FILE SYSTEM

1. Input / Output Basics

Java programs perform I/O operations through **streams**.

What is a sream ?

A stream is an abstraction for reading/writing a sequence of data.
However, stream does not represent data, instead stream is linked to a physical device by the Java I/O system to read and write data.

Two types of streams (based on direction of data flow)

1. **input** stream : A stream from which data is read from data end points.
2. **output** stream : A stream to which data is written to data end points.

The data end points are

- Keyboard/Console
- File
- Network

Stream Classes in Java

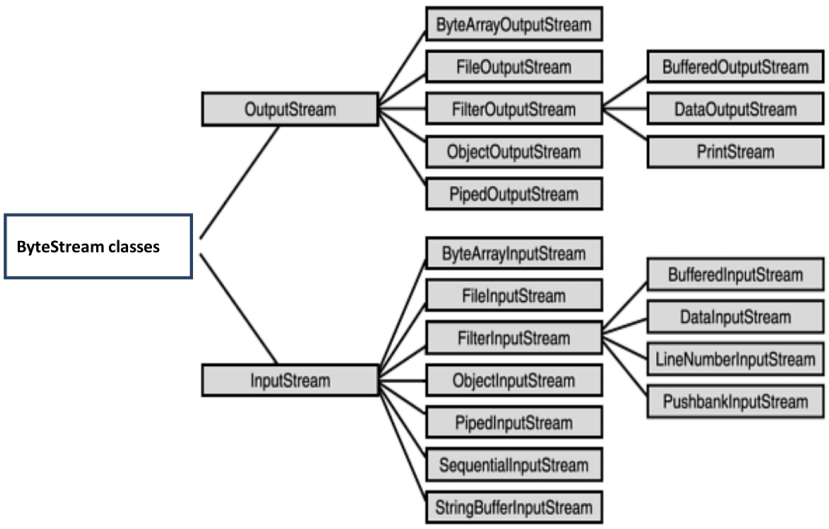
The **java.io** package contains all the classes required for input and output operations.
Java defines two types of streams (based on data type) for IO operation.

- B**yte** stream
- C**harater** stream

Byte Streams: reading/writing data as **bytes**. These streams are used for any kind of data, including text, images, audio, etc as 1 byte ASCII characters (English letters, numbers, etc).

Character streams: reading/writing data as **2-byte unicode characters** which means they can process not just ASCII characters but also characters from international languages (e.g., Chinese, Japanese, Arabic, Hindi...etc).

Byte Stream Classes



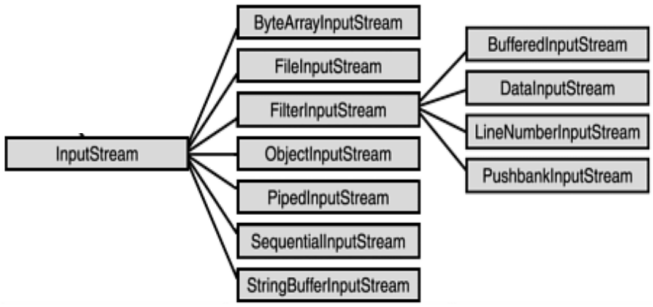
Byte streams are defined by using two class hierarchies as given below

1. **InputStream** – Used to read data from data end points.
2. **OutputStream** – Used to write data to data end points.

*Both these classes are **abstract classes**, that is programmer can not instantiate objects.*
As in diagram, each abstract classes got concreate subclasses to handle different data end points, devices such as Files, Network, Memory buffers.

Two of the most important are **read( )** and **write( )**,
– These methods are overridden by derived stream classes.

InputStream Hirarchy



Some important classes in the hirarchy are discussed below

1. FileInputStream: One of the few classes that can directly **read data** from Disk File.

Constructor and Description

FileInputStream(File file) :
Creates a FileInputStream by opening a connection to an actual file, the file named by the File object file in the file system.

Eg. opens a file input.txt to read

```
// create File object
File file = new File( "input.txt");
FileInputStream fis = new FileInputStream(file);
```

FileInputStream(String name)
Creates a FileInputStream by opening a connection to an actual file, the file named by the path name name in the file system

Eg. opens a file input.txt using file name to read instead of File object.

```
FileInputStream fis = new FileInputStream("input.txt");
```

Read functions and Description

int read()
Reads a byte of data from this input stream.
Eg. read data using above created fis object

```
int data = 0;
while ((data = fis.read()) != -1) {
        System.out.print((char) data);
}
```

int read(byte[] byteArray)
Reads up to **byteArray.length** bytes of data from this input stream into an array of bytes.
Eg. read data using above created fis object

```
byte[] buffer = new byte[1024];
int bytesRead;
while ((bytesRead = fis.read(buffer)) != -1) {
        System.out.print(new String(buffer, 0, bytesRead));
}
```

int read(byte[] b, int off, int len)
Reads up to **len** bytes of data from this input stream into an array of bytes.

Eg. read data using above created fis object

```
byte[] buffer = new byte[1024]; // Define a buffer to hold the data
int bytesRead;
int offset = 0;
int length = 20; // Set the length to read
while ((bytesRead = fis.read(buffer, offset, length)) != -1) {
        // Convert the read bytes to a string and print them
        String output = new String(buffer, 0, bytesRead);
        System.out.println(output);
}
```

Important Note:
*1. There is NO function available in FileInputStrean to read a **String data**.*
2. Classes like ***BufferedInputStream***, ***DataInputStream***, ***ObjectInputStream etc*** *are not capable of reading* data directly from Disk file. Hence these classes **wrap** a FileInputStream object to do **File** I/O operation on respective requirements.

