IO Programming

Content

- **Introduction to Stream,**
- **Byte Stream,**
- **Character stream,**
- **Readers and Writers,**
- **File Class,**
- **File InputStream,**
- **File Output Stream,**
- **InputStreamReader,**
- **OutputStreamWriter,**
- **FileReader,**
- **FileWriter,**
- **Buffered Reader.**

- **Java I/O** (Input and Output) is used *to process the input* and *produce the output*.
- Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.
- Stream
  - A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.
  - In Java, 3 streams are created for us automatically. All these streams are attached with the console.

    1) System.out: standard output stream

    2) System.in: standard input stream

    3) System.err: standard error stream

Byte Stream

- ByteStream classes are used to read bytes from the input stream and write bytes to the output stream. In other words, we can say that ByteStream classes read/write the data of 8-bits. We can store video, audio, characters, etc., by using ByteStream classes. These classes are part of the java.io package.
- The ByteStream classes are divided into two types of classes, i.e., InputStream and OutputStream. These classes are abstract and the super classes of all the Input/Output stream classes.
- InputStream Class
- The InputStream class provides methods to read bytes from a file, console or memory. It is an abstract class and can't be instantiated;

Character Streams

- The java.io package provides CharacterStream classes to overcome the limitations of ByteStream classes, which can only handle the 8-bit bytes and is not compatible to work directly with the Unicode characters. CharacterStream classes are used to work with 16-bit Unicode characters. They can perform operations on characters, char arrays and Strings.
- However, the CharacterStream classes are mainly used to read characters from the source and write them to the destination. For this purpose, the CharacterStream classes are divided into two types of classes, I.e., Reader class and Writer class.
- Reader Class
- Reader class is used to read the 16-bit characters from the input stream. However, it is an abstract class and can't be instantiated, but there are various subclasses that inherit the Reader class and override the methods of the Reader class. All methods of the Reader class throw an IOException. The subclasses of the Reader class are given in the following table.

Reader Class

- Java Reader is an abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods to provide higher efficiency, additional functionality, or both.
- Some of the implementation class are BufferedReader, CharArrayReader, Filter Reader, InputStreamReader, PipedReader, StringReader
- **Constructor**

protected	Reader()	It creates a new character-stream reader whose critical sections will synchronize on the reader itself.
protected	Reader(Object lock)	It creates a new character-stream reader whose critical sections will synchronize on the given object.

Methods

Modifier and Type	Method	Description
abstract void	close()	It closes the stream and releases any system resources associated with it.
void	mark(int readAheadLimit)	It marks the present position in the stream.
boolean	markSupported()	It tells whether this stream supports the mark() operation.
int	read()	It reads a single character.
int	read(char[] cbuf)	It reads characters into an array
abstract int	read(char[] cbuf, int off, int len)	It reads characters into a portion of an array.
int	read(CharBuffer target)	It attempts to read characters into the specified character buffer.
boolean	ready()	It tells whether this stream is ready to be read.
void	reset()	It resets the stream.
long	skip(long n)	It skips characters.

```
import java.io.*;
public class ReaderExample {
   public static void main(String[] args) {
      try {
         Reader reader = new FileReader("file.txt");
         int data = reader.read();
         while (data != -1) {
            System.out.print((char) data);
            data = reader.read();
         }
         reader.close();
      } catch (Exception ex) {
         System.out.println(ex.getMessage());
      }
   }
}
```

- Java Writer
- It is an abstract class for writing to character streams. The methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses will override some of the methods defined here to provide higher efficiency, functionality or both.
- **Constructor**

Modifier	Constructor	Description
protected	Writer()	It creates a new character-stream writer whose critical sections will synchronize on the writer itself.
protected	Writer(Object lock)	It creates a new character-stream writer whose critical sections will synchronize on the given object

Methods

Modifier and Type	Method	Description
Writer	append(char c)	It appends the specified character to this writer.
Writer	append(CharSequence csq)	It appends the specified character sequence to this writer
Writer	append(CharSequence csq, int start, int end)	It appends a subsequence of the specified character sequence to this writer.
abstract void	close()	It closes the stream, flushing it first.
abstract void	flush()	It flushes the stream.
void	write(char[] cbuf)	It writes an array of characters.
abstract void	write(char[] cbuf, int off, int len)	It writes a portion of an array of characters.
void	write(int c)	It writes a single character.
void	write(String str)	It writes a string.
void	write(String str, int off, int len)	It writes a portion of a string.

