Lesson-13

An **InputStreamReader** is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset.

FileReader: Reads text from character files. It Inherits from **InputStreamReader**. Need to give the name of the file to read. Throws FileNotFoundException if the named file does not exist/can't open the file.

An **OutputStreamWriter** is a bridge from character streams to byte streams. Characters written to it are encoded into bytes using charset. Inherits from **Writer** class. It's an Abstract class.

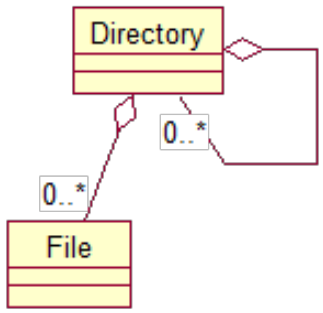
PrintWriter Prints formatted representations of objects to a text-output stream.

To read character streams, use a subclass of **Reader**. To write character streams, use a subclass of **Writer**

- Use **byte streams** for raw binary data. (Optional Topic)
  - Suitable for handling binary data, such as image files, audio files, video files, etc.
- Use **character streams** for text data.
  - Suitable for handling text data, such as text files, HTML files, etc.

File ob = new File("Hello.txt");

Diagram representation	Java Class

	Class Directory{
	
	List<File> files;
	List<Directory> dirs; // Reflexive Association
	
	}

Different Streams to work with Files

Java I/O Stream Types – Simplified Explanation

1. Byte Streams (`InputStream` / `OutputStream`)

- Used for reading and writing binary data (files, images, videos).
- Works with bytes (`byte[]`, `int`), not characters.
- Common classes: `FileInputStream`, `FileOutputStream`.
- Best for handling raw data like multimedia files.

2. Character Streams (`Reader` / `Writer`)

- Designed for handling text-based data (char, String).
- Supports character encoding (UTF-8, Unicode, ASCII).
- Common classes: FileReader, FileWriter.
- Ideal for reading text files and documents.

3. Data Streams (`DataInputStream` / `DataOutputStream`)

- Reads and writes primitive data types (`int`, `double`, `boolean`).
- Stores structured binary data in a portable format.
- Common methods: `readInt()`, `writeDouble()`, `readUTF()`.
- Best for storing numerical or structured data in files.

4. Buffered Streams (`BufferedInputStream` / `BufferedOutputStream`)

- Improves I/O performance by reducing direct disk access.
- Used to improve the performance of other streams by buffering data.
- Uses an internal buffer to read/write large chunks of data efficiently.
- Common classes `BufferedReader`, `BufferedWriter`, `BufferedInputStream`.
- Recommended for handling large files and frequent I/O operations.

5. Object Streams (`ObjectInputStream` / `ObjectOutputStream`)

- Used for serializing and deserializing Java objects.
- Allows storing objects in files or transmitting them over networks.
- Common methods: `writeObject()`, `readObject()`.
- Best for object persistence**.**