2. BufferedInputStream:
Wraps another InputStream, such as FileInputStream, to improve efficiency by buffering input data (reading larger chunks at once and reducing disk access).
It does not read from the disk directly but uses FileInputStream as its underlying source to get data from Disk file.

Constructor and Description
Eg Create Object of BufferedInputStream

```
FileInputStream fis = new FileInputStream("input.txt");
// wrap FileInputStream with BufferedInputStream
BufferedInputStream bis = new BufferedInputStream(fis);
```

Read functions and Description
Can use any of the read functions given below
int read()
int read(byte[] byteArray)
int read(byte[] b, int off, int len)
Eg. read data using above created **bis** object.

```
int read()
while ((data = bis.read()) != -1) {
        System.out.print((char) data);
        bos.write(data);
}
```

3. DataInputStream:

So far, the data is read as bytes. But some binary files (.bin or .dat extensions) need to read data as primitive data types (e.g., int, float, double, etc.). DataInputStream is ueed to fullfil such requiremets. DataInputStream also be used to read from Network (sockets).

It does not read from the disk directly but uses FileInputStream as its underlying source to get data from Disk file.

Eg Create Object of DataInputStream

```
FileInputStream fis = new FileInputStream("binary_data.dat");
DataInputStream din = new DataInputStream(fis);
```

Can use any of the read functions given below

```
int read()
int read(byte[] byteArray)
int read(byte[] b, int off, int len)
boolean readBoolean()
byte readByte()
char readChar()
double readDouble()
float readFloat()
int readInt()
```

Eg. read data using above created din object.

```
// read double value
double dVal = din.readDouble();
// read int value
int iVal = din.readInt();
// read boolean value
boolean bVal = din.readBoolean();
// read char value
char cVal = din.readChar();

System.out.println("dVal = " + dVal);
System.out.println("iVal = " + iVal);
System.out.println("bVal = " + bVal);
System.out.println("cVal = " + cVal);
```

4. ObjectInputStream:

ObjectOutputStream is used to serialize objects (save them in byte form).
ObjectInputStream is used to deserialize objects (read them back into their original form).Both streams are typically used for persistent storage or object transfer in Java.

```
FileInputStream fileInputStream = new FileInputStream("serialized_object.dat");
ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);
// Read and deserialize the object from the file
Person person = (Person) objectInputStream.readObject();
```

5. PipedInputStream : Used to create a **data pipe**.
It facilitates communication between two threads in the same program by creating a data pipe. One thread writes data to the pipe using a PipedOutputStream, and another thread reads the data from the pipe using a PipedInputStream

```
PipedOutputStream outputStream = new PipedOutputStream();
PipedInputStream inputStream = new PipedInputStream(outputStream);

String message = "Hello from the producer!";
outputStream.write(message.getBytes());

int data;
while ((data = inputStream.read()) != -1) {
      System.out.print((char) data);
}
```

6. ByteArrayInputStream: allows an application to create an input stream from a byte array. It is used when you want to **read data from memory** (i.e., a byte array) **rather than** from an external data source like a file, socket, or pipe**.**

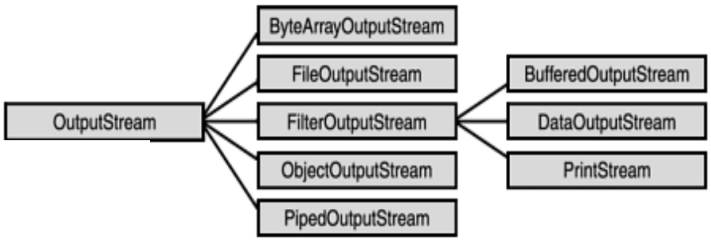
```
FileInputStream fileInputStream = new FileInputStream("firmware.bin");
byte[] fileData = new byte[fileInputStream.available()];
fileInputStream.read(fileData);

ByteArrayInputStream input = new ByteArrayInputStream(fileData);

// Calculate the SHA-256 checksum of the data
MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] buffer = new byte[4096];
int bytesRead;

while ((bytesRead = input.read(buffer)) != -1) {
   md.update(buffer, 0, bytesRead);
}
```

OutputStream Hirarchy



Some important classes in the hirarchy are discussed below

1. FileOutputStream: One of the few classes that can directly **write data** from Disk File.

Constructor and Description

FileOutputStream(File file)
Creates a file output stream to write to the file represented by the specified File object.

Open in write mode

```
File file = new File( "output.txt");
FileOutputStream fos = new FileOutputStream(file);
```

FileOutputStream(File file, boolean append)

Open in write mode, if boolean append = false
Open in append mode, if boolean append = true

```
File file = new File( "output.txt");
FileOutputStream fos = new FileOutputStream(file, append);
```

FileOutputStream(String name)
Creates a file output stream to write to the file represented by the specified f**ile name**.

```
FileOutputStream fos = new FileOutputStream(“output.txt”);
```

FileOutputStream(String name, boolean append)
Creates a file output stream to write to the file with the specified **file name**.

