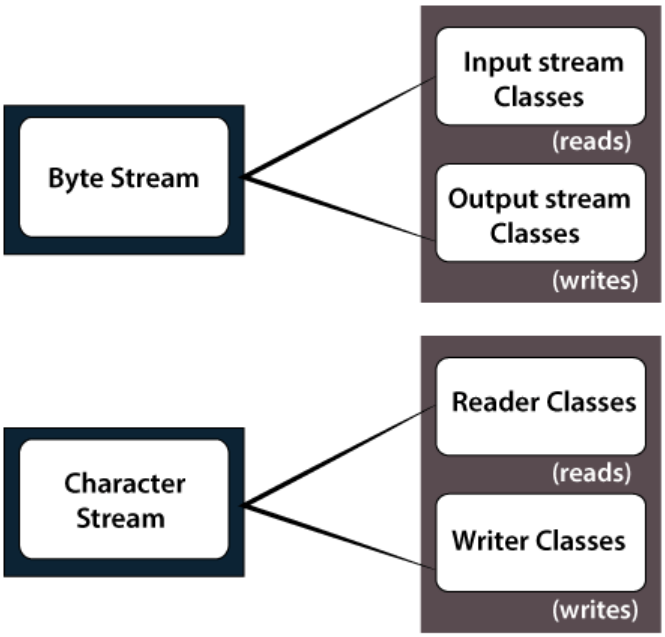
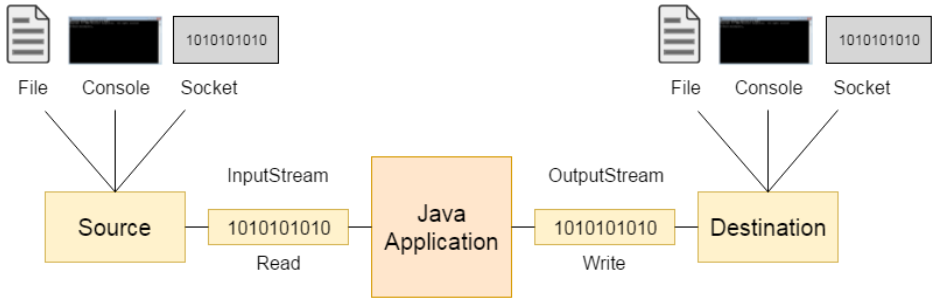
1. Introduction to Stream
    - In Java, a File is an abstract data type. A named location used to store related information is known as a File. There are several File Operations like creating a new File, getting information about File, writing into a File, reading from a File and deleting a File.
    - Stream
        - 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.

