29 Lecture - CS504

Important Subjective

Q: How do you open a file for reading in C++? **A**: You can open a file for reading in C++ using the std::ifstream class and the open() method with the file path as the argument. **Q: What is the purpose of using buffered file streams in Java?** **A**: Buffered file streams, such as BufferedReader and BufferedWriter, improve file I/O performance by reducing frequent disk access. **Q: How can you handle exceptions during file handling in Java?** **A**: In Java, you can use try-catch blocks to catch and handle exceptions that may occur during file operations, ensuring graceful error handling. **Q: What precautions should be taken when using relative file paths?** **A**: When using relative file paths, ensure that the current working directory is consistent across different environments to access files correctly. **Q: How do you close a file after reading or writing in C++?** **A**: In C++, you can close a file opened for reading or writing using the close() method or automatically by the file stream's destructor. **Q: What is the benefit of using the try-with-resources statement in Java for file handling?** **A**: The try-with-resources statement automatically closes resources (e.g., file streams) after execution, reducing manual cleanup code. **Q: Why should you check for file existence before opening it?** **A**: Checking file existence prevents potential exceptions and ensures that you are working with valid files, avoiding unexpected behavior. **Q: How does the DRY (Don't Repeat Yourself) principle relate to file handling?** **A**: The DRY principle encourages code reuse, leading to the creation of functions or classes that handle file operations, minimizing code duplication. **Q: How can you read a file line-by-line in C++?** **A**: In C++, you can use a std::ifstream object to read a file line-by-line using a loop and the getline() function. **Q: What are some common file handling pitfalls in both C++ and Java?** **A**: Common pitfalls include not closing files after usage, improper error handling, incorrect file paths, and insufficient memory management.