Open in write mode, if boolean append = false
Open in append mode, if boolean append = true

```
FileOutputStream fos = new FileOutputStream("output.txt", append);
```

If file doesnt exists, this will create an empty file with the given fine name.

Write functions and Description

Eg: Write data using above created **fos** object

void write(int b)
Writes the specified byte to this file output stream.

```
FileOutputStream fos = new FileOutputStream("output.txt"));
fos.write(65);  // Writes the byte for ASCII 'A'
System.out.println("Single byte (A) written to the file.");
```

void write(byte[] b)
Writes b.length bytes from the specified byte array to this file output stream.

```
FileOutputStream fos = new FileOutputStream("output.txt"));
String message = "Genesis 1:1 In the beginning God created the heavens and the earth";

byte[] byteArray = message.getBytes();
fos.write(byteArray);
```

void write(byte[] b, int offset, int len)
Writes len bytes from the specified byte array starting at offset off to this file output stream.

```
FileOutputStream fos = new FileOutputStream("output.txt"));
String message = "Genesis 1:1 In the beginning God created the heavens and the earth";
```

byte[] byteArray = message.getBytes();
fos.write(byteArray, 12, message.length());

offset = 12, len = message.length() = 74
Therefore bytes between 12 to 74 will be written into the file.
That is " In the beginning God created the heavens and the earth"

2. BufferedOutputStream:

Wraps another OutputStream, such as FileOutputStream, to improve efficiency by buffering output data (reading larger chunks at once and reducing disk access).

It does not wrte to disk directly but uses FileOutputStream as its underlying source to write data to Disk file.

Eg Create Object of BufferedOutputStream

FileOutputStream fos = new FileOutputStream("output.txt");
BufferedOutputStream bos = new BufferedOutputStream(fos);

Can use any of the read functions given below

int read()
int read(byte[] b, int off, int len)

Eg. read data using above created ***bos*** object.

Using int write()

```
// Example 1: Write a single byte using write(int b)
int singleByte = 65; // ASCII value of 'A'
bos.write(singleByte);
```

Using int write(byte[] byteArray)

```
// Example 2: Write a byte array using write(byte[] b, int off, int len)
String data = "Genesis 1:1 In the beginning God created the heavens and the
earth";
byte[] byteArray = data.getBytes();
int offset = 0; // Start writing from beginning
int length = data.length(); // Write till end
bos.write(byteArray, offset, length);
```

3. DataOutputStream:

So far, the data is written as bytes. But some binary files (.bin or .dat extensions) need to write data as primitive data types (e.g., int, float, double, etc.). DataOutputStream is ueed to fullfil such requiremets. DataOutputStream also be used to write data to Network (sockets).

It does not write to the disk directly but uses FileOutputStream as its underlying source to write data to Disk file.

Constructor

```
public DataOutputStream(OutputStream out)
```

Write Functions

```
void write(byte[] b, int off, int len)
void write(int b)
void writeBoolean(boolean v)
void writeByte(int v)
void writeBytes(String s)
void writeChar(int v)
void writeChars(String s)
void writeDouble(double v)
void writeFloat(float v)
void writeInt(int v)
void writeLong(long v)
void writeShort(int v)
```

```
//Eg, write to a binary file
FileOutputStream fos = new FileOutputStream("binary_data.dat");
DataOutputStream dout = new DataOutputStream(fos);

dout.writeDouble(3.14);
dout.writeInt(100);
dout.writeBoolean(true);
dout.writeChar('A');
```

4. ObjectOutputStream:

ObjectOutputStream is used to serialize objects (save them in byte form). ObjectInputStream is used to deserialize objects (read them back into their original form).Both streams are typically used for persistent storage or object transfer in Java.

Eg: Serialize object of class Person

```
FileOutputStream fileOutputStream = new
FileOutputStream("serialized_object.dat");
ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);

// Create an object to serialize
Person person = new Person("Alice", 30);

// Serialize and write the object to the file
objectOutputStream.writeObject(person);
```

5. ByteArrayOutputStream

This itself does not write directly to the file system. It writes data into an in-memory byte array (buffer). However, you can take the byte data stored in the ByteArrayOutputStream and manually write it to a file using other classes like FileOutputStream.

```
// Data to be written into ByteArrayOutputStream
String message = "Genesis 1:1 In the beginning God created the heavens and
the earth";
ByteArrayOutputStream baos = new ByteArrayOutputStream()) {

// Convert the string message to a byte array
byte[] byteArray = message.getBytes();
```

```
// Write the byte array into the ByteArrayOutputStream
baos.write(byteArray);

// Convert the ByteArrayOutputStream to byte array
byte[] outputBytes = baos.toByteArray();

// Now write the byte array to a file
FileOutputStream fos = new FileOutputStream("output.txt")
fos.write(outputBytes);
```

