Custom Virtual File System Interview Q&A

1. What is the purpose of the CVFS.java program?

    1. CVFS stands for Custom Virtual File System.
    2. It simulates a file system using Java classes and memory, not disk.
    3. Supports commands like create, read, write, ls, stat, and unlink.
    4. Mimics basic Unix-like file operations.
    5. Uses OOP principles such as encapsulation and modularity.
    6. Helps understand how operating systems handle file management internally.

---

2. Which Java packages are imported in this file and why?

    1. java.util.Scanner – for reading user input.
    2. java.util.Arrays – for working with arrays like UFDT.
    3. java.io.Console – used for console interactions.
    4. All are part of Java's standard library (no external dependencies).
    5. These packages simplify input and output handling in the shell environment.

---

3. What is a class in Java and how many main classes are defined in this file?

    1. A class is a blueprint for creating objects in Java.
    2. It groups data (fields) and behavior (methods) together.
    3. In CVFS.java, main classes include MarvellousConstants, Inode, SuperBlock, FileTable, UAREA, and MarvellousFileSystem.
    4. Each class represents a different component of the virtual file system.
    5. Promotes separation of concerns and modular design.

---

4. What is the role of the main() function in this project?

1. Acts as the program's entry point.
2. Calls initialization methods like StartAuxilaryDataInitialisation().
3. Launches the MainShell() for command interaction.
4. Controls program termination using exit command.
5. Runs indefinitely in a loop until the user exits.

---

5. Why does the CVFS use constants instead of hardcoded numbers?

1. Defined in the MarvellousConstants class for global use.
2. Improves code readability and maintainability.
3. Avoids magic numbers and ensures consistent configuration.
4. Easier to modify limits (like MAXINODE) later.
5. Promotes cleaner and standardized code.

---

6. What is the use of Scanner in the CVFS program?

1. Used to read commands entered by the user.
2. Reads strings like "create demo.txt 3".
3. Parses input using split(" ") into tokens.
4. Provides flexibility for dynamic user input.
5. Simplifies console-based command handling.

---

7. Explain the naming convention followed in this project.

1. Class names use PascalCase (e.g., MarvellousFileSystem).
2. Variables and methods use camelCase (e.g., createFile(), fileName).
3. Constants use UPPERCASE_WITH_UNDERSCORES (e.g., MAXFILESIZE).
4. Follows standard Java coding conventions.
5. Increases code readability and uniformity.

8. What is the function of the help command in this system?

1. Displays all available commands in CVFS.
2. Helps new users understand system features.
3. Provides a brief description of each command.
4. Implemented as a simple print-based method.
5. Similar to Unix help or man commands.

---

9. Why is the file system simulated in memory and not on disk?

1. To simplify implementation without complex I/O.
2. Avoids use of File or FileWriter APIs.
3. Allows fast execution and testing.
4. Focuses on logic rather than persistence.
5. Ideal for learning OS-level concepts in Java.

---

10. How does the system handle user errors?

1. Checks for invalid command names or missing arguments.
2. Displays descriptive error messages.
3. Returns error codes defined in constants.
4. Prevents program crashes due to bad input.
5. Ensures stability and robustness of the shell.

---

11. Explain the concept of an inode in CVFS.

1. Each file is represented by an Inode object.
2. Stores metadata like FileName, FileSize, and Permission.
3. Contains a Buffer for file data.
4. Tracks reference count and link count.
5. Linked together in a list called the DILB (Disk Inode Linked Block).

6. Acts like Unix inode structures.

---

12. What is the purpose of the SuperBlock class?
    1. Manages overall file system resources.
    2. Tracks TotalInodes and FreeInodes.
    3. Acts like a resource manager for inode allocation.
    4. Initialized at startup to default values.
    5. Critical for controlling file creation and deletion.

---

13. What is the use of the UAREA class?
    1. Represents the User Area of the file system.
    2. Contains an array called UFDT (User File Descriptor Table).
    3. Each element in UFDT is a FileTable object.
    4. Simulates how an OS tracks open files per user.
    5. Limits number of open files using constants.

---

14. Explain how file creation is handled in CVFS.
    1. Command syntax: creat <filename> <permission>.
    2. Validates filename and permission (1=read, 2=write, 3=read/write).
    3. Checks for duplicate file names using IsFileExists().
    4. Allocates free inode and initializes its metadata.
    5. Creates FileTable entry and updates UFDT.
    6. Returns file descriptor to the user.

