Here are detailed and structured notes from the uploaded document:

File Handling in Java

Introduction to File I/O

- Programs often need to communicate with the outside world via input (e.g., keyboard) or output (e.g., screen).
- Communication can also occur through stored data, such as files.

I/O Streams

- A stream is a communication channel for transferring data between a program and an external source.
- **Types of Data Streams:**
  - Byte streams (8-bit data).
  - Character streams (Unicode characters).
- **Stream Characteristics:**
  - Represent various data sources and destinations (files, devices, memory).
  - Perform data manipulation (e.g., filtering, transforming).

Java I/O API

- Found in the `java.io` package.
- **Key Features:**
  - Provides classes for sequential access streams.
  - Divided into byte-based and character-based streams.
  - Supports operations like reading, writing, and object serialization.

Types of Streams

1. **Byte Streams:**
   - Handle raw byte data.
   - Example: `InputStream` (for reading), `OutputStream` (for writing).
2. **Character Streams:**
   - Handle character data.
   - Example: `Reader` (for reading), `Writer` (for writing).

File Class in Java

- Represents file and directory paths.
- **Constructors:**
  1. `File(String name)` – Creates a `File` object for a specified file or directory.
  2. `File(String subDir, String name)` – Represents a file in a subdirectory.
  3. `File(File subDir, String name)` – Represents a file using a `File` object for the subdirectory.

Creating Files in Java

1. **Using `createNewFile()`:**
    - Creates a new file.
    - Example:

```
File file = new File("demo.txt");
file.createNewFile();
```

2. **Using `FileOutputStream`:**
    - Writes byte data to a file.
3. **Using `createFile()`:**
    - Creates new files through specific methods.

Important Methods in the File Class

Method	Description
`exists()`	Checks if a file or directory exists.
`createNewFile()`	Creates a new file if it doesn't already exist.
`mkdir()`	Creates a directory.
`isFile()`	Checks if the object represents a file.
`isDirectory()`	Checks if the object represents a directory.
`list()`	Lists all files and directories within a specified directory.
`length()`	Returns the number of characters in a file.
`delete()`	Deletes a file or directory.

FileWriter and FileReader

- **FileWriter:**
    - Used to write character data to files.
    - Example constructors:

```
FileWriter fw = new FileWriter("file.txt");
FileWriter fw = new FileWriter("file.txt", true); // Append mode
```

    - Methods: `write()`, `flush()`, `close()`.
- **FileReader:**
    - Reads character data from files.
    - Example constructor:

```
FileReader fr = new FileReader("file.txt");
```

    - Methods: `read()`, `close()`.

BufferedWriter and BufferedReader

- **BufferedWriter:**
  - Enhances `FileWriter` for efficient writing with buffering.
  - Allows appending new lines using `newLine()`.
- **BufferedReader:**
  - Enhances `FileReader` for efficient reading.
  - Reads data line-by-line with `readLine()`.

PrintWriter

- An advanced writer that can write any type of data (text, integers, etc.).
- Example constructors:

```
PrintWriter pw = new PrintWriter("file.txt");
```

- Methods include `print()` and `println()`.

Serialization and Deserialization

1. **Serialization:**
   - Converts objects into byte streams for storage or transfer.
   - Requires the class to implement the `Serializable` interface.
   - Example:

```
ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("file.obj"));
oos.writeObject(object);
```

2. **Deserialization:**
   - Converts byte streams back into objects.
   - Example:

```
ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("file.obj"));
Object obj = ois.readObject();
```

Points to Remember:

- The `Serializable` interface is a marker interface with no methods.
- Java classes must remain consistent during serialization and deserialization.

Programs and Practical Examples

1. Merge contents of two files into another file.
2. Perform line-by-line merging from multiple files.
3. List and filter file and directory names in a folder.

Let me know if you need further clarification or edits!