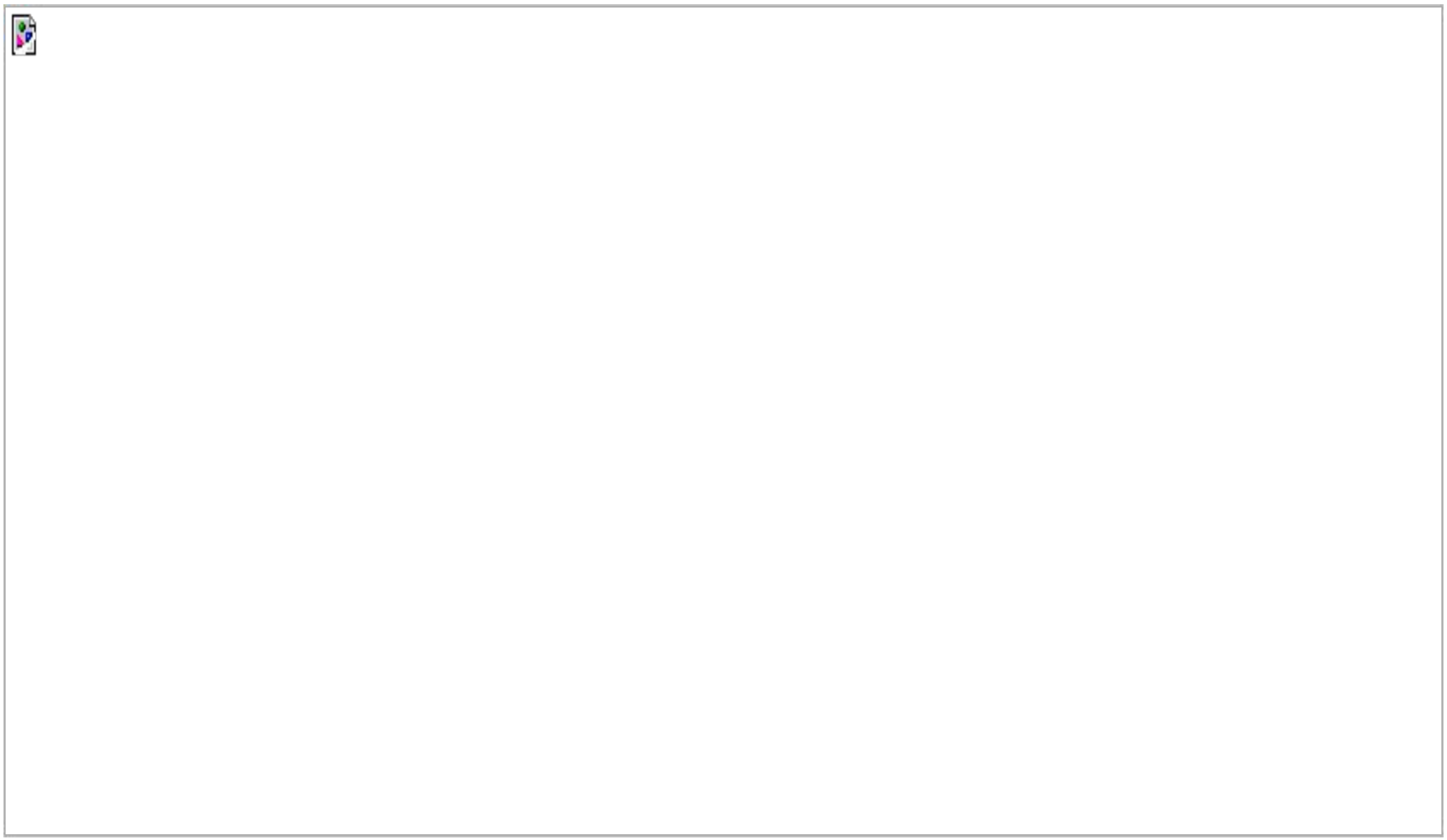
```
import java.io.*;
public class WriterExample {
    public static void main(String[] args) {
        try {
            Writer w = new FileWriter("output.txt");
            String content = "I love my country";
            w.write(content);
            w.close();
            System.out.println("Done");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

OutputStream vs InputStream

- OutputStream
- Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.
- InputStream
- Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

- OutputStream class
- OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.
- Useful methods of OutputStream

Method	Description
1) public void write(int)throws IOException	is used to write a byte to the current output stream.
2) public void write(byte[])throws IOException	is used to write an array of byte to the current output stream.
3) public void flush()throws IOException	flushes the current output stream.
4) public void close()throws IOException	is used to close the current output stream.



- InputStream class
- InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.
- Useful methods of InputStream

Method	Description
1) public abstract int read()throws IOException	reads the next byte of data from the input stream. It returns -1 at the end of the file.
2) public int available()throws IOException	returns an estimate of the number of bytes that can be read from the current input stream.
3) public void close()throws IOException	is used to close the current input stream.

File Class

- The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.
- The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.
- Fields

Modifier	Type	Field	Description
static	String	pathSeparator	It is system-dependent path-separator character, represented as a string for convenience.
static	char	pathSeparatorChar	It is system-dependent path-separator character.
static	String	separator	It is system-dependent default name-separator character, represented as a string for convenience.
static	char	separatorChar	It is system-dependent default name-separator character.