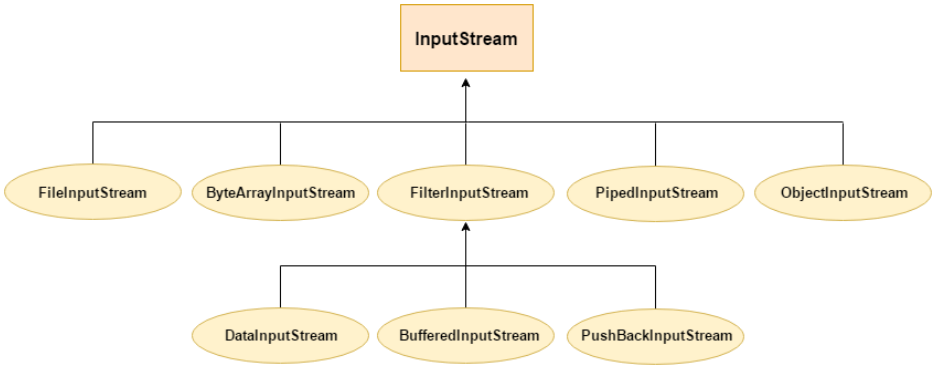


Brief classification of I/O streams

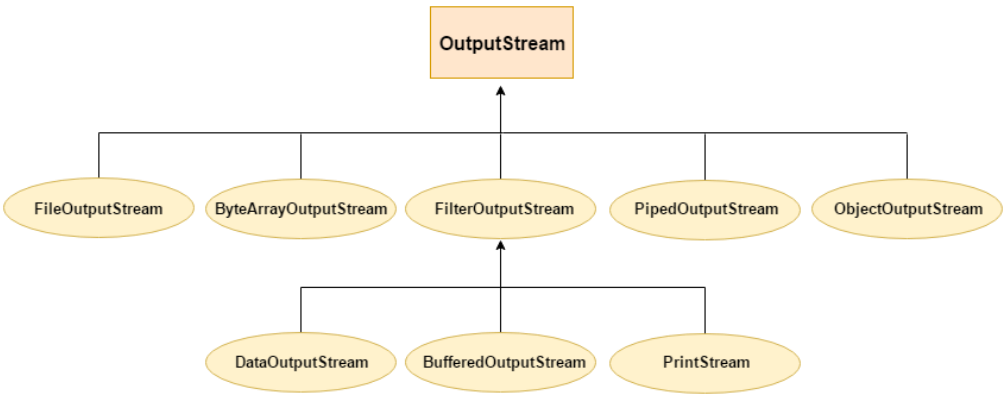
- Java I/O (Input and Output) is used to process the input and produce the output.
- In Java, 3 streams are created for us automatically. All these streams are attached with the console.
    - System.out: standard output stream
    - System.in: standard input stream
    - System.err: standard error stream
- InputStream vs OutputStream
    - Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.
    - Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.



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



- 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.



2. Byte Stream
    - 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.
3. Character stream
    - 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.
4. Readers and Writers
    - Readers
        - 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, FilterReader, InputStreamReader, PipedReader, StringReader.

Ex.

```
import java.io.*;
public class ReaderExample {
   public static void main(String[] args) {
      try {
         Reader reader = new FileReader("D:\\subjects\\JAVA\\2022
23\\lectcode\\Unit 6\\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());
      }
   }
}
```

Output: Hello ADIT Students

- Writers
  - 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.

    Ex.

```
import java.io.*;
public class WriterExample {
   public static void main(String[] args) {
      try {
         Writer w = new FileWriter("D:\\subjects\\JAVA\\2022 23\\lectcode\\Unit
6\\output.txt");
         String content = "I am from CVMU";
         w.write(content);
         w.close();
         System.out.println("Done");
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

5. 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.

Ex.

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

    try {
      File file = new File("D:\\subjects\\JAVA\\2022 23\\lectcode\\Unit 6\\MyFile.txt");
      if (file.createNewFile()) {
        System.out.println("New File is created!");
      } else {
        System.out.println("File already exists.");
      }
    } catch (IOException e) {
      e.printStackTrace();
    }

  }
}
```

6. FileOutputStream
   - 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.

Ex.

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
  public static void main(String args[]){
      try{
       FileOutputStream fout=new FileOutputStream("D:\\subjects\\JAVA\\2022
23\\lectcode\\Unit 6\\testout.txt");
       fout.write(65);
       fout.close();
       System.out.println("success...");
      }catch(Exception e){System.out.println(e);}
   }
}
```

Ex.

```
import java.io.FileOutputStream;
public class FileOutputStreamExample2 {
  public static void main(String args[]){
      try{
```

```
        FileOutputStream fout=new FileOutputStream("D:\\subjects\\JAVA\\2022
23\\lectcode\\Unit 6\\testout.txt");
        String s="Welcome to CVMU at ADIT.";
        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);}
    }
}
```

7. FileInputStream
   - 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.

```
Ex.
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
         FileInputStream fin=new FileInputStream("D:\\subjects\\JAVA\\2022
23\\lectcode\\Unit 6\\testout.txt");
         int i=fin.read();
        while(i!=-1){
         System.out.print((char)i);
         i=fin.read();
        }

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

8. FileWriter
   - 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.

```
Ex.
import java.io.FileWriter;
public class FileWriterExample {
   public static void main(String args[]){
       try{
         FileWriter fw=new FileWriter("D:\\subjects\\JAVA\\2022 23\\lectcode\\Unit
6\\testout.txt");
         fw.write("Wingardium Leviosa");
         fw.close();
        }catch(Exception e){System.out.println(e);}
```

```
        System.out.println("Success...");
    }
}
```

9. FileReader
    - Java FileReader class is used to read data from the file. It returns data in byte format like FileInputStream class.

