Java File Handling Overview

Dr. Mohammad Salah Uddin

August 24, 2023

Introduction

Java provides the **java.io** package for file I/O operations. Here are some common tasks we can accomplish with Java file handling:

- Reading from a File: We can read data from files using classes like **FileInputStream**, **FileReader**, **BufferedReader**, etc.
- Writing to a File: Writing data to files is achieved using classes like **FileOutputStream**, **FileWriter**, **BufferedWriter**, etc.
- Creating and Deleting Files and Directories: We can create, delete, and manage files and directories using the **File** class.

Introduction (Con't)

Java provides the **java.io** package for file I/O operations. Here are some common tasks we can accomplish with Java file handling:

- Checking File and Directory Information: The **File** class provides methods to check various attributes of files and directories, such as existence, size, last modified date, etc.
- Working with Binary Data: Java provides classes like **DataInputStream** and **DataOutputStream** for reading and writing primitive data types in binary format.

Key Classes and Methods

Here are some key classes and methods we'll use for file handling in Java:

- **java.io.File**
  - File(String pathname): Creates a new **File** instance with the given path.
  - **boolean exists()**: Checks if the file or directory exists.
  - **boolean isFile()**, **boolean isDirectory()**: Checks if the **File** instance represents a file or directory.
  - **String[] list()**: Returns an array of filenames in the directory represented by the **File** instance.

Key Classes and Methods (con't)

Here are some key classes and methods we'll use for file handling in Java:

- **java.io.FileReader** and **java.io.FileWriter**
    - **FileReader(String fileName)**, **FileWriter(String fileName)**: Creates file readers and writers.
    - **int read()**: Reads a single character from the reader. Returns -1 at end of file.
    - **int read(char[] buffer)**: Reads characters into a buffer.
    - **void write(int c)**, **void write(char[] cbuf)**: Writes characters to the writer.

Buffered Reader and Writer

These classes provide buffering for character input and output streams, improving performance.

- **java.io.BufferedReader**
  - Buffered character input stream.
  - Efficiently reads text from a character-input stream.
  - **readLine()** reads a line of text.
- **java.io.BufferedWriter**
  - Buffered character output stream.
  - Efficiently writes text to a character-output stream.
  - **newLine()** writes a platform-specific line separator.

FileInputStream and FileOutputStream

Used for reading from and writing to binary files.

- **java.io.FileInputStream**
    - Reads bytes from a file.
    - Provides low-level access to file contents.
    - **read(byte[] buffer)** reads bytes into a buffer.
- **java.io.FileOutputStream**
    - Writes bytes to a file.
    - Provides low-level access to file contents.
    - **write(byte[] buffer)** writes bytes from a buffer.

DataInputStream and DataOutputStream

Used for reading and writing primitive data types in binary format.

- **java.io.DataInputStream**
  - Reads primitive data types.
  - Reads from an underlying input stream.
  - **readInt()**, **readDouble()**, etc.
- **java.io.DataOutputStream**
  - Writes primitive data types.
  - Writes to an underlying output stream.
  - **writeInt()**, **writeDouble()**, etc.

Exception Handling

File handling operations can throw various exceptions, such as **IOException**. It's important to handle exceptions properly to ensure that our code is robust and handles unexpected scenarios.

- Always enclose file operations in try-catch blocks.
- Provide user-friendly error messages to aid debugging.

Conclusion

- Java file handling is fundamental for diverse applications.
- Proper exception handling ensures robust and reliable programs.
- Always close files after reading/writing to prevent resource leaks.