Constructor

Constructor	Description
File(File parent, String child)	It creates a new File instance from a parent abstract pathname and a child pathname string.
File(String pathname)	It creates a new File instance by converting the given pathname string into an abstract pathname.
File(String parent, String child)	It creates a new File instance from a parent pathname string and a child pathname string.
File(URI uri)	It creates a new File instance by converting the given file: URI into an abstract pathname.

Modifier and Type	Method	Description
static File	createTempFile(String prefix, String suffix)	It creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
boolean	createNewFile()	It atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
boolean	canWrite()	It tests whether the application can modify the file denoted by this abstract pathname.String[]
boolean	canExecute()	It tests whether the application can execute the file denoted by this abstract pathname.
boolean	canRead()	It tests whether the application can read the file denoted by this abstract pathname.
boolean	isAbsolute()	It tests whether this abstract pathname is absolute.
boolean	isDirectory()	It tests whether the file denoted by this abstract pathname is a directory.

boolean	isFile()	It tests whether the file denoted by this abstract pathname is a normal file.
String	getName()	It returns the name of the file or directory denoted by this abstract pathname.
String	getParent()	It returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
Path	toPath()	It returns a java.nio.file.Path object constructed from the this abstract path.
URI	toURI()	It constructs a file: URI that represents this abstract pathname.
File[]	listFiles()	It returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname
long	getFreeSpace()	It returns the number of unallocated bytes in the partition named by this abstract path name.
String[]	list(FilenameFilter filter)	It returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
boolean	mkdir()	It creates the directory named by this abstract pathname.

- Java File Example 1

