[0];
        int lineCount = 0;
        int wordCount = 0;
        int totalWordLength = 0;

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            String line;
            while ((line = br.readLine()) != null) {
                lineCount++;
                // Split on whitespace
                String[] words = line.trim().split("\\s+");
                for (String w : words) {
                    if (!w.isEmpty()) {
                        wordCount++;
                        totalWordLength += w.length();
                    }
                }
            }
        } catch (IOException e) {
            System.err.println("File not found or cannot be read.");
            return;
        }

        System.out.println("Line Count: " + lineCount);
        System.out.println("Word Count: " + wordCount);

        if (wordCount > 0) {
            double averageWordLength = (double) totalWordLength / wordCount;
            // Round to 2 decimal places
            System.out.printf("Average Word Length: %.2f%n", averageWordLength);
        } else {
            System.out.println("Average Word Length: 0");
        }
    }
}
```

Key Points:

- Reads a text file line by line, counting total **lines**, **words**, and **word length**.
- Splits on whitespace using `split("\\s+")`.
- Prints line count, word count, and average word length (rounded to 2 decimals).

How to Compile & Run

1. **FileEncryptor**:
   - **Compile**: `javac FileEncryptor.java`
   - **Run**: `java FileEncryptor sample.txt encrypted.dat MySecretPassword`
2. **FileStatsReader**:
   - **Compile**: `javac FileStatsReader.java`
   - **Run**: `java FileStatsReader sample.txt`

Ensure you're in the same directory as each `.java` file when compiling and running.

Further Reading

- For **decryption**, see our companion **FileDecryptor** utility (AES/GCM).
- For instructions on **Java installation**, refer to Ubuntu 24.04 Setup or your system's official documentation.

If you have questions or need additional support, contact **Alexander La Barge** at **Brilliant Solutions, Inc.**

Happy coding!