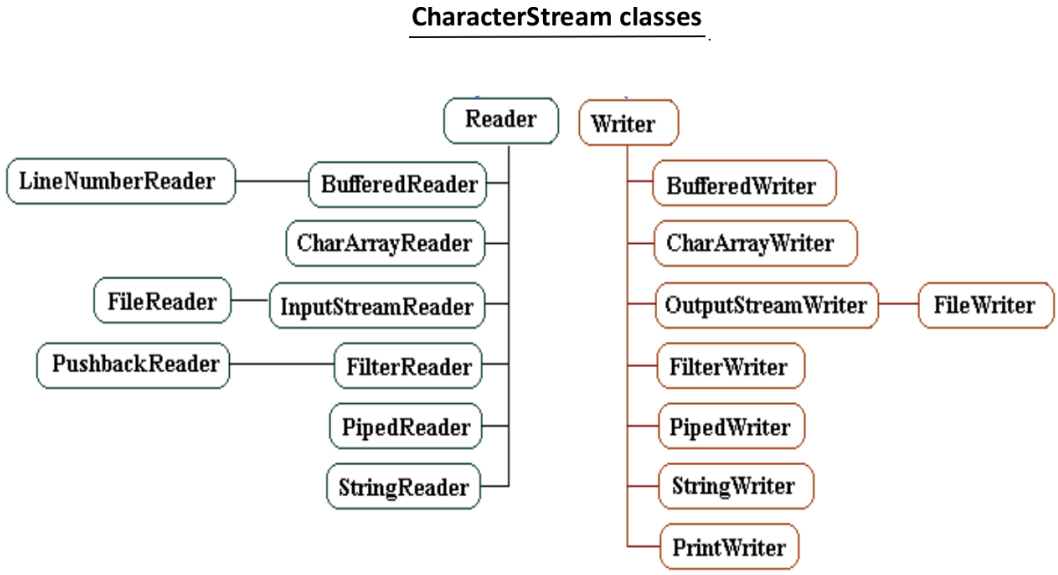
Character Stream Classes

Character streams are defined by using two class hierarchies.

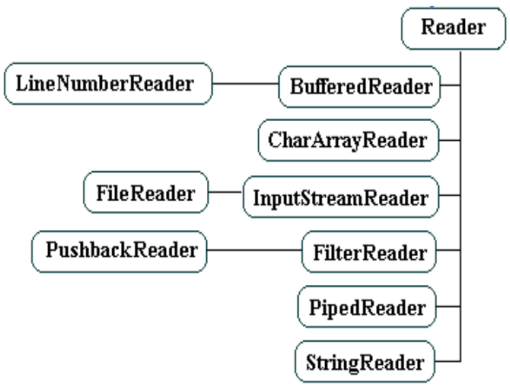
At the **top are two abstract** classes,

- Reader class
- Writer class .

Java has several concrete subclasses of each of these, as given in belo diagram



Reader Class Hiratchy



Some Important Classes are

1. FileReader
2. BufferedReader
3. InputStreamReader

1. FileReader

Only class capable of reading a disk file directly.
It read data as a sequence of 16 bit (2-byte) unicode characrer.
Suitable for reading languages other than English, for eg Chinese, Arabic etc.

Constructors

FileReader(File file) :
Creates a FileReader object by opening a connection to an actual file, the file named by the File object file in the file system.

Eg. opens a file input.txt to read

```
// create File object
File file = new File( "input.txt");
FileReader fin = new FileReader(file);
```

FileReader(String name)
Creates a FileReader by opening a connection to an actual file, the file named by the path name name in the file system

Eg. opens a file input.txt using file name to read instead of File object.

```
FileReader fin = new FileReader("input.txt");
```

Read functions and Description

*All below functions use **fin** object created above.*

int read()
Reads a character of data from this file reader.
Eg. read data using above created fis object

```
int data;
while ((data = fin.read()) != -1) {
        System.out.print((char)data);
}
```

int read(char[] cbuf)

```
char arr[] = new char[20];
int charRead = 0;
while ((charRead  = fin.read(arr)) != -1) {
    System.out.println(new String(arr));
}
```

Reads up to **charRead** bytes of data from this file reader
into an array of bytes.

int read(char[] b, int off, int len)
Reads up to **len** characters of data from this file reader into an array of bytes.

Eg. read data using above created fis object

```
char [] buffer = new char[1024]; // Define a buffer to hold the data
int charRead;
int offset = 0;
int length = 20; // Set the length to read
```

```
while (( charRead = fin.read(buffer, offset, length)) != -1) {
        // Convert the read bytes to a string and print them
        String output = new String(buffer, 0, charRead);
        System.out.println(output);
}
```

2. BufferedReader
is a Java class that reads text from an file, for eg, efficiently buffering the characters to improve performance. It's used to read data in large chunks rather than one character at a time, which can significantly speed up input operations, especially when reading from slower sources like files or network streams.

It does not read from the disk directly but uses **FileReader** as its underlying source to get data from Disk file.

Line-by-Line Reading: Provides a method **readLine()** that reads a full line of text at a time, making it easy to handle text input.

Note:
This is the **only class** that support **reading** as a **String** class , **String readLine()**

Constructors

BufferedReader(FileReader in)
Creates a buffering character-input stream that uses a default-sized input buffer.
BufferedReader(FileReader in, int sz)
Creates a buffering character-input stream that uses an input buffer of the specified size.

BufferedReader(FileReader in)
FileReader fis = new FileReader("malayalam.txt");
BufferedReader **reader** = new BufferedReader(fis);

Read functions and Description
*All below functions use **reader** object created above.*
int read()
int read(char[] array)
int read(char[] b, int off, int len)
String readLine()
Reads a line of text/String.