```
import java.io.*;
public class FileDemo {
   public static void main(String[] args) {

      try {
         File file = new File("javaFile123.txt");
         if (file.createNewFile()) {
            System.out.println("New File is created!");
         } else {
            System.out.println("File already exists.");
         }
      } catch (IOException e) {
         e.printStackTrace();
      }
```

```
import java.io.*;
public class FileDemo2 {
   public static void main(String[] args) {

      String path = "";
      boolean bool = false;
      try {
         // createing  new files
         File file  = new File("testFile1.txt");
         file.createNewFile();
         System.out.println(file);
         // createing new canonical from file ob
     ject
         File file2 = file.getCanonicalFile();
         // returns true if the file exists
         System.out.println(file2);
         bool = file2.exists();
         // returns absolute pathname
         path = file2.getAbsolutePath();
         System.out.println(bool);
         // if file exists
         if (bool) {
            // prints
            System.out.print(path + " Exists?
 " + bool);
         }
      } catch (Exception e) {
         // if any error occurs
         e.printStackTrace();
      }
   }
}
```

Output:

```
testFile1.txt
/home/Work/Project/File/testFile1.txt
true
/home/Work/Project/File/testFile1.txt
Exists? true
```

```
import java.io.*;
public class FileExample {
public static void main(String[] args) {
    File f=new File("/Users/Vaidehi/Documents");
    String filenames[]=f.list();
    for(String filename:filenames){
        System.out.println(filename);
    }
}
}
```

```
import java.io.*;
public class FileExample {
public static void main(String[] args) {
    File dir=new File("/Users/sonoojaiswal/Documents");
    File files[]=dir.listFiles();
    for(File file:files){
       System.out.println(file.getName()+" Can Write: "+fil
    e.canWrite()+"
       Is Hidden: "+file.isHidden()+" Length: "+file.length()+
    " bytes");
    }
}
}
```

Java FileOutputStream Class

- Java FileOutputStream is an output stream used for writing data to a file.
- If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class.
- But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.
- FileOutputStream class declaration
  - **public class** FileOutputStream **extends** OutputStream

- FileOutputStream class methods

Method	Description
protected void finalize()	It is used to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.
FileChannel getChannel()	It is used to return the file channel object associated with the file output stream.
FileDescriptor getFD()	It is used to return the file descriptor associated with the stream.
void close()	It is used to closes the file output stream.

- Java FileOutputStream Example 1: write byte

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
    public static void main(String args[]){
          try{
            FileOutputStream fout=new FileOutputStream("D
    :\\testout.txt");
           fout.write(65);
           fout.close();
           System.out.println("success...");
          }catch(Exception e){System.out.println(e);}
     }
}
```

- Java FileOutputStream example 2: write string

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
    public static void main(String args[]){
          try{
            FileOutputStream fout=new FileOutputStream("D:\\testout.txt"
     );
            String s="Welcome to Indus University";
            byte b[]=s.getBytes();//converting string into byte array
            fout.write(b);
            fout.close();
            System.out.println("success...");
           }catch(Exception e){System.out.println(e);}
      }
}
```

Java FileInputStream Class

- Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class.
- Java FileInputStream class declaration
  - **public class** FileInputStream **extends** InputStream

Java FileInputStream class methods

Method	Description
int available()	It is used to return the estimated number of bytes that can be read from the input stream.
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to **b.length** bytes of data from the input stream.
int read(byte[] b, int off, int len)	It is used to read up to **len** bytes of data from the input stream.
long skip(long x)	It is used to skip over and discards x bytes of data from the input stream.
FileChannel getChannel()	It is used to return the unique FileChannel object associated with the file input stream.
FileDescriptor getFD()	It is used to return the FileDescrip to object.
protected void finalize()	It is used to ensure that the close method is call when there is no more reference to the file input stream.
void close()	It is used to closes the stream

- Java FileInputStream example 1: read single character

```
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\testout.txt");
          int i=fin.read();
          System.out.print((char)i);

          fin.close();
         }catch(Exception e){System.out.println(e);}
        }
       }
```

- Java FileInputStream example 2: read all characters

