Java I/O

- 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.
- We can perform file handling in Java by Java I/O API.

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.

OutputStream vs InputStream

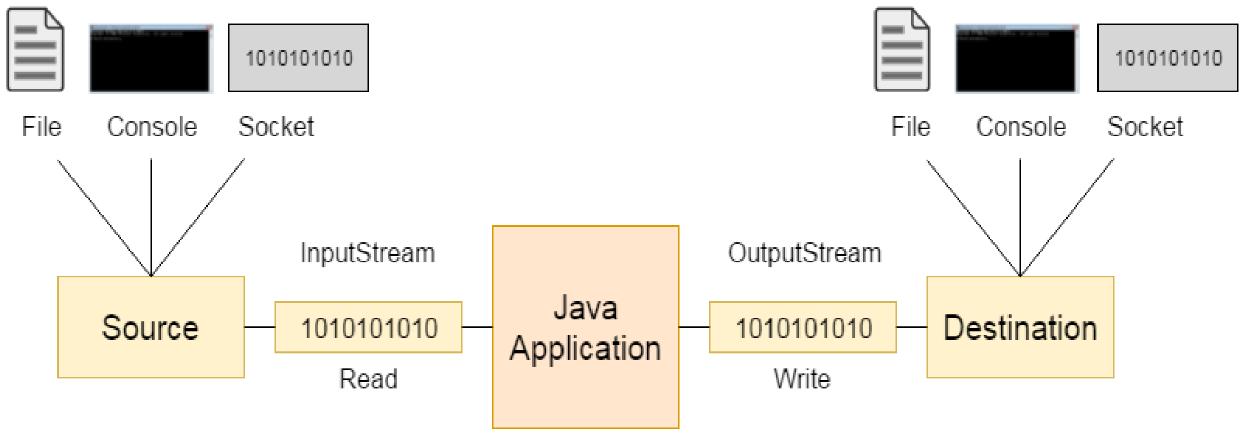
OutputStream

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.

Let's understand the working of Java OutputStream and InputStream by the figure given below.



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.

InputStream Hierarchy

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 outputstream.
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);}

      }

}
```

Output:

```
Success...
```

The content of a text file testout.txt is set with the data A. A

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 javaTpoint.";
             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);}
      }
}
```

Output:

```
Success...
```

The content of a text file testout.txt is set with the data Welcome to java. testout.txt

Welcome to java.

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 FileDescriptor 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);}
         }
        }
```

Note: Before running the code, a text file named as "testout.txt" is required to be created. In this file, we are having following content: Welcome to java

Welcome to java

After executing the above program, you will get a single character from the file which is 87 (in byte form). To see the text, you need to convert it into character.

Output:

```
W
```

Java FileInputStream example 2: read all characters

```
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);}
         }
        }
```

Output: Welcome to java

Java 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.

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();

        }

    }

}
```

Output:

```
New File is created!
```

Java - RandomAccessFile

- This class is used for reading and writing to random access file.
- A random access file behaves like a large array of bytes.
- There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations.
- If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

Method

Modifier and Type	Method	Method
void	close()	It closes this random access file stream and releases any system resources associated with the stream.
FileChannel	getChannel()	It returns the unique FileChannel object associated with this file.
int	readInt()	It reads a signed 32-bit integer from this file.
String	readUTF()	It reads in a string from this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
void	writeDouble(double v)	It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first.
void	writeFloat(float v)	It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first.
void	write(int b)	It writes the specified byte to this file.
int	read()	It reads a byte of data from this file.
long	length()	It returns the length of this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.

Example

```
import java.io.IOException; import
java.io.RandomAccessFile;
public class RandomAccessFileExample
{
    static final String FILEPATH ="myFile.txt"; public
    static void main(String[] args)
  {  try {
            System.out.println(new String(readFromFile(FILEPATH, 0, 18)));
            writeToFile(FILEPATH, "I love my country and my people", 31);
             } catch (IOException e) { e.printStackTrace();
        }
    }
    private static byte[] readFromFile(String filePath, int position, int size) throws
            IOException {
        RandomAccessFile file = new RandomAccessFile(filePath, "r");
        file.seek(position);
        byte[] bytes = new byte[size];
        file.read(bytes);
        file.close();
        return bytes;
    }
    private static void writeToFile(String filePath, String data, int position) throws
            IOException {
        RandomAccessFile file = new RandomAccessFile(filePath, "rw");
        file.seek(position);
        file.write(data.getBytes());
        file.close();
    }
}
```

The myFile.TXT contains text "This class is used for reading and writing to random access file."

after running the program it will contains

```
This class is used for reading I love my country and my people.
```