Eg, reads a malayalam file as lines/String.

```
String line;
while ((line = reader.readLine()) != null) {
        String str = line + "\n";
        System.out.println(str);
}
```

3. InputStreamReader
is a class in Java that serves as a **bridge between Byte streams and Character streams**. It reads bytes from an input source and decodes them into characters using a specified charset or the platform's default charset.

Converts byte streams (e.g., data from a file or network) into character streams.

Eg, reading a ASCII file, decoding into unicode character

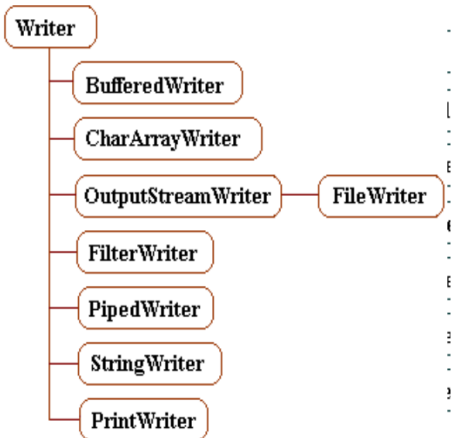
```
//create construcor
// FileInputStream is byte stream
InputStream inputStream = new FileInputStream("input.txt");

// its converted into Char stream, Specify character encoding UTF8
InputStreamReader inputStreamReader = new InputStreamReader(inputStream,
"UTF-8");

// Char Stream is wrapped by BufferReader
BufferedReader reader = new BufferedReader(inputStreamReader);

// reading file
String line;
while ((line = reader.readLine()) != null) {
        System.out.println(line);
}
```

Writer Class Hiratchy




```
Some Important classes
   1. FileWriter
   2. BufferedWriter
   3. OutputStreamWriter
```

1. FileWriter

Only class capable of writing a disk file directly.
It writes data as a sequence of 16 bit (2-byte) unicode characrer.
Suitable for writing Non ASCII languages, for Eg Chinese, Arabic etc.

Constructors

FileWriter(File file)
Creates a file output stream to write to the file represented by the specified File object. Open in write mode

```
File file = new File( "malayalam.txt");
FileWriter fw = new FileWriter(file);
```

FileWriter(File file, boolean append)
Open in write mode, if boolean append = false
Open in append mode, if boolean append = true

```
File file = new File( "malayalam.txt");
FileWriter fw = new FileWriter(file, append);
```

FileWriter(String name)
Creates a file output stream to write to the file represented by the specified file name.
FileWriter fw = new FileWriter(“malayalam.txt”);

FileWriter(String name, boolean append)
Creates a file output stream to write to the file with the specified file name.
Open in write mode, if boolean append = false
Open in append mode, if boolean append = true

FileWriter fw = new FileWriter("malayalam.txt", append);

If file doesnt exists, this will create an empty file with the given fine name.

write functions and Description

*All below functions use **fw** object created above.*

void write(int c)
void write(char[] cbuf)
void write(char[] cbuf, int off, int len)
void write(String cbuf)

void write(int c)
// Example 1: Write a single character using write(int c)
// int singleChar = 'ആ'; // Unicode for Malayalam character 'ആ'
//// also you can use corresponding hex values
int singleChar = 0x0D06; // Hexadecimal representation of ആ
fw.write(singleChar); // Writes the single character 'ആ'
fw.write('\n'); // Newline for clarity

void write(char[] cbuf)
// Example 2: Write an array of characters using write(char[] cbuf)
char[] verse1Array = "ആദിയിൽ ദൈവം ആകാശവും ഭൂമിയും സൃഷ്ടിച്ചു.\n".toCharArray();
// also you can use array of corresponding hex values
fw.write(verse1Array); // Writes the first verse as a char array

void write(char[] cbuf, int off, int len)
// Example 3: Write a portion of the character array using write(char[] cbuf, int off, int len)
// Writes first 10 characters of the first verse
fw.write(verse1Array, 0, 10);
fw.write('\n');

void write(String cbuf)
// Example 4: Write a full string using write(String str)
String verse2 = "ഭൂമി ശൂന്യവും പെരുപ്പും ആയിരുന്നു; അതിന്റെ മേൽ ഇരുട്ടും ആഴങ്ങളുടെ മേൽ ദൈവത്തിന്റെ ആത്മാവും ഉണ്ടായിരുന്നു.\n";
// also you can use array of corresponding hex values
// Writes the full string of verse 2
fw.write(verse2);

2. BufferedWriter
is a Java class that writes text from an file, for eg, efficiently buffering the characters to improve performance. It's used to write data in large chunks rather than one character at a time, which can significantly speed up input operations, especially when writing to slower sources like files or network streams.

It does not write tothe disk directly but uses **FileWriter** as its underlying source to purt data into Disk file.

Constructors

BufferedWriter(FileWriter in)
Creates a buffering character-input stream that uses a default-sized input buffer.
BufferedWriter(FileWriter in, int sz)
Creates a buffering character-input stream that uses an input buffer of the specified size.

***BufferedWriter*(*FileWriter* in)**
FileWriter fir = new FileWriter("malayalam.txt");
BufferedWriter **writer** = new BufferedReader(fis);

write functions and Description
*All below functions use **writer** object created above.*

void write(int c) : Writes a single character.
*void write(String s): w*rites a String.
*void write(String s, int off, int len): w*rites a portion of a String.
void write(char[] cbuf, int off, int len) : Writes a portion of an array of characters.