```
package com.javatpoint;

import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\testout.txt");
          int i=0;
          while((i=fin.read())!=-1){
           System.out.print((char)i);
          }
          fin.close();
         }catch(Exception e){System.out.println(e);}
        }
       }
```

Java InputStreamReader

- An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset.
- The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

Constructor name	Description
InputStreamReader(InputStream in)	It creates an InputStreamReader that uses the default charset.
InputStreamReader(InputStream in, Charset cs)	It creates an InputStreamReader that uses the given charset.
InputStreamReader(InputStream in, CharsetDecoder dec)	It creates an InputStreamReader that uses the given charset decoder.
InputStreamReader(InputStream in, String charsetName)	It creates an InputStreamReader that uses the named charset.

- Method

Modifier and Type	Method	Description
void	close()	It closes the stream and releases any system resources associated with it.
String	getEncoding()	It returns the name of the character encoding being used by this stream.
int	read()	It reads a single character.
int	read(char[] cbuf, int offset, int length)	It reads characters into a portion of an array.
boolean	ready()	It tells whether this stream is ready to be read.

```
public class InputStreamReaderExample {
   public static void main(String[] args) {
      try  {
         InputStream stream = new FileInputStream("file.txt");
         Reader reader = new InputStreamReader(stream);
         int data = reader.read();
         while (data != -1) {
            System.out.print((char) data);
            data = reader.read();
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Java OutputStreamWriter

- OutputStreamWriter is a class which is used to convert character stream to byte stream, the characters are encoded into byte using a specified charset.
- write() method calls the encoding converter which converts the character into bytes. The resulting bytes are then accumulated in a buffer before being written into the underlying output stream.
- The characters passed to write() methods are not buffered. We optimize the performance of OutputStreamWriter by using it with in a BufferedWriter so that to avoid frequent converter invocation.

Constructor

Constructor	Description
OutputStreamWriter(OutputStream out)	It creates an OutputStreamWriter that uses the default character encoding.
OutputStreamWriter(OutputStream out, Charset cs)	It creates an OutputStreamWriter that uses the given charset.
OutputStreamWriter(OutputStream out, CharsetEncoder enc)	It creates an OutputStreamWriter that uses the given charset encoder.
OutputStreamWriter(OutputStream out, String charsetName)	It creates an OutputStreamWriter that uses the named charset.

Methods

Modifier and Type	Method	Description
void	close()	It closes the stream, flushing it first.
void	flush()	It flushes the stream.
String	getEncoding()	It returns the name of the character encoding being used by this stream.
void	write(char[] cbuf, int off, int len)	It writes a portion of an array of characters.
void	write(int c)	It writes a single character.
void	write(String str, int off, int len)	It writes a portion of a String

```
public class OutputStreamWriterExample {
    public static void main(String[] args) {

        try {
            OutputStream outputStream = new FileOutputStream("output
    .txt");
            Writer outputStreamWriter = new OutputStreamWriter(outpu
    tStream);

            outputStreamWriter.write("Hello World");

            outputStreamWriter.close();
        } catch (Exception e) {
            e.getMessage();
        }
    }
}
```

Java FileWriter Class

- Java FileWriter class is used to write character-oriented data to a file. It is character-oriented class which is used for file handling in java.
- Unlike FileOutputStream class, you don't need to convert string into byte array because it provides method to write string directly.
- Java FileWriter class declaration
  - **public class** FileWriter **extends** OutputStreamWriter

Methods of FileWriter class

Constructor	Description
FileWriter(String file)	Creates a new file. It gets file name in string.
FileWriter(File file)	Creates a new file. It gets file name in File object.

Method	Description
void write(String text)	It is used to write the string into FileWriter.
void write(char c)	It is used to write the char into FileWriter.
void write(char[] c)	It is used to write char array into FileWriter.
void flush()	It is used to flushes the data of FileWriter.
void close()	It is used to close the FileWriter.

```
import java.io.FileWriter;
public class FileWriterExample {
    public static void main(String args[]){
        try{
          FileWriter fw=new FileWriter("D:\\testout.txt");
          fw.write("Welcome to javaTpoint.");
          fw.close();
         }catch(Exception e){System.out.println(e);}
         System.out.println("Success...");
    }
}
```

Java FileReader Class

- Java FileReader class is used to read data from the file. It returns data in byte format like FileInputStream class.
- It is character-oriented class which is used for file handling in java.
- Java FileReader class declaration
  - **public class** FileReader **extends** InputStreamReader

Constructor	Description
FileReader(String file)	It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.
FileReader(File file)	It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.

