Project Title: AES File Encryption and Decryption

Project Overview: The **AES File Encryption** project demonstrates how to securely encrypt and decrypt files using the AES (Advanced Encryption Standard) algorithm in **CBC (Cipher Block Chaining)** mode with **PKCS5Padding**. This project aims to showcase secure file handling using symmetric encryption, ensuring confidentiality and integrity of sensitive data stored in files.

Key Components of the Project:

1. **AES Algorithm with CBC Mode and PKCS5Padding:**
    - **AES (Advanced Encryption Standard):** A widely used symmetric encryption algorithm that ensures high-level data security.
    - **CBC (Cipher Block Chaining):** A mode of AES that improves encryption by ensuring that identical blocks of plaintext are encrypted into different ciphertexts, thus enhancing security.
    - **PKCS5Padding:** A padding scheme used to ensure that the length of the plaintext matches the block size required by the AES algorithm.
2. **Key Generation (SHA-256):**
    - A 256-bit key is derived using the **SHA-256** hash function. The password provided by the user is hashed using SHA-256 to generate a secure key. This key is used in both the encryption and decryption processes to ensure data confidentiality.
    - The key is then converted into a SecretKeySpec object, which is compatible with the AES encryption algorithm.
3. **Initialization Vector (IV) Generation:**
    - An **Initialization Vector (IV)** is generated randomly for each encryption session to ensure that even if the same data is encrypted multiple times, the output ciphertext is different each time.
    - The IV is stored along with the encrypted file to enable the decryption process to reconstruct the exact ciphertext.
4. **File Encryption Process:**
    - The file to be encrypted is read byte-by-byte, and the AES encryption algorithm is applied to each block of data.
    - The **IV** is written at the beginning of the encrypted file to facilitate decryption later.
    - The file is encrypted and saved in the specified output file.
5. **File Decryption Process:**
    - The encrypted file is read, and the IV is extracted from the start of the file.
    - The AES algorithm is initialized with the same secret key and the extracted IV.

- The ciphertext is decrypted, and the original file content is restored and saved as a new file.

Steps Involved in Encryption and Decryption:

1. **Key Generation:** A user-defined key (e.g., "mySecretKey12345") is passed through the **SHA-256** hash function to generate a 256-bit AES key.
2. **Encryption:**
   - A random IV is generated for each encryption operation.
   - The file contents are encrypted using the AES algorithm in CBC mode.
   - The encrypted file is saved, along with the IV at the beginning of the file.
3. **Decryption:**
   - The IV is read from the encrypted file.
   - The AES algorithm is initialized with the same secret key and the IV.
   - The file is decrypted, and the original content is restored and saved in the specified output file.

File Handling:

- The project reads input files, processes them byte-by-byte for encryption, and writes encrypted data to an output file.
- During decryption, it reads the encrypted file, extracts the IV, and uses it to decrypt the file content back to its original form.

Technologies Used:

- **Java Cryptography API:** The javax.crypto package is used to implement the AES encryption and decryption algorithms, key generation, and IV handling.
- **File I/O:** The project uses Java's java.nio.file.Files to read and write files, ensuring efficient handling of file data.
- **MessageDigest (SHA-256):** The MessageDigest class is used for generating the 256-bit encryption key from the provided passphrase.
- **Base64 Encoding:** (Optional, not fully implemented in the code, but typically used in cryptographic projects for encoding binary data into a readable format.)

Main Functions:

1. **getKey(String myKey):** This function takes a user-defined string and converts it into a 256-bit AES key using SHA-256.
2. **generateIv():** This function generates a random 16-byte Initialization Vector (IV) used for AES encryption.
3. **encryptFile(String inputFile, String outputFile, SecretKeySpec secretKey):** This function reads the input file, encrypts its content, and writes the encrypted content (along with the IV) to the output file.

4. **decryptFile(String inputFile, String outputFile, SecretKeySpec secretKey):** This function reads the encrypted file, extracts the IV, decrypts the file content, and writes the decrypted content to the output file.

Use Case:

This project can be used in scenarios where file confidentiality is crucial, such as securing sensitive documents, images, or videos. It ensures that files are encrypted with strong AES encryption, making them unreadable to unauthorized users without the correct decryption key.

Conclusion:

The **AES File Encryption and Decryption** project demonstrates an effective method to protect sensitive files using robust encryption techniques. By leveraging AES in CBC mode with a dynamically generated IV, it ensures that the encryption process is secure and the data is protected from unauthorized access. This project is a good example of implementing file security in Java using industry-standard cryptographic algorithms.