Eg. Write/append unicode char to a file.
void append(String str)
// Create a BufferedWriter wrapped around a FileWriter
BufferedWriter writer = new BufferedWriter(new FileWriter(fileName));

// Write a line of text
writer.write("Hello, World!");
writer.newLine(); // Using newLine() method to add a newline

// Write multiple lines using the append method
writer.append("This is a line added with append()").append("\n");
writer.write("This is another line.");

3. OutputStreamWriter
is a class in Java that bridges byte streams and character streams. It is used to write characters to an output stream by converting them into bytes using a specified character encoding.

Eg Program, char array encoded into UTF 8 and write unicode file
String fileName = "output.txt";

char[] charArray= "Genesis1:1 ആദിയിൽ ദൈവം ആകാശവും ഭൂമിയും സൃഷ്ടിച്ചു";

```
String textToWrite  =  charArray.toCharArray();

// Create a FileOutputStream to write to the file
FileOutputStream fos = new FileOutputStream(fileName);

// Create an OutputStreamWriter with UTF-8 encoding
OutputStreamWriter osw = new OutputStreamWriter(fos, "UTF-8");

// Write the text to the file
osw.write(textToWrite);
```

WORKING WITH FILES

1. Tiny Editor

Create a editor, that echows the keybord inputs and exit when “quit” is typed.

```
import java.io.*;
public class TinyEditor {
	public static void main(String[] args) {
		try {
			BufferedReader br = new BufferedReader(
					new InputStreamReader(System.in));
			System.out.println("Enter a line of text");
			System.out.println("Enter a quit to stop");

			String str[] = new String[100];
			for (int i = 0; i < 100; i++) {
				str[i] = br.readLine();
				if (str[i].equalsIgnoreCase("quit") == true) {
					break;
				}
			}
			System.out.println("Exit...!");
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

2, Read an ASCII file and write to another file – Use byte stream classes

```
import java.io.*;
public class BytesStremTest {
	public static void main(String[] args) {
		try {
			FileInputStream fis = new FileInputStream("input.txt");
			FileOutputStream fos = new FileOutputStream("output.txt");
			byte[] bArr = new byte[50];
			int b = 0;
			while ( (b = fis.read(bArr)) != -1) {
				System.out.println(new String(bArr));
				fos.write(bArr);
			}
```

```
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

3. Read any file and write to another file – Use char stream classes

```
import java.io.*;
public class CharStreamTest {
	public static void main(String[] args) {
		try {
			FileReader fis = new FileReader("malayalam.txt");
			BufferedReader reader = new BufferedReader(fis);

			FileWriter fos = new FileWriter("utf_output.txt");
			BufferedWriter writer = new BufferedWriter(fos);

			while ((line = reader.readLine()) != null) {
				System.out.println(line);
				writer.write(line);
			}
			writer.close();
			fos.close();

			reader.close();
			fis.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

4. Test DataInputStream & DataOutputStream

```
import java.io.*;
public class DataStreams {
	public static void main (String[] s) {
		try {
			//1. open a file
			File f1 = new File("out.dat");
			if (f1.exists() == false) {
				f1.createNewFile();
			}
			//2. create FileInputStream wrapped by DataInputStream
			FileOutputStream fos = new FileOutputStream(f1);
			DataOutputStream dout = new DataOutputStream(fos);

			//3. write primitive data (float, int, boolean, char)
			dout.writeDouble(3.14);
			dout.writeInt(100);
```

```
                dout.writeBoolean(true);
                dout.writeChar('A');

            // read data from the same file out.data
            //1. create FileOutputStream wrapped by DataOutputStream
            // using same file object f1

                    FileInputStream fis = new FileInputStream(f1);
                    DataInputStream din = new DataInputStream(fis);

                        //2 Read data
                    double dVal = din.readDouble();
                    int iVal = din.readInt();
                    boolean bVal = din.readBoolean();
                    char cVal = din.readChar();

                System.out.println("dVal = " + dVal);
                System.out.println("iVal = " + iVal);
                System.out.println("bVal = " + bVal);
                System.out.println("cVal = " + cVal);

                dout.close();
                fos.close();

                din.close();
                fis.close();
            } catch (IOException e) {
                    System.out.println(e.getMessage());
            }
        }
}
```

5. University exam question...!

Enter a string via command line argument. Write the string into a file. Read it from the file and verify is it a palindrome. ?

// using Character stream class

```
import java.io.*;
public class PalindromeCheck {
        public static void checkPalindrome(String fname) {
                boolean flag = true;
                try {
                        BufferedReader reader = new BufferedReader(new FileReader(fname));
                        if(reader != null) {
                                String line = reader.readLine();
                                if (line  != null) {
                                        int len = line.length();
                                        for (int i=0; i<len/2; i++) {
                                                if (line.charAt(i) != line.charAt(len-1-i)) {
                                                        flag = false;
                                                        break;
                                                }
```

```
                        }
                    }
                    reader.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
            if (flag == true) {
                System.out.println("Entered string  is a Palindrome...!");
            } else {
                System.out.println("Entered string  is NOT a Palindrome...!");
            }
        }

        public static void writeString(String fname, String line) {
            try {
                BufferedWriter writer = new BufferedWriter (new FileWriter(fname));
                if (writer != null) {
                    writer.write(line);
                    writer.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        public static void main(String[] args) {
            String line;
            if (args.length > 0) {
               line = args[0];
               System.out.println("String =" + line);

               writeString("test.txt", line);
               checkPalindrome("test.txt");
            }
        }
}
```

How to run

```
$ java PalindromeCheck malayalam
String : malayalam
Entered string  is a Palindrome...!
```

6. University exam question...!

Enter a string via command line argument. Write the string into a file. Read it from the file and verify is it a palindrome. ? Do it using Byte stream classes.