Method	Description
int read()	It is used to return a character in ASCII form. It returns -1 at the end of file.
void close()	It is used to close the FileReader class.

```
import java.io.FileReader;
public class FileReaderExample {
   public static void main(String args[])throws Exception{

        FileReader fr=new FileReader("D:\\testout.txt");
        int i;
        while((i=fr.read())!=-1)
        System.out.print((char)i);
        fr.close();
   }
}
```

Java BufferedWriter Class

- Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits Writer class. The buffering characters are used for providing the efficient writing of single arrays, characters, and strings.
- Class declaration
- Java.io.BufferedWriter class:
- **public class** BufferedWriter **extends** Writer

Constructor

Constructor	Description
BufferedWriter(Writer wrt)	It is used to create a buffered character output stream that uses the default size for an output buffer.
BufferedWriter(Writer wrt, int size)	It is used to create a buffered character output stream that uses the specified size for an output buffer.

Methods

Method	Description
void newLine()	It is used to add a new line by writing a line separator.
void write(int c)	It is used to write a single character.
void write(char[] cbuf, int off, int len)	It is used to write a portion of an array of characters.
void write(String s, int off, int len)	It is used to write a portion of a string.
void flush()	It is used to flushes the input stream.
void close()	It is used to closes the input stream

```
import java.io.*;
public class BufferedWriterExample {
public static void main(String[] args) throws Exception {

    FileWriter writer = new FileWriter("D:\\testout.txt");
    BufferedWriter buffer = new BufferedWriter(writer);
    buffer.write("Welcome to Indus University.");
    buffer.close();
    System.out.println("Success");
    }
}
```

Java BufferedReader Class

- Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits ReaderClass.
- Java BufferedReader class declaration
- Java.io.BufferedReader class:
- **public class** BufferedReader **extends** Reader

Constructor

Constructor	Description
BufferedReader(Reader rd)	It is used to create a buffered character input stream that uses the default size for an input buffer.
BufferedReader(Reader rd, int size)	It is used to create a buffered character input stream that uses the specified size for an input buffer.

Methods

Method	Description
int read()	It is used for reading a single character.
int read(char[] cbuf, int off, int len)	It is used for reading characters into a portion of an array.
boolean markSupported()	It is used to test the input stream support for the mark and reset method.
String readLine()	It is used for reading a line of text.
boolean ready()	It is used to test whether the input stream is ready to be read.
long skip(long n)	It is used for skipping the characters.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readAheadLimit)	It is used for marking the present position in a stream.
void close()	It closes the input stream and releases any of the system resources associated with the stream.

```
import java.io.*;
public class BufferedReaderExample {
   public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("D:\\testout.txt");
        BufferedReader br=new BufferedReader(fr);

        int i;
        while((i=br.read())!=-1){
        System.out.print((char)i);
        }
        br.close();
        fr.close();
   }
}
```

CJP PAPER - vaidehipatel.ce@inc | CJP 4th CE-B & IT-B B | Meet – qwg-fvzr-tkw

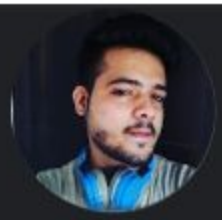
meet.google.com/qwg-fvzr-tkw?authuser=0

8 09:23

Dattani Shrey

P

Patel Maitri



PRUTHVI TIRMAL

Jaydip Thakkar

HARSH THAKER

Ridham Jivani

AAYUSHI TALAVIYA

d5cyuijyb5

Raise hand

Present now