---

15. How are files stored in memory?

1. Each file has a character array buffer.
2. Data written using write_file() and read using read_file().
3. Managed entirely in heap memory.
4. No real file is created on disk.
5. Provides a safe and fast virtual environment.

---

16. Explain the function of write_file() in detail.

1. Accepts file descriptor and data string.
2. Checks if file has write permission.
3. Writes data into inode's buffer.
4. Updates write offset and actual file size.
5. Prevents overflow beyond MAXFILESIZE.
6. Returns number of bytes written.

---

17. How does read_file() work internally?

1. Takes file descriptor and read size.
2. Validates file existence and read permission.
3. Calculates available bytes using offset.
4. Copies data from buffer into output string.
5. Updates read offset.
6. Returns data read or an error message.

---

18. What does the stat_file() command show?

    1. File name and inode number.
    2. File size and type (regular or special).
    3. Link count and reference count.
    4. Permission values.
    5. Current status of file in system memory.
    6. Similar to Unix stat command.

---

19. What are the permission modes available?

    1. READ (1)
    2. WRITE (2)
    3. READ + WRITE (3)
    4. Checked before performing file operations.
    5. Enforced in both read_file() and write_file().

---

20. How is file deletion implemented?

    1. Executed using the unlink command.
    2. Frees inode buffer and clears metadata.
    3. Removes reference from UFDT array.
    4. Increments FreeInodes count.
    5. Returns confirmation message to user.

---

21. How does CVFS track open files?

1. Using UFDT[] array in the UAREA class.
2. Each entry points to a FileTable object.
3. Each FileTable stores read/write offsets.
4. Keeps Count field to track open instances.
5. Simulates how OS manages file descriptors.

---

22. What happens during system initialization?

1. The method StartAuxilaryDataInitialisation() runs.
2. Creates SuperBlock, Inode list, and UAREA.
3. Sets default values for all components.
4. Displays startup message “Boot block created successfully”.
5. Ensures system is ready before accepting commands.

---

23. Explain how the Disk Inode Linked Block (DILB) is created.

1. Function CreateDILB() initializes multiple inode objects.
2. Assigns unique InodeNumber to each.
3. Links them together using next pointers.
4. Sets initial values for permission, size, etc.
5. Simulates a list of available files in memory.

---

24. How are commands parsed in the shell?

1. User input read as string using Scanner.
2. Split into tokens using split(" ").
3. First token identifies command type.
4. Remaining tokens passed as arguments.
5. Commands matched using if-else or switch blocks.

25. What does the ManPage() function do?

1. Provides documentation for each command.
2. Displays syntax and examples.
3. Helps users recall parameters and permissions.
4. Acts as internal manual for the CVFS.
5. Improves usability in command-line environment.

---

26. What is the difference between FileType and Permission?

1. FileType defines kind of file (regular or special).
2. Permission defines what operations are allowed.
3. FileType helps system logic; Permission helps access control.
4. Both are part of the Inode class.
5. Used together for safe file access.

---

27. How does the ls command work in CVFS?

1. Traverses the inode linked list.
2. Displays names of all active (non-zero FileType) files.
3. Counts and shows number of valid files.
4. Skips empty or unused inodes.
5. Equivalent to Unix ls command output.

---

28. What is the significance of offsets in file operations?

1. ReadOffset and WriteOffset store current pointer positions.
2. Prevents overwriting old data accidentally.
3. Maintains continuity during multiple reads/writes.
4. Allows partial reads or appends.

5. Resets on file reopen or close.

---

29. What are some improvements that can be made to this CVFS?
    1. Add persistent file storage using Java I/O.
    2. Implement sub-directories and hierarchical paths.
    3. Introduce user-based permissions and authentication.
    4. Allow file append and truncate operations.
    5. Support concurrent access and thread safety.
    6. Provide graphical user interface for better usability.

---

30. Why is this project a good example of OOP in Java?
    1. Uses multiple interrelated classes to represent real-world entities.
    2. Demonstrates encapsulation via private fields and getters/setters.
    3. Modular design for each system component.
    4. Uses abstraction for user commands vs. internal operations.
    5. Easy to extend or modify without rewriting entire logic.