```
import java.io.*;
public class PalindromeCheck {
        public static void checkPalindromeEx(String fname) {
            boolean flag = true;
            try {
                FileInputStream fis  = new FileInputStream(fname);
```

```
                    if(fis != null) {
                        byte [] b = new byte [1024];
                        int bRead = fis.read(b);
                        if (bRead  > 0) {
                           String line = new String(b, 0, bRead);
                           int len = line.length();
                              System.out.println("Line = " + line);
                              System.out.println("length = " + len);
                              for (int i=0; i<len/2; i++) {
                                 if (line.charAt(i) != line.charAt(len-1-i)) {
                                    flag = false;
                                    break;
                                 }
                              }
                        }
                        fis.close();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
                if (flag == true) {
                    System.out.println("Entered string  is a Palindrome...!");
                } else {
                    System.out.println("Entered string  is NOT a Palindrome...!");
                }
        }

public static void writeStringEx(String fname, String line) {
                try {
                    FileOutputStream fos = new FileOutputStream (fname);
                    if (fos != null) {
                        fos.write(line.getBytes());
                        fos.close();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
        }
        public static void main(String[] args) {
                String line;
                if (args.length > 0) {
                   line = args[0];
                   System.out.println("String =" + line);
                   writeStringEx("test.txt", line);
                   checkPalindromeEx("test.txt");
                }
        }
}
```

Reading Console Input / Writing Console Output

Relevent classes

Reading Console Input classes	Writing Console Output classes
Scanner	*System.out (PrintStream)*
BufferedInputSream	*PrintWriter*
DataInputStream	*BufferedWriter*
BuferedReader	
InputStreamReader	

The Predefined Streams

System.in
System.out
System.err

Package java.lang defines a class called System.
System contains three predefined stream variables
– in, out, and err
These are standard input, output, error streams.
These are difined in Syatem class.

```
static InputStream  in;
static PrintStream  out;
static PrintStream  err;
```

Reading Console Input classes

1. **Scanner class**

package java.utils

```
/*------Class 1: Scanner class------*/
Scanner scanner = new Scanner(System.in);
System.out.print("Enter your name: ");
// Read a line of text from the console
String name = scanner.nextLine();
// Display the input received
System.out.println("Hello, " + name + "!");
/*--------------------------------------*/
```

2. BufferedInputSream class

package java.io

```
/*------Class 2: BufferedReader class------*/
BufferedInputStream bis = new BufferedInputStream(System.in);
System.out.print("Enter data : ");
byte[] b = new byte[100];
bis.read(b);
System.out.println("You entered : " + new String(b));
/*--------------------------------------*/
```

3. BufferedReader class

package java.io

```
/*------Class 3: BufferedReader class------*/
```

```
InputStreamReader Isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(Isr);
System.out.print("Enter your name: ");
try {
        name = br.readLine();
        System.out.println("Hello, " + name + "!");
} catch (IOException e) {
System.out.println(e.getMessage());
}
/*-------------------------------------------------*/
```

4. DataInputStream class

```
package java.io
/*------Class 4: DataInputStream class------*/
DataInputStream dis = new DataInputStream(System.in);
System.out.print("Enter your name: ");
try {
        name = dis.readLine();
} catch (IOException e) {
System.out.println(e.getMessage());
}
System.out.println("Hello, " + name + "!");
/*-------------------------------------------------*/
```

5. **InputStreamReader class**

```
package java.io
/*------Class 5: DataInputStream class------*/
InputStreamReader Isr1 = new InputStreamReader(System.in);
char[] c = new char[20];
try {
        Isr1.read(c);
        System.out.println("Hello, " + new String(c));
} catch (IOException e) {
System.out.println(e.getMessage());
}
/*-------------------------------------------------*/
```

Note

*Why do we **wrap** InputStreamReader in BufferedReader as below*

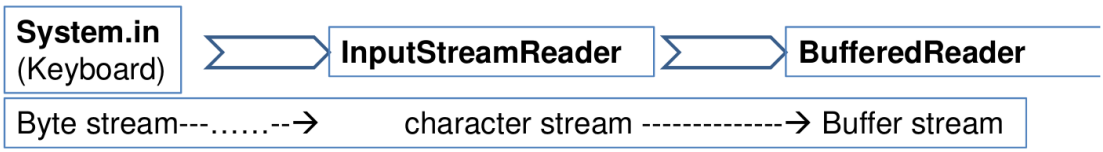
```
InputStreamReader Isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(Isr);
```

Ans:

refer

```
InputStreamReader Isr = new InputStreamReader(System.in);
```

*Here **System.in** is keyboard, which **byte stream**, that is transfer data in **bytes***
*BufferedReader is used to improve efficiency, but it works with **Character stream**, which can not process bytes.*
*Hence, InputStreamReader is **converts byte stream into character stream**, so that BufferedReader can be used to improve efficiently.*



InputStreamReaer in = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(in)

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Writing Console Output classes

1. **System.out (PrintStream) class**

package java.utils

That is, 'out' is an object of PrintStream class in System class.

