Practical - 8 File handling in Java

There are several **File Operations** like **creating a new File, getting information about File, writing into a File, reading from a File** and **deleting a File**.
A series of data is referred to as **a stream**. In Java, **Stream** is classified into two types, i.e., **Byte Stream** and **Character Stream**.

- **File MANAGEMENT :**
    - **Create a File :**

File(String *directoryPath*, String *filename*)
File defines many methods that obtain the standard properties of a **File** object.

canRead()	Boolean	Tests whether the file is readable or not
canWrite()	Boolean	Tests whether the file is writable or not
createNewFile()	Boolean	Creates an empty file
delete()	Boolean	Deletes a file
exists()	Boolean	Tests whether the file exists
getName()	String	Returns the name of the file
getAbsolutePath()	String	Returns the absolute pathname of the file
length()	Long	Returns the size of the file in bytes
list()	String[]	Returns an array of the files in the directory
mkdir()	Boolean	Creates a directory

e.g checking a file properties

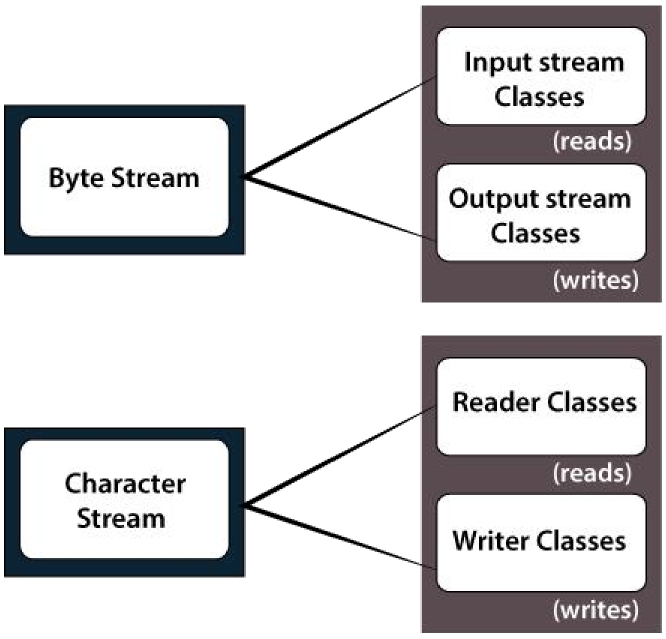
```
import java.io.File;
class FileDemo {
static void p(String s) {
System.out.println(s);
}
public static void main(String args[]) {
```

```
File f1 = new File("../basic/Arraytest.java");
  p("File Name: " + f1.getName());
  p("Path: " + f1.getPath());
  p("Abs Path: " + f1.getAbsolutePath());
  p("Parent: " + f1.getParent());
  p(f1.exists() ? "exists" : "does not exist");
  p(f1.canWrite() ? "is writeable" : "is not writeable");
  p(f1.canRead() ? "is readable" : "is not readable");
  p("is " + (f1.isDirectory() ? "" : "not" + " a directory"));
  p(f1.isFile() ? "is normal file" : "might be a named pipe");
  p("File last modified: " + f1.lastModified());
  p("File size: " + f1.length() + " Bytes");

  f1 = new File("../basic/Arraytest.class");
  p("deleted:" + f1.delete());
}
}
```

Reading and writing into the file

Java's stream-based I/O is built upon four abstract classes: **InputStream**, **OutputStream**, **Reader**, and **Writer**



Brief classification of I/O streams

Byte Stream is mainly involved with byte data. A file handling process with a byte stream is a process in which an input is provided and executed with the byte data.

Character Stream is mainly involved with character data. A file handling process with a character stream is a process in which an input is provided and executed with the character data.

- **InputStream and Outputstream**
  it is abstract class for input/read a byte stream
  It is abstract class for ouput/write a bytestream
  Most of the method thows the IOException.

- **FileInputStrem and FileOutputStream**

FileInputstream read the bytes from the file and extends the InputStream while FileOutputstream write the bytes into the file and extends the OutputStream.

FileInputStream Methods

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.
void close()	It is used to closes the stream.

FileOutputSteam Methods

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
void close()	It is used to closes the file output stream.

E. g Program for reading and writing a bytes from the file using FileInputStream and FileOutputStream

```
import java.io.*;
public class Filereadwritedemo {
   public static void main(String args[]){
        try{
          FileOutputStream fout=new FileOutputStream("./test.txt");
          String str = "testing of file demo";
          byte buf[] = str.getBytes();
```

```
            for(int i =0;i<buf.length;i++) {
                fout.write(buf[i]);
            }
           // fout.write("abc");
            fout.close();
            System.out.println("success...");

        // Reading data from file
            FileInputStream fin=new FileInputStream("./test.txt");
             int n;
             while((n=fin.read())!=-1){
                System.out.print((char)n);
             }
           } // end of try block
           catch(IOException e){System.out.println(e);}
      }
}
```

DataInputStream and DataOutputSteam

DataInputStream class allows an application to read primitive data from the input stream in a machine-independent way.
DataOutputSteam class allow to write a primitive data from output stream in machine independent way.