    Ex.

```
import java.io.FileReader;
public class FileReaderExample {
   public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("D:\\subjects\\JAVA\\2022 23\\lectcode\\Unit 6\\testout.txt");
        int i;
        while((i=fr.read())!=-1)
        System.out.print((char)i);
        fr.close();
   }
}
```

10. InputStreamReader
    - An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset.

    Ex.

```
import java.io.*;
public class InputStreamReaderExample {
   public static void main(String[] args) {
      try  {
         InputStream stream = new FileInputStream("D:\\subjects\\JAVA\\2022 23\\lectcode\\Unit 6\\testout.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();
      }
   }
}
```

11. Scanner
    - Scanner class in Java is found in the java.util package. Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.
    - The Java Scanner class breaks the input into tokens using a delimiter which is whitespace by default.

    Ex.

```
import java.util.*;
public class ScannerExample {
public static void main(String args[]){
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = in.nextLine();
        System.out.println("Name is: " + name);
        in.close();
        }
}
```

12. PrintWriter class
    - Java PrintWriter class is the implementation of Writer class. It is used to print the formatted representation of objects to the text-output stream.

    Ex.

```
import java.io.PrintWriter;

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

    String data = "This is a text inside the file.";

    try {
      PrintWriter output = new PrintWriter("D:\\subjects\\JAVA\\2022 23\\lectcode\\Unit 6\\output.txt");

      output.print(data);
      output.close();
    }
    catch(Exception e) {
      e.getStackTrace();
    }
  }
}
```

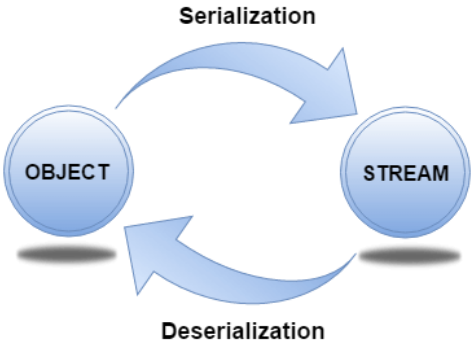
13. BuffredReader
    - Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits Reader class.

```
Ex.
import java.io.*;
public class BufferedReaderExample {
   public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("D:\\subjects\\JAVA\\2022 23\\lectcode\\Unit 6\\output.txt");
        BufferedReader br=new BufferedReader(fr);

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

14. Object Serialization

- Serialization in Java is a mechanism of writing the state of an object into a byte-stream.
- The reverse operation of serialization is called deserialization where byte-stream is converted into an object.
- The serialization and deserialization process is platform-independent, it means you can serialize an object on one platform and deserialize it on a different platform.
- For serializing the object, we call the writeObject() method of ObjectOutputStream class, and for deserialization we call the readObject() method of ObjectInputStream class.



- java.io.Serializable interface
    - The Serializable interface must be implemented by the class whose object needs to be persisted.

- The String class and all the wrapper classes implement the java.io.Serializable interface by default.

- ObjectOutputStream class
  - The ObjectOutputStream class is used to write primitive data types, and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.
- ObjectInputStream class
  - An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

Ex. Object serialization

```
import java.io.Serializable;
import java.io.*;


class Student implements Serializable
{
 int id;
 String name;
 public Student(int id, String name) {
  this.id = id;
  this.name = name;
 }
}
public class Persist{
 public static void main(String args[]){
  try{
  //Creating the object
  Student s1 =new Student(211,"ravi");
  //Creating stream and writing the object
  FileOutputStream fout=new FileOutputStream("D:\\subjects\\JAVA\\2022
23\\lectcode\\Unit 6\\f.txt");
  ObjectOutputStream out=new ObjectOutputStream(fout);
  out.writeObject(s1);
  out.flush();
  //closing the stream
  out.close();
  System.out.println("success");
  }catch(Exception e){System.out.println(e);}
 }
}
```

Ex. Object deserialization

```
import java.io.Serializable;
import java.io.*;

class Student implements Serializable
{
 int id;
 String name;
 public Student(int id, String name) {
  this.id = id;
  this.name = name;
 }
}

public class Depersist{
 public static void main(String args[]){
  try{
  //Creating stream to read the object
  ObjectInputStream in=new ObjectInputStream(new
FileInputStream("D:\\subjects\\JAVA\\2022 23\\lectcode\\Unit 6\\f.txt"));
  Student s=(Student)in.readObject();
  //printing the data of the serialized object
  System.out.println(s.id+" "+s.name);
  //closing the stream
  in.close();
  }catch(Exception e){System.out.println(e);}
 }
}
```