File Packer Unpacker - Interview Questions and Answers

1. Explain the flow and working of your project.

The project is a secure file packer and unpacker built using **Java** and **Swing** for the GUI. The process is as follows:

- **Authentication:** The user logs in using the fixed credentials "Marvellous" and "Marvellous."
- **Main Menu:** The user selects either **Pack** or **Unpack**.
- **Packing Process:** The user specifies the directory, a destination file name, and a **mandatory Encryption Key**. The program iterates through the source files, creates an **encrypted 100-byte header** (containing file name and size), and writes the **encrypted header** followed by the **encrypted file content** to the packed file.
- **Unpacking Process:** The user provides the packed file name and the corresponding **Decryption Key**. The program reads the file sequentially, using the key to **decrypt the header** to get the file size and name. It then reads and **decrypts the file data** and writes it to a newly created file.

2. What is mean by Packing And Unpacking?

Packing is the process of consolidating multiple source files into a single, combined archive file. This single file stores both the **encrypted content** of the original files and the necessary **encrypted metadata** (file names and sizes) in fixed 100-byte headers. **Unpacking** is the exact reverse operation: it reads the archive, decrypts the metadata and content, and recreates all original files in the system.

3. What is the use of Encryption? (*Updated*)

Encryption is a fundamental security measure used to convert data into a coded format, protecting it from unauthorized access or viewing. **In this project, encryption is used to secure both the file metadata (headers) and the file content** using a **user-defined key**. Without the correct decryption key, the packed file's contents cannot be recovered or understood.

4. How you can save memory by using packing?

The current project does not save memory. It combines files without implementing any form of **compression**. The total size of the packed file is the sum of the sizes of all the original files, plus the space consumed by the 100-byte encrypted header for each file.

5. Why you select Java as a developmental language for this project?

Java was selected primarily because it is a **platform-independent language**. This allows the compiled application to run consistently on any operating system (Windows, Linux, macOS) that has a Java Runtime Environment (JRE) installed, ensuring maximum portability.

6. Explain the concept of MD5 Checksum? (*Updated*)

An MD5 checksum is a unique **128-bit digital signature** or fingerprint generated for a file, used primarily to **verify data integrity**. Any tiny change to the file results in a completely different MD5 value. The project specification **requires** that the file name, size, and **checksum** be written to a log file, but this logging/checksum feature is currently **pending implementation**.

7. How you recreate all the files again in case of unpacking? (*Updated*)

The MarvellousUnpacker class handles file recreation by reading the packed file sequentially. The key steps are:

1. Read the **encrypted 100-byte header**.
2. **Decrypt** the header using the user's Decryption Key.
3. Extract the original file's name and size from the decrypted header.
4. Create a new file with the original name.
5. Read the exact number of bytes (the file size) of **encrypted data** from the packed file.
6. **Decrypt** the data and write it into the newly created file. This process repeats until the archive is exhausted.

8. How you maintain a record of each file in case of packing? (*Updated*)

The MarvellousPacker class maintains a record by creating a **fixed-size, 100-byte header** for each source file. This header contains the file's name and its size, padded with spaces. Critically, **this 100-byte header is encrypted** using the user's key and is written to the packed file immediately before the encrypted content of its corresponding file.

9. Which User interface you provide for this project?

The project provides a **graphical user interface (GUI)** built entirely using the standard **Java Swing** library. The GUI facilitates user interaction through a login frame, a main menu, a packing frame, and an unpacking frame.

10. Which is the targeted audience for this project?

The targeted audience includes students, developers, or any individual user who requires a simple, secure, and basic utility for grouping and extracting files across different platforms.

11. Are you generating any log of Packing and unpacking activity?

Yes, the project generates a log of its key activities by printing diagnostic messages and statistical reports to the console using System.out.println. This includes confirmation of directory access, file creation, and a final statistical report showing the number of files processed in each activity.

12. Which types of File Manipulations are used in this project?

The project relies on core file I/O operations from the java.io package:

- **File Reading:** Using FileInputStream to read data from source files and the packed file.
- **File Writing:** Using FileOutputStream to write data to the newly created packed file and to the extracted files during unpacking.
- **Directory Management:** Using the File class to traverse and list files in the source directory.

13. What happens if our folder contains duplicate files in case of packing?

If a folder contains duplicate file names, the packing process will sequentially include both files in the archive. However, during the unpacking process, the two files will be extracted sequentially with the same name. Since the system cannot hold two files with the exact same name in the same location, the second extracted file will **overwrite** the first one, leaving only the content of the file that was processed last.

14. Which types of files get packed and unpacked using your project?

The project can pack and unpack **any type of file** (text, binary, images, audio, etc.). This is because the core logic operates at the **byte level** using java.io, without relying on file extensions or needing to interpret the file content.

15. What are the resources that you refer during development of this project?

The primary resources used were the official **Java Documentation (Java API)** for the java.io and javax.swing packages, supplemented by various online tutorials and forums for specific implementation details regarding file handling and GUI design.

16. Which difficulty you faced in this project?

A key challenge was correctly implementing the logic for the **fixed-size 100-byte header**, especially ensuring that the file name and size information is accurately padded with spaces. Incorrect padding or parsing of the header (which is now also encrypted) is the most critical point, as any error here leads to corrupted files during the unpacking process.

17. Is there any chance of improvement in your project?

Yes, there is significant potential for improvement. Possible enhancements include:

1. **Compression:** Implementing a compression algorithm (e.g., Huffman or LZW) to genuinely save disk space.
2. **Required Features:** Implementing the pending features like the **3-attempt login limit** and **checksum logging**.
3. **User Experience:** Adding a file browser for easier input selection and a progress bar during I/O operations.

18. Can we use this project on different platforms?

Yes, absolutely. The project is **platform-independent** because it is built entirely using Java. It can run on any operating system, including Windows, Linux, and macOS, provided the Java Runtime Environment (JRE) is installed.

19. Explain File header in case of packed file? (*Updated*)

A file header is a **fixed-size 100-byte block of encrypted metadata** written at the beginning of each file's data within the packed archive. Its essential function is to store two pieces of information: the **original file name** and its **size in bytes**. This encrypted header is crucial, as the unpacker must successfully decrypt and read it to determine exactly how many bytes of encrypted data to read next to correctly recreate the original file.

20. Which type of Encryption technique is used for this project? (*Updated*)

This project uses a simple **Keyed XOR Cipher** (Exclusive OR) for encryption. The key used for the cipher is derived from the **user-provided string key** (it uses the byte value of the first character of the key string). The XOR operation is self-inverting, meaning the same simple operation with the same key is used for both encryption and decryption.