DataInputStream Method

int read(byte[] b)	It is used to read the number of bytes from the input stream.
int read(byte[] b, int off, int len)	It is used to read **len** bytes of data from the input stream.
int readInt()	It is used to read input bytes and return an int value.
byte readByte()	It is used to read and return the one input byte.
char readChar()	It is used to read two input bytes and returns a char value.
double readDouble()	It is used to read eight input bytes and returns a double value.
boolean readBoolean()	It is used to read one input byte and return true if byte is non zero, false if byte is zero.

DataOutputStream Method

int size()	It is used to return the number of bytes written to the data output stream.
void write(int b)	It is used to write the specified byte to the underlying output stream.
void write(byte[] b, int off, int len)	It is used to write len bytes of data to the output stream.

void writeBoolean(boolean v)	It is used to write Boolean to the output stream as a 1-byte value.
void writeChar(int v)	It is used to write char to the output stream as a 2-byte value.
void writeChars(String s)	It is used to write string to the output stream as a sequence of characters.
void writeByte(int v)	It is used to write a byte to the output stream as a 1-byte value.
void writeBytes(String s)	It is used to write string to the output stream as a sequence of bytes.
void writeInt(int v)	It is used to write an int to the output stream
void writeShort(int v)	It is used to write a short to the output stream.
void writeLong(long v)	It is used to write a long to the output stream.
void flush()	It is used to flushes the data output stream.

e.g Writing and Reading a integer values into the file.

```
class Datainoutstream
 {
   public static void main(String[] args) throws IOException {

  try{
      FileOutputStream file = new FileOutputStream("./test1.txt");
      DataOutputStream data = new DataOutputStream(file);
      int arr[] = {12,23,43,666,8,89};
      for(int i =0 ; i<arr.length;i++){
         data.writeInt(arr[i]);
       }
      data.close();

    // Reading data from file
       FileInputStream input = new FileInputStream("./test1.txt");
       DataInputStream din = new DataInputStream(input);
       //Count the total bytes form the input stream

        while(din.available()>0)
         {
           System.out.print(" "+ din.readInt());
         }
     }  catch(Exception e) {System.out.println(e);}

   }
 }
```

- ## Reader and Writer class to handle the Charcter/Text data of file

Reader is abstract class which read the char data from file. Writer is abstract class which write char data into file.

- ## **BufferedReader and BufferedWriter**

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.
Constructor
BufferedReader(Reader rd)

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.
String readLine()	It is used for reading a line of text.
long skip(long n)	It is used for skipping the characters.
void close()	It closes the input stream and releases any of the system resources associated with the stream.

BufferedWriter methods
Constructor BufferedWriter(Writer wrt)

