Character Stream Vs Byte Stream in Java

A stream is a sequence of data. I/O Stream refers to a stream that is unlikely a method to sequentially access a file. I/O Stream means an input source or output destination representing different types of sources e.g. disk files. The java.io package provides classes that allow you to convert between Unicode character streams and byte streams of non-Unicode text.

- Input Stream: reads data from the source.
- Output Stream: writes data to a destination.

When to use Character Stream over Byte Stream?

In Java, characters are stored using Unicode conventions. Character stream is useful when we want to process text files. These text files can be processed character by character. Character size is typically 16 bits.

When to use Byte Stream over Character Stream?

Byte oriented reads byte by byte. A byte stream is suitable for processing raw data like binary files.

Key points while using and dealing with any of the above streams are as follows:

1. Names of character streams typically end with Reader/Writer and names of byte streams end with InputStream/OutputStream
2. The streams used in example codes are unbuffered streams and less efficient. We typically use them with buffered readers/writers for efficiency. We will soon be discussing use BufferedReader/BufferedWriter (for character stream) and BufferedInputStream/BufferedOutputStream (for byte stream) classes.

3. It is always recommended to close the stream if it is no longer in use. This ensures that the streams won't be affected if any error occurs.
4. The above codes may not run in online compilers as files may not exist.

Character Stream

In Java, characters are stored using Unicode conventions. Character stream automatically allows us to read/write data character by character. For example, FileReader and FileWriter are character streams used to read from the source and write to the destination.

Example

```
// Java Program illustrate Reading
// a File in Human Readable
// Format Using FileReader Class

// Importing required classes
import java.io.*;

// Main class
public class GfG {

    // Main driver method
    public static void main(String[] args) throws IOException {
```

```
        // Initially assigning null as we have not read
        // anything
        FileReader sourceStream = null;

        // Try block to check for exceptions
        try {

            // Reading from file
            sourceStream = new FileReader(
                "/Users/mayanksolanki/Desktop/demo.rtf");

            // Reading sourcefile and writing content to
            // target file character by character.

            int temp;

            // If there is content inside file
            // than read
            while ((temp = sourceStream.read()) != -1)
                System.out.println((char)temp);

            // Display message for successful execution of
            // program
            System.out.print(
                "Program successfully executed");
        }

        // finally block that executes for sure
        // where we are closing file connections
        // to avoid memory leakage
        finally {

            // Closing stream as no longer in use
            if (sourceStream != null)
                sourceStream.close();
        }
    }
}
```

Output:

Writes content to the target file character by character

```
Program successfully executed
```

Byte Stream

Byte streams process data byte by byte (8 bits). For example, FileInputStream is used to read from the source and FileOutputStream to write to the destination.

Example:

```java
// Java Program Illustrate ByteStream Class to
// Copy Contents of One File to Another File

// Importing required classes
import java.io.*;

// Main class
public class GfG {

    // Main driver method
    public static void main(String[] args)
        throws IOException {

        // Initially assigning null ot objects for
        // reading and writing to file
        FileInputStream sourceStream = null;
        FileOutputStream targetStream = null;

        // Try block to check for exceptions
        try {

            // Passing the files via local directory
            sourceStream = new FileInputStream(
                "/Users/mayanksolanki/Desktop/demo.rtf");
            targetStream = new FileOutputStream(
                "/Users/mayanksolanki/Desktop/democopy.rtf");

            // Reading source file and writing content to
            // target file byte by byte
            int temp;

            // If there is content inside file
            // than read
            while ((temp = sourceStream.read()) != -1)
                targetStream.write((byte)temp);

            // Display message for successful execution of
```

```
            // program
            System.out.print(
                "Program successfully executed");
        }

        // finally block that executes for sure
        // where we are closing file connections
        // to avoid memory leakage
        finally {

            if (sourceStream != null)
                sourceStream.close();

            if (targetStream != null)
                targetStream.close();
        }
    }
}
```

Output:

```
Program successfully executed
```