**Introduction to File Handling in Java**

File handling in Java is a crucial part of the I/O (Input and Output) operations. Java provides several classes and methods to read, write, and manipulate files and directories.

**Key Classes for File Handling**

1. **File**: Represents a file or directory pathnames in an abstract manner.
2. **FileReader**: Used for reading streams of characters from files.
3. **FileWriter**: Used for writing streams of characters to files.
4. **BufferedReader**: Reads text from a character input stream, buffering characters for efficient reading.
5. **BufferedWriter**: Writes text to a character output stream, buffering characters for efficient writing.
6. **FileInputStream**: For reading streams of raw bytes from files.
7. **FileOutputStream**: For writing streams of raw bytes to files.
8. **Scanner**: For reading files using a delimiter pattern (such as new lines, spaces, etc.).

**Working with the File Class**

The File class is used to create and manage file and directory paths.

**Common Methods in the File Class**

java

Copy code

File file = new File("example.txt");

// Check if file exists

boolean exists = file.exists();

// Create a new file

boolean fileCreated = file.createNewFile();

// Delete a file

boolean fileDeleted = file.delete();

// Check if it's a file or directory

boolean isFile = file.isFile();

boolean isDirectory = file.isDirectory();

// List files in a directory

String[] fileList = file.list();

// Get file information

String fileName = file.getName();

String filePath = file.getPath();

long fileSize = file.length();

**Reading Files**

**Using FileReader and BufferedReader**

java

Copy code

FileReader fr = new FileReader("example.txt");

BufferedReader br = new BufferedReader(fr);

String line;

while ((line = br.readLine()) != null) {

System.out.println(line);

}

br.close();

fr.close();

**Using Scanner**

java

Copy code

File file = new File("example.txt");

Scanner scanner = new Scanner(file);

while (scanner.hasNextLine()) {

String line = scanner.nextLine();

System.out.println(line);

}

scanner.close();

**Writing to Files**

**Using FileWriter and BufferedWriter**

java

Copy code

FileWriter fw = new FileWriter("example.txt");

BufferedWriter bw = new BufferedWriter(fw);

bw.write("Hello, world!");

bw.newLine();

bw.write("Another line.");

bw.close();

fw.close();

**Using PrintWriter**

java

Copy code

PrintWriter pw = new PrintWriter("example.txt");

pw.println("Hello, world!");

pw.println("Another line.");

pw.close();

**Working with Byte Streams**

**Using FileInputStream and FileOutputStream**

java

Copy code

// Reading bytes

FileInputStream fis = new FileInputStream("example.txt");

int byteContent;

while ((byteContent = fis.read()) != -1) {

System.out.print((char) byteContent);

}

fis.close();

// Writing bytes

FileOutputStream fos = new FileOutputStream("example.txt");

String text = "Hello, world!";

fos.write(text.getBytes());

fos.close();

**Handling Exceptions**

File handling operations can throw several checked exceptions, including:

* IOException
* FileNotFoundException

Always use try-catch blocks or throw the exceptions as needed.

java

Copy code

try {

FileReader fr = new FileReader("example.txt");

BufferedReader br = new BufferedReader(fr);

String line;

while ((line = br.readLine()) != null) {

System.out.println(line);

}

br.close();

fr.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

**Summary**

1. Use the File class for file and directory management.
2. Use FileReader/FileWriter for reading/writing character streams.
3. Use BufferedReader/BufferedWriter for efficient reading/writing of text.
4. Use FileInputStream/FileOutputStream for reading/writing byte streams.
5. Handle exceptions properly to avoid runtime errors.