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

e.g Reading and writing text data using BufferedReader and bufferedWriter

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

  FileWriter writer = new FileWriter("./testreader.txt");
  BufferedWriter buffer = new BufferedWriter(writer);
  buffer.write("Welcome to new file .");
  String str = "This is the new line to be added";
  buffer.newLine();
  //from 5 the index it write 10 character
  // is the new
  buffer.write(str,5,10);
```

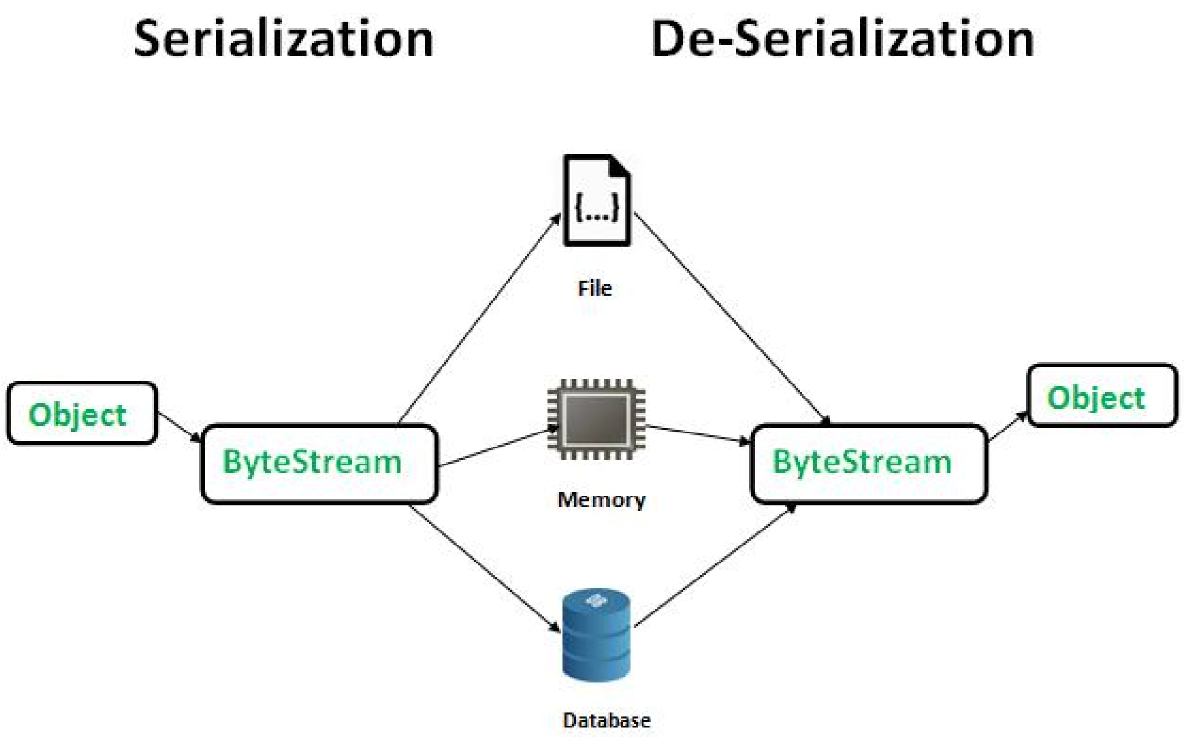
```
    buffer.close();
    System.out.println("Success");

    FileReader reader = new FileReader("./testreader.txt");
    BufferedReader buffer1 = new BufferedReader(reader);
    String s;
    /*while((s=  buffer1.readLine()) != null) {
    System.out.println(s);
     }*/
     System.out.println("other way");
      int c;
      while((c=buffer1.read())>0){
       System.out.print((char) c);
      }

    }
}
```

Serialization and Deserialization

Serialization is a mechanism of converting the state of an object into a byte stream.



Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory.

To make a Java object serializable we implement the java.io.Serializable **interface.**

- The ObjectOutputStream class contains **writeObject()** method for serializing an Object.
- This mechanism is used to persist the object. Over the network
- public final void writeObject(Object obj)
  throws IOException
- The ObjectInputStream class contains **readObject()** method for deserializing an object.

- public final Object readObject()
  throws IOException,
  ClassNotFoundException

- **Only non-static data members are saved via Serialization process.**

e.g

```
// Java code for serialization and deserialization
// of a Java object
import java.io.*;

class Demo implements Serializable
{
    public int a;
    public String b;

    // Default constructor
    public Demo(int a, String b)
    {
        this.a = a;
        this.b = b;
    }

}

class Serialdemo
{
    public static void main(String[] args)
    {
        Demo object = new Demo(1, "this");
        Demo object1 = new Demo(2, "that");

        String filename = "file.ser";

        // Serialization
        try
        {
            //Saving of object in a file
            FileOutputStream file = new FileOutputStream(filename);
            ObjectOutputStream out = new ObjectOutputStream(file);

            // Method for serialization of object
            out.writeObject(object);
            out.writeObject(object1);
            out.close();
            file.close();

            System.out.println("Object has been serialized");

        }

        catch(IOException ex)
        {
            System.out.println("IOException is caught");
        }


        Demo object2 = null;

        // Deserialization
        try
        {
            // Reading the object from a file
            FileInputStream file = new FileInputStream(filename);
            ObjectInputStream in = new ObjectInputStream(file);

            // Method for deserialization of object
            object2 = (Demo)in.readObject();

            in.close();
            file.close();

            System.out.println("Object has been deserialized ");
            System.out.println("a = " + object1.a);
            System.out.println("b = " + object1.b);
        }

        catch(IOException ex)
        {
            System.out.println("IOException is caught");
        }

        catch(ClassNotFoundException ex)
        {
            System.out.println("ClassNotFoundException is caught");
        }

    }
}
```

Exercise

1	Write a java program to Create a File and write an integer number.read the integer numbers from file and display the total of it.
2	Write a java program perform which write a text into the file and read the content of file. Whenever the character ‘d’ finds it skip the character. And display the content. Use BufferedReader and

	BufferedWriter class
3	Create class Item with itemno,name,price. Write a program to store the object into the file. and read the object from the file.