```
/*Class 1: System.out------------*/
// Print with a newline
System.out.println("This is a message with a newline.");
// Print formatted text using printf
int number = 5;
System.out.printf("Formatted output: The number is %d \n", number);
//Print without a newline
System.out.print("This is a message without a newline.");
/*-----------------------------------*/
```

2. **PrintWriter class**

package java.io

```
/*Class 2: PrintWriter------------*/
PrintWriter pWriter = new PrintWriter(System.out, true);
pWriter.println("This is a message using PrintWriter.");
pWriter.printf("Formatted message: %d apples.%n", 5);
/*--------------------------------*/
```

3. **BufferedWriter class**

package java.io

```
/*Class 3: BufferedWriter------------*/
OutputStreamWriter Osw = new OutputStreamWriter(System.out);
BufferedWriter bWriter = new BufferedWriter(Osw);
try {
        bWriter.write("BufferedWriter example: writing to console.");
        bWriter.newLine();
        bWriter.write("This is another line.");
        bWriter.flush();
} catch (IOException e) {
System.out.println(e.getMessage());
}
```

PrintWritter Class

PrintWriter used for console output in Java

PrintWriter(System.out, true): This creates a PrintWriter object that wraps System.out. The second argument (true) ensures that the stream automatically flushes the buffer after each write operation (e.g., after println() or printf()).
writer.println(): This writes a line of text with a newline.
writer.printf(): This allows formatted text output (like System.out.printf()), useful for variables and formatted data.
writer.print(): This prints text without appending a newline at the end.

```
// Create a PrintWriter that writes to the console (System.out)
PrintWriter writer = new PrintWriter(System.out, true); // 'true' enables auto-flush

// Write a simple message to the console
writer.println("This is a message written using PrintWriter.");

// Use formatted output with printf (similar to System.out.printf)
writer.printf("Formatted output: %d apples and %d oranges.%n", 5, 10);

// Use print (without newline)
writer.print("This is a single line printed without a newline.");

// Close the PrintWriter (optional when using System.out)
writer.close();
```

Note
what will happen if set auto-flush is set to false as below
*PrintWriter writer = new PrintWriter(System.out, **false**); // 'true' enables auto-flush*
Ans:
Without flush(), Output Will Not Appear Until Closing:
*If you don't call **flush()** or **close()**, the output will remain buffered and will not appear on the console immediately.*

System.out Vs PrintWriter

System.out

- System.out is a byte stream.
- **System.out** refers to the standard output stream(monitor).
- **System:** It is a final class defined in the java.lang package.
- **out:** This is an instance of PrintStream type, which is a public and static member field of the System class.

PrintWritter

- **PrintWriter** should be used to write a stream of characters
- PrintWriter is a subclass of **Writer** (character stream class)
- It is used in real world programs to make it easier to internationalize the program

Object Streams and Serialization

Serialization is the process of writing(converting) the state/data of an object to a byte stream.

– This is useful when we want to
  • save(store) the state/data of the program to file or
  • when we want to send it over network.

– The ObjectOutputStream is used to serialize an object, for eg **student** object of **Student class** and write it to a file, for eg, **student.ser**

Deserialization.

– Deserialization converts the stored byte streams in file to object
  The ObjectInputStream is used to read the object back from the file.
  The byte stream is converted back into a Person object.

The Person class implements **Serializable**, which is required for objects to be serialized in Java. The **Serializable** interface has no methods or fields and serves only to identify the semantics of being serializable

Step 1: Create a class, for eg Sudent implementing Serializable

```
import java.io.*;
class Student implements Serializable {
   private int id;
   private String name;
   public Student(int id, String name) {
      this.id = id;
      this.name = name;
   }
   public int getId() {
      return id;
   }
   public String getName() {
      return name;
   }
}
```

Step 2: Create a test class to do serialization

```
public class SerializationExample {
   public static void main(String[] args) {
      // Serialize the object
      try {
            FileOutputStream fos = new FileOutputStream("student.ser");
            ObjectOutputStream out = new ObjectOutputStream(fos);
            Student student = new Student(1, "Alice");
            out.writeObject(student);
            System.out.println("Student object serialized and saved to student.ser");
      }catch (IOException e) {
            System.out.println(e.getMessage());
      }
```

```
        // Deserialize the object
        try {
                FileInputStream fis = new FileInputStream("student.ser");
                ObjectInputStream Oin= new ObjectInputStream(fis);
                Student deserializedStudent = (Student) Oin.readObject();
                System.out.println("Deserialized Student: ID - " +
                                        deserializedStudent.getId());
                System.out.println("Name - " + deserializedStudent.getName());
           } catch(IOException e) {
                System.out.println(e.getMessage());
           } catch (ClassNotFoundException e) {
                System.out.println(e.getMessage());
           }
    }
}
```