**Topic:** File Handling  
  
**Introduction**

Java file handling refers to the set of classes and methods in the Java programming language that allow developers to read, write, and manipulate files on a computer's file system.

This includes opening, closing, reading from, and writing to files, as well as managing file and directory paths and permissions.  
  
**PATH**The Path class is used to programmatically represent a path in the file system (and can therefore point to files as well as directories, even to non-existent ones)

toString() returns the string representation of the path  
Path p1 = Paths.get("/var/www"); // p1.toString() returns "/var/www"

getFileName() returns the file name (or, more specifically, the last element of the pathPath p1 = Paths.get("/var/www"); // p1.getFileName() returns "www"

**Retrieving information using the**To interact with the filesystem you use the methods of the class Files.

**Checking existence:**  
To check the existence of the file or directory a path points to, you use the following methods:  
Files.exists(Path path) & Files.notExists(Path path)  
!Files.exists(path) does not neccesarily have to be equal to Files.notExists(path), because there are three possible scenarios

1. A file's or directory's existence is verified (exists returns true and notExists returns false in this case)  
2. A file's or directory's nonexistence is verfied (exists returns false and notExists returns true)   
3. Neither the existence nor the nonexistence of a file or a directory can be verified (for example due to access restrictions): Both exists and nonExists return false.

**Checking whether a path points to a file or a directory**This is done using Files.isDirectory(Path path) and Files.isRegularFile(Path path)  
**Path p1 = Paths.get("/var/www"); Path p2 = Paths.get("/home/testuser/File.txt");   
Files.isDirectory(p1) == true   
Files.isRegularFile(p1) == false   
Files.isDirectory(p2) == false**

**Reading files**Files can be read byte- and line-wise using the Files class  
Path p2 = Paths.get(URI.create("file:///home/testuser/File.txt")); byte[] content = Files.readAllBytes(p2); List linesOfContent = Files.readAllLines(p2);

Files.readAllLines() optionally takes a charset as parameter (default is StandardCharsets.UTF\_8):

List linesOfContent = Files.readAllLines(p2, StandardCharsets.ISO\_8859\_1);

**Writing files**  
Files can be written bite- and line-wise using the Files class

Path p2 = Paths.get("/home/testuser/File.txt");   
List lines = Arrays.asList( new String[]{"First line", "Second line", "Third line"});   
Files.write(p2, lines);   
Files.write(Path path, byte[] bytes)

Eisting files wile be overridden, non-existing files will be created.

**File I/O**

(Input and Output) is used to process the input and produce the output. Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations. Handling files is also done in java by Java I/O API.

**Point to a path**

// -> IO   
File file = new File("io.txt");

// -> NIO   
Path path = Paths.get("nio.txt");

**Paths relative to another path**

// Forward slashes can be used in place of backslashes even on a Windows operating system // -> IO   
File folder = new File("C:/");   
File fileInFolder = new File(folder, "io.txt");

// -> NIO   
Path directory = Paths.get("C:/");   
Path pathInDirectory = directory.resolve("nio.txt");

**Converting File from/to Path for use with libraries**

// -> IO to NIO   
Path pathFromFile = new File("io.txt").toPath();

// -> NIO to IO   
File fileFromPath = Paths.get("nio.txt").toFile();

**Check if the file exists and delete it if it does**

/ -> IO

if (file.exists()) {   
boolean deleted = file.delete();   
if (!deleted) {   
throw new IOException("Unable to delete file"); } }

// -> NIO   
Files.deleteIfExists(path);

**Interview Questions**

**1. What are File Channels and Byte Buffers in Java NIO, and how do you use them?**

**Answer:** File Channels and Byte Buffers are part of the Java NIO (New I/O) package. File Channels provide a faster way to read from and write to files compared to traditional I/O streams. Byte Buffers are containers for data that is read from or written to channels.

**2. How can you handle a NullPointerException in file handling operations?**

**Answer:** A NullPointerException can occur if you attempt to use an object reference that hasn't been initialized. In file handling, this often happens when trying to format or process a null date. You can handle this by checking for null values before performing operations.